

CpE 390: Microprocessor Systems

Lecture 12

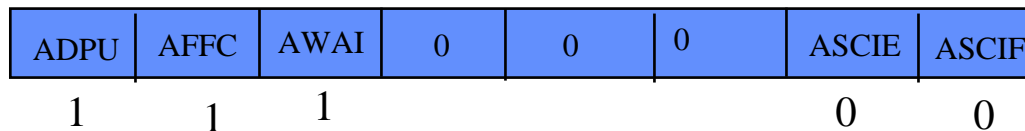
68HC12 Analog to Digital Converter (2)

Example 10.4 Write a subroutine to initialize the ATD converter for the 912BC32 and start the conversion with the following parameters:

- Nonscan mode
- Select channel 7
- enable ATD fast flag clear
- Stop ATD in wait mode
- Disable interrupt
- Finish current conversion then freeze when BDM becomes active
- 10-bit operation and 2 A/D clock periods of sample time
- Choose 2 MHz as the conversion frequency for 8 MHz E clock

Solution: The setting of ATD control register 2 to 5 are as follows:

- Enable ATD (set bit 7 to 1)
- Enable fast flag clear (set bit 6 to 1)
- Stop ATD when in wait mode (set bit 5 to 1)
- Disable ATD interrupt (set bit 1 to 0)
- Clear all other bits
- Write the value \$E0 to the ATDCTL2



ATDCTL3

- Complete the current instruction then freeze when BDM becomes active
- Write the value \$02 to this register

0	0	0	0	0	0	FRZ1	FRZ0
						1	0

ATDCTL4

- Select 10-bit operation (set bit 7 to 1)
- Set sample time to two ATD clock period (set bits 6 & 5 to 00)
- Set clock prescale factor to 4 to select 2 MHz as the ATD clock frequency. Set bits 4 to 0 to 00001.
- Write the value \$81 to this register.

S10BM	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
1	0	0	0	0	0	0	1

ATDCTL5

- Select four conversions as the conversion sequence length (set bit 6 to 0)
- Select nonscan mode (set bit 5 to 0)
- Select single channel mode (set bit 4 to 0)
- Select channel 7 (set bits 3..0 to 0111)
- Write the value \$07 to this register

0	S8CM	SCAN	MULT	CD	CC	CB	CA
	0	0	0	0	1	1	1

The following subroutine will perform the desired initialization:

```
#include "d:\miniide\hc12.inc"
ATD_init movb  #$E0,ATDCTL2
        ldaa   #10
; the next two instructions create 5 us delay
wait     deca           ; 1 E clock cycle execution time
        bne    wait     ; 3/1 E clock cycles execution time
        movb   #$02,ATDCTL3
        movb   #$81,ATDCTL4
        rts
```

Notes: this routine does not write into the ATDCTL5 register because that will start the conversion. We should write into the ATDCTL5 register only when we want to perform the conversion

Example 10.5 Write a subroutine to initialize the ATD0 module of the 912DG128 with the following parameters:

- Nonscan mode
- Select channels 0 to 7
- Enable fast flag clear feature
- Stop ATD in wait mode
- Disable interrupt
- Finish current conversion then freeze when BDM becomes active
- Select 10-bit operation and set sample time to 4 ATD clock periods
- Set prescale factor to 4 for 8 MHz E clock

Solution: Settings of ATD control registers as follows:

ATD0CTL2

- Enable ATD (set bit 7 to 1)
- Select fast flag clear all (set bit 6 to 1)
- Stop ATD when in wait mode (set bit 5 to 1)
- Disable ATD interrupt (set bit 1 to 0)
- Clear all other bits
- Write the value \$E0 into this register

ADPU	AFFC	AWAI	0	0	0	ASCIE	ASCIF
1	1	1				0	0

ATD0CTL3

- Complete the current instruction when the BDM becomes active
- Write the value \$02 into this register

0	0	0	0	0	0	FRZ1	FRZ0
						1	0

ATD0CTL4

- Select 10-bit operation (set bit 7 to 1)
- Set sample time to 4 ATD clock periods (set bits 6..5 to 01)
- Set prescale factor to 4 to select 2 MHz as the ATD clock source frequency.
- Write the value \$A1 into this register.

