

HC05 MCU Keypad Decoding Techniques Using the MC68HC705J1A

By David Yoder
CSIC Applications

Introduction

This application note demonstrates the use of a matrix keypad including wakeup from stop mode with HC05 J and K series microcontrollers. The MC68HC705J1A is used as an example.

The code is divided into a main routine and two subroutines. The main routine handles stop mode and the interrupt service routine that acts on the key being pressed. The keypad subroutine actually decodes the keypad. The delay subroutine is used by the interrupt service routine to debounce the key press and key release.

Features

4 x 4 Matrix Keypad

A matrix keypad allows a designer to implement a large number of inputs with a small number of port pins. 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 square a matrix as possible. As an example of a non-square matrix using more pins, if 16 keys were arranged in an 8 x 2 matrix, 10 keys would be required instead of eight.

Low-Component Count

A matrix keypad requires the use of pulldown resistors. These pulldowns have been built into the HC705J1A as well as several other Motorola MCUs. This minimizes the need for external components and their related cost.

In a battery-operated device, such as a remote control, current consumption is paramount. Stop mode in HC05 parts often is used to minimize current consumption when the microcontroller is not needed. This mode stops the crystal or ceramic resonator from running, thereby lowering the MCU's current draw. To exit STOP and resume processing, an external reset or interrupt is required. The MC68HC705J1A and several other HC05 MCUs contain circuitry that minimizes the need for external components to connect the keypad to the external interrupt pin or the hardware reset pin.

The ceramic resonator is not related to the keypad but demonstrates a low-component count. A three-pin device includes the resonator and load capacitors in one package. The MC68HC705J1A's internal bias resistor mask option eliminates the external resistor. The oscillator circuit requires only one external component in this arrangement.

Low-Power Consumption

Pulldowns also draw current. While a key is pressed, pulldowns are shorted to an output that is driving high. While waiting for the debounce delay, the current draw can be minimized by driving the outputs low after the decoding is complete. The outputs must be re-configured to high before the STOP instruction is executed so that they can pull a pulldown up and cause an interrupt.

Floating inputs are another source of excess current in CMOS circuitry. To ensure that floating inputs are tied to ground, the MC68HC705J1A has software programmable pulldown resistors on all input/output pins.

High-Current Sink Pins

This part has high-current sink capability on pins PA4 through PA7. This keypad code leaves those pins free for use and does not modify their state. The project in the appendices uses them to drive LEDs that show the code of the key that was pressed.

Computer Operating Properly Watchdog

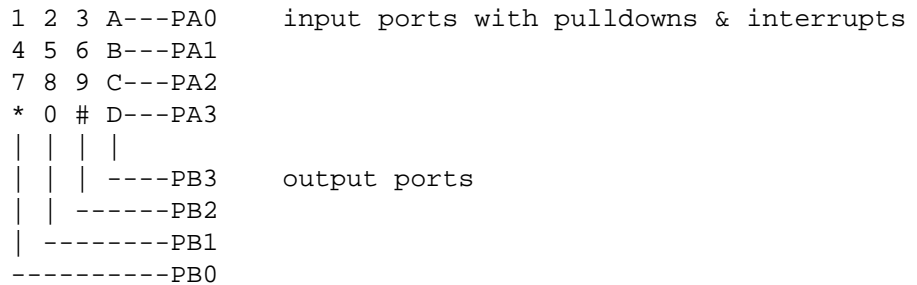
The COP watchdog is serviced during the delay routine used for debounce. This allows the watchdog to catch runaway code and reset the part if a problem occurs.

Key Repeat

Often a signal should be sent as long as the corresponding key is pressed. For that reason, this routine loops until the key is released.

Implementation

Keypad decoding works by combining a matrix of switches with resistor pull-downs. The keypad is to be connected in the following fashion:



The wakeup on keypress and keypad decoding can be considered separately. Wakeup from STOP requires an external \overline{IRQ} signal. The MC68HC705J1A has circuitry to create an interrupt if any one of the port A0 through A3 pins goes high. The \overline{IRQ} edge/level sensitivity bit applies to these pins also. In addition, all pins of the MC68HC705J1A have internal pull-down devices that are enabled when the ports are programmed as inputs.

