

**F<sup>2</sup>MC-16FX FAMILY**  
16-BIT MICROCONTROLLER  
**MB96340**

---

**KEY MATRIX INTERFACE  
USING I/O PORT**

APPLICATION NOTE

## Revision History

Date	Issue
2007-04-16	V1.0, First release, MPi
2007-06-06	V1.1, Updated with review findings from PHu, MPi
2007-09-04	V1.2, Fixed typos and syntax highlighting, MPi

This document contains 29 pages.

## Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for **all products delivered free of charge** (e.g. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC's etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. **Note, all these products are intended and must only be used in an evaluation laboratory environment.**

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days from the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.
2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH's entire liability and the customer's exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH's sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH in original packing and without further defects resulting from the customer's use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.
3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.
4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH's and its suppliers' liability is restricted to intention and gross negligence.

### **NO LIABILITY FOR CONSEQUENTIAL DAMAGES**

**To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.**

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect

## Contents

<b>REVISION HISTORY .....</b>	<b>2</b>
<b>WARRANTY AND DISCLAIMER .....</b>	<b>3</b>
<b>CONTENTS .....</b>	<b>4</b>
<b>1 INTRODUCTION .....</b>	<b>6</b>
1.1 Connection Diagram.....	7
1.2 Key Matrix Functionality and Limitation .....	8
<b>2 OPERATION .....</b>	<b>9</b>
2.1 Key Matrix (1-9) Scanning .....	9
2.1.1 Key-Debounce .....	10
2.2 Key '0' .....	12
2.3 Display Key .....	12
<b>3 RESOURCE USAGE .....</b>	<b>13</b>
3.1 Reload Timer.....	13
3.2 I/O Port.....	13
3.3 External Interrupt.....	13
<b>4 EXAMPLE CODE.....</b>	<b>14</b>
4.1 Initialization Routines .....	14
4.1.1 Flowchart.....	14
4.1.2 C Code .....	15
4.2 Functionality Routines .....	16
4.2.1 Flowchart.....	16
4.2.2 C Code .....	19
4.3 Main Function.....	22
4.3.1 Flowchart.....	22
4.3.2 C Code .....	23
4.4 Interrupt Service Routines .....	24
4.4.1 Flowchart.....	24
4.4.2 C Code .....	25
4.5 Interrupt Vector .....	26
4.5.1 Code.....	26
<b>5 ADDITIONAL INFORMATION .....</b>	<b>27</b>
<b>LIST OF FIGURES .....</b>	<b>28</b>

---

LIST OF TABLES .....	29
----------------------	----

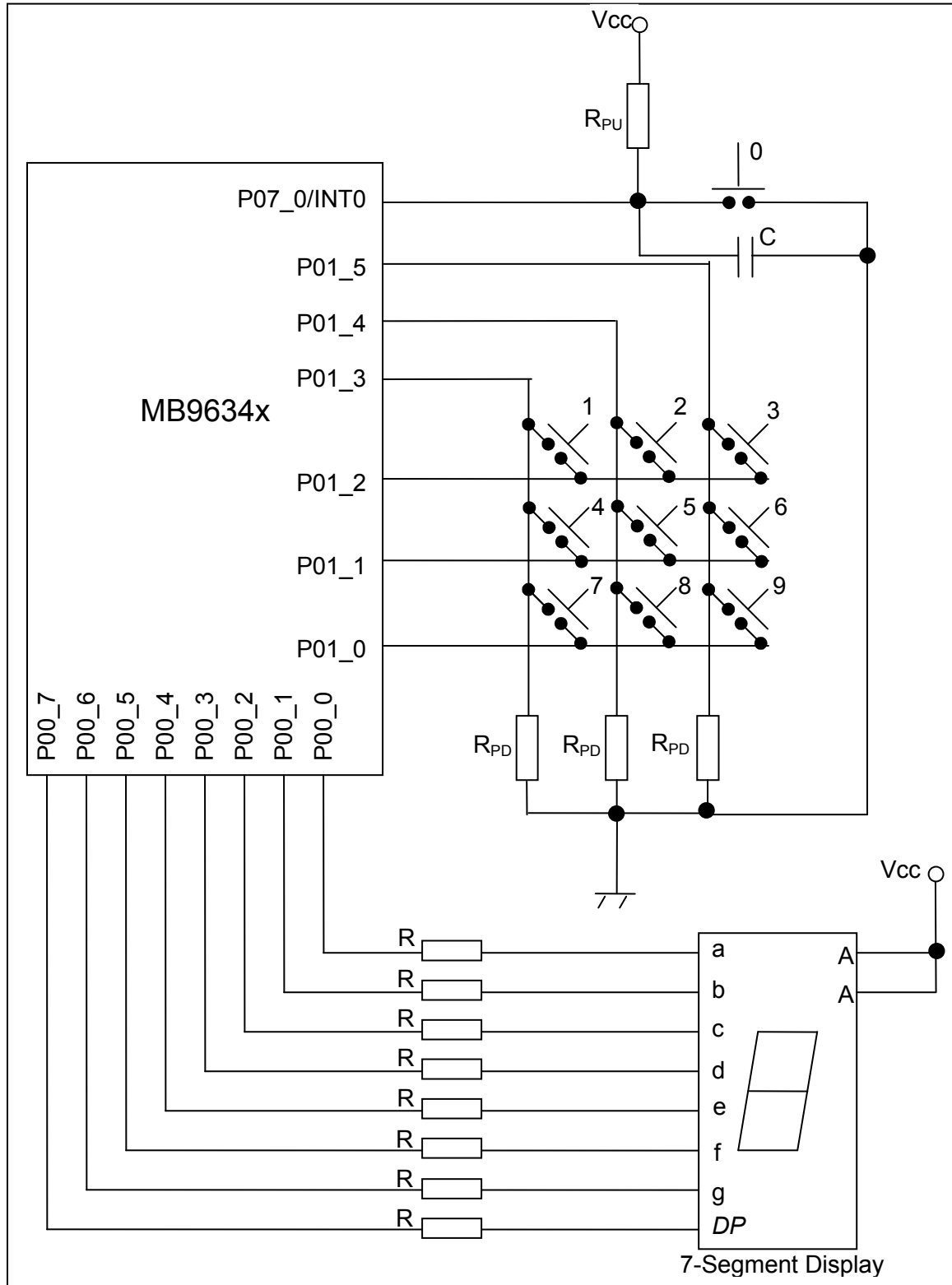
## 1 Introduction

This application note describes how to interface a 3x3 keyboard matrix to an MCU of 16FX Family and demonstrates how to decode the key-press using the scanning technique. The number corresponding to the pressed key would be displayed on the 7-Segment Display. Other than the key matrix it also contains a separate key connected to external interrupt pin INT0. The functionality of this key is similar to Power On button.