S10BM	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
1	0	1	0	0	0	0	1

ATD0CTL5

- Set conversion sequence length to 8 (set bit 6 to 1)
- Select nonscan mode (set bit 5 to 0)
- Select multiple channel mode (set bit 4 to 1)
- Select channels 7 to 0 (set bits 3..0 to 0000)
- Write the value \$50 to this control register

0	S8CM	SCAN	MULT	CD	CC	CB	CA
	1	0	1	0	0	0	0

The following subroutine will perform the desired ATD configuration:

```
#include    "d:\miniide\hc12.inc"
ATD0_init  movb    #$E0,ATDCTL2
           ldaa    #10
; the next two instructions create 5 us delay
wait0      deca                    ; 1 E clock cycle execution time
           bne     wait            ; 3/1 E clock cycles execution time
           movb    #$02,ATDCTL3
           movb    #$A1,ATDCTL4
           rts
```

The C language version of the subroutine is as follows:

```
void ATD0_init (void)
{
    int i;
    ATD0CTL2 = 0xE0;
    for (i = 0; i < 40; i++) {
        asm ("nop");
    }
    ATD0CTL3 = 0x02;
    ATD0CTL4 = 0xA1;
}
```

Example 10.6 Write a program to perform A/D conversion on the analog signal connected to the AN7 pin. Collect 20 A/D conversion results and store them at memory location starting from \$800. Use the same configuration as in Example 10.4.

Solution:

- Write to the ATDCTL5 register five times and collect four samples each time.
- Wait until the SCF flag of the ATDSTAT register is set to 1 and then collect the samples.

```
#include "d:\miniide\hc12.inc"
```

```

        org      $1000
        ldx      #$800          ; use index register X as a pointer to the buffer
        jsr      ATD_init      ; initialize the ATD converter
        ldy      #5
loop5    movb     #$07,ATDCTL5   ; start an A/D conversion sequence
        brclr    ATDSTAT0,$80,*
        movw     ADR0H,2,x+     ; collect and save the conversion result (left- justified)
        movw     ADR1H,2,x+     ; post-increment the pointer by 2
        movw     ADR2H,2,x+     ;      “
        movw     ADR3H,2,x+     ;      “
        dbne     y,loop5
        swi
        end
```


The C language version of the program is as follows:

```
#include <hc12.h>
void ATD_init (void);
int buf[20];
void main (void)
{
    int i;

    ATD_init();
    for (i = 0; i < 5; i++) {
        ATDCTL5 = 0x07;    /* start an A/D conversion */
        while (!(ATDSTAT0 & 0x80)); /* wait for the A/D conversion to complete */
        /* store the result right-justified */
        buf[4*i + 0] = ADR0H * 4 + ADR0L/64;
        buf[4*i + 1] = ADR1H * 4 + ADR1L/64;
        buf[4*i + 2] = ADR2H * 4 + ADR2L/64;
        buf[4*i + 3] = ADR3H * 4 + ADR3L/64;
    }
    asm ("swi");
}
```

The LM34 Precision Fahrenheit Temperature Sensors

- Voltage output is linearly proportional to the ambient temperature.
- No external calibration required.
- Very linear over the temperature range
- Accuracy is $\pm 0.5^\circ\text{F}$ at room temperature and 1.5°F over a full -50° to $+300^\circ\text{F}$ range.
- Draws about $75\ \mu\text{A}$ current from the power supply.

