SENSOR DE TEMPERATURA

Diseño e implementación de un Sistema de Bardware y Software, basado en una arquitectura ARM, que toma valores de un sensor de temperatura y, mediante comunicación UART, envía los resultados en una PC que los presenta en una interfaz de usuario.

Bollini Leonardo
Martínez Franco

Índice

Objetivo

Este trabajo consiste en el diseño e implementación de un sistema encargado de censar la temperatura de una sala y mostrar, mediante una aplicación desarrollada en Python, los valores medidos y un gráfico con el histórico de temperaturas.

Para este trabajo, se decidió trabajar con un potenciómetro ya que éste nos permite mostrar la funcionalidad del sistema con mayor facilidad. La implementación de éste con un sensor de temperatura es trivial y no representa mayores complicaciones.

Implementación

El sistema consiste de un potenciómetro que entrega un valor de tensión variable entre 0V y 3,3V. Este valor es tomado por el módulo ADC del microcontrolador LPC1769, convirtiendo esta señal a un valor digital de 12 bits. El LPC cada un intervalo de tiempo definido, lee el valor entregado por el potenciómetro.

El microcontrolador comunicará este valor a una computadora por UART. Para ello, convierte este valor binario a caracteres, para poder enviarlo correctamente.

La PC recibe el mensaje con el valor medido. Un programa desarrollado en Python toma esta medición, y muestra un gráfico de temperatura histórico con los últimos valores medidos.

Siendo que los valores entregados a la PC varían entre 0 y 4095, el programa agrupa las mediciones en rango que representan una variación entre 0°C y 45°C.

Elementos Principales Utilizados

Placa de Desarrollo LPCXpresso 1769. Microprocesador ARM Cortex M3.

Es el encargado de tomar el valor del sensor de temperatura (potenciómetro en este caso), prepararlo para comunicarlo, y enviarlo a la PC.

Potenciómetro de 1 KOhm.

Hace las veces de sensor de temperatura.

Driver Max 232 para la conexión serie entre ARM – PC.

Necesario para la comunicación serie. Adapta los niveles de tensiones de la señal que entrega el circuito del microprocesador y de lo que necesita la PC para interpretar correctamente el valor.

LPC

El microcontrolador se programó para que periódicamente cense el valor del potenciómetro. Esto se llevó a cabo utilizando interrupciones del Timer0.

Al momento de obtener el valor, se utiliza el módulo ADC, obteniendo un valor digital de 12 bits que representa esta medición.

Utilizando librerías de C, se convierte este valor a una cadena de caracteres, ya que, al ser la comunicación UART, enviamos caracteres por el puerto (cada carácter es de 8 bits).

En la PC, un programa desarrollado en Python escucha el puerto. Cuando llega un valor, lo toma, lo transforma en un entero, y lo almacena en una cola FIFO de tamaño fijo.

Esta cola hace las veces de buffer. Aquí se almacenarán los datos que llegan, y será desde aquí donde el programa lee los datos para mostrarlos en un gráfico.

Conexión

La conexión entre LPC y PC es a través de un Driver MAX232, el cual se transforma los valores de tensión a niveles TTL, interpretables por la PC.

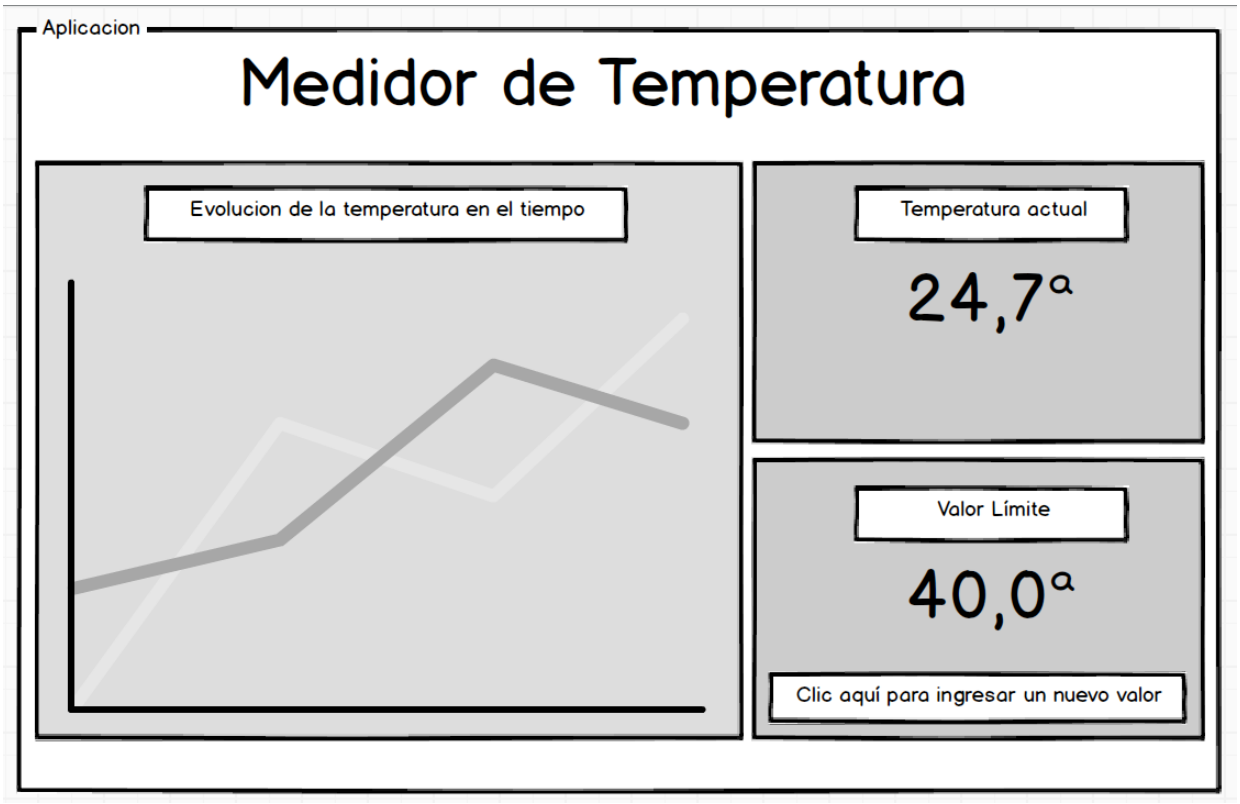
Se utilizó un cable Serial-USB y se instalaron drivers para emular el puerto COM, ya que las computadoras donde se probó el programa no contaban con puertos seriales.

