CPE 325: Embedded Systems Laboratory
Laboratory #8 Tutorial
UART Serial Communications

Aleksandar Milenković
Email: milenka@uah.edu
Web: http://www.ece.uah.edu/~milenka

Objective

This tutorial will introduce communication protocols used with MSP430 and other devices. Specifically, it will cover asynchronous serial communication using 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 serial communication with a workstation
Understanding of workstation clients interfacing serial communication ports (putty) and UAH serial communication application

Notes

All previous tutorials are required for successful completion of this lab. Read CPE323 lecture discussing UART Communication.

Contents

1 Serial Communication

An MSP430-based platform can communicate with another system, such as a personal computer, using either the 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 an 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 MSP430F5529 microcontroller and echoes the character back to the PC (Figure 1). Since we cannot connect the two systems (our PC and the microcontroller) to the same clock source, we should use the UART mode. The USCI peripheral can be utilized for that purpose. The communication speed is 115,200 bits/s (one-bit period is thus 1/115,200 or ~8.68 us). The USCI clock, UCLK, is connected to SMCLK running at 1,048,576 Hz. To achieve the baud rate of 115,200 bits per second, the internal divider registers are initialized to UCA0BR0=0x09, and UCABR1=0x00, because 1,048,576/115,200 = 9.1 ~ 9. Additionally, the modulation register, UCA0MCTL, is set to 0x02 (bit 0 of the field UCBRSx is set to 1). See the reference manual (Table 36.4) for more common combinations of clock source and baud rate and values for baud rate control registers. Also, the reference manual gives details on how the right value in UCA0MCTL is determined (they idea is to continuously minimize probability of erroneous detection at the receiver side).

Figure 1 shows an implementation using polling. The function UART_setup() is configuring USCI in UART mode: 8-bit characters, no parity, 1 stop bit, and the baud rate is set as described above. Please note that we follow recommended sequence of steps for USCI initialization – the SWRST bit in the control register remains set during initialization and it is cleared once he initialization is over. The main program loop is an infinite loop where we use polling to detect whether a new character is received. The program is waiting in line 59 for new character to be received. When a character is received in the UCA0RXBUF register, the UCA0RXIFG bit is set. Before the character is echoed back through the serial interface, we first check whether the USCI's transmit data buffer is empty (line 61). When the transmit buffer is empty, we proceed with copying the received character that is in UCA0RXBUF into UCA0TXBUF. The LED1 is toggled before we go back to the main loop.

```c
/*------------------------------------------------------------------------------
 * File:          Lab8_D1.c
 *
 * Function:      Echo a received character, using polling.
 *
 * Description:   This program echos the character received from UART back to UART.
 *                Toggle LED1 with every received character.
 *                Baud rate: low-frequency (UCOS16=0);
 *                1048576/115200 = ~9.1 (0x0009|0x01)
 *
 * Clocks:        ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO
 *
 * Board:         MSP-EXP430F5529
 *
 * Instructions: Set the following parameters in putty
 * Port: COMx
 * Baud rate: 115200
 * Data bits: 8
 * Parity: None
 * Stop bits: 1
 * Flow Control: None
 *
 * Note:       If you are using Adafruit USBtoTTL cable, look for COM port
 *             in the Windows Device Manager with the following text:
 *             Silicon Labs CP210x USB to UART Bridge (COM<x>).
 *             Connecting Adafruit USB to TTL:
 *              GND - black wire - connect to the GND pin (on the board or
BoosterPack)
 *              Vcc - red wire - leave disconnected
 *              Rx     white wire (receive into USB, connect on TxD of the board P3.3)
 *              Tx -   green wire (transmit from USB, connect to RxD of the board
P3.4)
 *        MSP430F5529
 *     -----------------
 * /|\ |            XIN|-
 *  |  |               | 32kHz
 *  |--|RST        XOUT|-
 *     |               |
 *     |    P3.3/UCA0TXD|------------>
 *     |               | 115200 - 8N1
 *     |    P3.4/UCA0RXD|<------------
 *     |           P1.0|----> LED1
 *
 * Input:     None (Type characters in putty/MobaXterm/hyperterminal)
 * Output:    Character echoed at UART
 * Author:    A. Milenkovic, milenkovic@computer.org
 * Date:      October 2018, modified August 2020
 *--------------------------------------------------------------------------------*/
#include <msp430.h>