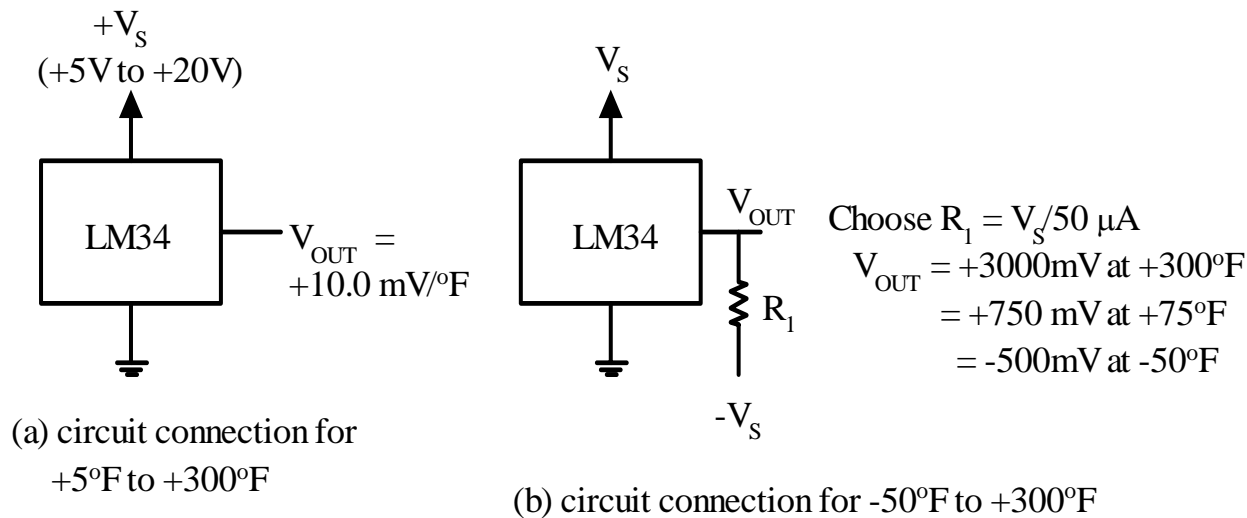


Figure 10.13 Circuit connection for the LM34

Example 10.7 Use the circuit in Fig 10.13 as a building block in a system to measure the room temperature. Display the result in three integral and one fractional digits using seven segment displays driven by the MAX7221. Measure and display the temperature in the range of -44°F to 212°F .

Solution:

- The LM34 voltage output will be from -440mV to 2120mV from -44°F to 212°F .
- Use the circuit shown in Figure 10.14 to shift and scale the sensor output to $0\sim 5\text{V}$.
- The room temperature is equal to A/D result divided by 4 and then subtracted by 44 (since 10 bit resolution of ATD).
- Temperature will be displayed in six seven-segment displays, which will be driven by the MAX7221. The circuit is shown in Figure 10.15.

