Interfacing a 4x4 Keyboard to an AT91 Microcontroller

Introduction

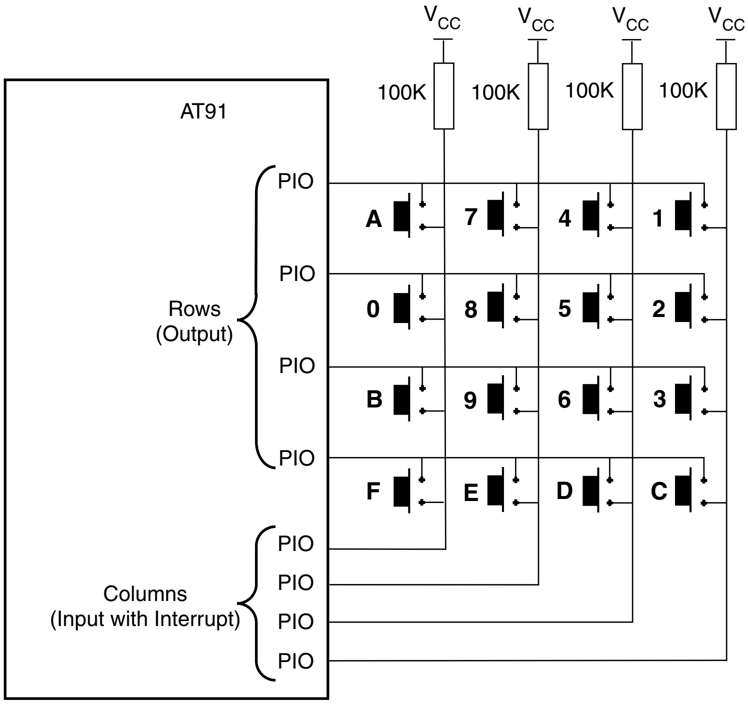
This Application Note describes programming techniques implemented on the AT91 ARM-based microcontroller for scanning a 4x4 Keyboard matrix usually found in both consumer and industrial applications for numeric data entry.

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.

Figure 1. Keyboard Interface



AT91 ARM® Thumb® Microcontrollers

Application Note

Rev. 2669A–ATARM–01/03

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

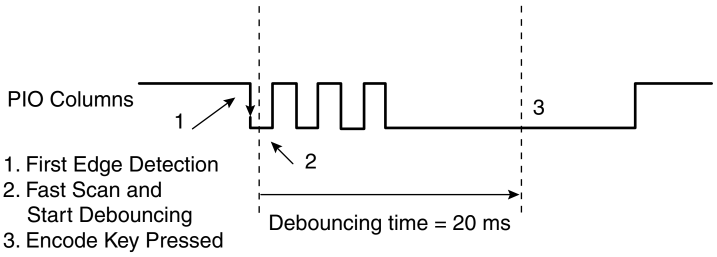
The Keyboard used is a 4x4 matrixed Keyboard. Columns are connected to pins configured as inputsand having the input change interrupt enabled. The initial state of these pins is high level due to four external pull-up resistors.

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



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

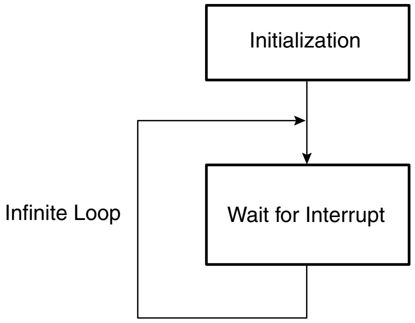
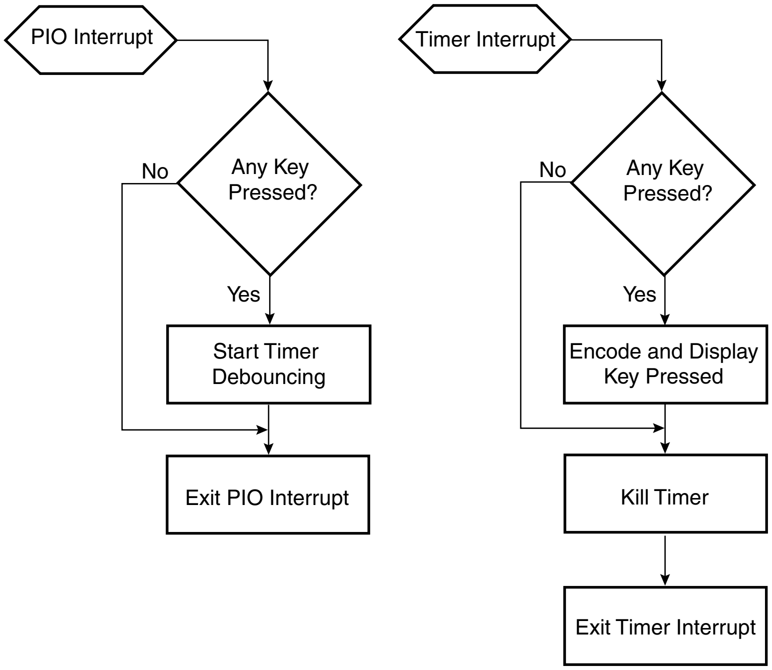


Figure 4. PIO and Timer Interrupts



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.
```

```
;---------------------------------------------------------------------------
;- File source: irq_pio.arm
;- Object: Assembler Interrupt Handler.
;---------------------------------------------------------------------------