```

```
void UART_setup(void) {

    P3SEL |= BIT3 + BIT4;   // Set USCI_A0 RXD/TXD to receive/transmit data
    UCA0CTL1 |= UCSWRST;    // Set software reset during initialization
    UCA0CTL0 = 0;           // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;   // Clock source SMCLK

    UCA0BR0 = 0x09;         // 1048576 Hz  / 115200 lower byte
    UCA0BR1 = 0x00;         // upper byte
    UCA0MCTL |= UCBRS0;     // Modulation (UCBRS0=0x01, UCOS16=0)

    UCA0CTL1 &= ~UCSWRST;   // Clear software reset to initialize USCI state machine
}

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;       // Stop WDT
    P1DIR |= BIT0;                  // Set P1.0 to be output
    UART_setup();                   // Initialize UART

    while (1) {
       while(!(UCA0IFG&UCRXIFG));   // Wait for a new character
       // New character is here in UCA0RXBUF
       while(!(UCA0IFG&UCTXIFG));   // Wait until TXBUF is free
       UCA0TXBUF = UCA0RXBUF;       // TXBUF <= RXBUF (echo)
       P1OUT ^= BIT0;               // Toggle LED1
    }
}

```

Figure 1. Echoing a Character Using the USCI in UART Mode and Polling

Figure 2 shows the program that performs the same task, but this time an interrupt service routine tied to the USCI receiver is used. In the main program the USCI is configured to generate an interrupt request when a new character is received. Whenever a character is received and loaded into UCA0RXBUF, the interrupt flag UCA0RXIFG is set and interrupt request is raised. The main program does nothing beyond initialization – the processor is in a low-power mode 0 (LPM0). What clock signals are down in this mode?

All actions in this implementation occurs inside the service routine. The processor wakes up when a new character is received and we find ourselves inside the service routine. In the ISR before writing the new character to UCA0TXBUF to transmit to back to the workstation, we need to make sure that it is indeed empty to avoid loss of data. The UCA0TXIFG interrupt flag is set by the transmitter when the UCA0TXBUF is ready to accept a new character. Note: here we do polling on transmit buffer inside the receiver ISR. When the UCA0TXBUF is ready (UCA0TXIFG flag is set), the content from UCA0RXBUF is copied into the UCA0TXBUF. The LED4 is toggled. When exiting the ISR, the original PC and SR are retrieved bringing the processor back in the LPM0.

```
/*------------------------------------------------------------------------------
 * File:          Lab8_D2.c
 *
```

```c
 * Function:      Echo a received character, using receiver ISR.
 * Description:   This program echos the character received from UART back to UART.
 *                Toggle LED1 with every received character.
 *                Baud rate: low-frequency (UCOS16=0);
 *                1048576/115200 = ~9.1 (0x0009|0x01)
 * Clocks:        ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO
 *
 * Instructions: Set the following parameters in putty
 * Port: COM1
 * Baud rate: 115200
 * Data bits: 8
 * Parity: None
 * Stop bits: 1
 * Flow Control: None
 *
 *        MSP430f5529
 *      -----------------
 * /|\ |            XIN|-
 *  |  |               | 32kHz
 *  |--|RST        XOUT|-
 *     |               |
 *     |   P3.3/UCA0TXD|------------>
 *     |               | 115200 - 8N1
 *     |   P3.4/UCA0RXD|<------------
 *     |           P1.0|----> LED1
 *
 * Input:     None (Type characters in putty/MobaXterm/hyperterminal)
 * Output:    Character echoed at UART
 * Author:    A. Milenkovic, milenkovic@computer.org
 * Date:      October 2018
 *--------------------------------------------------------------------------------*/
#include <msp430.h>

// Initialize USCI_A0 module to UART mode
void UART_setup(void) {

    P3SEL |= BIT3 + BIT4;   // Set USCI_A0 RXD/TXD to receive/transmit data
    UCA0CTL1 |= UCSWRST;    // Set software reset during initialization
    UCA0CTL0 = 0;           // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;   // Clock source SMCLK

    UCA0BR0 = 0x09;         // 1048576 Hz  / 115200 lower byte
    UCA0BR1 = 0x00;         // upper byte
    UCA0MCTL |= UCBRS0;     // Modulation (UCBRS0=0x01, UCOS16=0)

    UCA0CTL1 &= ~UCSWRST;   // Clear software reset to initialize USCI state machine
    UCA0IE |= UCRXIE;       // Enable USCI_A0 RX interrupt
}

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;// Stop WDT
    P1DIR |= BIT0;           // Set P1.0 to be output
    UART_setup();            // InitiAlize USCI_A0 in UART mode

    _BIS_SR(LPM0_bits + GIE); // Enter LPM0, interrupts enabled
```

```c
}

// Echo back RXed character, confirm TX buffer is ready first
#pragma vector = USCI_A0_VECTOR
__interrupt void USCIA0RX_ISR (void) {
    while(!(UCA0IFG&UCTXIFG));  // Wait until can transmit
    UCA0TXBUF = UCA0RXBUF;      // TXBUF <-- RXBUF
    P1OUT ^= BIT0;              // Toggle LED1
}