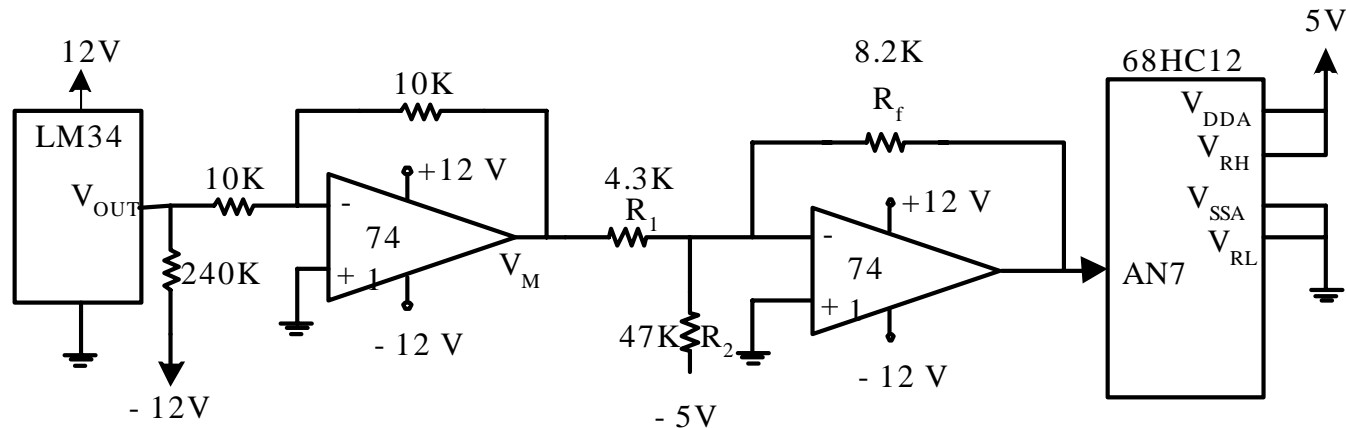


Figure 10.14 Circuit connection between the LM34 and the 68HC12

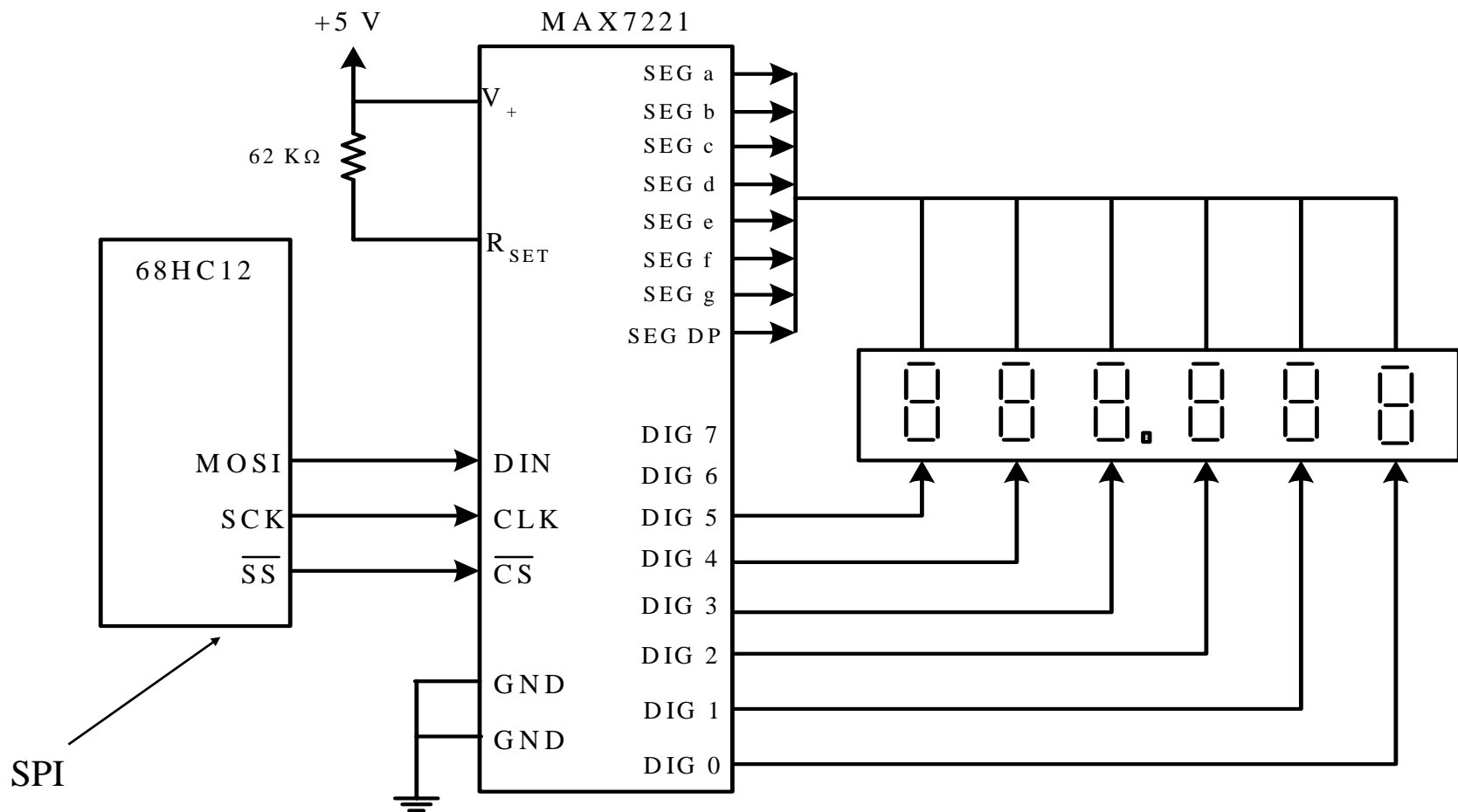


Figure 10.15 Digital thermometer display circuit

Temperature Display Consideration

- Temperature will be measured once every second and so will the update of the display.
- The characters ° and F will be displayed in digits 1 and 0, respectively.
- The minus sign will be displayed when the temperature is below zero. The minus sign will be shown in digit 5.
- When the temperature is between 0°F and –10°F, the negative sign will be displayed in digit 5 and digit 4 will be blanked.
- Leading zeros will be blanked for positive temperature.

SPI Configuration

- Same as Example 9.11.

MAX7221 Configuration

Scan limit register

- Need to display digits 5 to 0.
- Write the value \$0B05 to the MAX7221.

Decode mode register

- The leading four digits should be displayed using decode mode whereas ° and F should be displayed using non decode mode.
- The value to be sent to the MAX7221 is \$09FC.

Intensity register

- Temperature must be displayed in maximum intensity.
- The value to be sent to the MAX7221 is \$0A0F.

Shutdown register

- The chip should operate in normal mode.
- The value to be sent to the MAX7221 is \$0C01.

Display test register

- The chip should operate in normal mode.
- The value to be sent to the MAX7221 is \$0F00.

Digit 0 register

- Letter F is to be displayed using no-decode mode.
- The value to be sent to this register is \$0147.

Digit 1 register

- The degree character is to be displayed using non-decode mode.