Such kind of arrangement can be used in the industrial or automotive application usually for numeric key entry.

### 1.1 Connection Diagram

The Figure 1-1 shows the connection diagram. The key '0' is connected to the external interrupt pin INT0. This key is used to wake the microcontroller up from the STOP mode to RUN mode and also vice a versa. The pull up resistor  $R_{PU}$  is connected to limit the current when the key '0' is pressed. The capacitor is used to eliminate the bouncing of the key '0'.



**Figure 1-1: Key Matrix Connection Diagram**

The keyboard matrix containing keys '1' to '9' is connected to I/O port P01. This arrangement (as shown above) results in significant resource saving with a little software overhead. The port pins P01\_0 to P01\_2 are scan lines hence those are configured as digital outputs, whereas the port pins P01\_3 to P01\_5 are return lines hence those are configured as digital inputs. The pull-down resistors  $R_{PD}$  are connected to limit the current when any key within this matrix is pressed.

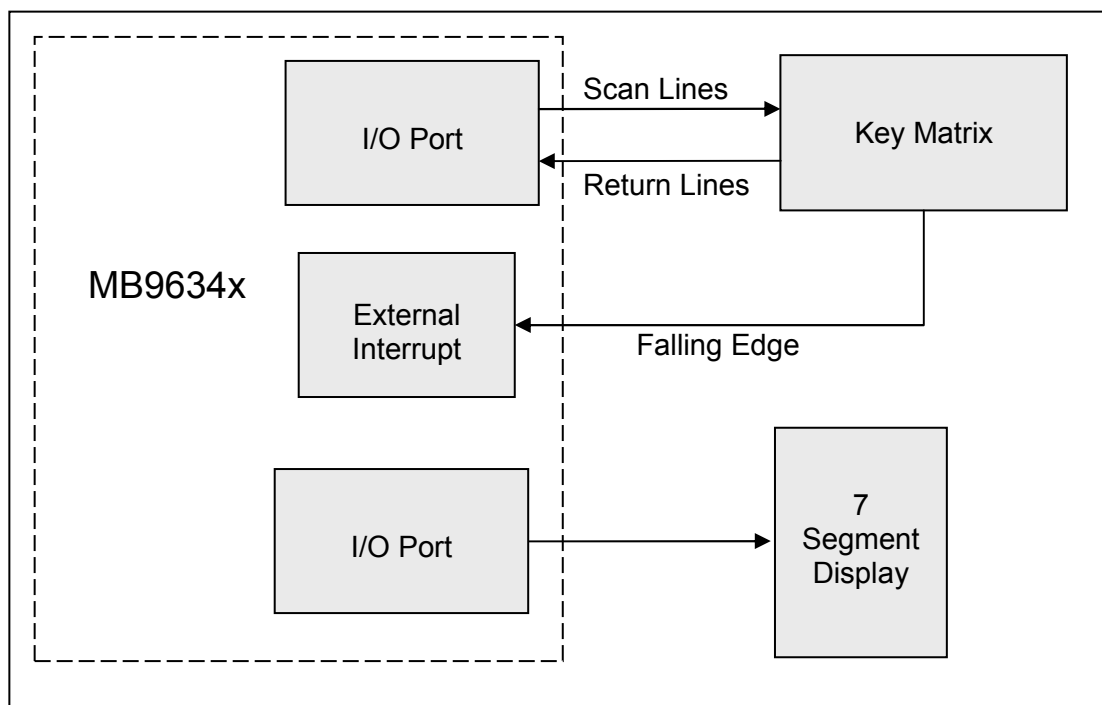
The common anode 7-Segment display is interfaced to the Port 0 via current limiting resistors (R). It is used to display the number of the pressed key.

## 1.2 Key Matrix Functionality and Limitation

The key matrix software is only able to detect a single key press at one instance. It is not capable of handling multiple simultaneous key presses. However such functionality can be added later, if required.

## 2 Operation

The below block diagram depicts the peripheral operation and dependency.



**Figure 2-1: Key Matrix Block Diagram**

### 2.1 Key Matrix (1-9) Scanning

One scan (digital output - P01\_0) line is set to HIGH at one instance while the other two scan lines are kept LOW and the return (digital input) lines are checked; if any key is pressed then the corresponding return line would be HIGH. Then after the scanning interval of 5 ms (which is configurable) the next scan line (P01\_1) is set to HIGH while the other two scan lines are kept LOW and the return lines are checked. Then the same process is repeated for the last scan line (P01\_2) after the scanning interval. After the last scan line the process would start all over again from the first scan line (P01\_0) after the scanning interval. The details about the key scan interval are discussed in section 2.1.1

The following table shows the keys and the corresponding key code (data on the Port 01):

Key Pressed	Return Lines			Scan Lines			Key Code
	P01_5	P01_4	P01_3	P01_2	P01_1	P01_0	
"1"	0	0	1	1	0	0	0x0C
"2"	0	1	0	1	0	0	0x14
"3"	1	0	0	1	0	0	0x24
"4"	0	0	1	0	1	0	0x0A
"5"	0	1	0	0	1	0	0x12
"6"	1	0	0	0	1	0	0x22
"7"	0	0	1	0	0	1	0x09
"8"	0	1	0	0	0	1	0x11
"9"	1	0	0	0	0	1	0x21

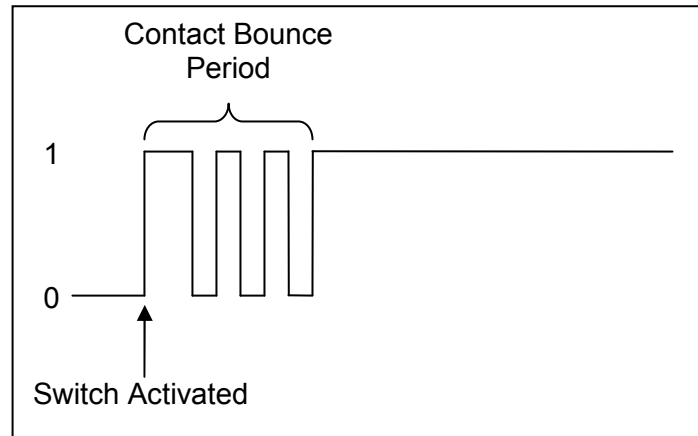
**Table 2-1: Key Codes**

The PDR01 register needs to be read to get these key codes.

With reference to the above table, let us consider that key “3” is pressed. To detect this key press the scan line P00\_2 needs to be HIGH and the return line P00\_5 would become HIGH, since the key connects these two lines once it is pressed. Hence the key code “0x24”.