PC. Interfaz Gráfica

Se definió la interfaz gráfica del usuario de la siguiente forma:

- La pantalla le muestra al usuario un resumen histórico de la temperatura.
- Un cuadro de entrada de texto permite que el usuario ingrese una nueva temperatura de umbral.

Cuando se pensó en la interfaz de usuario se planteó un sistema que pueda adquirir más características en el tiempo. Por ejemplo, que permita sonar una alarma en caso que se supere un valor de temperatura determinado. Este es una funcionalidad a agregar más adelante.



En el programa de la computadora, se muestra por pantalla una ventana dividida en dos partes.

En la primera parte, se observa el gráfico con el histórico de temperturas.

En la otra fila se observan tres displays, en los cuales se ven las temperaturas mínima, establecida en 0°, la actual, y la temperatura máxima, definida en 45° C.

Este rango de temperaturas se definió simplemente a los fines de mostrar el funcionamiento del sistema. No representa valores significativos.

Código

Para el desarrollo del código, se han utilizado al máximo las capacidades de las librerías que provee la suite de desarrollo, por lo que el código es muy compacto, modularizado y de fácil comprensión.

Código C del programa del microcontrolador:

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

//ADC
#include "freq.h"
#include "adc.h"
#include "debug.h"
#include "small_systick.h"

//UART
#include "type.h"
#include "uart.h"

//Utilidades relacionadas al manejo de cadenas y caracteres.
#include <string.h>
#include <stdlib.h>

#include <cr_section_macros.h>


volatile uint32_t SysTickCount;           /* counts 10ms timeTicks */
uint8_t cont;
volatile uint16_t adcCount;

// InitSysTick() sets systick timer to 10 mS
#define InitSysTick(MHz) SysTick_Config(MHz * (1000000/100))

/*-------------------------------------------------------------------------
  SysTick_Handler
  Cuando Timer0 interrumpe, medimos un valor en el ADC, lo convertimos a
  caracteres y lo enviamos por UART
 *-------------------------------------------------------------------------*/
void SysTick_Handler(void)  //interrumpe cada 10 milisegundos
{
        char mensaje[8];

        SysTickCount++; /* provee delay */

        if (SysTickCount < 100) return; //100 = 1 segundo

        //Tomamos el valor leido por el sensor
        adcCount = ADC0Read(0);  //Se lee el puerto P0[23]

        //Convierte el valor medido a caracteres (un caracter por cada
        //digito decimal)
        itoa(adcCount, mensaje, 10);

        //Agrega un retorno de carro y nueva línea al mensaje
        strcat(mensaje, "\r\n");

        //Enviamos por UART el valor:
        UARTSend(3, (uint8_t *)mensaje , strlen(mensaje) );
        SysTickCount = 0;
```

```
55
56          //Prende y apaga un LED para saber que el sistema está funcionando:
57          //Sirve como control
58          if (cont == 0){
59                  LPC_GPIO0->FIOCLR = (1 << 22); // P0[22] = 0
60                  cont = 1;
61          }
62          else {
63                  LPC_GPIO0->FIOSET = (1 << 22); // P0[22] = 1
64                  cont = 0;
65          }
66  }
67
68
69  int main(void) {
70
71      // TODO: insert code here
72          adcCount = 1000;
73
74          LPC_GPIO0->FIODIR |= (1 << 22); // P0[22] COMO SALIDA.
75
76
77          ADCInit(ADC_CLK);  //Inicializa el ADC. Entrada ADC por el puerto P0[23].
78          cont = 0;
79          InitSysTick(100);  //Inicializa el Timer para que interrumpa cada 10 ms.
80
81          //UART Setting
82          UARTInit(3, 9600);        /* baud rate setting */
83
84      while(1) {
85      }
86      return 0 ;
87  }
```

Mencionamos las librerías utilizadas, que se encuentran en el IDE del LPC en caso de que se necesite consultarlas. En el anexo se encuentra el código de las librerías incluidas en el IDE.

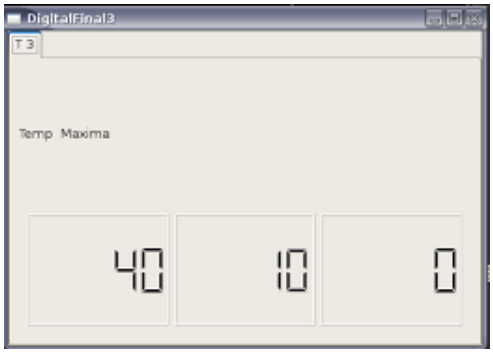
```
//ADC
"freq.h"
"adc.h"
"debug.h"
"small_systick.h"

