Application Report

SLAA139A – February 2002 – Revised May 2018

Implementing An Ultra-Low-Power Keypad Interface With MSP430™ MCUs

MSP430 Applications

ABSTRACT

In applications with keypads, a key can be held or stuck down, which causes excess current consumption and reduces the battery life of a battery-operated product. This application report describes a solution. The keypad interface in this report, based on the MSP430F123 microcontroller (MCU), draws 0.1 µA while waiting for a key press, is interrupt driven, requires no polling, and consumes a maximum of only 2 µA at 3 V if all keys are pressed and held simultaneously.

The source code described in this application report can be downloaded from http://www.ti.com/lit/zip/slaa139.

Contents

List of Figures

Trademarks

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

The keypad interface described in this report (see Figure 1) is based on the MSP430F123 MCU. Features of this design include:

- 100-nA typical current consumption while waiting for key press
- 2-µA maximum current consumption if all keys are held simultaneously
- No polling required
- No crystal required
- Minimum external components
- Suitable for any MSP430™ MCU

2 Implementation

The rows of the keypad are connected to port pins P3.0 to P3.3. The columns are connected to pins P1.0 to P1.2. Connecting the rows to port 3 pins, instead of port 1 pins, leaves the other port 1 pins for other interrupt sources, because the P1 pins have interrupt capability, but the P3 pins do not.

In normal mode, while the circuit is awaiting a key press (wait-for-press mode), the rows are driven high, and the P1.x column pins are configured as inputs, with interrupts enabled and set to interrupt on a rising edge. The 4.7-MΩ pulldown resistors hold the inputs low in this state. The MCU then enters low-power mode 4 (LPM4), in which the MCU current consumption is approximately 100 nA. This state is maintained indefinitely until a key is pressed. The circuit is interrupt-driven with no need for polling.