### 2.1.1 Key-Debounce

The mechanical keys do not open or close cleanly. When a key is pressed it makes and breaks contacts several times before settling into its final position. This causes several transitions or bounces to occur. The bounce period/time varies from key to key. This behavior is described in the below figure:



**Figure 2-2: Key Bounce**

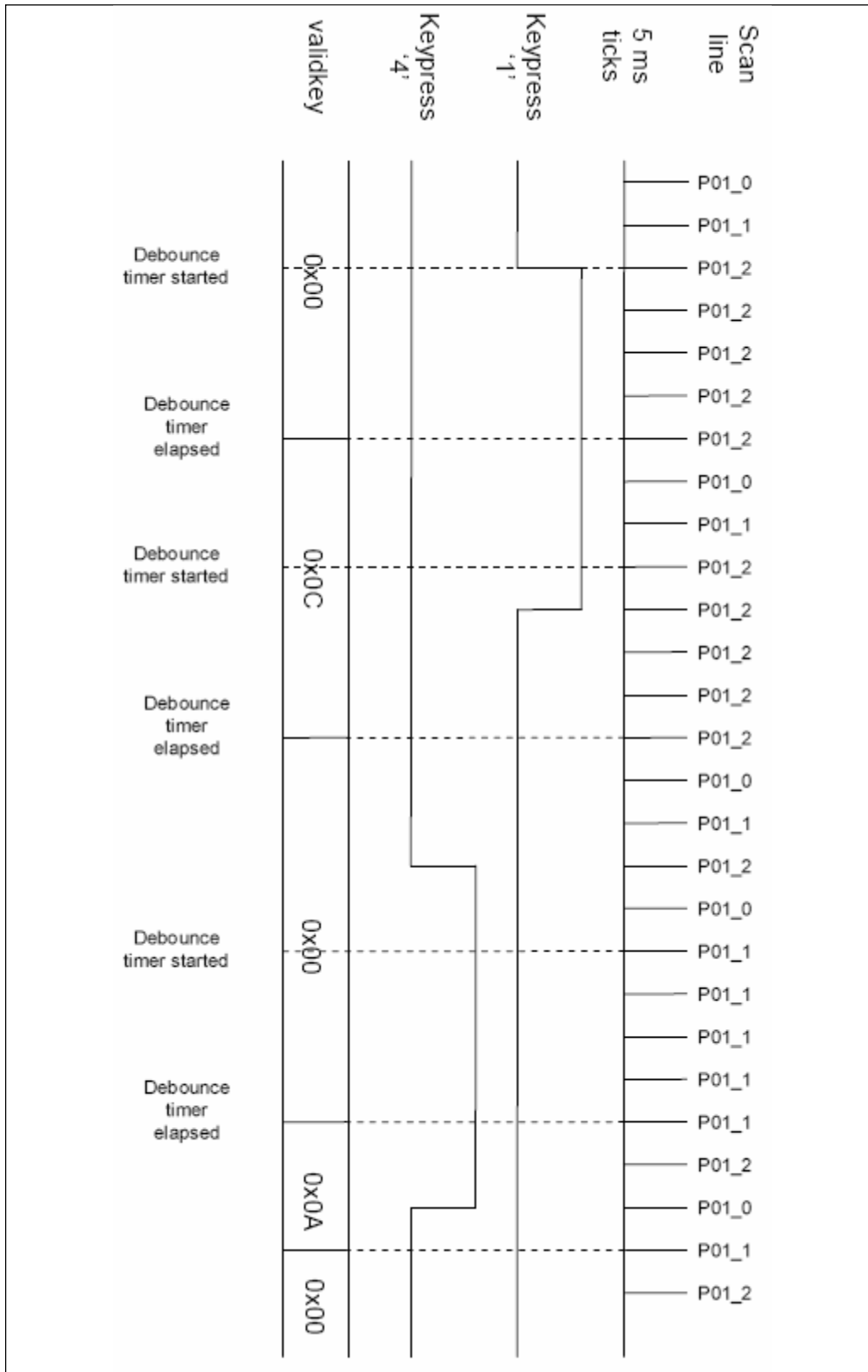
The above discussed phenomena can be taken care of by using software de-bouncing technique as described below.

Below steps explain the de-bouncing and the decision making logic involved in order to determine the key-press or key-release:

1. The key scanning interval is dependent on and needs to be less than the bounce time/debounce delay of the key/switch. Here the scanning interval is chosen such that the debounce delay is always multiple of the scanning interval i.e. the scanning interval in this case is 5 ms and the debounce delay is considered as 20 ms.
2. The return lines are polled / checked just once while a particular scan line is selected. If any of the return line is found HIGH, while a particular scan line is selected (HIGH), the corresponding key code is stored.
3. After the de-bounce delay of 20 ms if the same return line is found HIGH (while same scan line is selected (HIGH)), then the corresponding key is declared to be pressed (i.e. the corresponding key-code would be reflected in the variable `validkey`). Within this de-bounce delay any other key presses would be ignored, if any.
4. After the de-bounce delay, the variable `validkey` would continue to reflect the same key code, unless and until the same key is released or another key is pressed.
5. After the de-bounce delay another scan line would be selected and the steps 2 to 4 would be repeated, if required.
6. If in the step 2, no return line would be found HIGH then another scan line would be selected.
7. If no key is found pressed or the pressed key is released then the `validkey` would have value equal to `NONE`.

It should be noted that the multiple simultaneous key presses would not be taken care in this approach.

The following figure explains the key-matrix scanning with key de-bounce:



**Figure 2-3: Key-matrix Scanning with Key De-bounce**

## 2.2 Key '0'

Normally pin INT0 is at high level, once the key '0' is pressed, then the falling edge would appear on INT0 pin. This generates an interrupt.

As shown in the connection diagram, the de-bouncing of the key '0' is taken care by the RC circuit comprises of  $R_{PU}$  and C (instead of the software de-bouncing as discussed above). The values of these should be chosen such that the product of RC ( $R_{PU} * C$ ) is longer than the expected bounce time of the key. However, this puts a limitation on the rate of recurrence of the key press.

## 2.3 Display Key

The key which is pressed currently would be reflected on the 7-Segment Display connected to Port 00. That means if '1' is pressed, 7-Segment Display would display "1" (i.e. segments b and c would be lit). If '2' is pressed, it would display "2" (i.e. segments a, b, e, d and g would be lit), so on and so forth. If no key is pressed then it would not display anything. And in case of multiple simultaneous (invalid) key press, it would display "E." (Error).

### 3 Resource Usage

#### 3.1 Reload Timer

The Reload Timer 0 (RLT0) is used as the time base for key matrix scanning. RLT0 is configured to issue an interrupt at an interval of 5 ms at 16 MHz CLKP1. It should be chosen such that the de-bounce delay of the keys is always multiple of this scanning interval as discussed before. Here the de-bounce delay is considered as 20 ms.

In the RLT0 interrupt service routine (ISR) the scanning interval flag `time_5ms` is made `TRUE` so as to indicate the scanning function that the delay of 5 ms is elapsed and it should carry out the scanning again.