- The value to be sent to this register is \$0263.

Digit 2 register

- Fractional digit is to be displayed in this digit using decode mode.
- The fraction digit can be from 0 to 9.
- The value to be sent to the MAX7221 is \$030x (x = 0..9).

Digit 3 register

- This digit displays the one's digit using the decode mode.
- Decimal point is displayed in this digit.
- The value of this digit can be from 0 to 9.
- The value to be sent to the MAX7221 is \$040x (x = 0..9).

Digit 4 register

- This digit displays the ten's digit using decode mode.
- The value of this digit can be from 0 to 9.
- The value to be sent to the MAX7221 is \$050x (x = 0..9).
- If the temperature is between 0° and –10°F, then this digit should be blanked. Sent the value \$050F to blank this digit.

Digit 5 register

- This digit may display hundred's digit or minus sign, or be blanked.
- When displaying hundred's digit, send the value \$060x, where, $x = 1$ or 2 .
- When displaying minus sign, send the value \$060A.
- When being blanked, send the value \$060F.

The configuration of the A/D converter is identical to that in Example 10.4.

Procedure

Step 1

Initialize the SPI module.

Step 2

Initialize the MAX7221 chip.

Step 3

Configure the ATD module.

Step 4

Start A/D conversion and read the conversion result.

Step 5

Divide the conversion result by 4 and then subtract the quotient by 44 to obtain the current temperature. Separate each temperature digit by performing repetitive division by 10.

Step 6

Transfer temperature data to the MAX7221 to update the display.

