UART Temp Readings Using MSP 430 Launchpad

Seventh Laboratory Report for CENG 3331

Submitted by

Michael Lanford

1816337

Computer Engineering

University of Houston-Clear Lake

Houston, Texas 77058

10/30/22

Abstract

The goal of this lab was to introduce us to UART communication by using the MSP 430 Launchpad. During this lab, we answered some questions related to UART, which were given to us in the lab manual, and also used the MSP 430 Launchpad to look at the temperature.

Write-Up

Introduction

UART is short for Universal Asynchronous Receiver/Transmitter and is used to transmit bytes of data as individual bits in a sequential fashion. When they reach their destination, another UART will reassemble the bits into the message.

Task 1:

We began our experiment by answering some questions related to UART and the previous lab. For the first question, UART is an asynchronous serial communication with configurable data format and transmission speeds. The difference between UART and I2C is I2C uses the master/slave configuration along with clock signals while UART implements asynchronous serial data streams with no clock signal. For the second question, the difference between software UART and hardware UART is software UART needs code to communicate with other devices, which results in a buffer, compared to hardware UART which always has the communication built into the circuit. Besides this, software UART is extremely flexible compared to hardware UART.

Task 2:

For task 2, we continued the experiment by first setting up the HW UART. We then created a project and copied over code that we were given in the lab manual for the program we would use. We also finished off the code by writing in 2 lines that completed the program. A screenshot of this along with the debug window can be seen under Figure 1 of the appendix.

Task 3:

For task 3, we loaded code given to us in the lab manual into a new project and debugged it to make sure it worked. We then set a breakpoint and looked at the temperature given. The result of this can be seen in the appendix under Figure 2.

Task 4:

For this final task, we combined everything we had learned from tasks 2 and 3 to make a program that could read temperature and also send it to the computer. We created a new project and put the given code from the lab manual into the project, then filled in some missing code to get it to work. The figures of this can be seen in the appendix from Figure 3 to Figure 6. Finally, we ran and tested the program.