The below figure describes the behaviour described above:

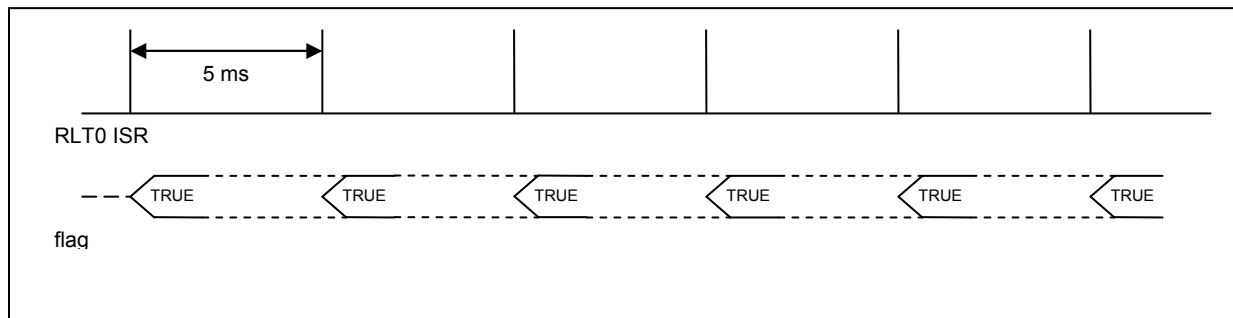


Figure 3-1: Reload Timer Ticks and ISR

The `time_5ms` would be made `FALSE` within the scanning function `Scan_Key()` once it is executed.

#### 3.2 I/O Port

The Port 00 is used for displaying the information related to the currently pressed key.

The Port 01 is used for key matrix scanning. Port pins P01\_0 to P01\_2 are configured as output and the port pins P01\_3 to P01\_5 configured as input.

#### 3.3 External Interrupt

The external interrupt pin 0 (INT0) is connected to key '0'. The INT0 pin is configured to generate an interrupt at every falling edge.

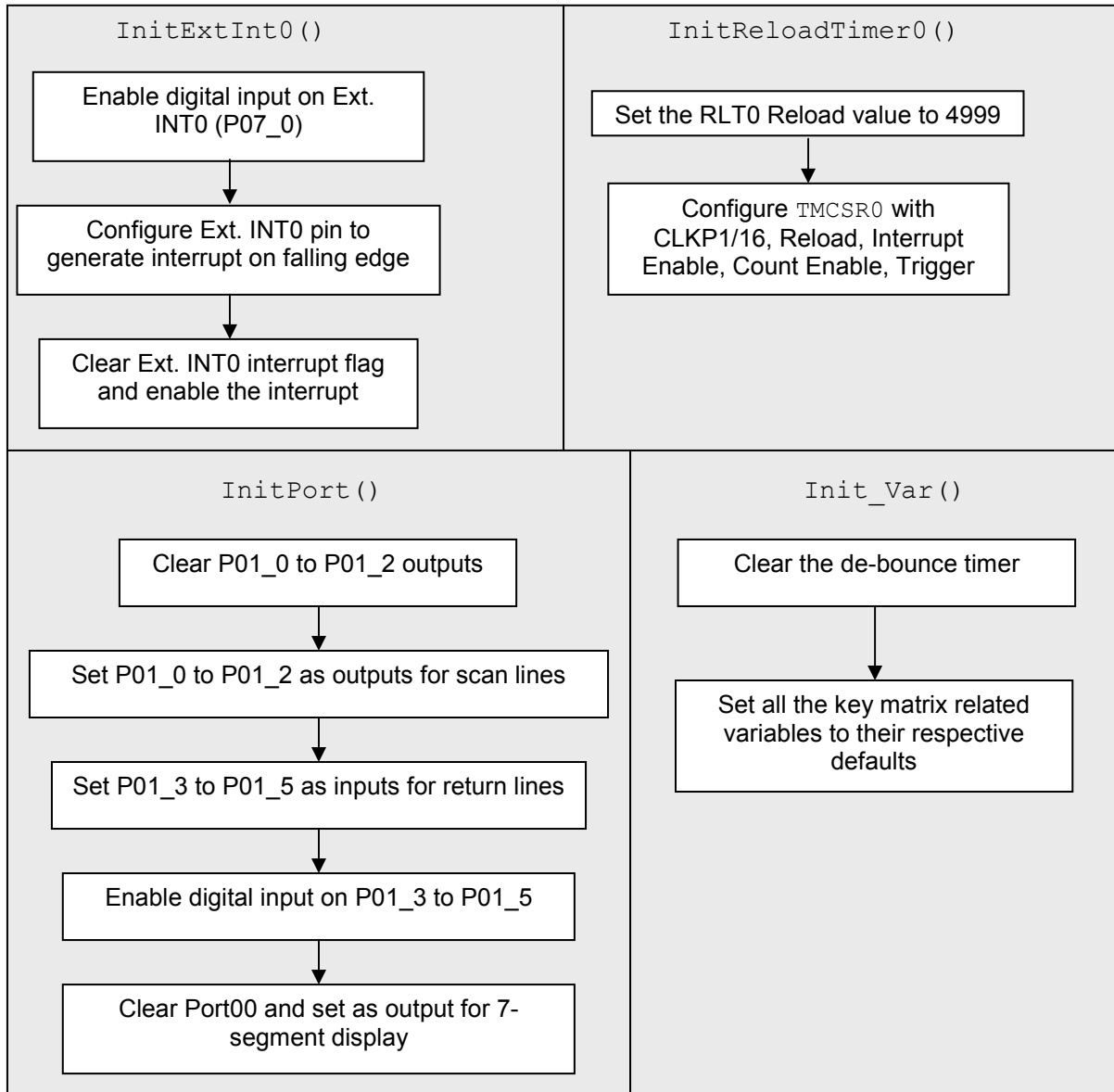
In the INT0 ISR then the request is made to switch to STOP mode or RUN mode depending upon the current operating mode. However, the transition from one mode to the other would happen in the function `Ctrl_Power()`.

The reason for this is, if the STOP mode is requested in the INT0 ISR itself then, after the wakeup, the CPU would execute the next instruction from where it was interrupted before going STOP mode. Because of this the software may take decision based on the old context (context before entering STOP mode) which may be erroneous / invalid after the wakeup.

