ECE3120: Computer Systems
Chapter 7: Interfacing I/O Devices
Lab-Class

Manjeera Jeedigunta
http://blogs.cae.tntech.edu/msjeedigun21
Email: msjeedigun21@tntech.edu
Tel: 931-372-6181, Prescott Hall 120
Today

- Interfacing with LEDs
- Interfacing with Seven-Segment Display
- Time-Multiplexing
- Interfacing with Keypad
- Debouncing
Example 1: Use Port B to drive eight LEDs using the circuit shown in Figure 7.30. Use the LEDs to display the value of a counter counting from 1 to 255 each LED lighting up for 1ms.

Set DDRB for O/P, i.e., DDRB=1

Enable LEDs, Pin 1 of Port J = 0

for i = 1 to 255

    PTA = i

end
The assembly program that performs the operation is as follows: File1.asm

```
#include "hcs12.inc"
org $1000
led_c dc.b $FF ; initial value in the counter
temp dc.b $1
org $1500
movb #$FF,DDRB ; configure port B for output
bset DDRJ,$02 ; configure PJ1 pin for output
bclr PTJ,$02 ; enable LEDs to light
forever ldaa led_c ; load a with counter
led_lp staa temp ; turn on one LED
movb temp,PTB ; wait for 1ms second
ldy #5
jsr delayby100ms ;
dbne a,led_lp ; reach the end of the table yet?
bra forever ; start from beginning
swi
#include "delay.asm"
end
```
Just LEDs – Example 2

- Disable the seven segment display if you need to work with LEDs alone
  - `movb #$FF,DDRP ; configuring the digit select port for o/p`
  - `movb #$FF,PTP ; disabling the digits`

- Adding these two instructions to your previous code does this.

- Try `File1a.asm`
Light up even numbered LEDs File2.asm- Example 3

```
#include "hcs12.inc"
org $1000

led_tab dc.b $01,$04,$10,$40 ;initial value in the counter

org $1500
movb #$FF,DDRB ; configure port B for output
bset DDRJ,$02 ; configure PJ1 pin for output
bclr PTJ,$02 ; enable LEDs to light
movb #$FF,DDRP ;configure Port P for O/P
movb #$FF,PTP ;Disable the Digits

forever ldaa #4 ;initialize the counter
ldx #led_tab ; load a with counter

led_lp movb 1,x+,PORTB ; turn on one LED
ldy #5 ; wait for 1ms second
jsr delayby100ms ; "
dbne a,led_lp ; reach the end of the table yet?
bra forever ; start from beginning
swi

#include "delay.asm"
end
```
Driving a Seven-Segment Display

Table 7.5 BCD to seven-segment decoder

<table>
<thead>
<tr>
<th>BCD digit</th>
<th>Segments</th>
<th>Corresponding Hex Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 1 1 1 1 1 1 0</td>
<td>$7E$</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 0 0 0 0 0</td>
<td>$30$</td>
</tr>
<tr>
<td>2</td>
<td>1 1 0 1 1 0 1 0</td>
<td>$6D$</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1 1 0 0 0 1</td>
<td>$79$</td>
</tr>
<tr>
<td>4</td>
<td>0 1 1 0 0 1 1 1</td>
<td>$33$</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1 1 1 1 0 1</td>
<td>$5B$</td>
</tr>
<tr>
<td>6</td>
<td>1 0 1 1 1 1 1 1</td>
<td>$5F$</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1 0 0 0 0 0</td>
<td>$70$</td>
</tr>
<tr>
<td>8</td>
<td>1 1 1 1 1 1 1 1</td>
<td>$7F$</td>
</tr>
<tr>
<td>9</td>
<td>1 1 1 1 1 0 1 1</td>
<td>$7B$</td>
</tr>
</tbody>
</table>

Figure 7.31 Driving a single seven-segment display

Figure 7.32 Port B and Port K together drive six seven-segment displays (MC9S12DP256)
Driving a Seven-Segment Display

This table is fine if the Port B pins were wired to the segments in the same way.. But guess what?? They are not!!