//UART
"type.h"
"uart.h"

//Utilidades relacionadas al manejo de cadenas y caracteres.
<string.h>
<stdlib.h>
```

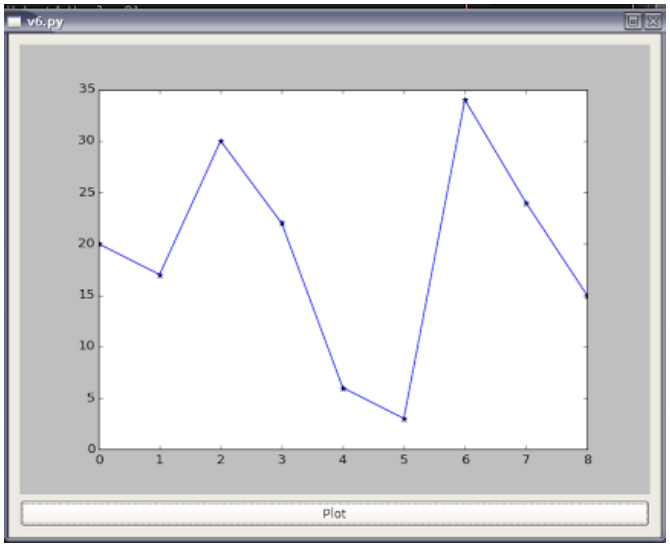
Código Python

Lectura del puerto serial

```
import serial
from Queue import Queue

for i in range(0, 10):
    try:
        ser = serial.Serial(i)  # open first serial port
        print "Conectado a", ser.name   # check which port was really used

        cola = Queue(10)

        while True:
            line = ser.readline()    # read a '\n' terminated line
            valor = int(line)

            try:
                cola.put_nowait(valor)

            except Exception, e:
                cola.get_nowait()
                cola.put_nowait(valor)

            print "el valor en la cola es:", list(cola.queue)

            #print line
        ser.close()             # close port
    except Exception, e:
        pass
```

Captura de Pantalla de la Interfaz gráfica





Conclusión

Este trabajo nos permitió ver ante todo el poder y la sencillez que nos da el trabajar con un LPC, más sumado a la robustez de un lenguaje como Python que nos permitió poder obtener los datos y mostrarlo en formato elegante.

El hecho de haber realizado un trabajo que integre tanto el diseño y conexión del sistema y el desarrollo del software nos da cuenta del poder y versatilidad de los microcontroladores y sistemas embebidos.

Anexo. Librerías

Uart.c

```
/****************************************************************************
 * $Id::                                                                    $
 *
 * Project:        uart: Simple UART echo for LPCXpresso 1700
 * File:  uart.c
 * Description:
 *                          LPCXpresso Baseboard uses pins mapped to UART3 for
 *                          its USB-to-UART bridge. This application simply echos
 *                          all characters received.
 *
 ****************************************************************************
 * Software that is described herein is for illustrative purposes only
 * which provides customers with programming information regarding the
 * products. This software is supplied "AS IS" without any warranties.
 * NXP Semiconductors assumes no responsibility or liability for the
 * use of the software, conveys no license or title under any patent,
 * copyright, or mask work right to the product. NXP Semiconductors
 * reserves the right to make changes in the software without
 * notification. NXP Semiconductors also make no representation or
 * warranty that such application will be suitable for the specified
 * use without further testing or modification.
 ****************************************************************************/

/*****************************************************************************
 *   History
 *   2010.07.01  ver 1.01    Added support for UART3, tested on LPCXpresso 1700
 *   2009.05.27  ver 1.00    Prelimnary version, first Release
 *
*****************************************************************************/
#include "LPC17xx.h"
#include "type.h"
#include "uart.h"

volatile uint32_t UART0Status, UART1Status, UART3Status;
volatile uint8_t UART0TxEmpty = 1, UART1TxEmpty = 1, UART3TxEmpty=1;
volatile uint8_t UART0Buffer[BUFSIZE], UART1Buffer[BUFSIZE], UART3Buffer[BUFSIZE];
volatile uint32_t UART0Count = 0, UART1Count = 0, UART3Count = 0;

/*****************************************************************************
** Function name:            UART0 IRQHandler
**
** Descriptions:             UART0 interrupt handler
**
** parameters:                       None
** Returned value:           None
**
*****************************************************************************/
void UART0_IRQHandler (void)
{
  uint8_t IIRValue, LSRValue;
  uint8_t Dummy = Dummy;

  IIRValue = LPC_UART0->IIR;