Step 7

Wait for one second and then go to step 4.

```

#include "d:\miniide\hc12.inc"

        org      $800
scan_limit  db      $0B,$05      ; data to set up scan limit register
decode_mode db      $09,$FC      ; data to set up decode mode register
intensity   db      $0A,$0F      ; data to set up intensity register
shutdown    db      $0C,$01      ; data to set up shut down register
display_test db      $0F,$00      ; data to set up display test register
digit0      db      $01,$47      ; data to update digit 0
digit1      db      $02,$63      ; data to update digit 1
digit2      rmw     1             ; data to update digit 2
digit3      rmw     1             ; data to update digit 3
digit4      rmw     1             ; data to update digit 4
digit5      rmw     1             ; data to update digit 5
f_is_zero   rmb     1             ; flag to indicate if fractional digit is 0

        org      $1000
        lds      #$8000          ; set up the stack pointer
        jsr      spi_init        ; configure SPI function
        jsr      m7221_init      ; configure MAX7221
        jsr      atd_init        ; configure ATD module
        clr      f_is_zero       ; fractional digit is not zero
forever     jsr      init_disp     ; initialize digits 3 to 5
            movb   #$07,ATDCTL5 ; start an A/D conversion sequence
            brclr  ATDSTAT0,$80,* ; wait until conversion is done

```

```

        ldd      ADR0H      ; read the conversion result
        ldx      #64        ; right-justify the conversion result
        idiv     ;          "
        xgdx     ;          "
        ldx      #4         ; convert to Fahrenheit temperature
        idiv     ;          "
; compute the value for displaying fractional digit to be sent to MAX7221
        cmpb     #0
        bne      check_1
        movw     #$0300,digit2 ; fractional digit is 0
        movb     #1,f_is_zero
        bra      ones_digit
check_1  cmpb     #1
        bne      check_2
        movw     #$0303,digit2 ; fractional digit is 3
        bra      ones_digit
check_2  cmpb     #2
        bne      is_3
        movw     #$0305,digit2 ; fractional digit is 5
        bra      ones_digit
is_3     movw     #$0308,digit2 ; fractional digit is 8
ones_digit xgdx
        subd     #44        ; shift by 44 degree

```

```

        bpl    above_zero    ; is temperature above zero?
        bmi    below_zero
        movb   #$0F,digit4+1 ; temperature is 0, so blank digit 4
        movb   #$0F,digit5+1 ; & blank digit 5
        jmp    update_T
below_zero movw   #$060A,digit5 ; set up the negative sign
        negb                    ; convert to positive temperature
        tst    f_is_zero
        bne    no_op          ; is fractional digit zero?
        incb                    ; add 1 to negative temperature
        ldaa   #10             ; take 10's complement of the
        suba   digit2+1        ; fractional digit
        staa   digit2+1        ;      "
no_op    clra                    ; clear A
        ldx    #10             ; separate one's digit
        idiv                    ;      "
        stab   digit3+1        ; set the digit3 value
        xgdx                    ; place ten's digit in B
        tstb                    ; is ten's digit zero?
        bne    not_zero
        ldab   #$0F
        stab   digit4+1        ; blank ten's digit
        jmp    update_T

```

```

not_zero    stab    digit4+1      ; fix ten's digit's value
            jmp     update_T
above_zero  clra
            cmpb    #9
            bhi     hi_than_9     ; is temperature equal or above ten degrees
            stab    digit3+1      ; between 0 and 9
            movb    #$0F,digit4+1 ; blank digit 4
            movb    #$0F,digit5+1 ; blank digit 5
            jmp     update_T
hi_than_9   cmpb    #99           ; is temperature between 10 and 100?
            bhi     three_dig
            ldx     #10
            idiv
            stab    digit3+1      ; set the one's digit value
            xgdx
            stab    digit4+1      ; set the ten's digit value
            movb    #$0F,digit5+1 ; blank the most significant digit
            jmp     update_T
three_dig   ldx     #10
            idiv
            stab    digit3+1      ; set the one's digit value
            xgdx
            ldx     #10

```

```

        idiv
        stab    digit4+1      ; set the ten's digit value
        xgdx
        stab    digit5+1      ; set the hundred's digit value
update_T  ldx     #digit2
        ldab    #4
        jsr     transfer      ; send data to MAX7221 to update temperature
        jsr     delay_1s
        jmp     forever
; *****
; The following routine initializes the SPI function
; *****
spi_init   movb   #$50,SP0CR1 ; enable SPI, shift data on rising edge, master mode
          movb   #$08,SP0CR2 ; enable internal pullup, normal drive
          movb   #$00,SP0BR   ; shift rate set to 4 MHz
          movb   #$CA,DDRS
          movb   #$07,PURDS   ; all port S pins have pullup
          rts
; *****
; The following routine initializes the MAX7221 chip including degree and letter F
; *****