Appendix

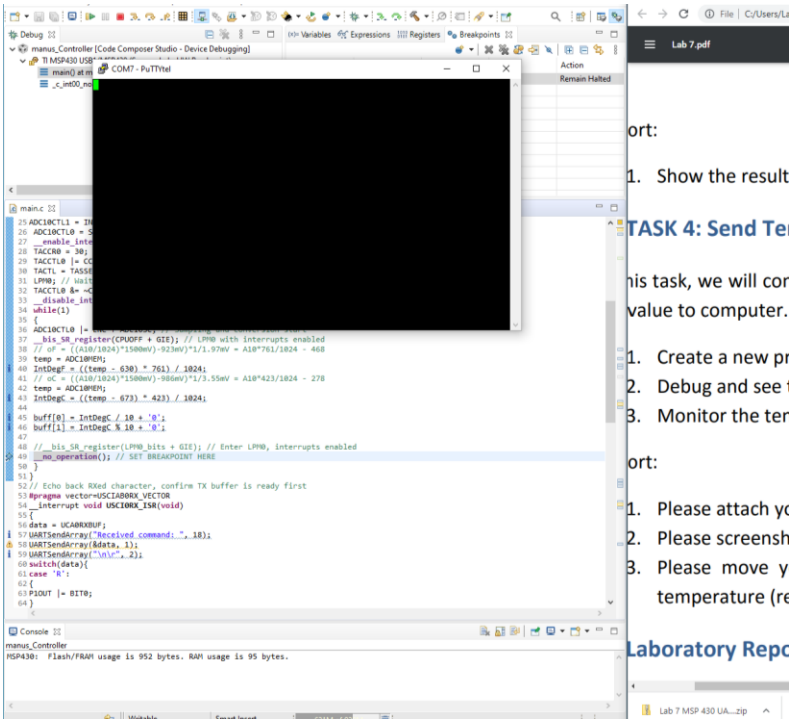

Fig. 1. PuTTYtel Screenshot of Code

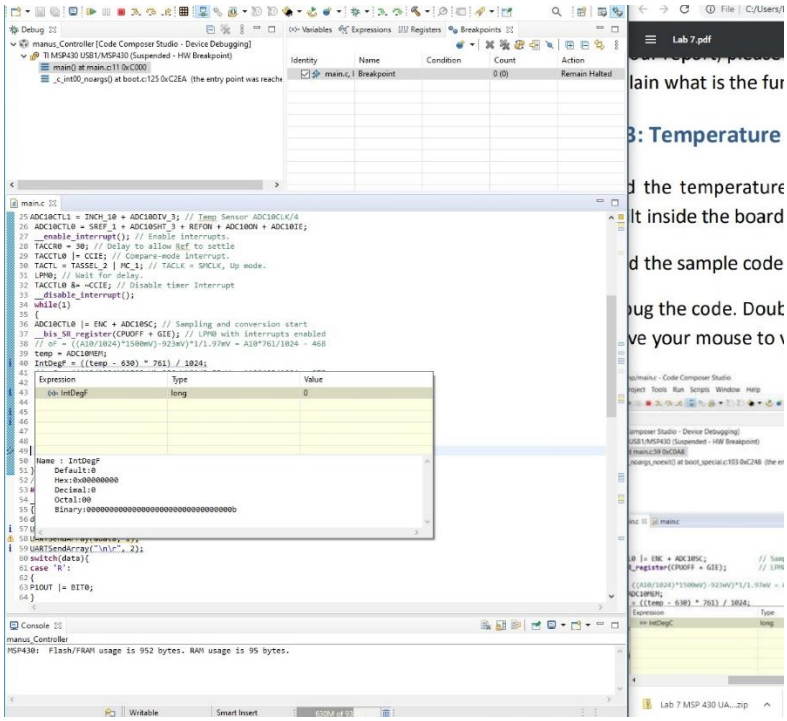

Fig. 2. Temperature Sensor

```
#include <msp430g2553.h>;
void UARTSendArray(unsigned char *TxArray, unsigned char ArrayLength);
static volatile char data;
long temp;
long IntDegF;
long IntDegC;
unsigned char buff[2];

void main(void)
{
WDTCTL = WDTPW + WDTHOLD; // Stop WDT
P1DIR |= BIT0 + BIT6; // Set the LEDs on P1.0, P1.6 as outputs
 P1OUT = BIT0; // Set P1.0
 BCSCTL1 = CALBC1_1MHZ; // Set DCO to 1MHz
 DCOCTL = CALDCO_1MHZ; // Set DCO to 1MHz
 /* Configure hardware UART */
 P1SEL = BIT1 + BIT2 ; // P1.1 = RXD, P1.2=TXD
 P1SEL2 = BIT1 + BIT2 ; // P1.1 = RXD, P1.2=TXD
 UCA0CTL1 |= UCSSEL_2; // Use SMCLK
 UCA0BR0 = 104; // Set baud rate to 9600 with 1MHz clock (Data Sheet 15.3.13)
 UCA0BR1 = 0; // Set baud rate to 9600 with 1MHz clock
 UCA0MCTL = UCBRS0; // Modulation UCBRSx = 1
 UCA0CTL1 &= ~UCSWRST; // Initialize USCI state machine
 IE2 |= UCA0RXIE; // Enable USCI_A0 RX interrupt
ADC10CTL1 = INCH_10 + ADC10DIV_3; // Temp Sensor ADC10CLK/4
 ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE;
 __enable_interrupt(); // Enable interrupts.
 TACCR0 = 30; // Delay to allow Ref to settle
 TACCTL0 |= CCIE; // Compare-mode interrupt.
 TACTL = TASSEL_2 | MC_1; // TACLK = SMCLK, Up mode.
 LPM0; // Wait for delay.
 TACCTL0 &= ~CCIE; // Disable timer Interrupt
 __disable_interrupt();
```

Fig. 3. Code Part 1

```
 while(1)
 {
 ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
 __bis_SR_register(CPUOFF + GIE); // LPM0 with interrupts enabled
 // oF = ((A10/1024)*1500mV)-923mV)*1/1.97mV = A10*761/1024 - 468
 temp = ADC10MEM;
 IntDegF = ((temp - 630) * 761) / 1024;
 // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
 temp = ADC10MEM;
 IntDegC = ((temp - 673) * 423) / 1024;

 buff[0] = IntDegC / 10 + '0';
 buff[1] = IntDegC % 10 + '0';

 //__bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled
 __no_operation(); // SET BREAKPOINT HERE
 }
}
// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
data = UCA0RXBUF;
UARTSendArray("Received command: ", 18);
UARTSendArray(&data, 1);
UARTSendArray("\n\r", 2);
switch(data){
case 'R':
{
P1OUT |= BIT0;
}
break;
case 'r':
{
P1OUT &= ~BIT0;
}
break;
```

Fig. 4. Code Part 2

```
case 'G':
{
P1OUT |= BIT6;
}
break;
case 'g':
{
P1OUT &= ~BIT6;
}
break;
case 'T':
 {
 UARTSendArray("Temperature is: ", 16);
 UARTSendArray(buff, 2); // Note two digits at least.
 UARTSendArray("\n\r", 2);
 }
 break;
default:
{
UARTSendArray("Unknown Command: ", 17);
UARTSendArray(&data, 1);
UARTSendArray("\n\r", 2);
}
break;
}
}
void UARTSendArray(unsigned char *TxArray, unsigned char ArrayLength){
// Send number of bytes Specified in ArrayLength in the array at using the hardware UART 0
// Example usage: UARTSendArray("Hello", 5);
// int data[2]={1023, 235};
// UARTSendArray(data, 4); // Note because the UART transmits bytes it is necessary to send two
//bytes for each integer hence the data length is twice the array length
while(ArrayLength--){ // Loop until StringLength == 0 and post decrement
while(!(IFG2 & UCA0TXIFG)); // Wait for TX buffer to be ready for new data
UCA0TXBUF = *TxArray; //Write the character at the location specified py the pointer
TxArray++; //Increment the TxString pointer to point to the next character
```

Fig. 5. Code Part 3

```
}
}
// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
  __bic_SR_register_on_exit(CPUOFF); // Clear CPUOFF bit from 0(SR)
}
#pragma vector=TIMER0_A0_VECTOR
__interrupt void ta0_isr(void)
{
 TACTL = 0;
 LPM0_EXIT; // Exit LPM0 on return
}
```

Fig. 6. Code Part 4

Conclusion

In conclusion, this lab was a perfect introduction to UART communication by way of the MSP 430 Launchpad. We were able to successfully get our temperature reader to work as intended and return the room temperature on the computer.