To use this feature, port B0 through B3 are programmed to output high logic levels. Now, if any one of the keys are pressed, an output high is shorted to an input with a pull-down. The output has enough drive current to defeat the pull-down and the result is a high on the input of the Port A pin. The internal circuitry latches an interrupt request, bringing the part out of STOP mode and executing code at the external interrupt vector.

Now that the interrupt service routine is executing, the keypad can be decoded to find out which key was pressed. This is done in the subroutine KEYPAD.SUB by matching a row with a column. Each column is set to output high while the other columns are output low. For each column, all rows are checked until one is found to be high. Rows that are not shorted by a keypress to the column that is driving high will be either pulled low by a pull-down or (if they are shorted to a column that is driving low) driven low. The matching is done by writing the columns to a value from a table, and then comparing the input value with another entry in the table. When a column and row are matched, the appropriate code is returned. If no match is found, a zero is returned.

The core of the keypad decoding subroutine is:

```

lda   portb           ;Get value in port B
and   #$f0           ;Do not allow high nibble to change
ora   KeyPad_Table+1,x ;Get key decode value from table
sta   portb           ;Write to port

lda   porta           ;Get value in port A
and   #$0F           ;Throw out columns to read only rows
cmp   KeyPad_Table,x ;See if high nibble bit was pulled low
beq   KeyPad030       ;If key found, branch

```

This code outputs an entry from the decode table on the low nibble of port B. A comparison is made between the low nibble of port A to another table entry to see if the matching column was pulled high. If a match was made, the code for that key is returned. Care is taken to retain the state of pins not used by the keypad.

After the decoding is done, several milliseconds will be spent just delaying for key debounce. Since it is likely that a key will be held down during this period, and that a key pressed will short an output high to a pulldown device and draw unnecessary current, the code should set the column outputs to low. That way, no current will be drawn by a pressed key.

The following code sets the low nibble of port B to the same level as the pulldowns:

```

KeyPad035:  lda   portb ;'Help' the pulldowns by driving the
            and   #$F0  ; lines low. This minimizes current
            sta   portb ; draw while debouncing.

```

The appendices show a framework for a project using a keypad and stop mode when not decoding. Operations to be performed when a key is pressed are placed in the interrupt service routine. The example simply outputs the code for each key pressed on LEDs attached to PA4 through PA7. These pins have high current sink capability. Therefore, setting the pin to output low turns the LED on. The codes, shown in the table at the end of the KEYPAD.SUB subroutine, are first complemented and then written to the high nibble of port A.

This project has been designed and implemented using Carnegie-Mellon Software Engineering Institute Level 2 requirements. The software is available on the Motorola CSIC BBS. To access the

software, set your modem software to eight data bits, no parity, and one stop bit. The BBS phone number is:

(512) 891-3733

The file is under the app notes file area and has the name an1239.zip.

Modifications

Using a repeat bit, the code can be changed to repeat only certain keys.

The last key pressed can be stored in a variable to give a longer repeat delay for the first repeat and then a fast repeat.

If low-power operation is not needed, the subroutine KeyPad_Body and its associated initialization, Key Pad_Init, can be called without the rest of the code to create a polled keypad routine.

An MC34064 low-voltage reset has been included to show the most robust $\overline{\text{RESET}}$ circuit. This provides protection from slow-ramping power supplies. Many bench-type power supplies ramp slowly, causing faulty power-on of MCUs. The MC34064 holds $\overline{\text{RESET}}$ pin low until the power supply is within a specified range. An internal pullup device on the MC68HC705J1A brings the $\overline{\text{RESET}}$ pin high when the MC34064 no longer drives it low. This also provides protection from brownout, when the MCU's minimum V_{DD} requirements are exceeded. If such robust protection is not required, engineering judgment may be used to design a more cost-effective circuit.

Appendix A: Software

```

*****
*****
*
*           Main Routine KeyPdInt - Low Power Keypad Interface
*
*****
*
* File Name: KeyPdInt.RTN           Copyright (c) Motorola 1994
*
* Full Functional Description Of Routine Design:
*   Program flow:
*     Reset:  Calls init routine to setup port DDR's and data regs
*             STOP to remain in low power mode when key is not pressed
*             Loop to STOP instruction after returning from interrupt
*     ISR:    Call Keypad routine to see if a key is down. Just return
*             if it was a 'ghost'
*             If key was there, debounce keypad with DelaymS routine
*             If no key was there, just return
*             If key was there, perform action based on value returned
*             by Keypad routine.
*             Branch to beginning of ISR to see if the key is still being
*             pressed.
*             Return path: delay to debounce the release of the key
*                         RTI to return to main loop
*
*****
*****
*
*           MOR Bytes Definitions for Main Routine
*
*****

org      MOR
db      PIRQ.+OSCRE.S. ;Enable Port A Interrupts
                          ;If used on a mask rom part,
                          ; be sure to specify this option.
    
```

Freescale Semiconductor, Inc.

