EE 319K
Introduction to Embedded Systems

Lecture 8: Fixed-Point Numbers, I/O Synchronization, LCD Interfacing

Announcements (1)

- Exam 2
  - Assembly programming
    - Up to and including pointers and indexed addressing
    - Arrays, strings, or FSMs
  - This Tuesday/Wednesday, Mar. 20/21
    - During regular lab section you are ***enrolled*** in
      - Not when you checkout!!
    - In lab (ACA 1.106)
      - Unique 16275: W (Mar. 21), 3-4pm, ACA 1.106
      - Unique 16280: W (Mar. 21), 4-5pm, ACA 1.106
      - Unique 16285: T (Mar. 20), 4-5pm, ACA 1.106
      - Unique 16290: T (Mar. 20), 5-6pm, ACA 1.106
  - Closed book, closed notes (scratch paper ok)
    - Familiarize yourself with instructions
      - Download, program & debug, submit on Blackboard

Announcements (2)

- Homework 7
  - Solve Exam 2 problems in C
- Lab 6
  - LCD driver
    - Local variables
    - Fixed-point
    - LCD interfacing

Agenda

- Recap
  - Local variables on the stack
  - Parameter passing using the stack
  - Stack frames
- Outline
  - Fixed-point numbers
  - I/O synchronization
  - LCD interface

Fixed Point Numbers

Why? (wish to represent non-integer values)

- Next lab measures distance from 0 to 3 cm
  E.g., 1.234 cm

When? (range is known, range is small)

- Range is 0 to 3cm

Resolution is 0.003 cm

How? (value = I*Δ)

- I (Variable Integer) is a 16-bit unsigned integer. It is stored and manipulated in memory.
- Δ (Fixed Constant) that represents the resolution. It is not stored but is usually written in comments ; implicit.

(What about negative numbers?)

Fixed Point Numbers: Decimal

Decimal

(Value = $I*10^m$)

I is a 16-bit unsigned integer

$\Delta = 10^m$ decimal fixed-point

For example with m=-3 (resolution of 0.001 or milli)
the value range is 0.000 to 65.535

What is π represented as, in Decimal Fixed Point?

π (3.14159…) = $I*10^{-3}$

=> I = Integral approximation of (3.14159…$*10^3$)

I = Integral approximation of (3141.59)

I = 3142

Decimal Fixed-point numbers are human-friendly

Fixed Point Numbers: Binary

Binary

(Value = $I*2^m$)

I is a 16-bit unsigned integer

$\Delta = 2^m$ binary fixed-point

For example with m=-8 (resolution of 1/256)

What is π represented as, in binary Fixed Point?

π (3.14159…)= $I*2^{-8}$

=> I = Integral approximation of (3.14159…$*2^8$)

I = Integral approximation of (804.2477)

=> I = 804

Binary fixed-point numbers are computer-friendly

Output

Output an integer:

Assume integer, n, is between 0 and 9999.

1. OutChar($30+n/1000)
   ;thousand's digit
2. n = n%1000
   OutChar($30+n/100)
   ;hundred's digit
3. n = n%100
   OutChar($30+n/10)
   ;ten's digit
4. OutChar ($30+n%10)
   ;one's digit

Output a fixed-point decimal number:

Assume the integer part of the fixed point number, n, is between 0 and 9999, and resolution is 0.001.

1. OutChar($30+n/1000)
   ;thousand's digit
2. n = n%1000
   OutChar($2E)
   ;decimal point
   OutChar($30+n/100)
   ;hundred's digit
3. n = n%100
   OutChar($30+n/10)
   ;ten's digit
4. OutChar ($30+n%10)
   ;one's digit