```

```

m7221_init    ldx      #scan_limit
               ldab     #7
               jsr      transfer
               rts

; *****
; The following routine transfer data to the MAX7221 chip.
; *****
transfer       bclr     PORTS,$80
               ldaa     1,x+
               staa     SP0DR
               brclr    SP0SR,$80,*    ; wait for SPI transfer to complete
               ldaa     1,x+
               staa     SP0DR
               brclr    SP0SR,$80,*
               bset     PORTS,$80      ; transfer data to appropriate register
               dbne     b,transfer
               rts

```

```

; *****
;
; The following function initializes the ATD module.
; *****
atd_init    movb    #$E0,ATDCTL2
            ldaa    #10
; the next two instructions create 5 us delay
wait        deca
            bne     wait
            movb    #$02,ATDCTL3
            movb    #$81,ATDCTL4
            rts
; *****
; The following function initializes display data.
; *****
init_disp   movw    #$0300,digit2
            movw    #$0480,digit3
            movw    #$0500,digit4
            movw    #$0600,digit5
            rts

```



```

; *****
; The following function creates one-second time delay
; *****
delay_1s    pshx
            movb    #$90,TSCR    ; enable TCNT & fast flags clear
            movb    #$03,TMSK2  ; configure prescale factor to 8
            bset    TIOS,$01    ; enable OC0
            ldx     #20          ; prepare to perform 20 OC actions
            ldd     TCNT
again       addd    #50000       ; start an output compare operation
            std     TC0          ; with 50 ms time delay
            brclr   TFLG1,$01,*
            ldd     TC0
            dbne    x,again
            pulx
            rts
            end

```

```

#include <hc12.h>
void spi_init (void);
void m7221_init(void);
void atd_init (void);
void transfer (char *ptr, char n);
void delay_1s (void);  /* create one-second time delay */
void init_disp(void);  /* initialize display data */
/* max7221 configuration data for registers and digits 0 and 1 */
char max_conf [14] =
    {0x0B,0x05,0x09,0xFC,0x0A,0x0F,0x0C,0x01,0x0F,0x00,0x01,0x47,0x02,0x63};
char temp_dat [8]; /* four leading temperature digits data */
void main ()
{
    int  temp;
    char xx, is_zero;
    spi_init ();
    m7221_init();
    atd_init ();
    while (1) {
        init_disp();           /* set up common temperature digit data */
        ATDCTL5 = 0x07; /* start an A/D conversion */
        while (!(ATDSTAT0 & 0x80)); /* wait for A/D conversion to complete */
    }
}

```

```
temp = ADR0H * 4 + ADR0L/64; /* combine the A/D result register */
xx = temp % 4; /* get the fractional digit value */
temp = temp / 4; /* convert to temperature (not shifted by 44 degrees yet ) */
```

```
switch (xx) {
    case 0: temp_dat[1] = 0x00;          /* fractional digit is 0 */
        is_zero = 1;
        break;
    case 1: temp_dat[1] = 0x03;          /* fractional digit is 3 */
        is_zero = 0;
        break;
    case 2: temp_dat[1] = 0x05;          /* fractional digit is 5 */
        is_zero = 0;
        break;
    case 3: temp_dat[1] = 0x08;          /* fractional digit is 8 */
        is_zero = 0;
        break;
    default: temp_dat[1] = 0x00;
        break;
}
```

```

if (is_zero && temp < 35) {      /* temperature is -44 ~ -10 F */
    temp_dat[7] = 0x0A;          /* set up negative sign */

    temp = abs (temp - 44);
    temp_dat[3] = temp % 10; /* compute one's digit */
    temp_dat[5] = temp / 10; /* compute ten's digit */
}
else if (!is_zero && temp < 35) {
    temp_dat[7] = 0x0A;          /* set up negative sign */

    temp_dat[1] = 10 - temp_dat[1]; /* complement the fractional digit */
    temp = abs(temp - 43);
    temp_dat[3] = temp % 10;
    temp_dat[5] = temp / 10;
}
else if (is_zero && temp < 45) { /* temperature is between 0 and -9 */
    temp_dat[7] = 0x0A;          /* set negative sign */
    temp_dat[3] = temp - 44;
    temp_dat[5] = 0x0F;          /* blank leading zeros */
}