```

Figure 2. Echoing a Character Using the USCI Device

2 Real-Time Clock

In this section we will describe a program that implements a real-time clock on the MSP430 platform (Figure 3). The time is measured from the beginning of the application with a resolution of 100 milliseconds (one tenth of a second). The time is maintained in two variables, unsigned int sec (for seconds) and unsigned char tsec for tenths 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 MSP EXP430F5529LP schematic) that connects through a serial cable to a PC. On the PC side we can open putty application and observe real-time clock that is sent from our development platform.

The first step is to initialize the USCI device in UART mode for communication using a baud rate 9,600 bits/sec. The next step is to initialize Timer_A to measure time and update the real-time clock variables. The Timer_A ISR is used to maintain the 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.

```c
/*------------------------------------------------------------------------------
/*------------------------------------------------------------------------------
 * File:          Lab8_D3.c
 * Function:      Displays real-time clock in serial communication client.
 * Description:   This program maintains real-time clock and sends time
 *                (10 times a second) to the workstation through
 *                a serial asynchronous link (UART).
 *                The time is displayed as follows: "sssss:tsec".
 *
 *                Baud rate divider with 1048576hz = 1048576/(16*9600) = ~6.8 [16
from UCOS16]
 * Clocks:        ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO = 1048576Hz
 * Instructions:  Set the following parameters in putty/hyperterminal
 * Port: COMx
 * Baud rate: 19200
 * Data bits: 8
 * Parity: None
 * Stop bits: 1
 * Flow Control: None
```

```
 *
 *        MSP430F5529
 *     -----------------
 * /|\ |            XIN|-
 *  |  |               | 32kHz
 *  |--|RST        XOUT|-
 *     |               |
 *     |   P3.3/UCA0TXD|------------>
 *     |               | 9600 - 8N1
 *     |   P3.4/UCA0RXD|<------------
 *     |           P1.0|----> LED1
 *
 * Author:      A. Milenkovic, milenkovic@computer.org
 * Date:        October 2018
--------------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>

// Current time variables
unsigned int sec = 0;              // Seconds
unsigned int tsec = 0;             // 1/10 second
char Time[8];                      // String to keep current time

void UART_setup(void) {
    P3SEL  = BIT3+BIT4;                         // P3.4,5 = USCI_A0 TXD/RXD
    UCA0CTL1 |= UCSWRST;                        // **Put state machine in reset**
    UCA0CTL1 |= UCSSEL_2;                       // SMCLK
    UCA0BR0 = 6;                                // 1MHz 9600 (see User's Guide)
    UCA0BR1 = 0;                                // 1MHz 9600
    UCA0MCTL = UCBRS_0 + UCBRF_13 + UCOS16;     // Mod. UCBRSx=0, UCBRFx=0,
                                                // over sampling
    UCA0CTL1 &= ~UCSWRST;                       // **Initialize USCI state machine**
}

void TimerA_setup(void) {
    TA0CTL = TASSEL_2 + MC_1 + ID_3; // Select SMCLK/8 and up mode
    TA0CCR0 = 13107;                 // 100ms interval
    TA0CCTL0 = CCIE;                 // Capture/compare interrupt enable
}

void UART_putCharacter(char c) {
    while (!(UCA0IFG&UCTXIFG));    // Wait for previous character to transmit
    UCA0TXBUF = c;                  // Put character into tx buffer
}

void SetTime(void) {
    tsec++;
    if (tsec == 10){
        tsec = 0;
        sec++;
        P1OUT ^= BIT0;              // Toggle LED1
    }
}

void SendTime(void) {
```

```
    int i;
    sprintf(Time, "%05d:%01d", sec, tsec);// Prints time to a string

    for (i = 0; i < sizeof(Time); i++) {  // Send character by character
        UART_putCharacter(Time[i]);
    }
    UART_putCharacter('\r');        // Carriage Return
}

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
    UART_setup();                   // Initialize UART
    TimerA_setup();                 // Initialize Timer_B
    P1DIR |= BIT0;                  // P1.0 is output;

    while (1) {
        _BIS_SR(LPM0_bits + GIE);   // Enter LPM0 w/ interrupts
        SendTime();                 // Send Time to HyperTerminal/putty
    }
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void TIMERA_ISA(void) {
    SetTime();                       // Update time
    _BIC_SR_IRQ(LPM0_bits);          // Clear LPM0 bits from 0(SR)
}