Application Note

```

*****
*
*           Program Initialization
*
* This routine sets up the high nibble of port a to drive LED's
* with it's high sink current. Due to the use of sink current,
* the LED's will be on when an low is output and off when a high
* is output.
*
* It then calls the Keypad_Init routine to setup the ports to
* interrupt the processor when a key is pressed.
*
* To prevent floating inputs and associated high current draw,
* the HC705J1A has pulldown devices on all I/O pins. This
* initialization should enable these pulldowns on unused I/O
* pins. RESET_ enables the pulldowns, so no code is required.
*
*****

```

```

                org      EPROM
Start:
KeyPdInt_Init:  lda      #00                ; This is for JICS only.
                                           ; JICS gives an error if an
                                           ; uninitialized register is used.
                                           ; X is "used" when it is stacked
                                           ; during a keypad interrupt service.
                lda      #$F0            ;Set the high nibble as output
                sta      PORTA           ; high. This enables output drive
                STA      DDRA            ; for LED's but turns them off.
                jsr      Keypad_Init     ;Set up the ports to interrupt
                                           ; on a keypress.

```

Freescale Semiconductor, Inc.


```
*****  
*                                                                 *  
*           KeyPdInt Main Program Loop                           *  
*                                                                 *  
* This section simply services the COP watchdog and then enters STOP mode. *  
* All other program execution is contained in the KeyPdInt_Isr, the *  
* external interrupt service routine for this code.               *  
*                                                                 *  
*****
```

```
KeyPdInt_Body:
    STOP                ;Execute STOP instruction to put
                        ; MCU in lowest power mode.
                        ; The keypad can exit from STOP.
                        ; STOP clears the I bit so CLI is
                        ; not needed.
                        ;When RTI returns from ISR, I bit
                        ; will be clear, enabling ints.
    bra    KeyPdInt_Body ;Infinite loop to stay in STOP.
```

```

*****
*
*           IRQ Interrupt Service Routine
*
* This is the external interrupt service routine. Both the external
* interrupt pin IRQ_ and the keypad interrupts use this routine. The real
* work of the program is done withing this service routine.
*
*****
KeyPdInt_Isr:      ; Any decoding of external interrupts should be done here.
                  ; The external and keypad interrupt share this vector.
KeyPdInt_Isr010:
    jsr    KeyPad_Body          ;See if a key is pressed
                                      ;If no key down, return
                                      ; to save power
    beq    KeyPdInt_Isr090
    lda    #$4                  ;Debounce key for 4mS
    jsr    DelaymS2_Body       ;Jump to delay routine
    jsr    KeyPad_Body         ;Get the keypress
KeyPdInt_Isr020:
    beq    KeyPdInt_Isr090      ;If no key down, return

    ;Operations that are to be performed based on a key should
    ; be placed here. This example will just flash the code.
    coma                                ;Complement the result
                                      ; because the LED's are
                                      ; negative logic.
    lsla                                ;Move the 4bit result into
    lsla                                ; the high nibble.
    lsla
    lsla
    sta    PORTA                  ;Output the result for view.
    lda    #!200                 ;Show the result for 200mS.
    jsr    DelaymS2_Body
    lda    #$F0                  ;Turn off the LED's
    sta    PORTA
KeyPdInt_Isr080:
    bra    KeyPdInt_Isr010       ;Back to beginning to repeat

KeyPdInt_Isr090  lda    #!10      ;Delay 10 mS
    jsr    DelaymS2_Body         ;Debounce the release
    jsr    KeyPad_Init          ;Set up the port to interrupt
    bset   IRQR,ISCR            ;Clear any interrupt requests
                                      ; generated due to key bounce
    rti                          ;Return from Interrupt.
                                      ;Interrups can happen in any
                                      ; code in the main routine
                                      ; after this ISR has been
                                      ; called once.
                                      ;Remember this when changing
                                      ; the main routine!

