Lecture 36

April 25, 2012

Review for Exam 3

Linking Assembly Subroutine with a C Program

A/D Converter

- Power-up A/D converter (ATD1CTL2)
- Write 0x05 to ATD1CTL4 to set at fastest conversion speed and 10-bit conversions
- Write 0x85 to ATD1CTL4 to set at fastest conversion speed and 8-bit conversions
- Select number of conversions in a sequence (ATD1CTL3)
- Select type of conversion sequence and the analog channels sampled (ATD1CTL5)
  - Right/left justified
  - signed/unsigned
  - Continuous Scan vs. Single Scan
  - Multichannel vs. Single Channel conversions
- How to tell when conversion is complete - ATD1STAT0 register
- How to read results of A/D conversions - ATD1DR$[7-0]$H (8-bit left-justified conversions)
- How to read results of A/D conversions - ATD1DR$[7-0]$L (8-bit right-justified conversions)
- How to read results of A/D conversions - ATD1DR$[15-6]$ (10-bit left-justified conversions)
- How to read results of A/D conversions - ATD1DR$[9-0]$ (10-bit right-justified conversions)
  - Be able to convert from digital number to voltage, and from voltage to digital number (need to know V_{RH} and V_{RL}).
- How long does it take to make a conversion?

Serial Communications and the IIC Bus

- Pins used – SDA and SCl
- Difference of use in Master and Slave mode
- IIC serial format for writing to slave
  - Start condition, 7-bit slave address, R/$\overline{\text{W}}$, wait for acknowledge
  - Send eight data bits, wait for ACK, repeat, send stop condition
- IIC serial format for reading from slave
  - Start condition, 7-bit slave address, R/$\overline{\text{W}}$, wait for acknowledge
  - Receive eight data bits, send ACK, repeat, after receiving last byte, send NACK instead of ACK, send stop condition
- IIC IBAD (Bus Address) register
  - Set address when used as slave
  - To use as master, write something like 0x01 (any address not assigned to a slave)
- IIC IBFD (Bus Frequency Divide) Register
  - Set clock speed to match slave
- IIC IBCR (Bus Control Register) Register
  - IBEN — Enable IIC bus
  - IBIE — Enable interrupts
  - MS/$\overline{\text{SL}}$ Switch to master mode
  - TX/$\overline{\text{RX}}$ Switch between transmit and receive
  - TKAK — Send an acknowledge
  - RSTA — Send an restart (didn't discuss)
  - IBSWAI — Specify if IIC clock should operate in WAIT mode (didn't discuss)
- IIC IBSR (Bus Status Register) Register
  - TCF — Transmit Complete Flag
  - IAAS — Did not discuss
  - IBB — Did not discuss
  - IBAL — Did not discuss
  - SRW — Did not discuss
  - IBIF — Interrupt flag. Clear by writing a 1 to this bit.
  - RXAK — Did not discuss
- IIC IBDR (Bus Data Register) Register
  - Write data to this register to send to slave
  - Read data from this register to receive from slave

Interfacing

- Getting into expanded mode — MODA, MODB, MDOC pins or MODE Register
- PEAR Register — enable ECLK, LSTRB, R/W on external pins
- Ports A and B in expanded mode
  - Port A – AD 15-8 (Port A is for data for high byte, even addresses)
  - Port B – AD 7-0 (Port B is for data for low byte, odd addresses)
- E clock
  - Address on AD 15-0 when E low, Data on AD 15-0 when E high
  - Need to latch address on rising edge of E clock
  - On write (output), external device latches data on signal initiated by falling edge of E
  - On read (input), HCS12 latches data on falling edge of E
  - E-clock stretch - MISC register
- R/W Line
- LSTRB line
- Single-byte and two-byte accesses
  - 16-bit access of even address – A0 low, LSTRB low – accesses even and odd bytes
  - 8-bit access of even address – A0 low, LSTRB high – accesses even byte only
  - 8-bit access of odd address – A0 high, LSTRB low – accesses odd byte only
  - A0 high and LSTRB high never occurs on external bus.
- Address Decoding – interfacing using MSI chips
- Timing – Be sure to meet setup and hold times of device receiving data
  - For a write, meet setup and hold of external device
  - For a read, meet setup and hold of HC12
- Timing — Be sure to meet address access time (length of time address needs to be on bus before external device is ready)