## 4 Example Code

### 4.1 Initialization Routines

#### 4.1.1 Flowchart



### 4.1.2 C Code

```

/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */
/*-----*/
volatile unsigned char power_stat = POWER_ON; // Power Status
volatile unsigned char time_5ms = FALSE; // 5 ms elapsed flag

unsigned char timer_debounce = 0; // Debounce timer
unsigned char keyflag = FALSE; // Flag to indicate if any key is pressed currently

unsigned char keymask = 1; // Keymask for the current key pressed
unsigned char keyin = NONE; // Keycode for the current key pressed

unsigned char last_keymask = 1; // Keymask for the last key pressed
unsigned char last_keyin = NONE; // Keycode for the last key pressed

unsigned char validkey = NONE; // Keycode for the current key pressed
// after considering debounce time
/*-----*/
/* Initialize RLTO */
/*-----*/
void InitReloadTimer0 (void)
{
    TMRLR0 = 4999; // Set reload value
// Hence interrupt at 4999+1 * 1/1MHz = 5 ms
    TMCSTR0 = 0x041B; // Clock CLKP1/16 =16MHz/16 = 1MHz, reload,
// Interrupt enable, count enable, trigger
}
/*-----*/
/* Initialize Port */
/*-----*/
void InitPort (void)
{
    PDR01 &= ~0x07; // Clear P01_0 to P01_2
    DDR01 |= 0x07; // Set P01_0 to P01_2 as output for scan lines
    DDR01 &= ~0x38; // Set P01_3 to P01_5 as input for return lines
    PIER01 |= 0x38; // Enable digital input on P01_3 to P01_5
    PDR00 = 0x00; // Clear P00_0 to P00_7
    DDR00 = 0xFF; // Set P00_0 to P00_7 as output for 7-Segment Display
}
/*-----*/
/* Initialize INT0 */
/*-----*/
void InitExtInt0 (void)
{
    ADER2 &= 0xFE; // Port I/O mode on P07_0
    PIER07_IE0 = 1; // Enable digital input on P07_0
    ELVRL0_LB0 = 1; // LB0, LA0 = 11 -> Falling edge
    ELVRL0_LA0 = 1;
    EIRRO_ER0 = 0; // Reset interrupt flag
    ENIRO_EN0 = 1; // Enable interrupt request
}
/*-----*/
/* Initialize variables */
/*-----*/
void Init_Var (void)
{
    timer_debounce = 0; // Clear debounce timer
    keyflag = FALSE; // No key pressed currently

    keymask = 1; // Select default scan line i.e. P00_1
    keyin = NONE; // Clear keyin

    last_keymask = 1; // Select default scan line i.e. P00_1
    last_keyin = NONE; // Clear last_keyin

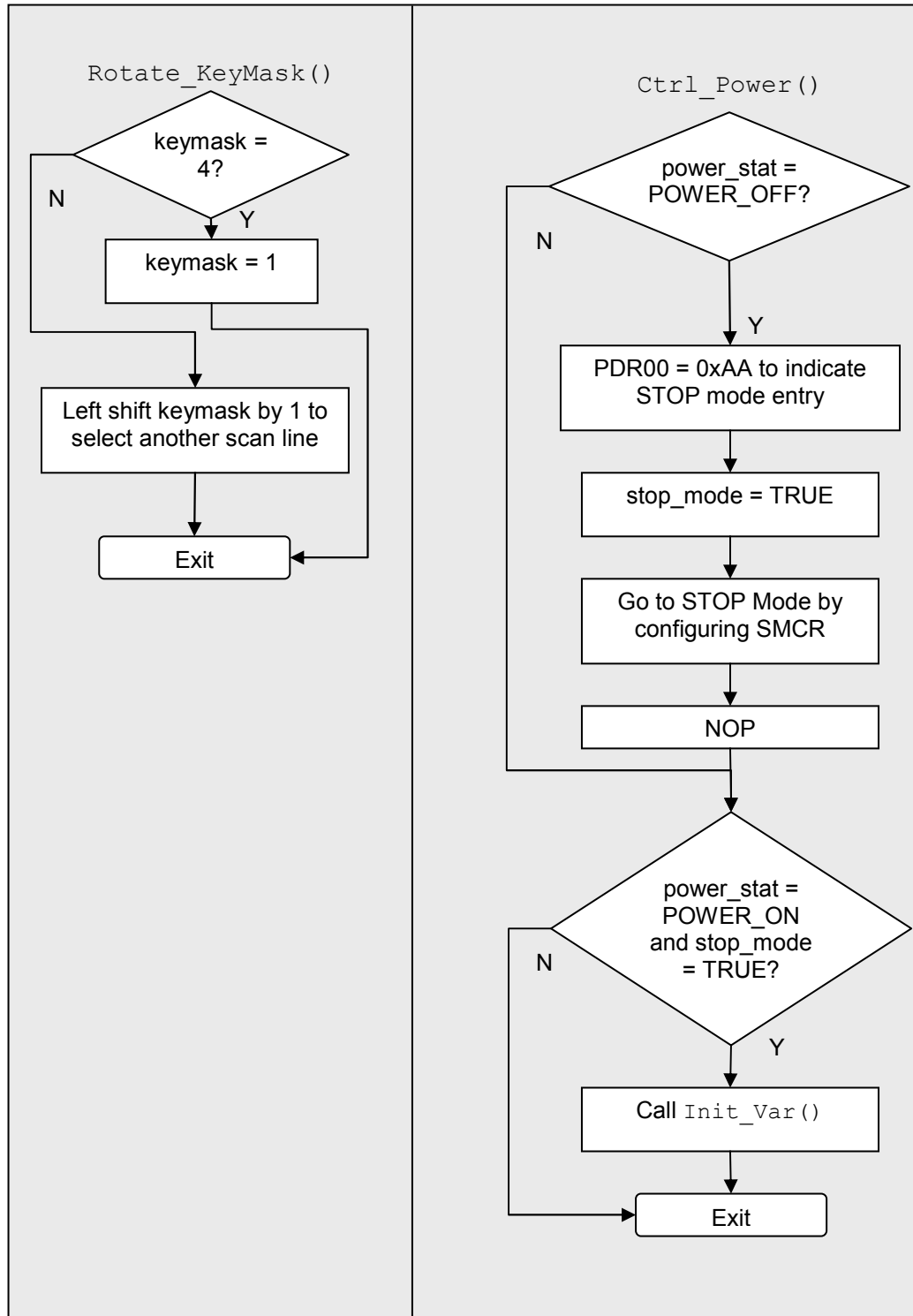
    validkey = NONE; // Clear validkey
}

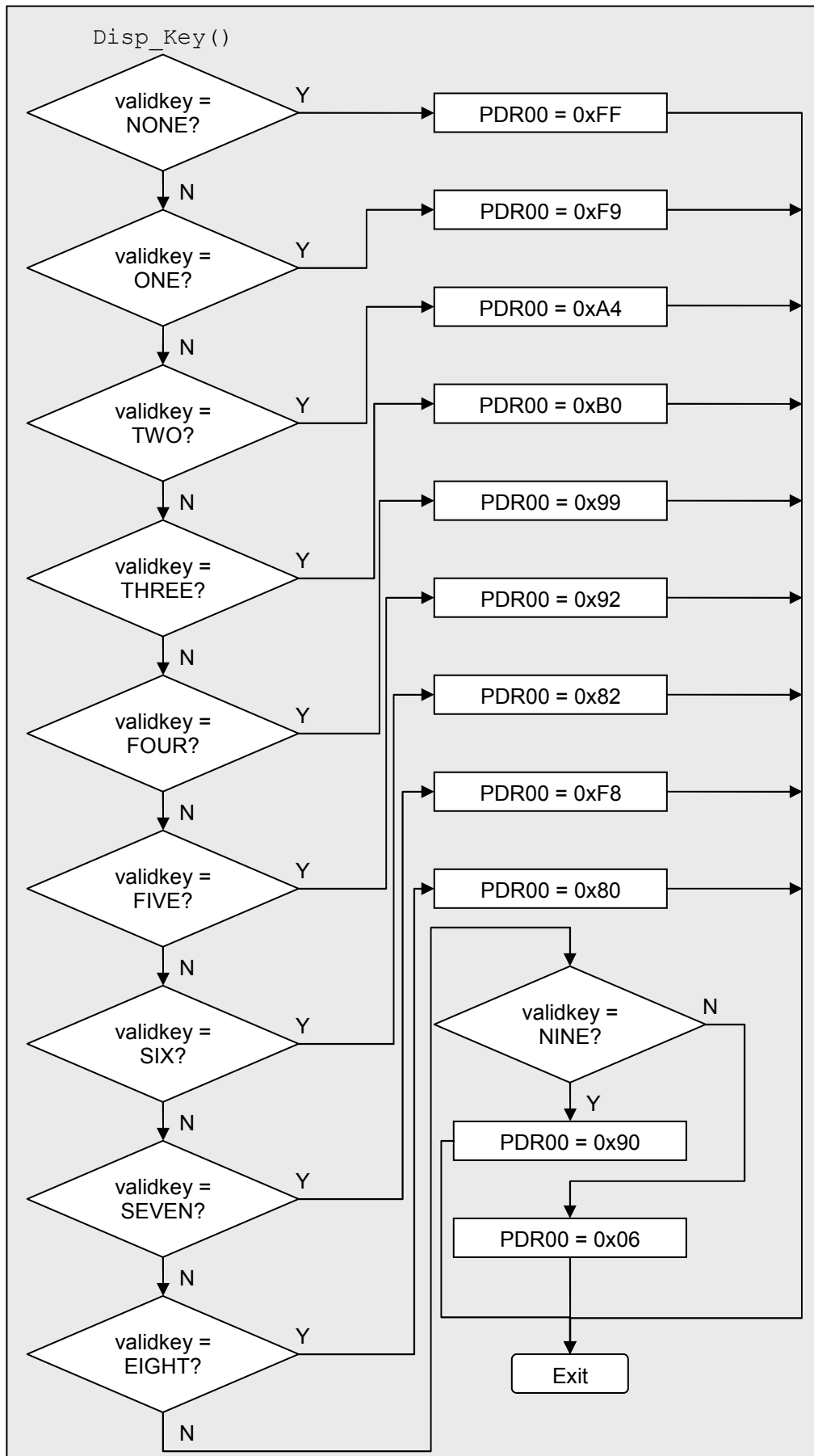
```