```

```
*****
*
*           Subroutine Body Includes Section
*
* These include statements include the subroutines that are called by
* this program.
* KeyPad.SUB      actually decodes the keypad
* KelaymS.SUB     delays operation in increments of milliseconds
*
*****
```

```
#INCLUDE 'DelaymS2.SUB'           ;Millisecond delay subroutine
```

```
#INCLUDE 'KeyPad.SUB'           ;Keypad decode subroutine
```

```
*****
*
*           Interrupt and Reset vectors for Main Routine
*
*****
```

```
org     RESET
fdb     Start
org     IRQ_INT
fdb     KeyPdInt_Isr
```



```

*      1 2 3 A---PA0   input ports with pulldowns & interrupts      *
*      4 5 6 B---PA1                                     *
*      7 8 9 C---PA2                                     *
*      * 0 # D---PA3                                     *
*      | | | |                                     *
*      | | | ----PB3   output ports                          *
*      | | -----PB2                                     *
*      | -----PB1                                     *
*      -----PB0                                     *
*
*      Key      Row      Col      PA      PB      *
*      1         0         0         1         1      *
*      2         0         1         1         2      *
*      3         0         2         1         4      *
*      A         0         3         1         8      *
*      4         1         0         2         1      *
*      5         1         1         2         2      *
*      6         1         2         2         4      *
*      B         1         3         2         8      *
*      7         2         0         4         1      *
*      8         2         1         4         2      *
*      9         2         2         4         4      *
*      C         2         3         4         8      *
*      *         3         0         8         1      *
*      0         3         1         8         2      *
*      #         3         2         8         4      *
*
*****

```

Freescale Semiconductor, Inc.

```
*****
*
*                               Keypad Initialization                               *
*
* This code sets up the low nibble of ports A and B to decode a 4x4 matrix *
* keypad. This does not affect the high nibble of the port data or data *
* direction registers. *
*
*****
```

```
KeyPad_Init:
    lda    ddra           ;Set the low nibble of port a as input
    and    #$F0           ; without affecting the high nibble.
    sta    ddra           ;This also enables the pulldowns.

    lda    portb          ;Set the low nibble of port b to high.
    ora    #$0F           ; This will defeat the pulldowns on
    sta    portb          ; port A if a key is pressed.

    lda    ddrb           ;Set the low nibble of Portb as output.
    ora    #$0F
    sta    ddrb

    clr    PDRA           ;Ensure that the pulldowns on port a
                        ; are not disabled.

    rts                  ;Return to calling program.
```

```
*****
*
* KeyPad_Body *
*
* This subroutine decodes a 4 x 4 matrix keypad on port B. *
*
*****
```

```
KeyPad_Body:                                ;Load X with the offset of the last
                                           ; entry in the table
    ldx   #{KeyPad_Table_Top - KeyPad_Table}

KeyPad010:
    lda   portb           ;Get value in port B
    and   #$f0           ;Do not allow high nibble to change
    ora   KeyPad_Table+1,x ;Get key decode value from table
    sta   portb           ;Write to port

    lda   porta           ;Get value in port A
    and   #$0F           ;Throw out columns to read only rows
    cmp   KeyPad_Table,x ;See if high nibble bit was pulled low
    beq   KeyPad030       ;If key found, branch
```

```

    decx                ;Decrement X three times to point to
    decx                ; next value in table
    decx
    bpl    KeyPad010    ;If not below bottom of table
                    ; try again.
    ldx    #$00         ;A key was not decoded, so:
    bra    KeyPad035    ;Return with null character

KeyPad030:
    lda    KeyPad_Table+2,x ;Load key code into Acc.
    tax
                    ;Store in X for now.

KeyPad035:
    lda    portb        ;'Help' the pulldowns by driving the
    and    #$F0         ; lines low. This minimizes current
    sta    portb        ; draw while debouncing.

                    ;Get result back to Acc.
    txa
    tsta                ;Set the flags so calling routine
                    ; can use them for decisions.

KeyPad040
    rts                ;Return with result value in Acc

                    ;Table of keypad decode values and codes.
                    ;Fill in your own key codes. Codes must be 1
                    ; byte each. Currently limited to 4 bits to
                    ; display on PA[4..7].

KeyPad_Table
    DB    $01,$01,$1    ; Row    Column
    DB    $01,$02,$2    ; PA0    PB0
    DB    $01,$04,$3    ; PA0    PB1
    DB    $01,$08,$A    ; PA0    PB2
    DB    $02,$01,$4    ; PA0    PB3
    DB    $02,$02,$5    ; PA1    PB0
    DB    $02,$04,$6    ; PA1    PB1
    DB    $02,$08,$B    ; PA1    PB2
    DB    $04,$01,$7    ; PA1    PB3
    DB    $04,$02,$8    ; PA2    PB0
    DB    $04,$04,$9    ; PA2    PB1
    DB    $04,$08,$C    ; PA2    PB2
    DB    $08,$01,$F    ; PA2    PB3
    DB    $08,$02,$E    ; PA3    PB0
    DB    $08,$04,$F    ; PA3    PB1
    KeyPad_Table_Top
    DB    $08,$08,$D    ; PA3    PB2
                    ; PA3    PB3