```

Figure 3. Display Real-Time Clock Through UART

Please note that sprintf with modifiers requires full printf support. This should have been already set by you when creating the project. If you did not, it is under MSP430 Compiler->Advanced Options->Language Options as shown in Figure 4.

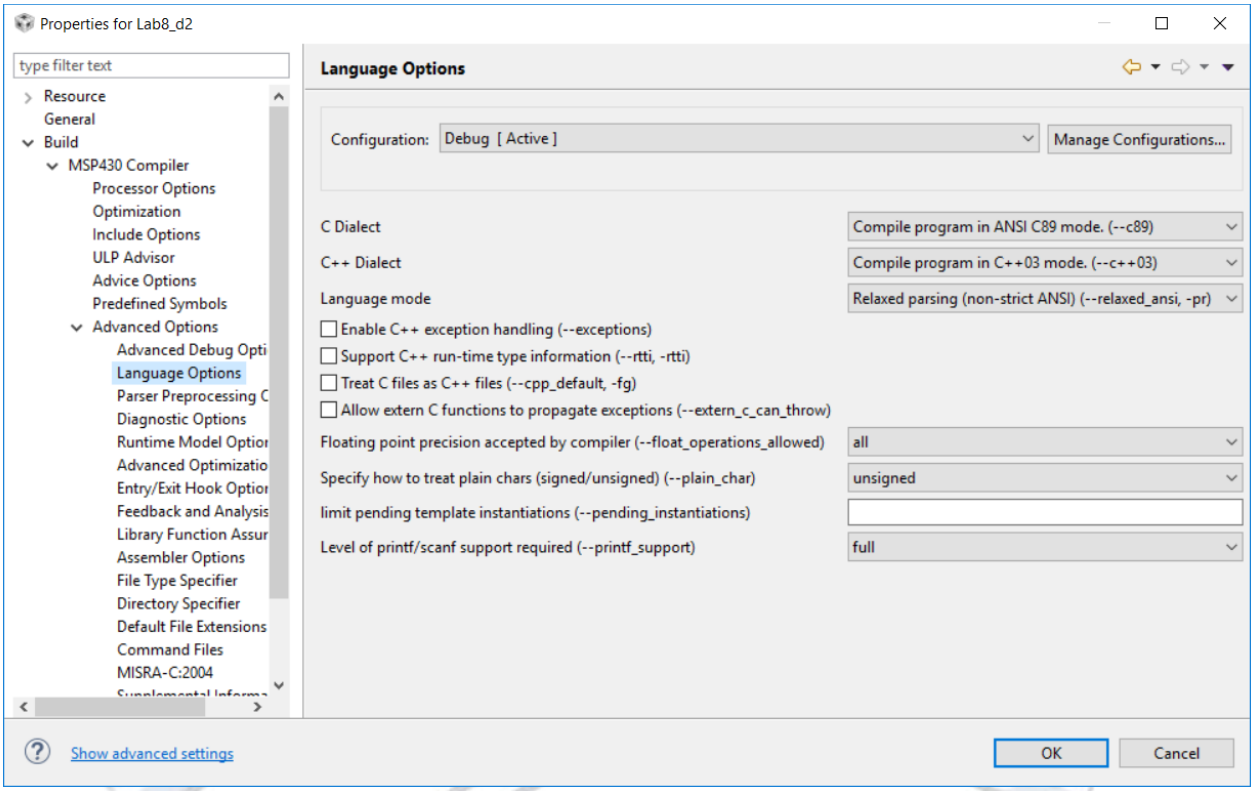

Figure 4. Setting Code Composer to Support sprintf

3 Putty 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 Putty application (available for download at http://portal.mhealth.uah.edu/public/index.php/serial-port-application). The Putty 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. Putty 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 want to send and view data of different types (ints, floats, etc.). To view this type of information, we can use the convenient UAH Serial Application developed by our former student Mladen Milosevic. This application 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 communication.

Because the UART protocol specifies that data is sent in 1-byte chunks, we must 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 Application 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 was 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.

Figure 5 shows a demo program for sending a floating-point variable through UART. The 4-byte float variable is sent in a 5-byte packet: header (1-byte) and 4-byte data (LSB byte is sent first). The variable is increased by 0.1 every second with modulus 10.0 and reported through UART as shown in the WDTISR.

```
/*------------------------------------------------------------------------------
 * File:        Demo8_D4.c
 * Function:    Send floating data to Serial port
 * Description: UAH serial app expects lower byte first so send each byte at a
 *              time sending Lowest byte first
 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO
 *
 * Instructions: Set the following parameters in putty
 * Port: COMx
 * Baud rate: 115200
 * Data bits: 8
 * Parity: None
 * Stop bits: 1
 * Flow Control: None