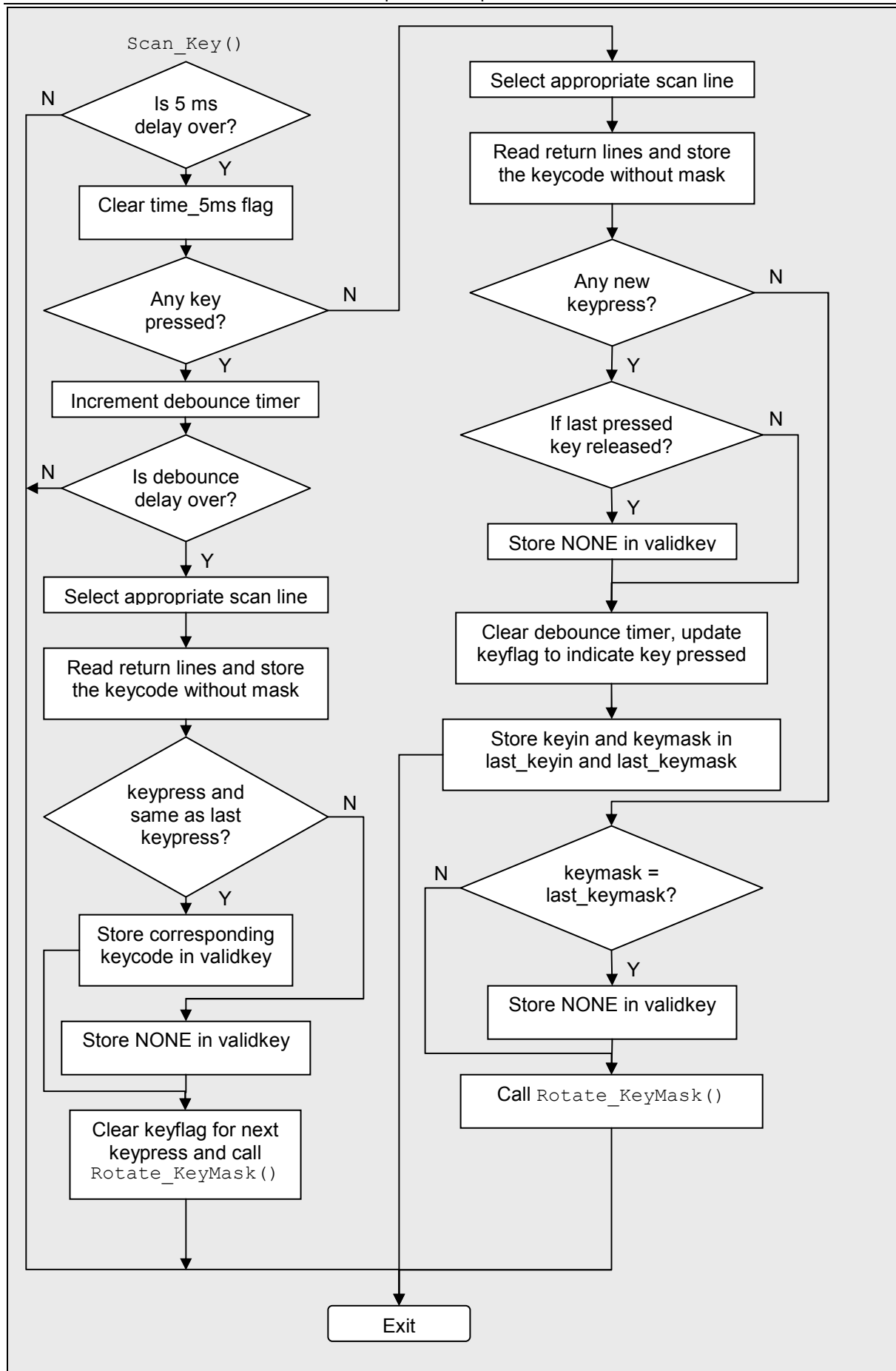
## 4.2 Functionality Routines

### 4.2.1 Flowchart

It should be noted that all the functions described in this section except Rotate\_KeyMask() should be called once in at least 5ms time. Calling of Rotate\_KeyMask() would be taken care by Scan\_Key() function.