```

Application Note

```

*****
*****
*
*       Subroutine Delaysms2 - Delay for whole number of milliseconds
*
*****
*
* File Name: delaysms2.SUB                      Copyright (c) Motorola 1994
*
* Full Functional Description of Module Design:
*
* This routine delays operation for a whole number of milliseconds.
* The number of milliseconds to delay is passed in the accumulator
* The routine alters Acc, X and CCR.
* A 4 MHz clock (2 MHz bus) is assumed.
* The smallest delay is 2012 cycles which occurs when Acc = 1. (1 ms)
* The largest delay is 512012 cycles which occurs when Acc = 0. (256 ms)
*
* Please note that passing 0 will NOT result in zero delay, but 256 ms delay.
*
* The number of milliseconds to delay is passed in the accumulator. The
* routine is formed by two loops. The inner loop (Delaysms020) executes in
* 1986 cycles. The outer loop executes once for each millisecond and adds
* 14 bus cycles each time through the loop. This creates 2000 cycles for
* each millisecond of delay. The RTS used to exit the routine add 6 bus
* cycles to the total time. The JSR used to enter the routine may add 5
* or 6 bus cycles, for direct or extended addressing, respectively.
*
* The exact number of cycles for this routine to execute may be calculated
* from (Assuming extended addressing):
*
*           cycles = 6+Acc(2+248(3+2+3)+5+3+3+3)+6           order of execution
*
* or:
*           cycles = 12 + ( Acc * 2000 )                     simplified
*
* Upon exit, the accumulator and index register will be zero.
*
*****

```

Freescale Semiconductor, Inc.