```

```

else if (!is_zero && temp < 45) { /* temperature is -9 ~ 0 F */
    temp_dat[1] = 10 - temp_dat[1]; /* complement the fractional digit */
    temp_dat[3] = abs(temp - 43); /* compute the one's digit */
    temp_dat[5] = 0x0F;          /* blank the ten's digit because it is zero */
    temp_dat[7] = 0x0A;          /* set up negative sign */
}
else if (temp < 54) {           /* temperature is 1~9 F */
    temp_dat[3] = temp - 44;     /* compute the one's digit */
    temp_dat[5] = 0x0F;          /* blank the leading zeros */
    temp_dat[7] = 0x0F;
}
else if (temp < 144) {          /* temperature is 10 ~ 99 F */
    temp_dat[3] = (temp - 44) % 10; /* compute ones digit */
    temp_dat[5] = (temp - 44) / 10; /* compute ten's digit */
    temp_dat[7] = 0x0F;          /* blank hundred's digit */
}
else {
    temp_dat[3] = (temp - 44) % 10; /* compute one's digit */
    temp = (temp - 44) / 10;
    temp_dat[5] = temp % 10; /* compute ten's digit */
    temp_dat[7] = temp / 10; /* compute hundred's digit */
};

```

```

        transfer (&temp_dat[0], 4);           /* update display data */
        delay_1s ();                          /* wait for one second */
    }
}
void spi_init(void)
{
    SP0CR1 = 0x50;
    SP0CR2 = 0x08;
    SP0BR = 0x00;
    DDRS = 0xCA;
    PURDS = 0x07;
}
void m7221_init (void)
{
    transfer (&max_conf[0],7);
}
/* transfer n bytes of data to MAX7221 */
void transfer (char *ptr, char n)
{
    char i;
    for (i = 0; i < n; i++) {
        PORTS &= 0x7F; /* set pin 7 to low */
    }
}

```

```

        SP0DR = *ptr++;
        while (!(SP0SR & 0x80));
        SP0DR = *ptr++;
        while (!(SP0SR & 0x80));
        PORTS |= 0x80; /* pull pin 7 to high */
    }
}
/* this function initialize the digit registers of the MAX7221 driver chip */
void init_disp(void)
{
    temp_dat[0] = 0x03;          /* fractional digit */
    temp_dat[1] = 0x00;
    temp_dat[2] = 0x04;          /* one's digit */
    temp_dat[3] = 0x80;
    temp_dat[4] = 0x05;          /* ten's digit */
    temp_dat[5] = 0x00;
    temp_dat[6] = 0x06;          /* hundred's digit */
    temp_dat[7] = 0x00;
}

```

```

void atd_init (void)
{
    int i;
    ATDCTL2 = 0xE0;
    for (i = 0; i < 40; i++)                /* delay for 5 us */
        asm ("nop");
    ATDCTL3 = 0x02;
    ATDCTL4 = 0x81;
}
/* this function uses OC0 function to create 1 second time delay */
void delay_1s (void)
{
    int i;
    TSCR = 0x90; /* enable TCNT & fast flag clear */
    TMSK2 = 0x03; /* set the clock prescale factor to 8 */
    TIOS = 0x01; /* select OC0 function */
    TC0 = TCNT + 50000u;
    i = 20;
    while (i) {
        while (!(TFLG1 & 0x01)); /* wait for 50 ms */
        TC0 = TC0 + 50000u;
        i--;
    }
}

```