1. CPE 325: Laboratory Assignment #8 Communications Part 1 - UART

Objectives: This tutorial will introduce communication protocols used with the MSP430 and other devices. Specifically, it will cover asynchronous serial communication using the USCI peripheral. You will learn the following topics:

- Configuration of the USCI peripheral device for UART mode
- Utilization of the USCI in UART mode for PC communications
- Understanding the hyperterminal and UAH serial communication apps and packet creation

Note: All previous tutorials are required for successful completion of this lab. Especially, the tutorials introducing the TI experimenter's board and the IAR software development environment.

1.1. Serial Communication

An MSP430-based platform can communicate with another system, such as a personal computer, using either synchronous or asynchronous communication mode. For two devices to communicate synchronously, they must share a common clock source. In this lab, we are going to interface a MSP430 with a personal computer using asynchronous communication mode. Since the two devices do not share a clock signal, there should be an agreement between the devices on the speed of the communication before the actual interface starts. To configure the MSP430 in UART mode, the internal divider and the modulation register should be initialized appropriately. The internal divider is calculated by dividing the clock by the baud rate. But, the division of the clock by the baud rate is usually not a whole number. Therefore, to take account of the fraction part of the division, we use the modulation register. The value in the modulation register is calculated in such a way that the time it takes to receive/transmit each bit is as close as possible to the exact time given by the baud rate. If the appropriate modulation value is not used, the fraction part of the division of clock frequency by the baud rate will accumulate and eventually make the two devices unable to communicate. An MSP430-based platform can be connected to a PC machine using the HyperTerminal application in Windows. Let us consider a program that sends a character from the PC to the MSP430FG4618 microcontroller and echo the character back to the PC (Figure 1). Since we cannot connect the two systems to the same clock source, we should use the UART mode. The device USCI can be utilized for that purpose. The communication speed is 115200 bits/sec (one bit period is 8.68 us). The USCI clock UCLK is connected to SMCLK that is 1048576 Hz. To achieve the baud rate of the 115200, the internal divider registers are initialized to UCA0BR0=0x09, and UCABR1=0x00, because 1048576/115200 = 9.1 ~ 9. The modulator register UCA0MCTL0=0x01. See the reference manual for the details how the value in UCA0MTCL is determined. The format of a character is 8-bit ASCII. After the USCI is configured to the requirements of the communication discussed above and interrupt are enabled, there will be an infinite loop. In the loop, the MSP430 gets into LPM0 low-power mode. What clock signals are down in this mode? When a new character is received the

USCIA0RX_ISR raises an interrupt. In the ISR the MSP430 status register on the stack is modified so that the processor is active after executing the ISR. In the main program the received character is sent back to the HyperTerminal (echo), and the processor goes back in the LPM3. The whole process of raising an interrupt when a new character is received, waking up from low power mode, echoing the character and going into low power mode will be done repeatedly.

```
//*******************************************************************
**
// MSP430xG46x Demo - USCI_A0, 115200 UART Echo ISR, DCO SMCLK
//
// Description: Echo a received character, RX ISR used. Normal mode is LPM0.
// USCI_A0 RX interrupt triggers TX Echo. Toggle LED4 with every received
// character
//
// Baud rate divider with 1048576hz = 1048576/115200 = ~9.1 (009h|01h)
// ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO = 32 x ACLK = 1048576Hz
// //* An external watch crystal between XIN & XOUT is required for ACLK *//
//
//      MSP430xG461x
//     -----------------
// /|\ |            XIN|-
//  |  |                | 32kHz
//  |--|RST         XOUT|-
//     |                |
//     |    P2.4/UCA0TXD|------------>
//     |                | 115200 - 8N1
//     |    P2.5/UCA0RXD|<------------
//
// A. Milenkovic, edited by Scott Marconnet
//*******************************************************************
**
#include <msp430xG46x.h>
void main(void)
{
  WDTCTL = WDTPW+WDTHOLD;     // Stop WDT
  P5DIR |= BIT1;              // Set P5.1 to be output
  P2SEL |= BIT4 + BIT5;       // P2.4,5 = USCI_A0 RXD/TXD
  UCA0CTL1 |= UCSSEL_2;       // SMCLK
  UCA0BR0 = 0x09;             // 1MHz/115200 (lower byte)
  UCA0BR1 = 0x00;             // 1MHz/115200 (upper byte)
  UCA0MCTL = 0x02;            // Modulation (UCBRS0=0x01)(UCOS16=0)
  UCA0CTL1 &= ~UCSWRST;       // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;            // Enable USCI_A0 RX interrupt
  _BIS_SR(LPM0_bits + GIE);  // Enter LPM0, interrupts enabled
}
// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCIA0RX_ISR (void)
{
  while(!(IFG2&UCA0TXIFG));  // Wait until can transmit
  UCA0TXBUF = UCA0RXBUF;      // TX -> RXed character
  P5OUT^=BIT1;                // Toggle LED4
}
```

Figure 1. Echoing a character using the USCI device

1.2. HyperTerminal Clock

In this section we will describe a program that implements a real-time clock on the MSP430 platform (Figure 2). The time is measured from the beginning of the application with a resolution of 100 ms (one tenth of a second). The time is maintained in two variables, unsigned int sec (for seconds) and unsigned char tsec for tents of a second. What is the maximum time you can have in this case? To observe the clock we can display it either on the LCD or send it serially to a workstation using a serial communication interface. In our example we send time through a serial asynchronous link using the MSP430's USCI (Universal Serial Communication Interface) device. This device is connected to a RS232 interface (see TI experimenter's board schematic) that connects through a serial cable to a PC. On the PC side we can open Hyperterminal application and observe real-time clock that is send from our development platform.