### 4.2.2 C Code

```

/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */
/*-----*/
/* Rotate KeyMask */
/*-----*/
void Rotate_KeyMask (void)
{
    // Keymask sequence P01_0->P01_1->P01_2->P01_0->P01_1->P01_2
    if (4 == keymask) // If scan line P01_2 was selected last
    {
        keymask = 1; // Select scan line P01_0
    }
    else
    {
        keymask = keymask << 1; // Else, select the next scan line
    }
    keyin = 0; // Clear keyin
}
/*-----*/
/* Display Key Pressed */
/*-----*/
// This function needs to be called at least once in every 5 ms
void Disp_Key (void)
{
    switch (validkey)
    {
        case NONE : PDR00 = 0xFF; // If no key pressed, display nothing
                    break;
        case ONE : PDR00 = 0xF9; // If 1 key pressed, display 1
                    break;
        case TWO : PDR00 = 0xA4; // If 2 key pressed, display 2
                    break;
        case THREE : PDR00 = 0xB0; // If 3 key pressed, display 3
                    break;
        case FOUR : PDR00 = 0x99; // If 4 key pressed, display 4
                    break;
        case FIVE : PDR00 = 0x92; // If 5 key pressed, display 5
                    break;
        case SIX : PDR00 = 0x82; // If 6 key pressed, display 6
                    break;
        case SEVEN : PDR00 = 0xF8; // If 7 key pressed, display 7
                    break;
        case EIGHT : PDR00 = 0x80; // If 8 key pressed, display 8
                    break;
        case NINE : PDR00 = 0x90; // If 9 key pressed, display 9
                    break;
        default : PDR00 = 0x06; //If invalid key pressed, display E.
                    break;
    }
}
/*-----*/
/* Power On/off Routine */
/*-----*/
// This function needs to be called at least once in every 5 ms
void Ctrl_Power (void)
{
    unsigned char stop_mode = FALSE;
    if ( POWER_OFF == power_stat ) // If stop mode is requested
    {
        PDR00 = 0xAA; // Indication of going to STOP mode
        stop_mode = TRUE; // set the flag
    }
}

```

```

        SMCR |= 0x03;                // Go to STOP mode
        __asm(" NOP");              // NOP
    }

    if ( POWER_ON == power_stat && TRUE == stop_mode) // If woke up from stop mode
    {
        Init_Var();                  // Initialize all variables
    }
}
/*-----*/
/*                               Key Matrix Scanning Routine                               */
/*-----*/
// This function needs to be called at least once in every 5 ms
void Scan_Key (void)
{
    unsigned char keywomask;

    if ( TRUE == time_5ms )        // If 5 ms delay elapsed
    {
        time_5ms = FALSE;         // Clear the flag for next 5 ms delay

        if ( TRUE == keyflag ) // If any key is pressed currently
        {
            timer_debounce++;     // Increment the debounce timer

            //If the debounce delay elapsed
            if ( DEBOUNCE_DELAY <= timer_debounce)
            {
                PDR01 &= ~0x07;    // Clear the scan line

                // Select the appropriate scan line (P01_0-P00_2)
                PDR01 |= keymask;
                keyin = PDR01 & 0x3F; // Read the return lines

                // Store keycode of the key pressed without mask
                keywomask = keyin & 0x38;

                // If any key is pressed and current keypress is same as
                // last key press
                if ( 0 != keywomask && keyin == last_keyin )
                {
                    // Keypress was longer than debounce time hence
                    // valid, store it
                    validkey = keyin;
                }
                else
                {
                    // Keypress wasnt longer than debounce time hence
                    // invalid, discard it
                    validkey = NONE;
                }
            }

            keyflag = FALSE;      // Next key press
            Rotate_KeyMask ();    //Rotate the keymask for next scan line
        }
    }
    else // If no key is pressed currently
    {
        PDR01 &= ~0x07; // Clear the scan line
        PDR01 |= keymask; // Select the appropriate scan line (P01_0-P01_2)

        keyin = PDR00 & 0x3F;    // Read the return lines
        keywomask = keyin & 0x38; // Store keycode of the key pressed
                                // without mask

        if ( 0 != keywomask )    // If any key is pressed
        {
            // If the last pressed key released
            if ( keymask == last_keymask && keyin != last_keyin )
            {
                validkey = NONE; // Discard the old keycode
            }
        }
    }
}

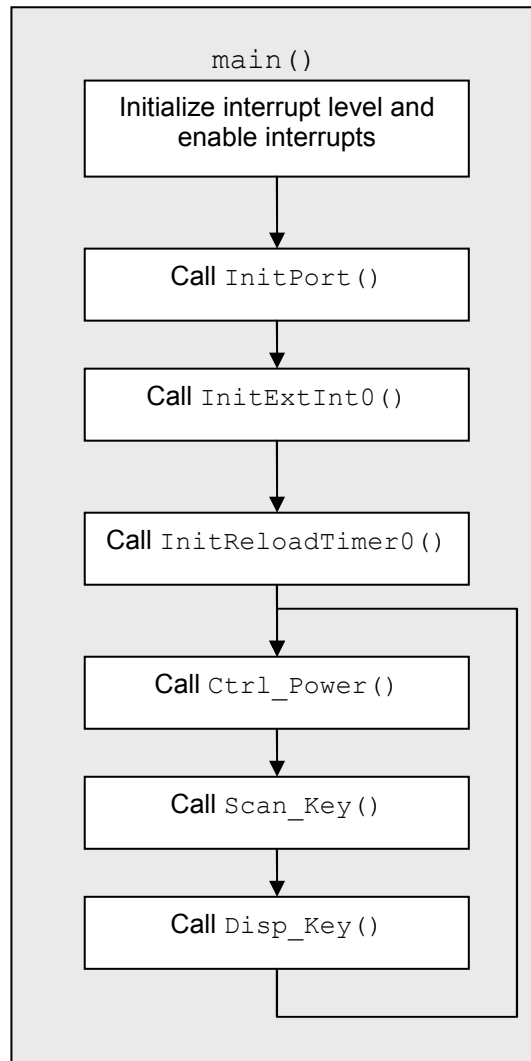
```

```
        timer_debounce = 0;           // Clear the debounce timer
        keyflag = TRUE;               // Flag to indicate key pressed
        last_keyin = keyin;           // Store the keyin for comparing
        last_keymask = keymask;       // Store the keyin for detecting
        // after debounce
        // key release
    }
    else
    {
        // If no key is pressed and keymask is same, means the last
        // pressed key released
        if ( keymask == last_keymask )
        {
            validkey = NONE;           // Discard the old keycode
        }

        Rotate_KeyMask (); // Rotate the keymask for next scan line
    }
}
}
```