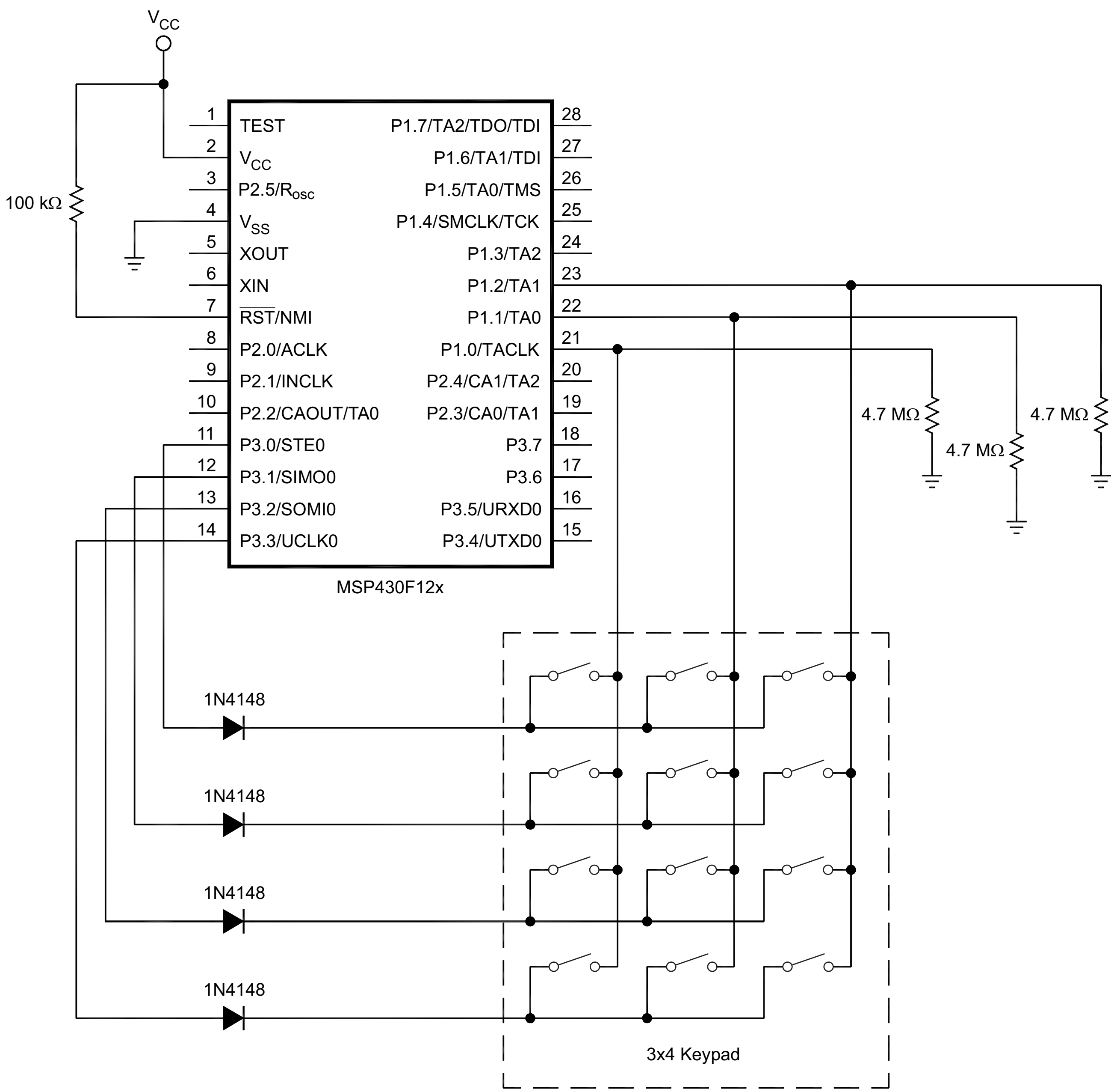

Figure 1. Keypad Schematic Diagram

When a key is pressed, the column associated with that key receives a rising edge, waking the MSP430 MCU. At that time, Timer_A is configured to perform a debounce delay of approximately 40 ms. The timer for the delay uses the internal digitally controlled oscillator (DCO) of the MSP430 MCU, which is an RC-type oscillator. The DCO is subject to tolerances, so a debounce delay was chosen to give a worst-case-minimum delay of 25 ms. That translates to a worst-case maximum delay of approximately 86 ms and a typical delay of approximately 40 ms. This is a useable range for keypad debounce.

After a key has been pressed, the MCU enters a wait-for-release mode in which it drives high only the necessary row for the key being pressed (other rows are driven low). The software reconfigures the P1.x I/O lines to interrupt on a falling edge, and the MCU returns to LPM4, waiting for the release of the key. Again, there is no polling necessary. The detection of the key release is interrupt driven, which allows the microcontroller to stay in LPM4 while the key is held, thus reducing current consumption. When the key is released, the debounce delay is again executed. After the debounce delay, the keypad is scanned again to determine if any other keys are being held. If so, the wait-for-release mode continues, waiting for all keys to be released. When all keys are released, the MSP430 MCU returns to the wait-for-press mode.

During the wait-for-release mode, only one row of the keypad is driven high, therefore limiting the maximum amount of current consumption to the condition where all three keys on a single row are pressed and held. For a 3-V system, that equates to approximately 2 µA. Any other key press does not result in increased current consumption, because the corresponding row is not driven high.

In this 3×4 keypad example, the rows are driven rather than the columns to limit the maximum current consumption by the circuit when all keys are pressed and held simultaneously. If the columns were driven instead, the rows would have had the pulldown resistors, therefore increasing the number of paths to ground when all the keys are held and increasing the possible current consumption.

3 Software

Figure 2 shows the software flow. The complete code listing follows and is can be downloaded from http://www.ti.com/lit/zip/slaa139.