The first step is to initialize the USCI device in UART mode for communication of 19200 bits/sec. The next step is to initialize the TimerA to measure time and update the real-time clock variables. The TimerA ISR is used to maintain clock and wake up the processor. In the main program, the local variables are taken and converted into a readable string that is then sent to the USCI device.

```
//*****************************************************************************
// FG4618 Board Demo - Real-time clock in Hyperterminal
//
// Filename: rtClock_uart.c
//
// Description: This program maintains real-time clock
// and sends time (10 times a second) to the workstation
// through a serial asynchronous link (uart).
// This program displays the system clock in "sssss:tsec" format to HyperTerminal.
// Instructions: Set the following parameters in hyperterminal
// Port : COM1
// Baud rate : 19200
// Data bits: 8
// Parity: None
// Stop bits: 1
// Flow Control: None
//
// Authors: Max Avula (ma0004@uah.edu), A. Milenkovic, milenkovic@computer.org
// UAH - October 2010
// Built with IAR Embedded Workbench IDE Version: 4.20.1.30017
//*****************************************************************************

#include <msp430xG46x.h>
#include <stdio.h>

// Current time variables
unsigned int sec=0; // seconds


unsigned char tsec=0; // 1/10 second
char Time[8]; // string to keep current time


//Function Declarations


void SetTime(void);
```

```
void SendTime(void);

//////////////////////////////////////////////////////////////
// UART0 Initialization                                     //
//////////////////////////////////////////////////////////////

void UART_Initialize(void) // USART in UART Mode
{
  P2SEL |= BIT4+BIT5; // Set UC0TXD and UC0RXD to transmit and receive data
  UCA0CTL1 |= BIT0; // Software reset
  UCA0CTL0 = 0; // USCI_A0 control register
  UCA0CTL1 |= UCSSEL_2; // Clock source SMCLK
  UCA0BR0=54; // 1048576 Hz / 19200
  UCA0BR1=0; //
  UCA0MCTL=0x0A; // Modulation
  UCA0CTL1 &= ~BIT0; // Software reset
}

// Sets the real-time clock variables
void SetTime(void)
{
  tsec++;
  if(tsec == 10)
  {
    tsec = 0;
    sec++;
    P5OUT^=BIT1;
  }
}

// Sends the time to the HyperTerminal
void SendTime(void)
{
  sprintf(Time, "%05d:%01d", sec, tsec); // prints time to a string
  for(int i=0;i<8;i++) // Send character by character
  {
    while (!(IFG2 & UCA0TXIFG));
    UCA0TXBUF = Time[i];
  }
  while (!(IFG2 & UCA0TXIFG));
  UCA0TXBUF = 0x0D; // Carriage Return
}
```

```
//////////////////////////////////////////////////////////////
//                          MAIN                            //
//////////////////////////////////////////////////////////////
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  UART_Initialize(); //Initialize UART
  /* Initialize Timer A to measure 1/10 sec */
  TACTL = TASSEL_2 + MC_1+ ID_3; // Select smclk/8 and up mode
  TACCR0 = 13107; // 100ms interval
  TACCTL0 = CCIE; // Capture/compare interrupt enable

  P5DIR |=0x02;
  while(1) // Infinite loop
  {
    _BIS_SR(LPM0_bits + GIE); // Enter LPM0 w/ interrupts
    SendTime(); // Send Time to HyperTerminal
  }
}

// Interrupt for the timer
#pragma vector=TIMERA0_VECTOR
__interrupt void TIMERA_ISA(void)
{
  SetTime(); // Set Clock
  _BIC_SR_IRQ(LPM0_bits); // Clear LPM0 bits from 0(SR)
}
```

Figure 2. Real time clock using the HyperTerminal application

1.3. HyperTerminal versus Serial App

As a final note, it's important to keep in mind how information is being sent through the UART connection. As we begin this lab, we will generally use the HyperTerminal application. The HyperTerminal application can only display ASCII characters. Since the UART communication protocol sends 8-bit chunks of information, the USCI peripheral has buffers that are best suited to sending or receiving 1-byte size data (with the added stop bits, etc.). It is simplest, therefore, to send and receive ASCII characters as they are a convenient 8-bit size. HyperTerminal can only handle character data types. If it receives non-character information, it will be interpreted as characters and gibberish will appear on the screen.

However, we do not always want to send characters – we often times want to send and view data of all different types (ints, floats, etc.). In order to view this type of information, we can use the convenient UAH serial app. This app translates serial packets that are sent to it, and it

can graphically represent the data versus time. Being able to construct packets with the MSP430 and read them with a software application is an important part of communications. Because the UART protocol specifies that data is sent in 1-byte chunks, we have to create a larger structure of information that we'll send. This is called a packet. The packet consists of predetermined bytes that we construct and tell the receiving software application how to interpret. The UAH serial app expects a packet that has a 1 byte header followed by the data followed by an optional checksum. The software must be told how many bytes of information to expect as well as the type and number of data were sending and how it's ordered. To send the data from the MSP430, we first send our header byte followed by our data that has been broken up into 1-byte chunks. The USCI UART buffer will then be fed each byte at a time. It is important in this process to ensure that the packet that you are sending has the same structure that the receiving device is expecting.

1.4. References

In order to understand more about UART communication and the USCI peripheral device, please access the following references:

- Davies Text, pages 493-497 and pages 574-590
- MSP430 User's Guide, Chapter 19, pages 551-586
- UAH Serial App Getting Started Guide on ANGEL