  IIRValue >>= 1;                     /* skip pending bit in IIR */
  IIRValue &= 0x07;                   /* check bit 1~3, interrupt identification */
  if ( IIRValue == IIR_RLS )          /* Receive Line Status */
  {
        LSRValue = LPC_UART0->LSR;
        /* Receive Line Status */
        if ( LSRValue & (LSR_OE|LSR_PE|LSR_FE|LSR_RXFE|LSR_BI) )
        {
```

```
            /* There are errors or break interrupt */
            /* Read LSR will clear the interrupt */
            UART0Status = LSRValue;
            Dummy = LPC_UART0->RBR;             /* Dummy read on RX to clear
                                                                    interrupt, then bail out */
            return;
          }
          if ( LSRValue & LSR_RDR )    /* Receive Data Ready */
          {
            /* If no error on RLS, normal ready, save into the data buffer. */
            /* Note: read RBR will clear the interrupt */
            UART0Buffer[UART0Count] = LPC_UART0->RBR;
            UART0Count++;
            if ( UART0Count == BUFSIZE )
            {
                    UART0Count = 0;             /* buffer overflow */
            }
          }
    }
    else if ( IIRValue == IIR_RDA )     /* Receive Data Available */
    {
            /* Receive Data Available */
            UART0Buffer[UART0Count] = LPC_UART0->RBR;
            UART0Count++;
            if ( UART0Count == BUFSIZE )
            {
              UART0Count = 0;           /* buffer overflow */
            }
    }
    else if ( IIRValue == IIR_CTI )     /* Character timeout indicator */
    {
            /* Character Time-out indicator */
            UART0Status |= 0x100;               /* Bit 9 as the CTI error */
    }
    else if ( IIRValue == IIR_THRE )    /* THRE, transmit holding register empty */
    {
            /* THRE interrupt */
            LSRValue = LPC_UART0->LSR;          /* Check status in the LSR to see if
                                                                        valid data in U0THR
or not */
            if ( LSRValue & LSR_THRE )
            {
              UART0TxEmpty = 1;
            }
            else
            {
              UART0TxEmpty = 0;
            }
    }
}

/*****************************************************************************
** Function name:           UART1_IRQHandler
**
** Descriptions:            UART1 interrupt handler
**
** parameters:                      None
** Returned value:          None
**
*****************************************************************************/
void UART1_IRQHandler (void)
{
  uint8_t IIRValue, LSRValue;
  uint8_t Dummy = Dummy;

  IIRValue = LPC_UART1->IIR;

  IIRValue >>= 1;                   /* skip pending bit in IIR */
  IIRValue &= 0x07;                 /* check bit 1~3, interrupt identification */
  if ( IIRValue == IIR_RLS )        /* Receive Line Status */
  {
```

```
          LSRValue = LPC_UART1->LSR;
          /* Receive Line Status */
          if ( LSRValue & (LSR_OE|LSR_PE|LSR_FE|LSR_RXFE|LSR_BI) )
          {
            /* There are errors or break interrupt */
            /* Read LSR will clear the interrupt */
            UART1Status = LSRValue;
            Dummy = LPC UART1->RBR;             /* Dummy read on RX to clear
                                                                    interrupt, then bail out */
            return;
          }
          if ( LSRValue & LSR_RDR )    /* Receive Data Ready */
          {
            /* If no error on RLS, normal ready, save into the data buffer. */
            /* Note: read RBR will clear the interrupt */
            UART1Buffer[UART1Count] = LPC UART1->RBR;
            UART1Count++;
            if ( UART1Count == BUFSIZE )
            {
                    UART1Count = 0;             /* buffer overflow */
            }
          }
    }
    else if ( IIRValue == IIR_RDA )     /* Receive Data Available */
    {
          /* Receive Data Available */
          UART1Buffer[UART1Count] = LPC UART1->RBR;
          UART1Count++;
          if ( UART1Count == BUFSIZE )
          {
            UART1Count = 0;             /* buffer overflow */
          }
    }
    else if ( IIRValue == IIR_CTI )     /* Character timeout indicator */
    {
          /* Character Time-out indicator */
          UART1Status |= 0x100;                 /* Bit 9 as the CTI error */
    }
    else if ( IIRValue == IIR_THRE )    /* THRE, transmit holding register empty */
    {
          /* THRE interrupt */
          LSRValue = LPC_UART1->LSR;            /* Check status in the LSR to see if
                                                                    valid data in U0THR or not
*/
          if ( LSRValue & LSR_THRE )
          {
            UART1TxEmpty = 1;
          }
          else
          {
            UART1TxEmpty = 0;
          }
    }

}
/*****************************************************************************
** Function name:           UART0_IRQHandler
**
** Descriptions:            UART0 interrupt handler
**
** parameters:                      None
** Returned value:          None
**
*****************************************************************************/
void UART3_IRQHandler (void)
{
  uint8_t IIRValue, LSRValue;
  uint8_t Dummy = Dummy;

