ADC12 and DAC12 Modules

by Alex Milenkovich, milenkovic@computer.org

Objective: This tutorial discusses the use of MSP430's ADC12 and DAC12 peripheral devices.

Note: Required are all previous tutorials.

1. On-Chip Temperature Sensor

Let us consider a C application shown in Figure 1 that samples the on-chip temperature sensor, converts the sampled voltage from the sensor to temperature in degrees Celsius and Fahrenheit, and sends the temperature sensor through a RS232 link to the Hyper-terminal application. Note: The source code for this program can be downloaded from the course secure directory (file temp_uart.c).
Analyze the program and test it on the EasyWeb2 platform. Answer the following questions.

What does the program do?
How the basic block module is configured?
How the ADC12 module is configured?
How the USART0 module is configured?

```
//  Description: fMCLK = fACLK = 8MHz
//
//               MSP430F149
//             -----------------
//         /|\|              XIN|-
//          | |                 |  XTAL (8Mhz)
//          --|RST          XOUT|-
//            |                 |
//            |             P2.1|-->LED
//
//  @A. Milenkovic, milenkovic@computer.org
//
//*****************************************************************************

#include "msp430x14x.h"
#include "stdio.h"

unsigned char thr_char;  /* hold char from UART RX*/
unsigned char rx_flag;   /* receiver rx status flag */
char gm1[67] = "Hello! I am an MSP430. Would you like to know my temperature? (Y|N)";
char gm2[9] = "Bye, bye!";
char gm3[15] = "Type in Y or N!";

long int temp;
long int IntDegF;
long int IntDegC;

char NewKey[25];

// initialize basic clock module
void InitOsc(void);
// UART Initializaion
void UART_Initialize(void);
//send char function
void UART0_putchar(char c);
```

```
void main(void)
{
  int i = 0;

  WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer

  InitOsc();
  UART_Initialize();

  ADC12CTL0 = SHT0_8 + REFON + ADC12ON;
  ADC12CTL1 = SHP;                      // enable sample timer
  ADC12MCTL0 = 0x01A;
  ADC12IE = 0x001;

  rx_flag = 0;            // rx default state "empty"
  _EINT();                // enable global interrupts

  // send greeting messages
  for(i = 0; i < 67; i++) {
     thr_char = gm1[i];
     UART0_putchar(thr_char);
  }

Wait:
    while(!(rx_flag&0x01)); // wait until receive the character from HyperTerminal
    rx_flag = 0;            // clear rx_flag
    UART0_putchar(thr_char);
    UART0_putchar('\n');    // newline
    UART0_putchar('\r');    // carriage return
    // echo to HyperTerminal twice
    if ((thr_char != 'y') && (thr_char != 'n') && (thr_char !='Y') && (thr_char !='N')) {
      //UART0_putchar(thr_char);
      for(i = 0; i < 15; i++) {
        thr_char = gm3[i];
        UART0_putchar(thr_char);
      }
      UART0_putchar('\n');    // newline
      UART0_putchar('\r');    // carriage return
      goto Wait;
    }
    if ((thr_char == 'y') || (thr_char == 'Y')) {
       ADC12CTL0 |= ENC + ADC12SC;         // Sampling and conversion start
      _BIS_SR(CPUOFF + GIE);              // LPM0 with interrupts enabled

      //  oF = ((x/4096)*1500mV)-923mV)*1/1.97mV = x*761/4096 - 468
      //  IntDegF = (ADC12MEM0 - 2519)* 761/4096
      IntDegF = (temp - 2519) * 761;
      IntDegF = IntDegF / 4096;

      //  oC = ((x/4096)*1500mV)-986mV)*1/3.55mV = x*423/4096 - 278
      //  IntDegC = (ADC12MEM0 - 2692)* 423/4096
      IntDegC = (temp - 2692) * 423;
      IntDegC = IntDegC / 4096;
      sprintf(NewKey, "T(F)=%ld\tT(C)=%ld\n", IntDegF, IntDegC);
      for(i = 0; i < 25; i++) {
         thr_char = NewKey[i];
         UART0_putchar(thr_char);
      }
      UART0_putchar('\n');    // newline
      UART0_putchar('\r');    // carriage return
      goto Wait;
    }

    if ((thr_char == 'n') || (thr_char == 'N')) {
      for(i = 0; i < 9; i++) {
        thr_char = gm2[i];
        UART0_putchar(thr_char);
      }
      UART0_putchar('\n');    // newline
      UART0_putchar('\r');    // carriage return
```

```
   }
}

void InitOsc(void)
{
  int i;

  IFG2 = 0;
  IFG1 = 0;
  //using a 8MHz clock from LFXT1, and ACLK=LFXT1
  BCSCTL1 |= XTS;
  /*wait in loop until crystal stabalizes*/
  do
  {
    IFG1 &= ~OFIFG;
  }
  while (OFIFG & IFG1);

  for (i = 0xFF; i > 0; i--);
  //Reset osc. fault flag again
  IFG1 &= ~OFIFG;
}


void UART_Initialize(void)
{
//////////////////////////////////////////////////////////////
//   UART0 Initialization
//////////////////////////////////////////////////////////////

// UART0 - 38400 bps
// IMPORTANT NOTICE: the following configuration works only for 8MHZ clock rate
// AND 38400 bps baud rate
// You have to recalculate these values if the cpu clock rate or baud rate has changed
  U0MCTL = 146;
  U0BR0  = 208;
  U0BR1  = 0;

  P3SEL |= 0x30; // bits 4-5 are for special function UART0
  ME1 |= URXE0 + UTXE0; // UART module enable
  U0CTL = CHAR;     /* 8-bit characters */
  U0TCTL = SSEL0; /* clock source ACLK, 8MHZ */
  U0RCTL = 0;

  IE1 |= URXIE0;  // enable RX interrupts
}


void UART0_putchar(char c) {
  // wait for other character to transmit
  while (!(IFG1 & UTXIFG0));
  U0TXBUF = c;
}

#pragma vector=UART0RX_VECTOR
__interrupt void UART0_RX_interrupt(void)
{
  thr_char = U0RXBUF;
  rx_flag=0x01;          // signal main function receiving a char
}


#pragma vector=ADC_VECTOR
__interrupt void ADC12ISR (void)
{
    temp = ADC12MEM0;                   // Move results, IFG is cleared
    _BIC_SR_IRQ(CPUOFF);                // Clear CPUOFF bit from 0(SR)
}
```