## 4.3 Main Function

### 4.3.1 Flowchart



### 4.3.2 C Code

```

/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */
/*-----*/
/*-----*/
/* M A I N */
/*-----*/
void main(void)
{
    InitIrqLevels();
    __set_il(7); // allow all levels
    __EI(); // globally enable interrupts

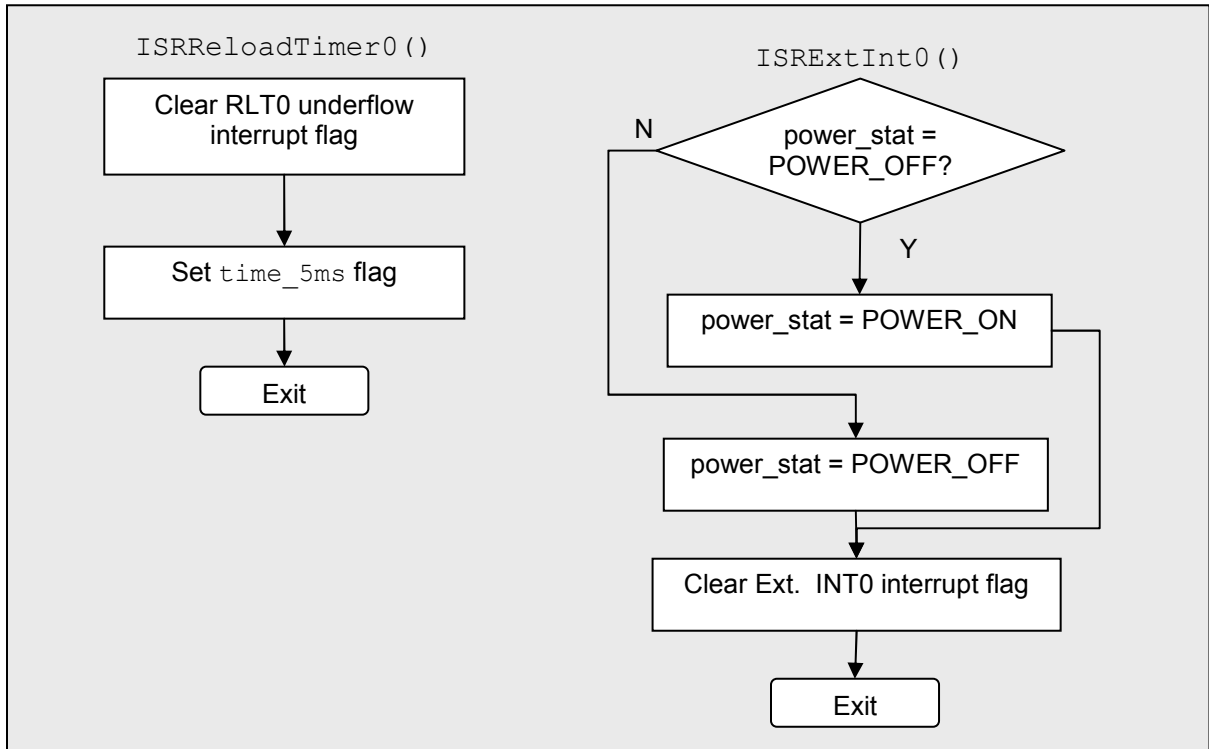
    InitPort(); // Initialize port 0 & 1
    InitExtInt0(); // Initialize external interrupt 0
    InitReloadTimer0(); // Initialize reload timer 0

    while(1)
    {
        Ctrl_Power(); // Manage power on/off
        Scan_Key(); // Scan the key matrix
        Disp_Key(); // Display the key currently pressed
    }
}

```

4.4 Interrupt Service Routines

4.4.1 Flowchart



#### 4.4.2 C Code

```

/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                          */
/*          (C) Fujitsu Microelectronics Europe GmbH                      */
/*-----*/
/*          I S R s                                                         */
/*-----*/

/*-----*/
/*          Reload Timer 0 ISR                                             */
/*-----*/

__interrupt void ISRReloadTimer0(void)
{
    TMCSSRO_UF = 0;                // Reset underflow interrupt request flag
    time_5ms = TRUE;              // Set the 5ms timer flag
}

/*-----*/
/*          External Interrupt 0 ISR                                       */
/*-----*/

__interrupt void ISRExtInt0(void)
{
    if ( POWER_OFF == power_stat )           // If it is powered off
    {
        power_stat = POWER_ON;              // Request power on
    }
    else
    {
        power_stat = POWER_OFF;            // Else request power off
    }
    EIRRO_ER0 = 0;                        // Clear interrupt flag
}

```

## 4.5 Interrupt Vector

### 4.5.1 Code

```

/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/*----- (C) Fujitsu Microelectronics Europe GmbH -----*/
void InitIrqLevels(void)
{
    . . .

    ICR = (17 << 8) | 2; /* Priority Level 2 for Ext Int0 of MB9634x Series */
    ICR = (51 << 8) | 3; /* Priority Level 2 for RLTO of MB9634x Series */

    . . .
}
. . .

/* ISR prototype */
__interrupt void ISRExtInt0(void);
__interrupt void ISRReloadTimer0(void);
. . .

#pragma intvect ISRExtInt0          17          /* Ext Int0 of MB9634x Series */
#pragma intvect ISRReloadTimer1    51          /* RLTO of MB9634x Series */
. . .

```

## 5 Additional Information

Information about FUJITSU Microcontrollers can be found on the following Internet page:

<http://mcu.emea.fujitsu.com/>

The software example related to this application note is:

*96430\_key\_matrix\_io*

It can be found on the following Internet page:

[http://mcu.emea.fujitsu.com/mcu\\_product/mcu\\_all\\_software.htm](http://mcu.emea.fujitsu.com/mcu_product/mcu_all_software.htm)

## List of Figures

Figure 1-1: Key Matrix Connection Diagram.....	7
Figure 2-1: Key Matrix Block Diagram.....	9
Figure 2-2: Key Bounce .....	10
Figure 2-3: Key-matrix Scanning with Key De-bounce .....	11
Figure 3-1: Reload Timer Ticks and ISR .....	13

## List of Tables

Table 2-1: Key Codes ..... 9