<table>
<thead>
<tr>
<th>IMAX = 70 mA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

Works great with that kind of circuit
But we have only 4 digit displays & Port P is the digit select port and not Port k
On Dragon12

Figure 7.32 Port B and Port K together drive six seven-segment displays (MC9S12DP256)
Driving the seven segment display in our case

- Port B = used to display the pattern
  - Should be configured for the o/p

- Port P = used to select the digit for display
  - \( P3=D0, P2=D1, P1=D2, P0=D3 \)
  - Port P should first be configured for output
  - One digit should be enabled for display
    - Set it to 0
  - The other 3 digits should be disabled
    - Set them to 1
Driving the seven segment display in our case

- Port B = used to display the pattern
  - Should be configured for the o/p

- Port P = used to select the digit for display
  - P3=D0, P2=D1, P1=D2, P0=D3
  - Port P should first be configured for output
  - One digit should be enabled for display
    - Set it to 0
    - The other 3 digits should be disabled
      - Set them to 1

Make sure you take down the circuit for our case!! And also take note of the new table of values for patterns and digit selects
Example 4: Write a sequence of instructions to display 0 on the seven-segment display #2 in Figure 1. File3.asm

Solution: To display the digit 0 on the display #2, we need to:

- Output the hex value $3F to port B
- Set the PP1 pin to 0
- Clear pins PP3, PP2, PP0 to 1

```assembly
#include "hcs12.inc"

org $1000
four equ $33 ; seven-segment pattern of digit 0
org $1500

movb #$0F,DDRP ; configure PORT P for output
movb #$FF,DDRB ; configure PORT B for output
bset PTP,$0D ; disable the remaining digits by setting the digits to 1
bclr PTP,$02 ; enable the required digit by setting it to 0
movb #$3f,PTB ; output the seven-segment pattern to PORTB
swi
end
```

Now try displaying the same number on the other digits ONE AT A TIME
Example 4 Write a program to display 1234 on the six seven-segment displays shown in Figure 1.

File4.asm

Solution: Display 1234 on display #3, #2, #1, #0, respectively.

The values to be output to Port B and Port P to display one digit at a time is shown in Table

<table>
<thead>
<tr>
<th>Seven-Segment</th>
<th>Displayed BCD Digit</th>
<th>Port B</th>
<th>PortP</th>
</tr>
</thead>
<tbody>
<tr>
<td>#3</td>
<td>1</td>
<td>$06</td>
<td>$0E</td>
</tr>
<tr>
<td>#2</td>
<td>2</td>
<td>$5B</td>
<td>$0D</td>
</tr>
<tr>
<td>#1</td>
<td>3</td>
<td>$4F</td>
<td>$0B</td>
</tr>
<tr>
<td>#0</td>
<td>4</td>
<td>$66</td>
<td>$07</td>
</tr>
</tbody>
</table>

Try File5.asm for a different pattern

Also try increasing the delay to 1 sec
#include "hcs12.inc"

org $1000

pat_port equ PORTB ; Port that drives the segment pattern
pat_dir equ DDRB ; direction register of the segment pattern
sel_port equ PTP ; Port that selects the digit
sel_dir equ DDRP ; data direction register of the digit select port

org $1500

movb #$FF,pat_dir ; configure pattern port for output
movb #$0F,sel_dir ; configure digit select port for output

forever
  ldx #disp_tab ; use X as the pointer
  loop
    movb 1,x+,pat_port ; output digit pattern and move the pointer
    movb 1,x+,sel_port ; output digit select value and move the pointer
  ldy #1 ; wait for 1 ms
  jsr delayby1ms ;
  cpx #disp_tab+12 ; reach the end of the table
  bne loop
  bra forever

swi

#include "delay.asm"

disp_tab
dc.b $06,$0E ; seven-segment display table Digit 3 (1110)
dc.b $5b,$0D ; Digit2 (1101)
dc.b $4f,$0B ; Digit1 (1011)
dc.b $66,$07 ; Digit0 (0111)
end
Time Multiplexing

- If Refresh Rate is 50 hz
- Total Time period = 1/50 = 20ms
- # of things to multiplex (leds, digits etc) = say 5
- Time for each thing = Total period/# of things = 4ms

```
Enable leds
Disable digits
Light the pattern

Enable dig3
Disable leds
Light the pattern

delay

delay

delay

Enable dig2
Disable leds
Light the pattern

Enable dig1
Disable leds
Light the pattern

delay

delay

Enable dig3
Disable leds
Light the pattern

Try File8.asm & File9.asm
```
Keypad Input Process

• A keyboard input is divided into three steps:

• Scan the keyboard to discover which key has been pressed

• Debounce the keyboard to determine if a key is indeed pressed. Both hardware and software approaches for key debouncing are available.