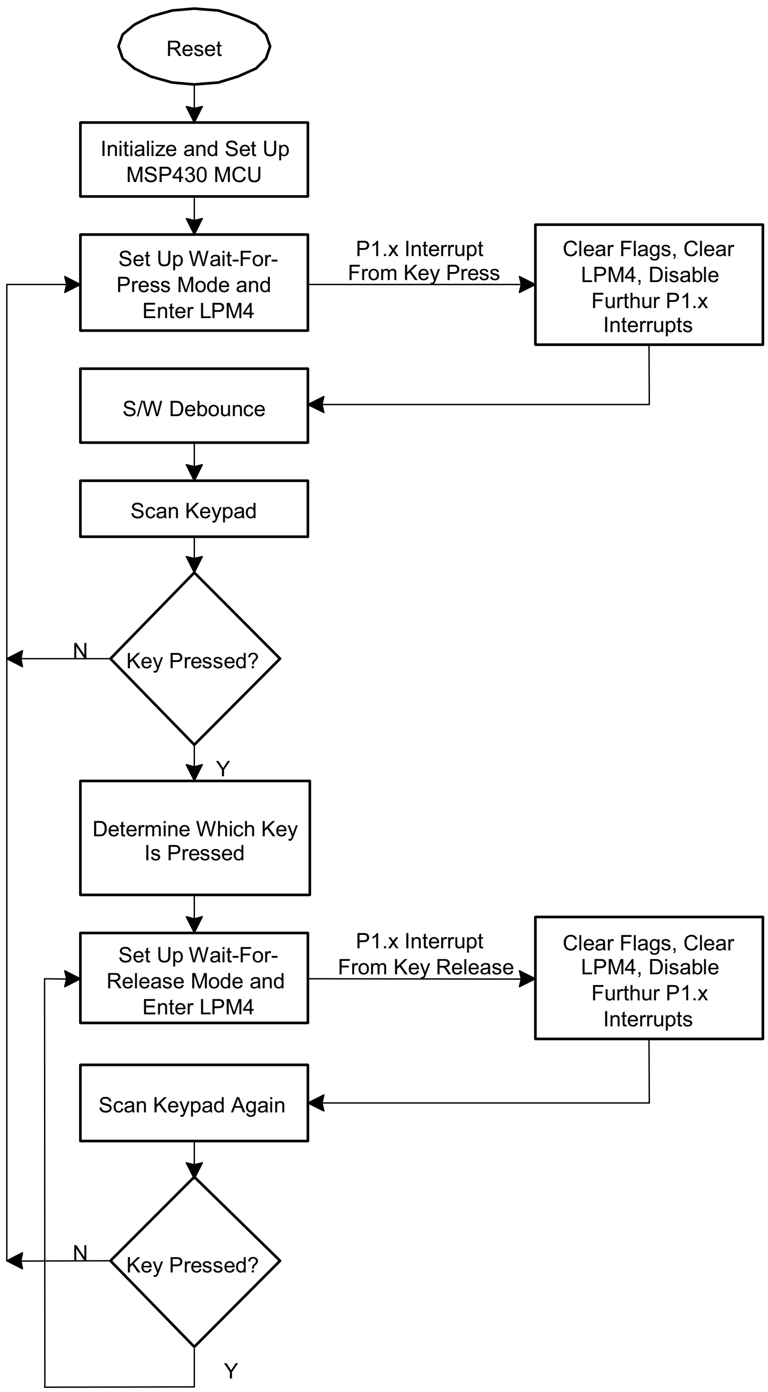

Figure 2. Software Flow

```
; THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
; REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
; INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
; FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
; COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
; TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
; POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
; INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
; YOUR USE OF THE PROGRAM.
;
; IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
; CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
; THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
; OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
; OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
; EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
; REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
; OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
; USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
; AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
; YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
; (U.S.$500).
;
; Unless otherwise stated, the Program written and copyrighted
; by Texas Instruments is distributed as "freeware".  You may,
; only under TI's copyright in the Program, use and modify the
; Program without any charge or restriction.  You may
; distribute to third parties, provided that you transfer a
; copy of this license to the third party and the third party
; agrees to these terms by its first use of the Program. You
; must reproduce the copyright notice and any other legend of
; ownership on each copy or partial copy, of the Program.
;
; You acknowledge and agree that the Program contains
; copyrighted material, trade secrets and other TI proprietary
; information and is protected by copyright laws,
; international copyright treaties, and trade secret laws, as
; well as other intellectual property laws.  To protect TI's
; rights in the Program, you agree not to decompile, reverse
; engineer, disassemble or otherwise translate any object code
; versions of the Program to a human-readable form.  You agree
; that in no event will you alter, remove or destroy any
; copyright notice included in the Program.  TI reserves all
; rights not specifically granted under this license. Except
; as specifically provided herein, nothing in this agreement
; shall be construed as conferring by implication, estoppel,
; or otherwise, upon you, any license or other right under any
; TI patents, copyrights or trade secrets.
;
; You may not use the Program in non-TI devices.

#include  "msp430x12x.h"

;*******************************************************************************
;  Ultralow-Power Keypad Interface
;
;  Description: This program implements and ultralow-power keypad interface
;  on the MSP430F12x.  The circuit consumes .1uA in normal mode while waiting
;  for a key press.  After a key press, a s/w debounce is performed and the
;  uC then waits for the key to be released.  The ciruict consumes a maximum
```

```
;  of 2uA in the even the keys are accidentally pressed and held.  The circuit
;  is completely interrupt driven, requires no polling, and requires no
;  external crystal.
;
;
;  Mike Mitchell
;  MSP430 Applications
;  Texas Instruments, Inc
;  January, 2002
;
;*****************************************************************************
             RSEG    CSTACK                     ; System stack
             DS      0

;*****************************************************************************
             RSEG    UDATA0                     ; RAM Locations
;*****************************************************************************


NoKey                EQU  01h
NoMatch              EQU  02h
Error_Flags          DS      1       ; Error Flags
                                     ; xxxx xxxx
                                     ;         ||
                                     ;         ||-- No Key being depressed
                                     ;         |----- No key match found

;*****************************************************************************
             RSEG    CODE                       ; Program code
;*****************************************************************************

Reset        mov     #SFE(CSTACK),SP            ; Initialize stackpointer
SetupWDT     mov     #WDTPW+WDTHOLD,&WDTCTL     ; Stop WDT
SetupPorts   mov.b   #0F8h,&P1DIR               ; Unused P1.x as Outputs
             mov.b   #0FFh,&P2DIR               ; Unused P2.x as outputs
             mov.b   #0FFh,&P3DIR               ; All P3.x as outputs

             eint                               ; Enable Interrupts

SetupDCO     mov.b   #0,&BCSCTL1                ; Set Rsel=0, leave DCO=3
                                                ; This gives nom MCLK of
                                                ; 130KHz at 3V, 25C.