Linking Assembly Subroutine with a C Program

- To link an assembly subroutine to a C program, you have to understand how parameters are passed.
- For the CodeWarrior C compiler, one parameter is passed in the registers. The other parameters are passed on the stack.
  - The left-most parameter is pushed onto the stack first, then the next-to-left, etc.
  - The right-most parameter is passed in the registers
    - An 8-bit parameter is passed in the B regisiter
    - A 16-bit parameter is passed in the D register
    - A 32-bit parameter is passed in the {X:D} register combination.
- A value returned from the assembly language program is returned in the registers:
  - An 8-bit parameter is returned in the B regisiter
  - A 16-bit parameter is returned in the D register
  - A 32-bit parameter is returned in the {X:D} register combination.
- In the assembly language program, declare things the C program has to know about as `XDEF`:

```
XDEF     foo
```

- In the C program, declare things in the assembly program as you would an other functions:

```
int foo(int x);
```

- In the assembly language program, use the stack to store local variables
  - Need to keep close track of stack frame

Consider an assembly-language function `fuzzy` which uses two 8-bit arguments `arg1` and `arg2`, and returns an 8-bit argument `result`.

- Declare the function in the C program as

```
char fuzzy(char arg1, char arg2);
```

- Here is how the function may be called in the C program:

```
char x,y,result;

result = fuzzy(x, y);
```

- When the program is compiled, the value of the variable `x` is pushed onto the stack, the value of the variable `y` is loaded into the B register, and the function is called with a `JSR` instruction:

```
       15:     result = fuzzy(x,y); /* call the assembly function */
000b f60000        [3]     LDAB  x
000e 37            [2]     PSHB
000f f60000        [3]     LDAB  y
0012 160000        [4]     JSR   fuzzy
```

- In the assembly language function, you may need to use some local variables, which need to be allocated on the stack. If the `fuzzy` function needs two local 8-bit variables `var1` and `var2`, you will need to allocate two bytes on the stack for them. Here's what the start of the assembly language program will look like:

```
fuzzy:
      leas      -2,sp        ; Room on stack for var1 and var2

; Stack frame after leas -2,sp
;
;     SP      -> var1
;     SP + 1  -> var2
;     SP + 2  -> Return address high
;     SP + 3  -> Return address low
;     SP + 4  -> 1st parameter of function (arg1)
;
;     2nd paramter (arg2) passed in B register
```

- In the assembly language program, you access `arg1`, `var1` and `var2` with indexed addressing mode:

```
         stab   1,SP     ; Save arg2 into var2
         ldaa   4,SP     ; Put arg1 into ACCA
         staa   0,SP     ; Save arg1 into var1
```

- When you return from the assembly language function, put the value you want to return into B, add two to the stack (to deallocate `var1` and `var2`), and return with an RTS. For example, it you want to return the value of the variable `var2`, you would do the following:

```
ldab   1,SP       ; Put var2 into B
leas   2,SP       ; Deallocate local variables
rts               ; Return to calling program
```

- Any global variables used by the program should be declared in a separate section:

```
; section for global variables
FUZZY_RAM: SECTION
; Locations for the fuzzy input membership values
I_E_PM:           ds.b     1
I_E_PS:           ds.b     1
```

- Any global constants used by the program should be declared in a separate section:

```
; section for global variables
FUZZY_RAM: SECTION
; Locations for the fuzzy input membership values
I_E_PM:           ds.b     1
I_E_PS:           ds.b     1

FUZZY_CONST: SECTION
; Fuzzy input membership function definitions for speed error
E_Pos_Medium:     dc.b    170, 255,   6,   0
E_Pos_Small:      dc.b    128, 208,   6,   6
```

- The assembly language code should be put in its own section:

```
; code section
MyCode:     SECTION
; this assembly routine is called by the C/C++ application
fuzzy:
            leas      -2,sp        ; Room on stack for ERROR and d_ERROR
```