  IIRValue = LPC UART3->IIR;
```

```
  IIRValue >>= 1;                       /* skip pending bit in IIR */
  IIRValue &= 0x07;                     /* check bit 1~3, interrupt identification */
  if ( IIRValue == IIR_RLS )            /* Receive Line Status */
  {
        LSRValue = LPC_UART3->LSR;
        /* Receive Line Status */
        if ( LSRValue & (LSR_OE|LSR_PE|LSR_FE|LSR_RXFE|LSR_BI) )
        {
          /* There are errors or break interrupt */
          /* Read LSR will clear the interrupt */
          UART3Status = LSRValue;
          Dummy = LPC_UART3->RBR;             /* Dummy read on RX to clear
                                                                  interrupt, then bail out */
          return;
        }
        if ( LSRValue & LSR_RDR )    /* Receive Data Ready */
        {
          /* If no error on RLS, normal ready, save into the data buffer. */
          /* Note: read RBR will clear the interrupt */
          UART3Buffer[UART3Count] = LPC UART3->RBR;
          UART3Count++;
          if ( UART3Count == BUFSIZE )
          {
                UART3Count = 0;             /* buffer overflow */
          }
        }
  }
  else if ( IIRValue == IIR_RDA )      /* Receive Data Available */
  {
        /* Receive Data Available */
        UART3Buffer[UART3Count] = LPC_UART3->RBR;
        UART3Count++;
        if ( UART3Count == BUFSIZE )
        {
          UART3Count = 0;            /* buffer overflow */
        }
  }
  else if ( IIRValue == IIR_CTI )      /* Character timeout indicator */
  {
        /* Character Time-out indicator */
        UART3Status |= 0x100;                /* Bit 9 as the CTI error */
  }
  else if ( IIRValue == IIR_THRE )     /* THRE, transmit holding register empty */
  {
        /* THRE interrupt */
        LSRValue = LPC_UART3->LSR;           /* Check status in the LSR to see if
                                                                        valid data in U0THR
or not */
        if ( LSRValue & LSR_THRE )
        {
          UART3TxEmpty = 1;
        }
        else
        {
          UART3TxEmpty = 0;
        }
  }
}


/*****************************************************************************
** Function name:           UARTInit
**
** Descriptions:            Initialize UART port, setup pin select,
**                                                  clock, parity, stop bits, FIFO, etc.
**
** parameters:                        portNum(0 or 1) and UART baudrate
** Returned value:          true or false, return false only if the
**                                                  interrupt handler can't be installed to the
**                                                  VIC table
**
```

```
*****************************************************************************/
uint32 t UARTInit( uint32 t PortNum, uint32 t baudrate )
{
  uint32 t Fdiv;
  uint32_t pclkdiv, pclk;

  if ( PortNum == 0 )
  {
        LPC_PINCON->PINSEL0 &= ~0x000000F0;
        LPC_PINCON->PINSEL0 |= 0x00000050;  /* RxD0 is P0.3 and TxD0 is P0.2 */
        /* By default, the PCLKSELx value is zero, thus, the PCLK for
        all the peripherals is 1/4 of the SystemFrequency. */
        /* Bit 6~7 is for UART0 */
        pclkdiv = (LPC SC->PCLKSEL0 >> 6) & 0x03;
        switch ( pclkdiv )
        {
          case 0x00:
          default:
                pclk = SystemCoreClock/4;
                break;
          case 0x01:
                pclk = SystemCoreClock;
                break;
          case 0x02:
                pclk = SystemCoreClock/2;
                break;
          case 0x03:
                pclk = SystemCoreClock/8;
                break;
        }

    LPC_UART0->LCR = 0x83;            /* 8 bits, no Parity, 1 Stop bit */
          Fdiv = ( pclk / 16 ) / baudrate ;     /*baud rate */
    LPC_UART0->DLM = Fdiv / 256;
    LPC_UART0->DLL = Fdiv % 256;
          LPC_UART0->LCR = 0x03;                /* DLAB = 0 */
    LPC_UART0->FCR = 0x07;            /* Enable and reset TX and RX FIFO. */

          NVIC EnableIRQ(UART0 IRQn);

    LPC_UART0->IER = IER_RBR | IER_THRE | IER_RLS;       /* Enable UART0 interrupt */
    return (TRUE);
  }
  else if ( PortNum == 1 )
  {
        LPC_PINCON->PINSEL4 &= ~0x0000000F;
        LPC_PINCON->PINSEL4 |= 0x0000000A;    /* Enable RxD1 P2.1, TxD1 P2.0 */

        /* By default, the PCLKSELx value is zero, thus, the PCLK for
        all the peripherals is 1/4 of the SystemFrequency. */
        /* Bit 8,9 are for UART1 */
        pclkdiv = (LPC SC->PCLKSEL0 >> 8) & 0x03;
        switch ( pclkdiv )
        {
          case 0x00:
          default:
                pclk = SystemCoreClock/4;
                break;
          case 0x01:
                pclk = SystemCoreClock;
                break;
          case 0x02:
                pclk = SystemCoreClock/2;
                break;
          case 0x03:
                pclk = SystemCoreClock/8;
                break;
        }

    LPC UART1->LCR = 0x83;            /* 8 bits, no Parity, 1 Stop bit */
          Fdiv = ( pclk / 16 ) / baudrate ;     /*baud rate */
```

```
    LPC_UART1->DLM = Fdiv / 256;
    LPC_UART1->DLL = Fdiv % 256;
          LPC_UART1->LCR = 0x03;                /* DLAB = 0 */
    LPC_UART1->FCR = 0x07;            /* Enable and reset TX and RX FIFO. */

          NVIC_EnableIRQ(UART1_IRQn);