Mainloop     call    #Set_For_Press             ; Setup to wait for key press
             bis     #LPM4,SR                   ; Wait for key press
             call    #Debounce                  ; Call debounce delay
             call    #KeyScan                   ; Scan Keypad
             bit.b   #NoKey,Error_Flags         ; Test if no key was depressed
             jnz     Mainloop                   ; False interrupt, no key pressed
             call    #KeyLookup                 ; Lookup Key value
             call    #Wait_For_Release          ; Wait for key(s) to be released
             jmp     Mainloop                   ;

;-----------------------------------------------------------------------------
Set_For_Press       ; Setup to wait for key press
;-----------------------------------------------------------------------------
             bis.b   #BIT0+BIT1+BIT2+BIT3,&P3OUT ; Enable keypad
             bic.b   #BIT0+BIT1+BIT2,&P1IES     ; L-to-H interrupts
             clr.b   &P1IFG                     ; Clear any pending flags
             mov.b   #BIT0+BIT1+BIT2,&P1IE      ; Enable interrupts
```

```
              clr.b   Error_Flags             ; Clear error flags

              ret
;-------------------------------------------------------------------------------
Debounce  ; Debounce Delay Routine
;-------------------------------------------------------------------------------
SetupTA       mov     #TASSEL1+TACLR,&TACTL   ; SMCLK, Clear TA
              mov     #CCIE,&TACCTL0          ; Enable CCR0 interrupt
              mov     #5125,&TACCR0           ; Value for typ delay of ~40mS
              bis     #MC0,&TACTL             ; Start TA in up mode
              bis     #LPM0,SR                ; Sleep during debounce delay

              ret                             ; Return
;-------------------------------------------------------------------------------
KeyScan    ; Keypad Routine
;-------------------------------------------------------------------------------
#define   KeyMask      R15
#define   LoopCount    R14
#define   KeyHex       R13
#define   KeyVal       R5

              mov        #1,KeyMask           ; Initialize scan mask
              mov        #4,LoopCount         ; Initialize loop counter
              clr        KeyHex               ; Clear register
              bic.b      #07h,&P1OUT          ; Clear column bits in P1OUT reg
Scan_1        bic.b      #0Fh,&P3OUT          ; Stop driving rows
              bis.b      #07h,&P1DIR          ; Set column pins to output and low
              bic.b      #07h,&P1OUT          ; To bleed off charge and avoid
                                              ; erroneous reads
              bic.b      #07H,&P1DIR          ; Set column pins back to input
              Mov.b      KeyMask,&P3OUT       ; Drive row
              bit.b      #7h,&P1IN            ; Test if any key pressed
              jz         Scan_2               ; No key pressed
              bis.b      KeyMask,KeyHex       ; If yes, set bit for row
              mov.b      &P1IN,R12            ; Read column inputs
              and.b      #07h,R12             ; Clear unused bits
              rla.b      R12                  ;
              rla.b      R12                  ; Rotate column bit
              rla.b      R12                  ;
              rla.b      R12                  ;
              bis.b      R12,KeyHex           ; Set column bit in KeyHex
Scan_2        rla.b      KeyMask              ; Rotate mask
              dec        LoopCount            ; Decrement counter
              jnz        Scan_1               ; Continue scanning if not done


; Check to see if any key is being pressed.  If not, set flag and return.
              tst.b      KeyHex               ; Test KeyHex
              jnz        EndScan              ; If not 0 return
              bis.b      #NoKey,Error_Flags   ; Set flag


EndScan       bis.b      #0Fh,&P3OUT          ; Drive rows again
              ret
;-------------------------------------------------------------------------------
KeyLookup    ; Table look-up to determine what key was pressed.
;-------------------------------------------------------------------------------
              mov        #10,KeyVal           ; Initial key value
LookLoop      cmp.b      Key_Tab(R5),KeyHex   ; Compare
              jeq        EndLU                ; If equal end look-up
```

```
            dec       KeyVal                  ; decrement pointer/counter
            jnz       LookLoop                ; Continue until find key or
                                              ; count to zero.

EndError  ;  If get here, Did not find match, so more than one key is pressed.
          ;  return error condition
            bis.b     #NoMatch,Error_Flags    ; Set Error Flag
            ret                               ; Return

EndLU     ; Done with Key look-up - found key successfully.
            dec       KeyVal                  ; Adjust because using same