I/O Synchronization

Blind Cycle Counting Synchronization

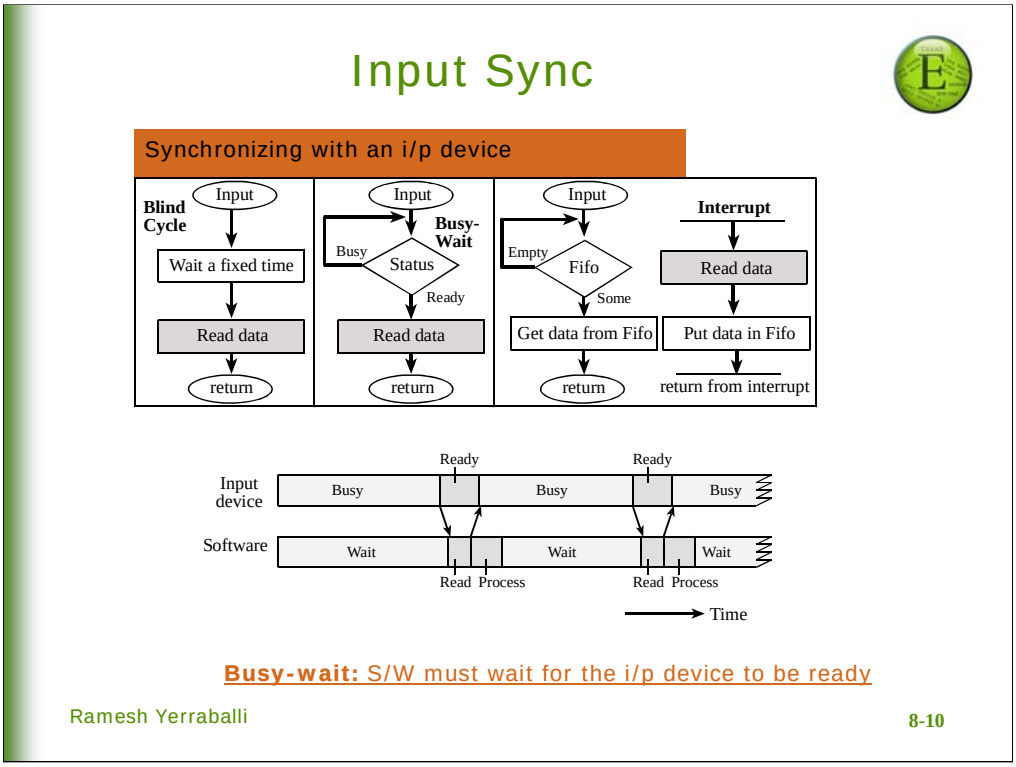
Blind cycle counting is appropriate when the I/O delay is fixed and known. This type of synchronization is blind because it provides no feedback from the I/O back to the computer.

Gadfly or Busy Waiting Synchronization

Check busy/ready flag over and over until it is ready

Interrupt Synchronization

Request interrupt when busy/ready flag is ready



In Interrupt-based synchronization for input, the input sub-routine repeatedly checks the FIFO to see if there is FIFO. Asynchronously, when inputs arrive (indicated by an Interrupt) the corresponding ISR gets executed the input data are read and put into the FIFO (possibly overwriting unconsumed previous data if the FIFO is full)

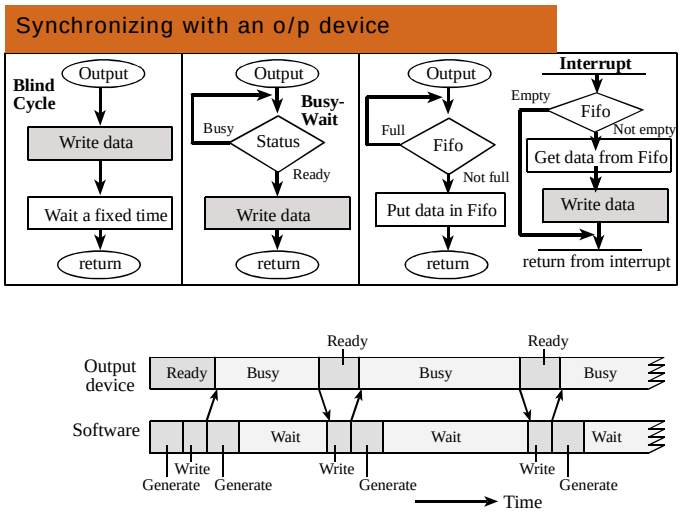
The timeline shows how when the device is busy the s/w has to wait. Once the device is "ready" it sets the "ready flag" and the s/w can read the available data. The s/w reading usually tends to be faster than the input device can create new input - We call this situation I/O-bound meaning the b/w is limited by the speed of the I/O h/w.

If on the other hand the device is faster than the s/w the waiting time would be zero because the s/w will always have data ready from the device. Such a situation is called CPU-bound meaning the b/w is limited by the speed of the s/w executing.

Busy-wait methods are classified as un-buffered.

Interrupt-based methods are classified as buffered.