    LPC_UART1->IER = IER_RBR | IER_THRE | IER_RLS;       /* Enable UART1 interrupt */
    return (TRUE);
  }
  else if ( PortNum == 3 )
  {
          LPC_PINCON->PINSEL0 &= ~0x0000000F;
          LPC_PINCON->PINSEL0 |=  0x0000000A;  /* RxD3 is P0.1 and TxD3 is P0.0 */
          LPC_SC->PCONP |= 1<<4 | 1<<25; //Enable PCUART1
          /* By default, the PCLKSELx value is zero, thus, the PCLK for
                 all the peripherals is 1/4 of the SystemFrequency. */
          /* Bit 6~7 is for UART3 */
          pclkdiv = (LPC_SC->PCLKSEL1 >> 18) & 0x03;
          switch ( pclkdiv )
          {
          case 0x00:
          default:
                    pclk = SystemCoreClock/4;
                    break;
          case 0x01:
                    pclk = SystemCoreClock;
                    break;
          case 0x02:
                    pclk = SystemCoreClock/2;
                    break;
          case 0x03:
                    pclk = SystemCoreClock/8;
                    break;
          }
          LPC_UART3->LCR = 0x83;              /* 8 bits, no Parity, 1 Stop bit */
          Fdiv = ( pclk / 16 ) / baudrate ;   /*baud rate */
          LPC_UART3->DLM = Fdiv / 256;
          LPC_UART3->DLL = Fdiv % 256;
          LPC_UART3->LCR = 0x03;              /* DLAB = 0 */
          LPC_UART3->FCR = 0x07;              /* Enable and reset TX and RX FIFO. */

          NVIC_EnableIRQ(UART3_IRQn);

          LPC_UART3->IER = IER_RBR | IER_THRE | IER_RLS;         /* Enable UART3 interrupt */
          return (TRUE);
  }
  return( FALSE );
}

/*****************************************************************************
** Function name:           UARTSend
**
** Descriptions:            Send a block of data to the UART 0 port based
**                                                  on the data length
**
** parameters:                      portNum, buffer pointer, and data length
** Returned value:          None
**
*****************************************************************************/
void UARTSend( uint32_t portNum, uint8_t *BufferPtr, uint32_t Length )
{
  if ( portNum == 0 )
  {
    while ( Length != 0 )
    {
          /* THRE status, contain valid data */
          while ( !(UART0TxEmpty & 0x01) );
          LPC_UART0->THR = *BufferPtr;
          UART0TxEmpty = 0;/* not empty in the THR until it shifts out */
          BufferPtr++;
```

```
            Length--;
          }
  }
  else if (portNum == 1)
  {
          while ( Length != 0 )
    {
            /* THRE status, contain valid data */
            while ( !(UART1TxEmpty & 0x01) );
            LPC UART1->THR = *BufferPtr;
            UART1TxEmpty = 0;/* not empty in the THR until it shifts out */
            BufferPtr++;
            Length--;
    }
  }
  else if ( portNum == 3 )
  {
    while ( Length != 0 )
    {
            /* THRE status, contain valid data */
            while ( !(UART3TxEmpty & 0x01) );
            LPC UART3->THR = *BufferPtr;
            UART3TxEmpty = 0;/* not empty in the THR until it shifts out */
            BufferPtr++;
            Length--;
          }
  }
  return;
}

/*****************************************************************************
**                            End Of File
*****************************************************************************/
```

Uart.h

```
/*****************************************************************************
 * $Id::                                                                     $
 *
 * Project:        uart: Simple UART echo for LPCXpresso 1700
 * File: uarttest.c
 * Description:
 *                          LPCXpresso Baseboard uses pins mapped to UART3 for
 *                          its USB-to-UART bridge. This application simply echos
 *                          all characters received.
 *
 *****************************************************************************
 * Software that is described herein is for illustrative purposes only
 * which provides customers with programming information regarding the
 * products. This software is supplied "AS IS" without any warranties.
 * NXP Semiconductors assumes no responsibility or liability for the
 * use of the software, conveys no license or title under any patent,
 * copyright, or mask work right to the product. NXP Semiconductors
 * reserves the right to make changes in the software without
 * notification. NXP Semiconductors also make no representation or
 * warranty that such application will be suitable for the specified
 * use without further testing or modification.
 *****************************************************************************/

/*****************************************************************************
 *   History
 *   2010.07.01  ver 1.01    Added support for UART3, tested on LPCXpresso 1700
 *   2009.05.27  ver 1.00    Prelimnary version, first Release
 *
*****************************************************************************/
#ifndef __UART_H
#define __UART_H
```

```
#define IER_RBR             0x01
#define IER_THRE    0x02
#define IER_RLS             0x04

#define IIR_PEND    0x01
#define IIR_RLS             0x03
#define IIR_RDA             0x02
#define IIR_CTI             0x06
#define IIR_THRE    0x01

#define LSR_RDR             0x01
#define LSR_OE              0x02
#define LSR_PE              0x04
#define LSR_FE              0x08
#define LSR_BI              0x10
#define LSR_THRE    0x20
#define LSR_TEMT    0x40
#define LSR_RXFE    0x80

#define BUFSIZE             0x40

uint32_t UARTInit( uint32_t portNum, uint32_t Baudrate );
void UART0_IRQHandler( void );
void UART1_IRQHandler( void );
void UARTSend( uint32_t portNum, uint8_t *BufferPtr, uint32_t Length );