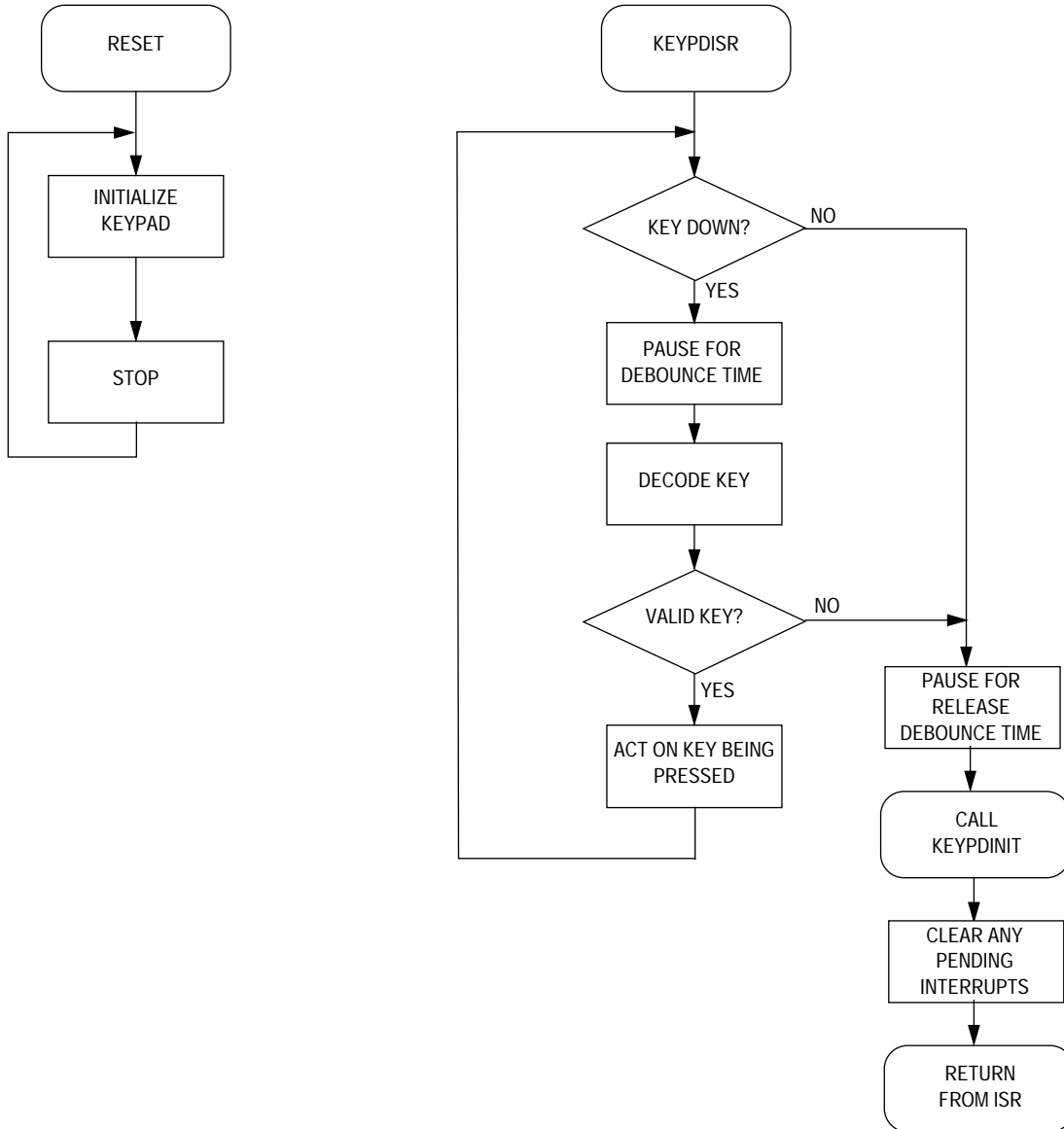

```

*****
*
* Delay for Xms
*
* Inner loop delays 1 ms. Outer loop counts ms.
* Number of ms in passed through the accumulator.
*
*****
Delayms2_Body:                ;JSR EXT to get here      6
Delayms2010    ldx    #$F8    ;Load delay into X      2--\
Delayms2020    decx   ;    Decrement delay      3-\|
               nop      ;    burn 2 bus cycles  2 ||
               bne    DelaymS2020 ;    Branch if not done 3-/\
               stx    COPR    ;Service the WDOG      5 |
               ;Note that X will
               ;always be zero here
               brn    *      ;Burn 3 bus cycles    3 |
               deca   ;decrement # of mS        3 |
               bne    DelaymS2010 ;branch if not done 3--/
Delayms2030    rts      ;return                    6

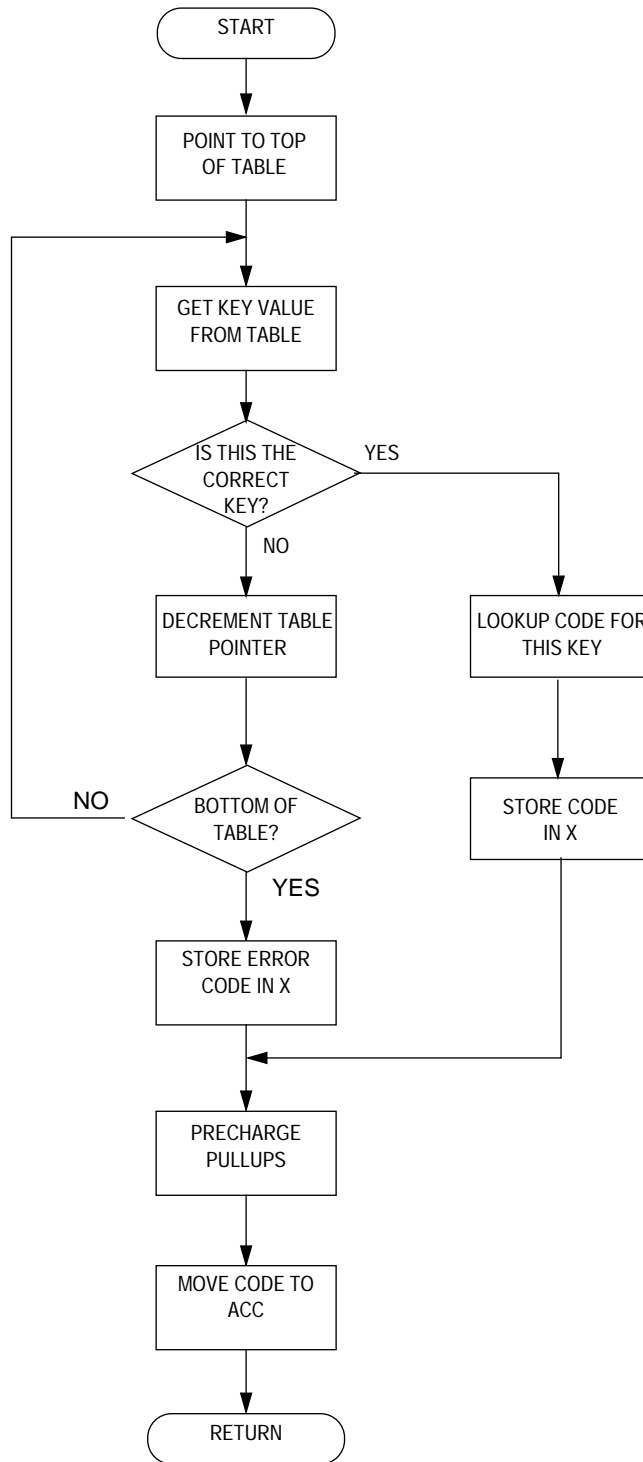
```

