CPE 325: Intro to Embedded Computer System

Lab08

Serial Communication and UART

Submitted by: Gianna Foti

Date of Experiment:_____30 October 2024_____

Report Deadline:_____31 October 2024_____

Lab Section: _____CPE353-02_____

Demonstration Deadline: _____31 October 2024_____

Introduction

In this lab, I programmed the MSP430F5529 microcontroller to interface with the ADXL335 accelerometer, sampling x, y, and z axis data at 10 Hz and displaying the values as separate lines in the UAH serial app. Additionally, I implemented a crash detection feature: when the total acceleration magnitude reaches 2g, the RED LED turns on to signal potential airbag deployment.The lab also involved generating waveforms with digital-to-analog conversion. At 30 Hz, the program outputs a sine wave by default, a saw-tooth wave when SW1 is pressed, and triples the frequency with SW2. This assignment enhanced my skills in sensor data acquisition, event-based signaling, and waveform generation on embedded systems.

Theory Topics

1. Serial Communication and UART
    a. Serial communication is a method of transferring data between two devices one bit at a time over a single communication line. The MSP430 microcontroller supports both synchronous and asynchronous communication. In this lab, asynchronous communication is used, specifically through UART (Universal Asynchronous Receiver-Transmitter).
        i. Baud Rate Calculation: The clock (1,048,576 Hz) is divided by the baud rate, with the modulation register used to adjust for non-integer values to prevent errors.
        ii. Implementation: The lab explores polling-based and interrupt-based communication. Interrupts allow the MSP430 to enter low-power mode until data is received, improving energy efficiency.
2. UAH Serial App
    a. The UAH Serial App interprets non-ASCII data (e.g., floats or integers) and visualizes it over time, unlike Putty, which only handles character data.

b. Packet Structure: Data is sent in packets with a header byte and data bytes (e.g., 4-byte float). This ensures the app can correctly display complex data, like the triangular waveform generated in the lab.
c. Usage: Proper configuration of packet size and sample rate ensures the app interprets the transmitted data correctly, making it ideal for real-time signal monitoring.

Results & Observation

Part 1:

Program Description:

This program implements an interactive chatbot called "Market Bot" on an MSP430 microcontroller using UART communication at 115,200 baud. It allows users to place orders for items like eggs, chicken, and milk. The chatbot prompts the user for their name, takes orders, calculates the total price, and thanks the user. Users can order multiple items, and invalid inputs trigger re-prompts. For the bonus, ANSI colors are used for text formatting: red for the bot and purple for the user. It also allows for backspacing. The program handles UART setup, sending/receiving data, and basic input validation.

Process:

1. First, I set up UART communication:
    a. I configured the RX (P3.4) and TX (P3.3) pins and initialized UART to run at 115,200 baud for smooth data exchange.
2. Next, I added ANSI colors for fun formatting for the bonus:
    a. I used escape codes to make Market Bot's messages red and user input purple, giving the interaction more personality and fulfilling a part of the bonus. This was for the bonus.
3. Then, I built the main chat loop:
    a. The bot asks for the user's name, greets them, and lists available items (eggs, chicken, and milk) for selection.

4. Next, I handled item selection and pricing:
    a. Using a ternary operator, I assigned prices and validated inputs to make sure only valid items are chosen. If the input was not valid, it gave an error message and listed the valid options again.
5. Then, I added quantity input and total calculation:
    a. The bot asks how many units the user wants, converts the input to an integer, and keeps a running total.
6. Finally, I made sure multiple orders were possible:
    a. After each selection, the bot asks if the user wants more items.
        i. If they select no, then it shows the total price and thanks them.
            1. Then loops back to username and welcome message.
        ii. If they select yes, it loops back to selection of valid items.

Flowchart:

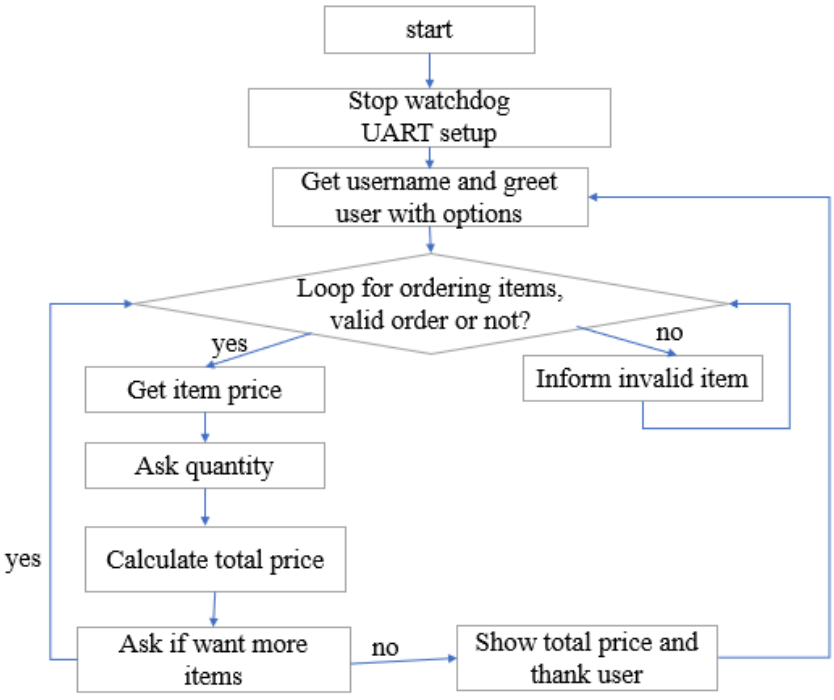

Figure 1: Program 1 Flowchart

Program Output:

```
[Market Bot]:
Hi, I am Market Bot. What is the name for your order?
[User]: Gianna
[Market Bot]: Hello Gianna! Today we have eggs, chicken, and milk. What would you like to order?
[User]: eggs
[Market Bot]: eggs costs $2. How many would you like?
[User]: 2
[Market Bot]: Do you want to order more (yes/no)?
[User]: yes
[Market Bot]: Today we have eggs, chicken, and milk. What would you like to order?
[User]: chicken
[Market Bot]: chicken costs $1. How many would you like?
[User]: 1
[Market Bot]: Do you want to order more (yes/no)?
[User]: yes
[Market Bot]: Today we have eggs, chicken, and milk. What would you like to order?
[User]: milk
[Market Bot]: milk costs $2. How many would you like?
[User]: 1
[Market Bot]: Do you want to order more (yes/no)?
[User]: no
[Market Bot]: The total for all items is $7. Thank you for shopping with me, Gianna!

[Market Bot]:
Hi, I am Market Bot. What is the name for your order?
[User]:
```

Figure 2: Program 1 Output

Part 2:

Program 2 Description:

This program generates a triangular waveform on an MSP430 microcontroller and transmits it via UART at 57,600 baud. The waveform has an amplitude of 8 units and a frequency of 2.5 Hz, incrementing and decrementing by 0.04 units per step. It completes 4 full periods (8 cycles) before stopping.

Process:

1. Define Requirements and Set Up:
    a. First, I defined the purpose of my program: to generate a triangular waveform with an amplitude of 8 units and a frequency of 2.5 Hz, then transmit the data via UART. Next, I set up my development environment for the MSP430 microcontroller, ensuring all necessary tools and hardware connections were in place.
2. Initialize Variables and Functions:

    a. I started by declaring global variables to hold important values: myData for the waveform, i to count the cycles, and peak to track the waveform's rising and falling states. Then, I implemented the UART_setup() function to configure the UART settings, including pin selection, baud rate, and modulation.

3. Incremental changes in waveform value:
    a. Increases by 0.04 units per step during the rising phase.
    b. Decreases by 0.04 units per step during the falling phase.
    c. The wave completes 4 full periods (8 cycles total: 4 rising + 4 falling).
4. Implement Character Transmission:
    a. Next, I wrote the sendChar(char c) function to handle character transmission over UART. I made sure it waited for the buffer to be ready before sending any data, ensuring reliable communication.
5. Develop the Main Function:
    a. In the main() function, I initialized the watchdog timer and UART, set the starting waveform value to 0, and entered Low Power Mode to conserve energy while waiting for interrupts.
6. Create the Watchdog Timer ISR:
    a. I then implemented the ISR to manage the watchdog timer interrupts. In this routine, I transmitted a start byte followed by the floating-point waveform data. I updated myData to create the triangular waveform, switched direction at the limits, and counted the completed cycles, disabling interrupts after four cycles as stated in the lab.
7. Testing and Finalization:
    a. Finally, I compiled the code and uploaded it to the MSP430. I connected it to a serial terminal for testing and verified that the waveform was generated and transmitted correctly.

Calculations:

1. T = 1/2.5Hz → 400ms → 200 sample up/down → 8/x = 200 → x = 0.04
    a. Breakdown:
        i. The frequency of the triangular wave is 2.5 Hz, meaning the wave completes 2.5 cycles per second.
        ii. Each half-cycle (either rising or falling) takes 400ms/2 = 200ms
        iii. The peak-to-peak amplitude of the triangular waveform is 8 units (from 0 to 8).
        iv. To smoothly increase from 0 to 8 over 200 samples, we divide the total amplitude by the number of samples: 8/200 = 0.04 units per sample
2. The increment of 0.04 units per sample ensures that the waveform smoothly transitions from 0 to 8 during the rising phase and back from 8 to 0 during the falling phase, completing a full period in 400 ms with precise timing.