Output Sync



Busy-wait: S/W must wait for the o/p device finish the previous operation

Ramesh Yerraballi

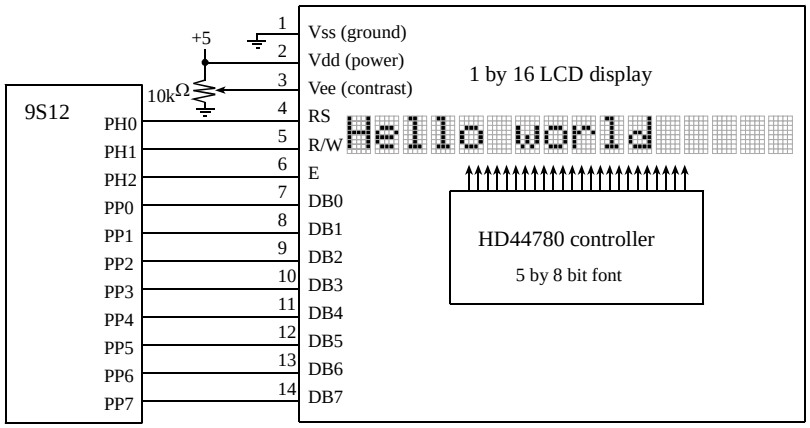
8-11

In Interrupt-based synchronization for output, the output sub-routine repeatedly checks the FIFO to see if the it is already full and when there is sapce it writes the output to the FIFO. Asynchronously, when output device is ready for output it indicates this by an Interrupt. The corresponding ISR gets execute. The ISR checks to make sure there is data to output. If there is output data then it is retrieved from the FIFO and written to the output device. The ISR returns to where the processor was when the interrupt occurred.

The timeline shows how when the s/w wants to output to the device it checks if the device is ready and when ready it writes the output and goes back to gnerating new output. The device processes the written data and while it is doing this sets the busy flag so no new inputs are accepted.

LCD Interface

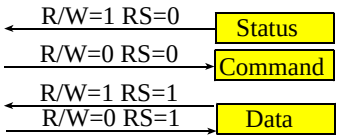
Parallel port LCD interface with the HD44780 Controller



Access Cycles

There are four types of access cycles to the HD44780 depending on RS and R/W(Control signals)

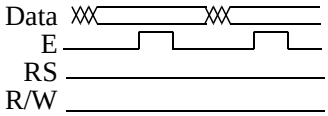
RS	R/W	Cycle
0	0	Write to Instruction Register
0	1	Read Busy Flag (bit 7)
1	0	Write data from µP to the HD44780
1	1	Read data from HD44780 to the µP



Programming the LCD interface

4-bit protocol write command (outCsr)

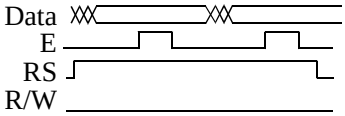
1. E=0, RS=0
2. 4-bit DB7,DB6,DB5,DB4 = most sig nibble of command
3. E=1
4. E=0 (latch 4-bits into LCD)
5. 4-bit DB7,DB6,DB5,DB4 = least sig nibble of command
6. E=1
7. E=0 (latch 4-bits into LCD)
8. blind cycle 90 us wait



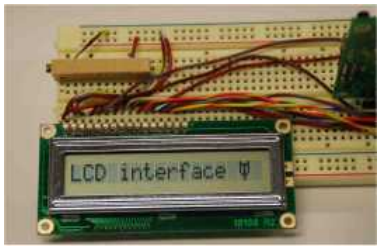
Programming the LCD interface

4-bit protocol write ASCII data (LCD_OutChar)

1. E=0, RS=1
2. 4-bit DB7,DB6,DB5,DB4 = most significant nibble of data
3. E=1
4. E=0 (latch 4-bits into LCD)
5. 4-bit DB7,DB6,DB5,DB4 = least significant nibble of data
6. E=1
7. E=0 (latch 4-bits into LCD)
8. blind cycle 90 us wait



LCD Programming



Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear display	0	0	0	0	0	0	0	0	0	1
Return home	0	0	0	0	0	0	0	0	1	—
Entry mode set	0	0	0	0	0	0	0	1	I/D	S
Display on/off control	0	0	0	0	0	0	1	D	C	B
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—
Function set	0	0	0	0	1	DL	N	F	—	—
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD

I/D = 1: Increment
I/D = 0: Decrement
S = 1: Accompanies display shift
S/C = 1: Display shift
S/C = 0: Cursor move
R/L = 1: Shift to the right
R/L = 0: Shift to the left
DL = 1: 8 bits, DL = 0: 4 bits
N = 1: 2 lines, N = 0: 1 line
F = 1: 5 × 10 dots, F = 0: 5 × 8 dots
BF = 1: Internally operating
BF = 0: Instructions acceptable