Appendix B: Flowcharts

Main Routine and External Interrupt Service Routine:

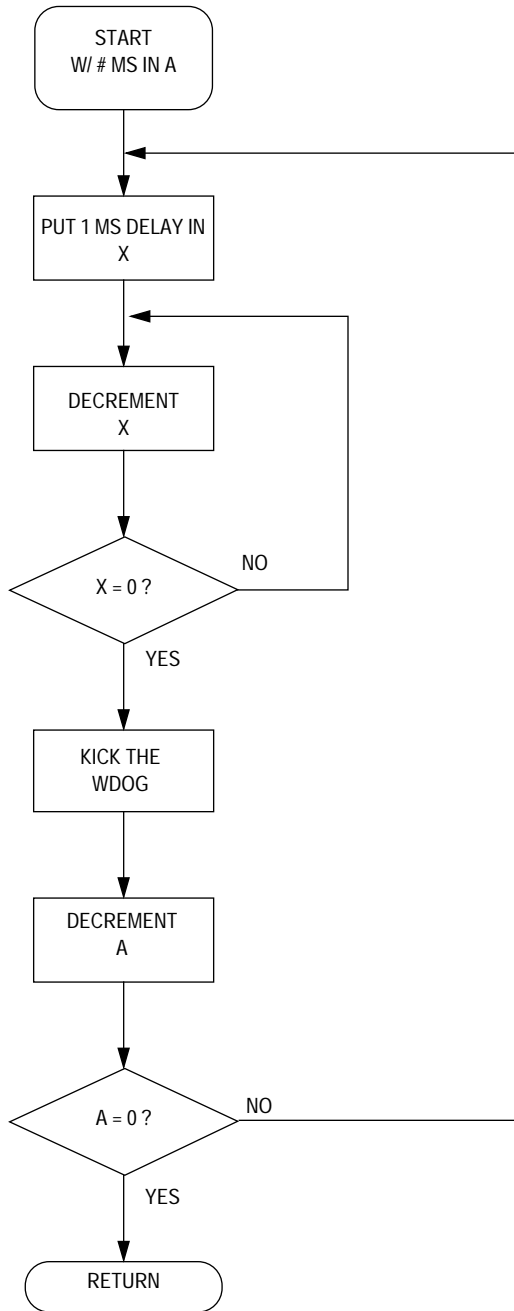


Keypad Decode Subroutine:

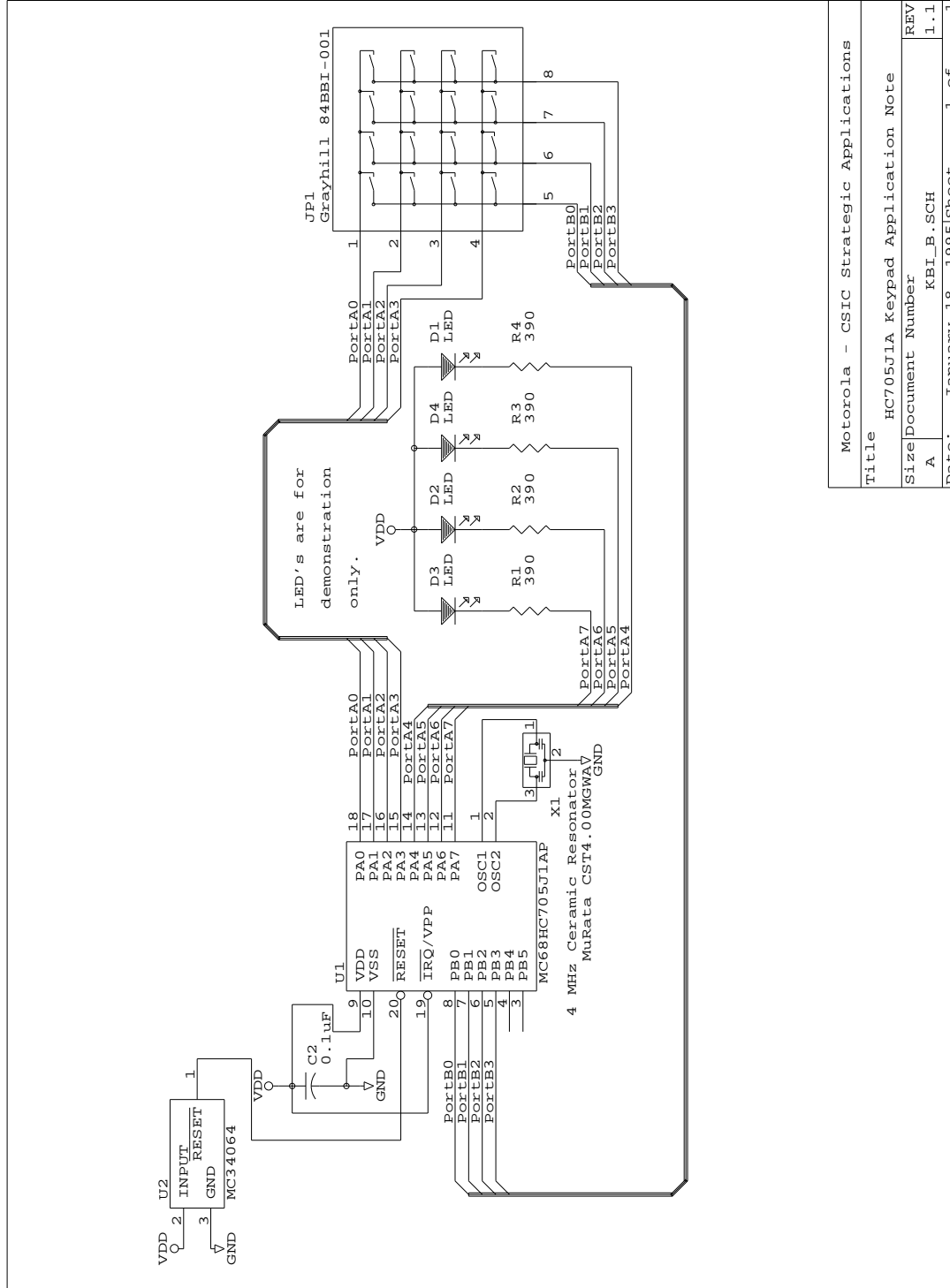


Application Note

Delay Subroutine:



Appendix C: Schematic



Motorola - CSIC Strategic Applications	
Title	HC705J1A Keypad Application Note
Size/Document Number	A KBI_B.SCH
REV	1.1
Date:	January 18, 1995
Sheet	1 of 1

NOTES

NOTES

Application Note

How to Reach Us:

Home Page:

www.freescale.com

E-mail:

support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