• Lookup the ASCII table to find out the ASCII code of the pressed key.
Keypad Scanning

- **PA7~PA4** → O/P, Row selection, row being [(0,1,2,3),(4,5,6,7) ..]
- Row being scanned is driven low → either one of PA7~PA4=0
- **PA3~PA0** → I/P, Decide which key is pressed
  - Initially High, when pressed the corr row and column will be shorted
  - When pressed the corresponding PA Pin would be low

![Keypad Diagram]

Table 7.16 Sixteen-key keypad row selections

<table>
<thead>
<tr>
<th></th>
<th>PA7</th>
<th>PA6</th>
<th>PA5</th>
<th>PA4</th>
<th>Selected keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0, 1, 2, and 3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4, 5, 6, and 7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8, 9, A, and B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>C, D, E, and F</td>
</tr>
</tbody>
</table>

Figure 7.41 Sixteen-key keypad connected to the HCS12
Example 5 Write a program to perform keypad scanning, debouncing, and returns the ASCII code in accumulator A to the caller. *File6.asm*

Solution

- Pins PA4..PA7 each control one row of four keys.
- Scanning is performed by setting one of the PA7…PA4 pins to low, the other three pins to high and testing one key at a time.

```assembly
#include "c:\miniide\hcs12.inc"
keyboard equ PTA

get_char          movb #$F0,DDRA ; set PA7~PA4 for output, PA3~PA0 for input
      movb #$EF,keyboard ; scan the row containing keys 0123

scan_k0           brclr keyboard,$01,key0 ; is key 0 pressed?
scan_k1           brclr keyboard,$02,key1 ; is key 1 pressed?
scan_k2           brclr keyboard,$04,key2 ; is key 2 pressed?
scan_k3           brclr keyboard,$08,key3 ; is key 3 pressed?
      bra scan_r1
key0              jmp db_key0
key1              jmp db_key1
```
key2      jmp      db_key2
key3      jmp      db_key3
scan_r1   movb     #$DF,keyboard ; scan the row containing keys 4567
scan_k4   brclr    keyboard,$01,key4 ; is key 4 pressed?
scan_k5   brclr    keyboard,$02,key5 ; is key 5 pressed?
scan_k6   brclr    keyboard,$04,key6 ; is key 6 pressed?
scan_k7   brclr    keyboard,$08,key7 ; is key 7 pressed?
       bra      scan_r2
key4      jmp      db_key4
key5      jmp      db_key5
key6      jmp      db_key6
key7      jmp      db_key7
scan_r2   movb     #$BF,keyboard ; scan the row containing keys 89AB
       bclr     keyboard,$40 ; “
scan_k8   brclr    keyboard,$01,key8 ; is key 8 pressed?
scan_k9   brclr    keyboard,$02,key9 ; is key 9 pressed?
scan_kA   brclr    keyboard,$04,keyA ; is key A pressed?
scan_kB   brclr    keyboard,$08,keyB ; is key B pressed?
       bra      scan_r3
key8      jmp      db_key8
key9      jmp      db_key9
keyA jmp db_keyA
keyB jmp db_keyB
scan_r3 movb #$7F,keyboard ; scan the row containing keys CDEF
scan_kC brclr keyboard,$01,keyC ; is key C pressed?
scan_kD brclr keyboard,$02,keyD ; is key D pressed?
scan_kE brclr keyboard,$04,keyE ; is key E pressed?
scan_kF brclr keyboard,$08,keyF ; is key F pressed?
jmp scan_r0
keyC jmp db_keyC
keyD jmp db_keyD
keyE jmp db_keyE
keyF jmp db_keyF
; debounce key 0
db_key0 jsr delay10ms
    brclr keyboard,$01,getc0
    jmp scan_k1
getc0 ldaa #$30 ; return the ASCII code of 0
    rts
; debounce key 1
db_key1  jsr    delay10ms
           brclr keyboard,$02,getc1
           jmp    scan_k2
getc1     ldaa #$31 ; return the ASCII code of 1
           rts