4-bit Interface

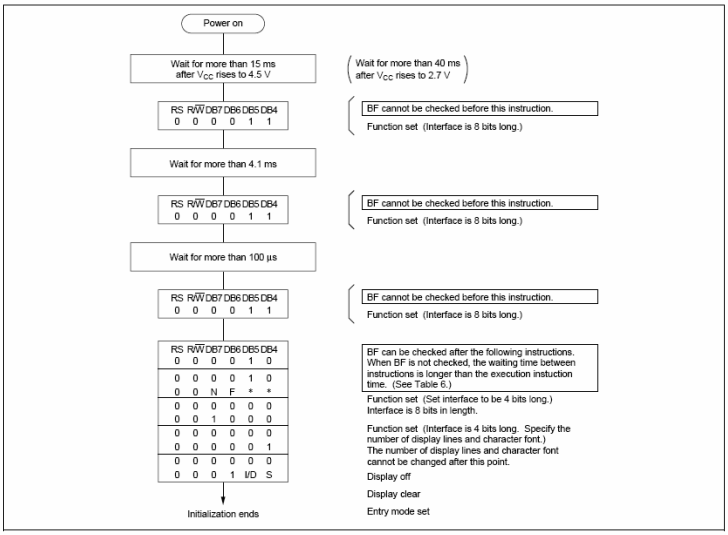


Figure 26 4-Bit Interface

Operations

Clear display

;1) outCsr($01) causes Clear

;2) blind cycle 1.64ms wait

;3) outCsr($02) sends the Cursor to home

;4) blind cycle 1.64ms wait

Move cursor to

;1) outCsr(DDaddr+$80)

first row (left-most 8) **DDaddr** is 0 to 7

second row (right-most 8) **DDaddr** is $40 to $47

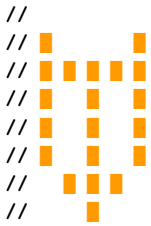
Fonts

Define up to 8 new fonts

```
// font is the font number 0 to 7
// n0-n7 is the 5 by 8 pixel map
      CGaddr=((font&0x07)<<3)+0x40;
  outCsr(CGaddr); // set CG RAM address
  LCD_OutChar(n0);
  LCD_OutChar(n1);
  LCD_OutChar(n2);
  LCD_OutChar(n3);
  LCD_OutChar(n4);
  LCD_OutChar(n5);
  LCD_OutChar(n6);
  LCD_OutChar(n7);
  outCsr(0x80); // revert back to DD RAM
```

Fonts Example

Say we want to store the following graphic (pixel configuration) in font number 5



```
//          n0 = $00
//          n1 = $11
//          n2 = $1F
//          n3 = $15
//          n4 = $15
//          n5 = $15
//          n6 = $0E
//          n7 = $04
```

```
// set font=5 to this graphic
// CGaddr=((5&0x07)<<3)+0x40 = $68
   outCsr(0x68); // set CG RAM address
   LCD_OutChar(0x00);
   LCD_OutChar(0x11);
   LCD_OutChar(0x1F);
   LCD_OutChar(0x15);
   LCD_OutChar(0x15);
   LCD_OutChar(0x15);
   LCD_OutChar(0x0E);
   LCD_OutChar(0x04);
   outCsr(0x80); // revert back to DD
   RAM

// To output this graphic
   LCD_OutChar(5);
```

LCD_OutFix

```
0) save any registers that will be destroyed by pushing on the stack
1) allocate local variables letter and num on the stack
2) initialize num to input parameter, which is the integer part
3) if number is less or equal to 999, go the step 6
4) output the string "*.** " calling LCD_OutString
5) go to step 19
6) perform the division num/100, putting the quotient in letter,
    and the remainder in num
7) convert the ones digit to ASCII, letter = letter+$30
8) output letter to the LCD by calling LCD_OutChar
9) output '.' to the LCD by calling LCD_OutChar
10)perform the division num/10, putting the quotient in letter,
    and the remainder in num
11)convert the tenths digit to ASCII, letter = letter+$30
12)output letter to the LCD by calling LCD_OutChar
13)convert the hundredths digit to ASCII, letter = num +$30
14)output letter to the LCD by calling LCD_OutChar
15)output ' ' to the LCD by calling LCD_OutChar
16)deallocate variables
17)restore the registers by pulling off the stack
```