#endif /* end __UART_H */
/*****************************************************************************
**                            End Of File
*****************************************************************************/
```

Adc.c

```
/*****************************************************************************
 *   adc.c:  ADC module file for NXP LPC17xx Family Microprocessors
 *
 *   Copyright(C) 2009, NXP Semiconductor
 *   All rights reserved.
 *
 *   History
 *   2009.05.25  ver 1.00    Prelimnary version, first Release
 *
*****************************************************************************/
#include "LPC17xx.h"
#include "type.h"
#include "adc.h"

volatile uint32_t ADC0Value[ADC_NUM];
volatile uint32_t ADC0IntDone = 0;

#if BURST_MODE
volatile uint32_t channel_flag;
#endif

#if ADC_INTERRUPT_FLAG
/*****************************************************************************
** Function name:           ADC_IRQHandler
**
** Descriptions:            ADC interrupt handler
**
** parameters:                      None
** Returned value:          None
**
*****************************************************************************/
void ADC_IRQHandler (void)
{
```

```
uint32_t regVal;

regVal = LPC_ADC->ADSTAT;           /* Read ADC will clear the interrupt */
if ( regVal & 0x0000FF00 ) /* check OVERRUN error first */
{
        regVal = (regVal & 0x0000FF00) >> 0x08;
        /* if overrun, just read ADDR to clear */
        /* regVal variable has been reused. */
        switch ( regVal )
        {
          case 0x01:
                regVal = LPC_ADC->ADDR0;
          break;
          case 0x02:
                regVal = LPC_ADC->ADDR1;
          break;
          case 0x04:
                regVal = LPC_ADC->ADDR2;
          break;
          case 0x08:
                regVal = LPC_ADC->ADDR3;
          break;
          case 0x10:
                regVal = LPC_ADC->ADDR4;
          break;
          case 0x20:
                regVal = LPC ADC->ADDR5;
          break;
          case 0x40:
                regVal = LPC_ADC->ADDR6;
          break;
          case 0x80:
                regVal = LPC ADC->ADDR7;
          break;
          default:
          break;
        }
        LPC_ADC->ADCR &= 0xF8FFFFFF; /* stop ADC now */
        ADC0IntDone = 1;
        return;
}

if ( regVal & ADC_ADINT )
{
        switch ( regVal & 0xFF )     /* check DONE bit */
        {
          case 0x01:
                ADC0Value[0] = ( LPC_ADC->ADDR0 >> 4 ) & 0xFFF;
          break;
          case 0x02:
                ADC0Value[1] = ( LPC_ADC->ADDR1 >> 4 ) & 0xFFF;
          break;
          case 0x04:
                ADC0Value[2] = ( LPC_ADC->ADDR2 >> 4 ) & 0xFFF;
          break;
          case 0x08:
                ADC0Value[3] = ( LPC_ADC->ADDR3 >> 4 ) & 0xFFF;
          break;
          case 0x10:
                ADC0Value[4] = ( LPC_ADC->ADDR4 >> 4 ) & 0xFFF;
          break;
          case 0x20:
                ADC0Value[5] = ( LPC_ADC->ADDR5 >> 4 ) & 0xFFF;
          break;
          case 0x40:
                ADC0Value[6] = ( LPC_ADC->ADDR6 >> 4 ) & 0xFFF;
          break;
          case 0x80:
                ADC0Value[7] = ( LPC_ADC->ADDR7 >> 4 ) & 0xFFF;
          break;
          default:
```

```
            break;
          }
#if BURST_MODE
          channel_flag |= (regVal & 0xFF);
          if ( (channel_flag & 0xFF) == 0xFF )
          {
            /* All the bits in have been set, it indicates all the ADC
            channels have been converted. */
            LPC_ADC->ADCR &= 0xF8FFFFFF;        /* stop ADC now */
            ADC0IntDone = 1;
          }
#else
          LPC_ADC->ADCR &= 0xF8FFFFFF; /* stop ADC now */
          ADC0IntDone = 1;
#endif
    }
    return;
}
#endif

/*****************************************************************************
** Function name:           ADCInit
**
** Descriptions:            initialize ADC channel
**
** parameters:                      ADC clock rate
** Returned value:          true or false
**
*****************************************************************************/
uint32_t ADCInit( uint32_t ADC_Clk )
{
  uint32_t pclkdiv, pclk;

  /* Enable CLOCK into ADC controller */
  LPC_SC->PCONP |= (1 << 12);

  /* all the related pins are set to ADC inputs, AD0.0~7 */
  LPC_PINCON->PINSEL0 |= 0x0F000000;  /* P0.12~13, A0.6~7, function 11 */
  LPC_PINCON->PINSEL1 &= ~0x003FC000; /* P0.23~26, A0.0~3, function 01 */
  LPC_PINCON->PINSEL1 |= 0x00154000;
  LPC_PINCON->PINSEL3 |= 0xF0000000;  /* P1.30~31, A0.4~5, function 11 */