C program which calls fuzzy logic assembly function

```
#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
#include <stdio.h>
#include <termio.h>

int fuzzy(unsigned char e, unsigned char de);
unsigned char ERROR[]  =  {128,128,128,100,150};
unsigned char d_ERROR[] = {128,100,150,128,128};

void main (void)
{
    char dPWM;
    int i;

/* Set up SCI for using printf() */
    SCI0BDH = 0x00;           /* 9600 Baud */
    SCI0BDL = 0x9C;
    SCI0CR1 = 0x00;
    SCI0CR2 = 0x0C;           /* Enable transmit, receive */

    for (i=0;i<5;i++) {
        dPWM = fuzzy(ERROR[i],d_ERROR[i]);
        (void) printf("ERROR = %3d, d_ERROR = %3d, ", ERROR[i],d_ERROR[i]);
        (void) printf("dPWM: %4d\r\n", dPWM);
    }
    asm(" swi");
}
```

The Assembly program

```
;*************************************************************
;* This stationery serves as the framework for a             *
;* user application. For a more comprehensive program that   *
;* demonstrates the more advanced functionality of this      *
;* processor, please see the demonstration applications      *
;* located in the examples subdirectory of the               *
;* Freescale CodeWarrior for the HC12 Program directory      *
;*************************************************************

; export symbols
            XDEF fuzzy
            ; we use export 'Entry' as symbol. This allows us to
            ; reference 'Entry' either in the linker .prm file
            ; or from C/C++ later on

; Include derivative-specific definitions
        INCLUDE 'derivative.inc'

; Offset values for input and output membership functions
E_PM              equ           0  ; Positive medium error
E_PS              equ           1  ; Positive small error
E_ZE              equ           2  ; Zero error
E_NS              equ           3  ; Negative small error
E_NM              equ           4  ; Negative medium error
dE_PM             equ           5  ; Positive medium differential error
dE_PS             equ           6  ; Positive small differential error
dE_ZE             equ           7  ; Zero differential error
dE_NS             equ           8  ; Negative small differential error
dE_NM             equ           9  ; Negative medium differential error
O_PM              equ          10  ; Positive medium output
O_PS              equ          11  ; Positive small
O_ZE              equ          12  ; Zero output
O_NS              equ          13  ; Negative small
O_NM              equ          14  ; Negative medium output
MARKER            equ         $FE  ; Rule separator
END_MARKER        equ         $FF  ; End of Rule marker

; variable/data section
FUZZY_RAM: SECTION
; Locations for the fuzzy input membership values for speed error
I_E_PM:           ds.b     1
I_E_PS:           ds.b     1
I_E_ZE:           ds.b     1
I_E_NS:           ds.b     1
I_E_NM:           ds.b     1
```

```
; Locations for the fuzzy input membership values for speed error diff
I_dE_PM:          ds.b     1
I_dE_PS:          ds.b     1
I_dE_ZE:          ds.b     1
I_dE_NS:          ds.b     1
I_dE_NM:          ds.b     1

; Output fuzzy membership values - initialize to zero
M_PM:             ds.b     1
M_PS:             ds.b     1
M_ZE:             ds.b     1
M_NS:             ds.b     1
M_NM:             ds.b     1


FUZZY_CONST: SECTION
; Fuzzy input membership function definitions for speed error
E_Pos_Medium:     dc.b    170, 255,   6,   0
E_Pos_Small:      dc.b    128, 208,   6,   6
E_Zero:           dc.b     88, 168,   6,   6
E_Neg_Small:      dc.b     48, 128,   6,   6
E_Neg_Medium:     dc.b      0,  80,   0,   6
; Fuzzy input membership function definitions for speed error
dE_Pos_Medium:    dc.b    170, 255,   6,   0
dE_Pos_Small:     dc.b    128, 208,   6,   6
dE_Zero:          dc.b     88, 168,   6,   6
dE_Neg_Small:     dc.b     48, 128,   6,   6
dE_Neg_Medium:    dc.b      0,  80,   0,   6

; Fuzzy output memership function definition
PM_Output:        dc.b    192
PS_Output:        dc.b    160
ZE_Output:        dc.b    128
NS_Output:        dc.b     96
NM_Output:        dc.b     64
; Rule Definitions
Rule_Start:       dc.b      E_PM,dE_PM,MARKER,O_NM,MARKER