AREA        Irq, CODE, READONLY, INTERWORK

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

;---------------------------------------------------------------------------
;- Function: pio_asm_irq_handler
;- Treatments: Parallel IO Controller Interrupt Handler.
;- Called Functions: Keyboard_pioHandlerInt
;- Called Macros: IRQ_ENTRY, IRQ_EXIT
;---------------------------------------------------------------------------
IMPORT      Keyboard_pioHandlerInt
EXPORT      pio_asm_irq_handler
pio_asm_irq_handler

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

;---------------------------------------------------------------------------
;- Function: timer0_asm_irq_handler
;- Treatments         : Timer 0 interrupt handler.
;- Called Functions   : Keyboard_timer0HandlerInt
;- Called Macros      : IRQ_ENTRY, IRQ_EXIT
;---------------------------------------------------------------------------
EXPORT      timer0_asm_irq_handler
IMPORT      Keyboard_timer0HandlerInt
IMPORTTC0_DESC
```

```
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.

```
//*-------------------------------------------------------------------------
//* File Name: Keyboard.h
//* Object: Keyboard Definition File
//*-------------------------------------------------------------------------

//*-------------------------------------------------------------------------
//*                                 Keyboard
//*-------------------------------------------------------------------------
//*   EB40A
//*                          Vcc      Vcc      Vcc      Vcc
//*                           |        |        |        |
//*                           R        R        R        R
//*                           |        |        |        |
//*                           |        |        |        |
//*   P1         -------------------------------------   |
//*                        |  |     |  |     |  |     |  |
//*                     A |      7 |      4 |      1 |   |
//*                        |  |     |  |     |  |     |  |
//*                         --|      --|      --|      --|
//*                           |        |        |        |
//*   P2         -------------------------------------   |
//*                        |  |     |  |     |  |     |  |
//*                     0 |      8 |      5 |      2 |   |
//*                        |  |     |  |     |  |     |  |
//*                         --|      --|      --|      --|
//*                           |        |        |        |
//*   P3         -------------------------------------   |
//*                        |  |     |  |     |  |     |  |
//*                     B |      9 |      6 |      3 |   |
//*                        |  |     |  |     |  |     |  |
//*                         --|      --|      --|      --|
//*                           |        |        |        |
//*   P4         -------------------------------------   |
//*                        |  |     |  |     |  |     |  |
//*                     F |      E |      D |      C |   |
//*                        |  |     |  |     |  |     |  |
//*                         --|      --|      --|      --|
//*                           |        |        |        |
//*                           |        |        |        |
//*   P5         -------------|        |        |        |
//*   P6         ----------------------|        |        |
//*   P7         -------------------------------|        |
//*   P8         ----------------------------------------|
//*-------------------------------------------------------------------------
#define NB_COLUMN4
#define NB_ROW4
```

```
//* 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.

```
//*---------------------------------------------------------------------------
//* 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;
u_char Keyboard_Column;
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)
{ //* Begin
  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
//* Object: Translate the Key buffer Row
//* Input Parameters: read- PIO read value
//* Output Parameters: row- Active row value
//*-------------------------------------------------------------------------
u_char Get_Keyboard_Row(u_int read)
{ //* Begin
  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;
}//* End

//*-------------------------------------------------------------------------
//* Function Name: Read_Keyboard
//* Object: Encode and Display Key pressed
//* Input Parameters: none
//* Output Parameters: none
//*-------------------------------------------------------------------------
void Read_Keyboard (void)
{ //* Begin

//* Check if Keyboard PIO interrupt
  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));

    //* Rows configured as PIO input
    at91_pio_open ( &PIO_DESC, KEYBOARD_ROW_MASK, PIO_INPUT );

    //* Columns configured as PIO input
    at91_pio_open ( &PIO_DESC,KEYBOARD_COLUMN_MASK,PIO_INPUT );
    at91_pio_write (&PIO_DESC,KEYBOARD_COLUMN_MASK,PIO_CLEAR_OUT );

//* 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 );

    at91_pio_open ( &PIO_DESC, KEYBOARD_COLUMN_MASK, PIO_INPUT_IRQ_BIT );

    //* 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;

}//* End

//*----------------------------------------------------------------------------
//* Function Name: Keyboard_Initialization
//* Object: Keyboard initialization
//* Input Parameters: none
//* Output Parameters: none
//*----------------------------------------------------------------------------
void Keyboard_Initialization (void)
{//* Begin

  //* Rows configured as PIO output
    at91_pio_open ( &PIO_DESC, KEYBOARD_ROW_MASK, PIO_OUTPUT );
at91_pio_write (&PIO_DESC, KEYBOARD_ROW_MASK, PIO_CLEAR_OUT );

    //* Column configured as PIO input
    at91_pio_open ( &PIO_DESC, KEYBOARD_COLUMN_MASK, PIO_INPUT_IRQ_BIT );

   //* set PIO interrupt
    //* open external PIO interrupt
at91_irq_open(PIO_DESC.periph_id,5,AIC_SRCTYPE_INT_EDGE_TRIGGERED,
pio_asm_irq_handler);

    //* Enable the PIO Clock
    at91_clock_open (PIO_DESC.periph_id);

  //* TIMER configuration
    //* Open the clock of the timer
    at91_clock_open(TC0_ID);

    //* Initialize the mode of the channel 0
    TC0_CMR =
    TC_WAVE|/* WAVE    : Waveform mode */
    TC_CLKS_MCK32;/* TCCLKS  : MCKI/32 */

    //* disable interrupts
    TC0_IDR = 0x1FF;//* disable interrupt

    //* Initialize the RC Register value
    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
}
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-76-58-30-00
FAX (33) 4-76-58-34-80

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.