 *
 *        MSP430f5529
 *     -----------------
 * /|\ |             XIN|-
 *  |  |                | 32kHz
 *  |--|RST         XOUT|-
 *     |                |
 *     |    P3.3/UCA0TXD|------------>
 *     |                | 115200 - 8N1
 *     |    P3.4/UCA0RXD|<------------
 *     |                |
 * Input:     None
 * Output:    Ramp signal in UAH Serial app
 * Author:    Prawar Poudel
 * Date:      October 2018
 *------------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdint.h>

volatile float myData;

void UART_setup(void) {

    P3SEL |= BIT3 + BIT4;    // Set USCI_A0 RXD/TXD to receive/transmit data
    UCA0CTL1 |= UCSWRST;     // Set software reset during initialization
```

```
    UCA0CTL0 = 0;           // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;   // Clock source SMCLK

    UCA0BR0 = 0x09;         // 1048576 Hz  / 115200 lower byte
    UCA0BR1 = 0x00;         // upper byte
    UCA0MCTL = 0x02;        // Modulation (UCBRS0=0x01, UCOS16=0)

    UCA0CTL1 &= ~UCSWRST;   // Clear software reset to initialize USCI state machine
}

void UART_putCharacter(char c) {
    while (!(UCA0IFG&UCTXIFG)); // Wait for previous character to transmit
    UCA0TXBUF = c;              // Put character into tx buffer
}

int main() {
    WDTCTL = WDT_ADLY_1000;
    UART_setup();               // Initialize USCI_A0 module in UART mode
    SFRIE1 |= WDTIE;            // Enable watchdog interrupts

    myData = 0.0;
    __bis_SR_register(LPM0_bits + GIE);
}

// Sends a ramp signal; amplitude of one period ranges from 0.0 to 9.9
#pragma vector = WDT_VECTOR
__interrupt void watchdog_timer(void) {
    char index = 0;
    // Use character pointers to send one byte at a time
    char *myPointer = (char* )&myData;

    UART_putCharacter(0x55);                // Send header
    for(index = 0; index < 4; index++) {    // Send 4-bytes of myData
        UART_putCharacter(myPointer[index]);
    }

    // Update myData for next transmission
    myData = (myData + 0.1);
    if(myData >= 10.0) {
        myData = 0.0;
    }
}
```