                  dc.b      E_PM,dE_PS,MARKER,O_NM,MARKER
                  dc.b      E_PM,dE_ZE,MARKER,O_NM,MARKER
                  dc.b      E_PM,dE_NS,MARKER,O_NS,MARKER
                  dc.b      E_PM,dE_NM,MARKER,O_ZE,MARKER

                  dc.b      E_PS,dE_PM,MARKER,O_NM,MARKER
                  dc.b      E_PS,dE_PS,MARKER,O_NM,MARKER
                  dc.b      E_PS,dE_ZE,MARKER,O_NS,MARKER
                  dc.b      E_PS,dE_NS,MARKER,O_ZE,MARKER
                  dc.b      E_PS,dE_NM,MARKER,O_PS,MARKER
```

```
                dc.b    E_ZE,dE_PM,MARKER,O_NM,MARKER
                dc.b    E_ZE,dE_PS,MARKER,O_NS,MARKER
                dc.b    E_ZE,dE_ZE,MARKER,O_ZE,MARKER
                dc.b    E_ZE,dE_NS,MARKER,O_PS,MARKER
                dc.b    E_ZE,dE_NM,MARKER,O_PM,MARKER

                dc.b    E_NS,dE_PM,MARKER,O_NS,MARKER
                dc.b    E_NS,dE_PS,MARKER,O_ZE,MARKER
                dc.b    E_NS,dE_ZE,MARKER,O_PS,MARKER
                dc.b    E_NS,dE_NS,MARKER,O_PM,MARKER
                dc.b    E_NS,dE_NM,MARKER,O_PM,MARKER

                dc.b    E_NM,dE_PM,MARKER,O_ZE,MARKER
                dc.b    E_NM,dE_PS,MARKER,O_PS,MARKER
                dc.b    E_NM,dE_ZE,MARKER,O_PM,MARKER
                dc.b    E_NM,dE_NS,MARKER,O_PM,MARKER
                dc.b    E_NM,dE_NM,MARKER,O_PM,END_MARKER

; code section
MyCode:     SECTION
; this assembly routine is called by the C/C++ application

; Stack frame after leas -2,sp
;
;               SP      -> ERROR
;               SP + 1  -> d_ERROR
;               SP + 2  -> Return address high
;               SP + 3  -> Return address low
;               SP + 4  -> 1st parameter of function (d_ERROR)
;
;               2nd paramter (ERROR) passed in B register
fuzzy:
                leas    -2,sp       ; Room on stack for ERROR and d_ERROR
                stab    1,sp        ; d_ERROR passed in B register
                ldab    4,sp        ; ERROR passed on stack
                stab    0,sp        ; Save in space reserved on stack

; Fuzzification
                LDX     #E_Pos_Medium ; Start of Input Mem func
                LDY     #I_E_PM     ; Start of Fuzzy Mem values
                LDAA    0,SP        ; Get ERROR value
                LDAB    #5          ; Number of iterations
Loop_E:         MEM                 ; Assign mem value
                DBNE    B,Loop_E    ; Do all five iterations
                LDAA    1,SP        ; Get d_ERROR value
                LDAB    #5          ; Number of iterations
Loop_dE:        MEM                 ; Assign mem value
```

```
                DBNE    B,Loop_dE     ; Do all five iterations

; Process rules
                LDX     #M_PM         ; Clear output membership values
                LDAB    #5
Loopc:          CLR     1,X+
                DBNE    B,Loopc

                LDX     #Rule_Start   ; Address of rule list -> X
                LDY     #I_E_PM       ; Address of input membership list -> Y
                LDAA    #$FF          ; FF -> A, clear V bit of CCR
                REV                   ; Rule evaluation

; Defuzzification
                LDX     #PM_Output    ; Address of output functions -> X
                LDY     #M_PM         ; Address of output membership values -> Y
                LDAB    #5            ; Number of iterations
                WAV                   ; Defuzzify
                EDIV                  ; Divide
                TFR     Y,D           ; Quotient to D; B now from 0 to 255
                SUBB    #128          ; Subtract offset from d_PWM
                                      ; dPWM returned in B; already there
                leas    2,sp          ; Return stack frame to entry value

                RTS
```

Output of Program

```
load
**************
>
>g 2029
hello, world
ERROR = 128, d_ERROR = 128, dPWM:    0
ERROR = 128, d_ERROR = 100, dPWM:   22
ERROR = 128, d_ERROR = 150, dPWM:  -18
ERROR = 100, d_ERROR = 128, dPWM:   22
ERROR = 150, d_ERROR = 128, dPWM:  -18
User Bkpt Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2B9E  10FD  0005  0001     00:0C        1001 0100
xx:2B9E  1B83          LEAS  3,SP

>
```