  /* By default, the PCLKSELx value is zero, thus, the PCLK for
  all the peripherals is 1/4 of the SystemFrequency. */
  /* Bit 6~7 is for UART0 */
  pclkdiv = (LPC_SC->PCLKSEL0 >> 6) & 0x03;
  switch ( pclkdiv )
  {
        case 0x00:
        default:
          pclk = SystemCoreClock/4;
        break;
        case 0x01:
          pclk = SystemCoreClock;
        break;
        case 0x02:
          pclk = SystemCoreClock/2;
        break;
        case 0x03:
          pclk = SystemCoreClock/8;
        break;
  }

  LPC_ADC->ADCR = ( 0x01 << 0 ) |     /* SEL=1,select channel 0 on ADC0 */
                  ( ( pclk  / ADC_Clk - 1 ) << 8 ) |  /* CLKDIV = Fpclk / ADC_Clk - 1 */
                  ( 0 << 16 ) |               /* BURST = 0, no BURST, software controlled */
                  ( 0 << 17 ) |               /* CLKS = 0, 11 clocks/10 bits */
                  ( 1 << 21 ) |               /* PDN = 1, normal operation */
                  ( 0 << 24 ) |               /* START = 0 A/D conversion stops */
                  ( 0 << 27 );                /* EDGE = 0 (CAP/MAT singal falling,trigger A/D
conversion) */
```

```
  /* If POLLING, no need to do the following */
#if ADC_INTERRUPT_FLAG
  LPC_ADC->ADINTEN = 0x1FF;           /* Enable all interrupts */
  NVIC_EnableIRQ(ADC_IRQn);
#endif
  return (TRUE);
}

/*****************************************************************************
** Function name:            ADC0Read
**
** Descriptions:             Read ADC0 channel
**
** parameters:                       Channel number
** Returned value:           Value read, if interrupt driven, return channel #
**
*****************************************************************************/
uint32_t ADC0Read( uint8_t channelNum )
{
#if !ADC_INTERRUPT_FLAG
  uint32_t regVal, ADC_Data;
#endif

  /* channel number is 0 through 7 */
  if ( channelNum >= ADC_NUM )
  {
        channelNum = 0;             /* reset channel number to 0 */
  }
  LPC_ADC->ADCR &= 0xFFFFFF00;
  LPC_ADC->ADCR |= (1 << 24) | (1 << channelNum);
                                    /* switch channel,start A/D convert */
#if !ADC_INTERRUPT_FLAG
  while ( 1 )                       /* wait until end of A/D convert */
  {
        regVal = *(volatile unsigned long *)(LPC_ADC_BASE
                          + ADC_OFFSET + ADC_INDEX * channelNum);
        /* read result of A/D conversion */
        if ( regVal & ADC_DONE )
        {
          break;
        }
  }

  LPC_ADC->ADCR &= 0xF8FFFFFF;        /* stop ADC now */
  if ( regVal & ADC_OVERRUN )         /* save data when it's not overrun, otherwise, return zero */
  {
        return ( 0 );
  }
  ADC_Data = ( regVal >> 4 ) & 0xFFF;
  return ( ADC_Data );          /* return A/D conversion value */
#else
  return ( channelNum );      /* if it's interrupt driven, the ADC reading is
                                                          done inside the handler. so, return
channel number */
#endif
}

/*****************************************************************************
** Function name:            ADC0BurstRead
**
** Descriptions:             Use burst mode to convert multiple channels once.
**
** parameters:                       None
** Returned value:           None
**
*****************************************************************************/
void ADCBurstRead( void )
{
  if ( LPC_ADC->ADCR & (0x7<<24) )
  {
```

```
        LPC_ADC->ADCR &= ~(0x7<<24);
  }
  /* Test channel 5,6,7 using burst mode because they are not shared
  with the JTAG pins. */
  LPC_ADC->ADCR &= ~0xFF;
  /* Read all channels, 0 through 7. */
  LPC_ADC->ADCR |= (0xFF);
  LPC_ADC->ADCR |= (0x1<<16);         /* Set burst mode and start A/D convert */
  return;                                                 /* the ADC reading is done inside the
                                                                                    handler, return 0. */
}

/*****************************************************************************
**                            End Of File
*****************************************************************************/
```

Adc.h

```
/*****************************************************************************
 *   adc.h:  Header file for NXP LPC17xx Family Microprocessors
 *
 *   Copyright(C) 2009, NXP Semiconductor
 *   All rights reserved.
 *
 *   History
 *   2009.05.25  ver 1.00    Prelimnary version, first Release
 *
*****************************************************************************/
#ifndef __ADC_H
#define __ADC_H

/* If Burst mode is enabled, make sure interrupt flag is set. */
#define ADC_INTERRUPT_FLAG   0        /* 1 is interrupt driven, 0 is polling */
#define BURST_MODE                    0   /* Burst mode works in interrupt driven mode only. */

#define ADC_OFFSET           0x10
#define ADC_INDEX            4

#define ADC_DONE             0x80000000
#define ADC_OVERRUN          0x40000000
#define ADC_ADINT            0x00010000

#define ADC_NUM                       8                 /* for LPCxxxx */
#define ADC_CLK                       1000000           /* set to 1Mhz */


extern uint32_t ADCInit( uint32_t ADC_Clk );
extern uint32_t ADC0Read( uint8_t channelNum );
extern void ADCBurstRead( void );
#endif /* end __ADC_H */
/*****************************************************************************
**                            End Of File
*****************************************************************************/
```