db_key2  jsr    delay10ms
           brclr keyboard,$04,getc2
           jmp    scan_k3
getc2     ldaa #$32 ; return the ASCII code of 2
           rts

db_key3  jsr    delay10ms
           brclr keyboard,$08,getc3
           jmp    scan_r1
getc3     ldaa #$33 ; return the ASCII code of 3
           rts

db_key4  jsr    delay10ms
           brclr keyboard,$01,getc4
jmp scan_k5
ldaa #$34 ; return the ASCII code of 4

rts

jsr delay10ms
brclr keyboard,$02,getc5
jmp scan_k6

getc5 ldaa #$35 ; return the ASCII code of 5
rts

db_key5 jsr delay10ms
brclr keyboard,$04,getc6
jmp scan_k7

crtc6 ldaa #$36 ; return the ASCII code of 6
rts

db_key6 jsr delay10ms
brclr keyboard,$08,getc7
jmp scan_r2

getc7 ldaa #$37 ; return the ASCII code of 7
rts

db_key7 jsr delay10ms
brclr keyboard,$0A,getc8
jmp scan_r2

getc8 ldaa #$38 ; return the ASCII code of 8
rts

db_key8 jsr delay10ms
brclr keyboard,$0C,getc9
jmp scan_r2

getc9 ldaa #$39 ; return the ASCII code of 9
rts

db_key9 jsr delay10ms
brclr keyboard,$0E,getc0
jmp scan_r2

getc0 ldaa #$30 ; return the ASCII code of 0
rts

db_key0 jsr delay10ms
brclr keyboard,$10,getc1
jmp scan_r2

getc1 ldaa #$31 ; return the ASCII code of 1
rts

db_key1 jsr delay10ms
brclr keyboard,$12,getc2
jmp scan_r2

getc2 ldaa #$32 ; return the ASCII code of 2
rts

db_key2 jsr delay10ms
brclr keyboard,$14,getc3
jmp scan_r2

getc3 ldaa #$33 ; return the ASCII code of 3
rts

jmp scan_r2
getc7 ldaa #$37 ; return the ASCII code of 7
rts

db_key8 jsr delay10ms
brclr keyboard,$01,getc8
jmp scan_k9
getc8 ldaa #$38 ; return the ASCII code of 8
rts

db_key9 jsr delay10ms
brclr keyboard,$02,getc9
jmp scan_kA
getc9 ldaa #$39 ; return the ASCII code of 9
rts

db_keyA jsr delay10ms
brclr keyboard,$04,getcA
jmp scan_kB
getcA ldaa #$41 ; get the ASCII code of A
rts

db_keyB jsr delay10ms
brclr keyboard,$08,getcB
jmp scan_r3
getcB ldaa #$42 ; get the ASCII code of B
rts
db_keyC jsr delay10ms
brclr keyboard,$01,getcC
jmp scan_kD
getcC ldaa #$43 ; get the ASCII code of C
rts
db_keyD jsr delay10ms
brclr keyboard,$02,getcD
jmp scan_kE
getcD ldaa #$44 ; get the ASCII code of D
rts
db_keyE jsr delay10ms
brclr keyboard,$04,getcE
jmp scan_kF
getcE ldaa #$45 ; get the ASCII code of E
rts
db_keyF jsr delay10ms
brclr keyboard,$08,getcF
jmp scan_r0
getcF ldaa #$46 ; get the ASCII code of F
rts
delay10ms  movb  #$90,TSCR1 ; enable TCNT & fast flags clear
         movb  #$06,TSCR2 ; configure prescale factor to 64
         movb  #$01, TIOS  ; enable OC0
         ldd TCNT
         addd #3750 ; start an output compare operation
         std TC0 ; with 10 ms time delay
         wait_lp2 brclr TFLG1,$01, wait_lp2
         rts
Next…

- Interfacing with LCD