Figure 1. C program for sampling and displaying temperature from the MSP430's temperature sensor.

2. Sinusoidal Wave Generator

Let us consider the following application. Your task is to develop a sinusoidal wave generator using the Softbaugh DRFG4618 board. The function you should implement is described as follows: y = 1.25*(1+sin(x)). The maximum voltage is consequently 2.5 V and the minimum voltage is 0 V. The frequency of the sine wave should be 10 Hz, that this one period should take 10 ms.

The signal is generated using the MSP430's DAC12 digital-to-analog converter. Periodically we will be sending a new digital value to the DAC12. These values are prepared in advance in a constant table. Let us first discuss how to create this lookup table. Our first step is to determine the number of samples we want to have and the number of bits to represent them. More samples, means a better quality of the generated sine wave. Let us assume that we want to have 256 samples of the function *y* in a lookup table, each sample with a value in the range of [0 ... 4095]. The sample with value 0 corresponds to 0 V, and samples with the value 0xFFF (4095) corresponds to 2.5 V. We can use Matlab to generate the lookup table for x=0, Δx, 2Δx,..2*pi, where Δx=2*pi/256. Figure 2 shows the Matlab program. We generate 256 samples (actually 257) in the range x=[0...2*pi] and calculate the integer values that correspond to each sample x, using the original function y=1.25*(1+sin(x)). The values are rounded to nearest integers and the samples are written back to a file. Note: the actual number of samples is 257, so we should remove the last one (the same as the first). We should also ensure that no samples are larger than 4095 (the largest number that can be represented by 12-bits). These constants will be used to initialize the lookup table visible to your program (here we create a header file named sine_lut_256.h). The sine_lut_256.h file can be found in the course secure directory.

```
x=(0:2*pi/256:2*pi)
y=1.25*(1+sin(x))
dac12=y*4096/2.5
dac12r = round(dac12)
dlmwrite('sine_lut_256.h',dac12r, ',')
```

Figure 2. Matlab program to generate a 256-entry lookup table.