                                              ; register for key counter
                                              ; and table pointer
          ; --> The key that was pressed is now in R5.  The applicaion
          ; can now move it for furthur handling, display, etc.
          ; This example doesn't actually do anything with the key information.

            ret
;-------------------------------------------------------------------------------
Wait_For_Release      ; Setup to wait for key release
;-------------------------------------------------------------------------------
; Isolate one row that is in use

            mov.b     #1,R11                  ; row counter
L$1         and.b     #0Fh,KeyHex             ; And off column info in KeyHex
            rrc       KeyHex                  ; Rotate row info through C
            jc        proceed                 ; Looking for a '1'
            rla       R11                     ; Shift to next bit and
            jmp       L$1                     ; continue looking

proceed     inv.b     R11                     ; Invert
            and       #0Fh,R11                ; Clear upper bits
            bic.b     R11,&P3OUT              ; Turn off all but one row

; Setup for interrupt on key release
            bis.b     #07h,&P1DIR             ; Set column pins to output and low
            bic.b     #07h,&P1OUT             ; To bleed off charge and avoid
                                              ; erroneous reads
            bic.b     #07H,&P1DIR             ; Set column pins back to input
            bis.b     #07h,&P1IES             ; H-L Interrupts
            clr.b     &P1IFG                  ; Clear any pending flags
            bis.b     #07h,&P1IE              ; Enable Interrupts
            bis       #LPM4,SR                ; Sleep waiting for release
            Call      #Debounce               ; Debounce release of key
            call      #KeyScan                ; Scan keypad again
            bit.b     #NoKey,Error_Flags      ; Test if any key pressed
            jz        Wait_For_Release        ; If so, repeat

End_Wait    bic.b     #NoKey,Error_Flags      ; Clear flag
            ret                               ; Return
;-------------------------------------------------------------------------------
P1ISR  ; P1.x Interrupt service Routine
;-------------------------------------------------------------------------------
            bic     #LPM4,0(SP)               ; Return active
            clr.b   &P1IFG                    ; Clear interrupt flag
            clr.b   &P1IE                     ; Disable furthur P1 interrupts
            reti
;-------------------------------------------------------------------------------
CCR0_ISR  ; CCR0 Interrupt Service Routine
;-------------------------------------------------------------------------------
```

```
            bic     #LPM0,0(SP)             ; Return Active
            mov     #TACLR,&TACTL           ; Stop and clear TA
            clr     &TACCTL0                ; Clear CCTL0 register
            reti
;-------------------------------------------------------------------------------
Key_Tab   ; Key look-up table
;-------------------------------------------------------------------------------
            DB      00h   ; Dummy value.  Allows use of same register for
                          ; both table pointer and key counter
            DB      028h  ; '0' key
            DB      011h  ; '1' key
            DB      021h  ; '2' key
            DB      041h  ; '3' key
            DB      012h  ; '4' key
            DB      022h  ; '5' key
            DB      042h  ; '6' key
            DB      014h  ; '7' key
            DB      024h  ; '8' key
            DB      044h  ; '9' key
;-------------------------------------------------------------------------------
            COMMON  INTVEC                  ; Interrupt vectors
;-------------------------------------------------------------------------------

            ORG     RESET_VECTOR
            DW      Reset
            ORG     TIMERA0_VECTOR
            DW      CCR0_ISR
            ORG     PORT1_VECTOR
            DW      P1ISR
;-------------------------------------------------------------------------------
            END
```

4 Low-Power Implementation on MSP430 FRAM MCUs

These resources give additional information about keypad applications based on the MSP430 FRAM-based microcontrollers:

Low-Power Hex Keypad Using MSP430 MCUs implements a completely interrupt-driven approach with minimal use of external components.

Infrared (IR) BoosterPack Plug-In Module includes a low-power hex keypad implementation.

MSP430 Capacitive Sensing Microcontrollers enable capacitive-touch keypad implementations.

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from February 5, 2002 to May 22, 2018	Page
• Editorial changes throughout document	1
• Added Section 4, *Low-Power Implementation on MSP430 FRAM MCUs*	10