Figure 5. MSP430 Program for Sending Floating-Point Data (UAH Serial App)

Figure 6 shows how to properly configure UAH Serial App for viewing the RAMP signal. We are using a single channel, the size of the packet is 5 bytes, we are plotting one sample at a time (they are arriving rather slowly in this example). Figure 7 shows the RAMP signal in the UAH Serial App sent by the program from Figure 5.

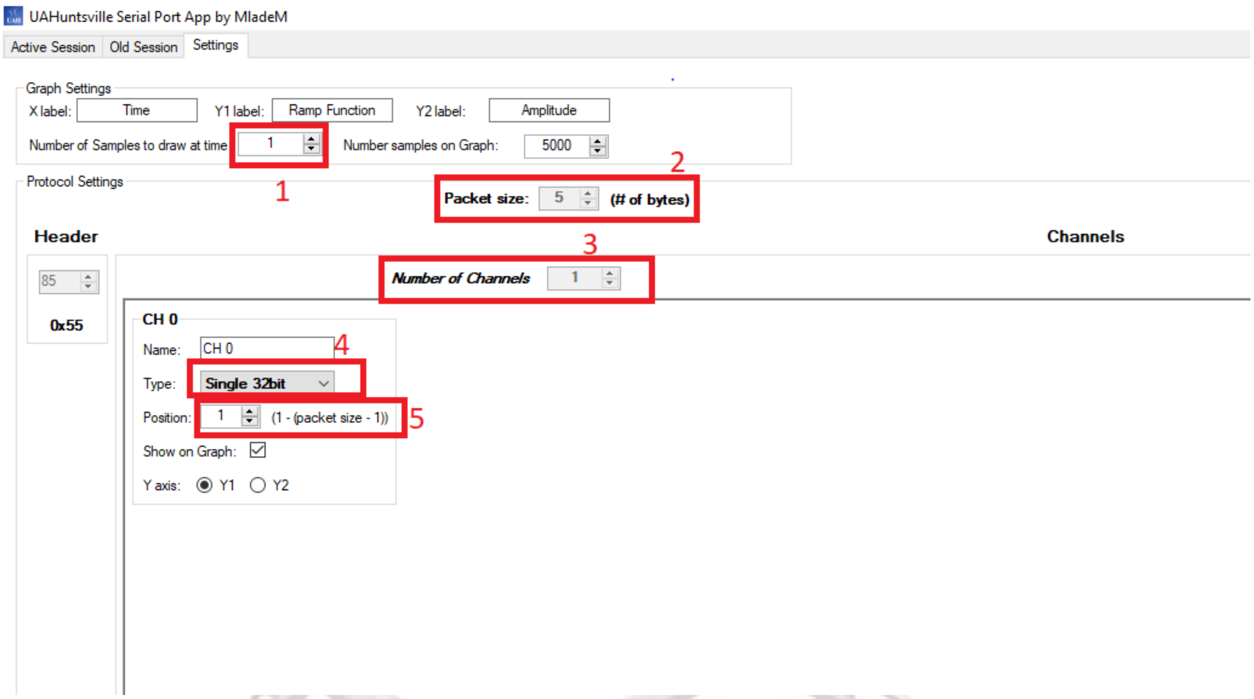