The next step is to determine the trigger period. We want 256 samples to spread over 2*pi range of x. The required period of the sine wave is 10 Hz (0.1sec). That means that the trigger period is 0.1/256 sec. We will use a TimerA device to generate triggers. Assuming that the default clock frequency on SMCLK is used as the timer clock (1048576 Hz), we can determine the value that needs to be written to the TimerA counter $(0.1/256)*F_{SMCLK} = 410$.

Figure 3 shows the complete program. We stop the watchdog time, initialize the ADC12 to give a reference voltage of 2.5 V, and initialize the timer to raise an interrupt every 0.1/256 sec. The TimerA ISR wakes the processor, we read the next sample from the table, and output it to the DAC12.

```
//******************************************************************************
//   MSP430xG461x Demo - DAC12_0, Output Voltage SINEWAVE on DAC0
//
//   Description: Using DAC12_0 and 2.5V ADC12REF reference with a gain of 1,
//   output sinisoidal wave on P6.6, y=1.25(1+sin(x)).
//   Normal mode is LPM0 with CPU off.  WDT used
//   to provide ~0.064ms interrupt used to wake up the CPU and update the DAC
//   with software. Use internal 2.5V Vref.
//   ACLK = 32kHz, SMCLK = MCLK = WDTCLK = default DCO 1048576Hz
//
//
//               MSP430xG461x
//            -----------------
//         /|\|              XIN|-
//          | |                 | 32kHz
//          --|RST          XOUT|-
//            |                 |
//            |        DAC0/P6.6|--> sine (10Hz)
//            |                 |
//
// Author: Aleksandar Milenkovic, milenkovic@computer.org
// Date:  October 2009
//******************************************************************************
#include "msp430xG46x.h"
#include "sine_lut_256.h"  /*256 samples are stored in this table */

void main(void) {
  unsigned int i;

  WDTCTL = WDTPW + WDTHOLD;                   // Stop WDT
  ADC12CTL0 = REF2_5V + REFON;                // Internal 2.5V ref on
  for (i=50000; i>0; i--);                    // Delay to allow Ref to settle
  __disable_interrupt();                      // Disable Interrupts
  DAC12_0CTL = DAC12IR + DAC12AMP_5 + DAC12ENC;   //Sets DAC12
  CCTL0 = CCIE;                                // CCR0 interrupt enabled
  CCR0 = 410;                                   // Sets Timer Freq (1048576*0.1sec/256)
  TACTL = TASSEL_2 + MC_1;                     // set clock to SMCLK, up mode
  i=0;
  while (1)
  {
    __bis_SR_register(LPM0_bits + GIE);     // Enter LPM0, interrupts enabled
    DAC12_0DAT = LUT256[i];
    DAC12_0DAT &= 0xFFF;
    i=(i+1)%256;
  }
}

#pragma vector = TIMERA0_VECTOR
__interrupt void TA0_ISR(void)
{
  __bic_SR_register_on_exit(LPM0_bits);     // Exit LPMx, interrupts enabled
}
```

Figure 3. Sine Wave Generator.

3. Assignments

Assignment #1: Serial communication (PC to EasyWeb2 and EasyWeb2 to PC)
Write a C program that samples the integrated on-chip temperature sensor (analog input channel INCHx = 1010, see User's Guide, Chapter 17 for details) once every second, calculates the temperature in degrees of Celsius and Fahrenheit, time-stamps it, and sends it to the HyperTerminal. The format of a reading on the HyperTerminal should be as follows: 01:23 <78 F/ 26 C> ("Minutes:Seconds <Degrees Fahrenheit / Degrees Celsius>"). The time keeping starts from 00:00.

Assignment #2: DAC12
Write a C program that will generate a sinusoidal wave with frequency of 100 Hz. You should use the lookup table initialized with 256 values for the first quadrant of the sine wave, that is: 0, Δx, $2\Delta x$,.. pi/2, where Δx=pi/512. You should implement the control in the program to get the correct sine wave. What are advantages and disadvantages of this approach?