Figure 6. Configuring UAH Serial App for Viewing Ramp Signal

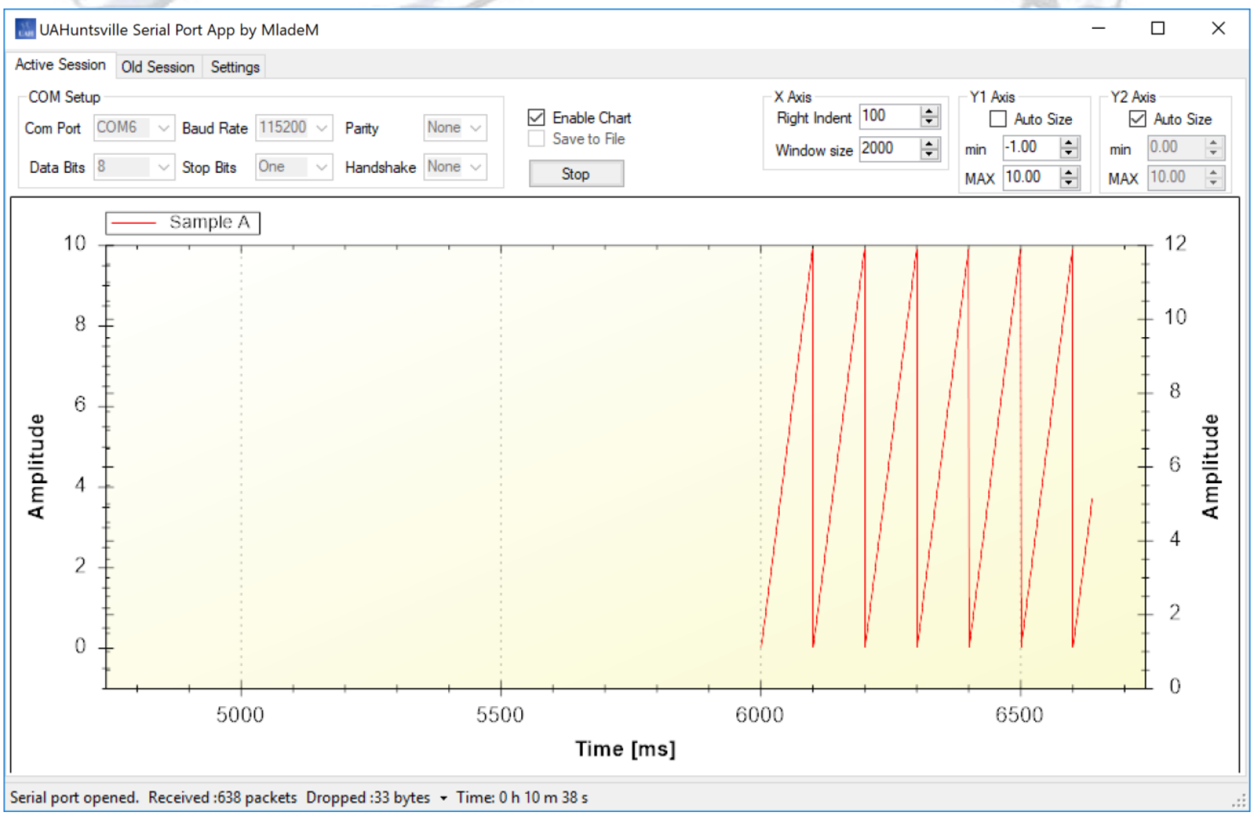

Figure 7. The RAMP signal in UAH Serial App

4 References

To understand more about UART communication and the USCI peripheral device, please read the following references:

- Davies' *MSP430 Microcontroller Basics*, pages 493 – 497 and pages 574 – 590
- MSP430 User's Guide, Chapter 36, pages 937– 966