Program Output:

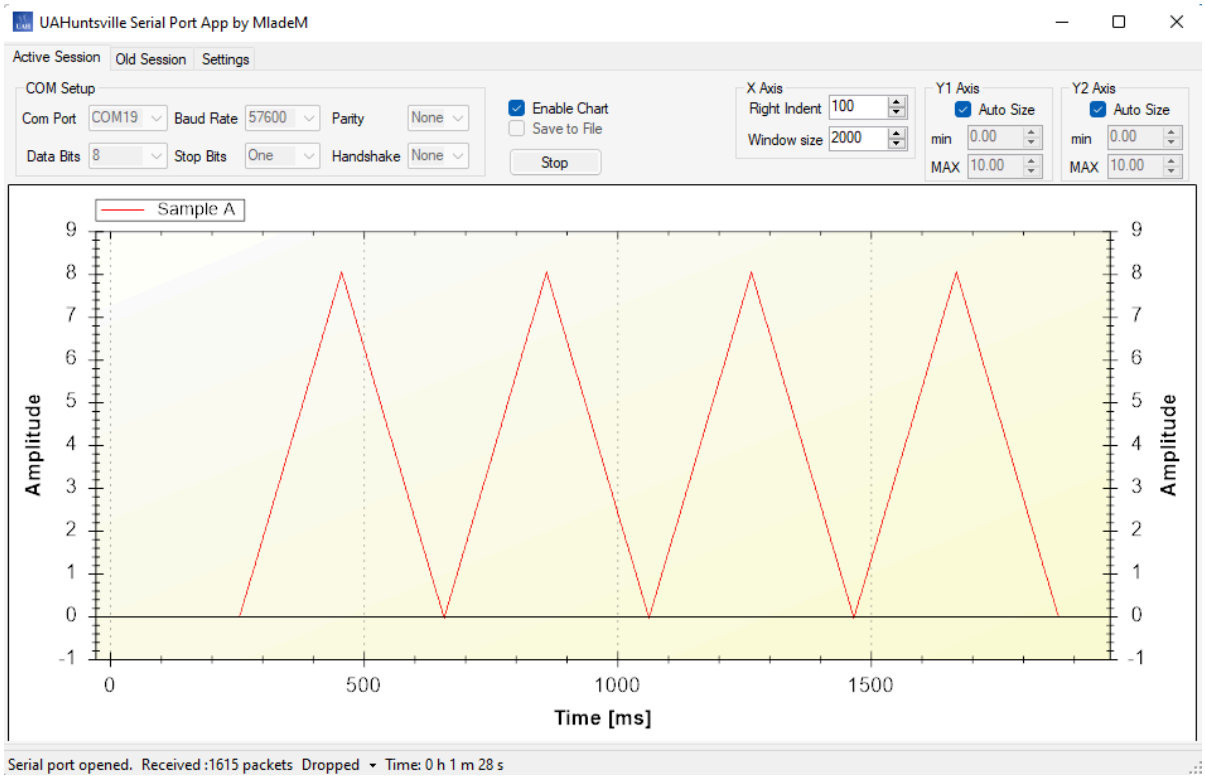

Figure 3: Program 2 Output

Demo Questions:

1. What clock signals are stopped in LPM0?
    a. MCLK is stopped. This allows peripherals, like the USCI module for UART communication, to continue operating while saving power.
2. What is the maximum time you can have on the real-time clock (demo #3)?
    a. In Demo #3, the time is tracked using two variables:
        i. sec (unsigned int) for seconds
        ii. tsec (unsigned char) for tenths of a second
    b. Since sec is a 16-bit unsigned integer, the maximum value is 65,535 seconds.
    c. Additionally, since “tsec” increments for 0 to 9, each second consists of 10 tenths.
    d. Therefore, the maximum time is 65,535 + 0.9 = 65,535.9 seconds

Conclusion

In this lab, I developed two programs on the MSP430 microcontroller: MarketBot and a triangular waveform generator.For Program 1, MarketBot used UART at 115,200 baud to take user orders for eggs, chicken, and milk. It handled input validation, allowed multiple items to be ordered, and displayed the total price at the end. This program provided experience with interactive communication and input handling. I also completed the bonus with different colors for each user as well as a backspacing functionality In Program 2, I generated a triangular waveform with an amplitude of 8 units and a frequency of 2.5 Hz, transmitted via UART at 57,600 baud. Using Watchdog Timer interrupts at 32 ms intervals, the waveform completed 4 full periods, with the system entering Low Power Mode between updates.This lab strengthened my skills in UART communication, timing control, and power-efficient programming. Both programs functioned as required, demonstrating reliable embedded system design. In all, this lab was time consuming but not overly difficult.

Appendix

Table 1: Program 1 with Bonus source code

```
/*-------------------------------------------------------------------------------------------
 * File: Lab08_p1.c
 * Description: Uses a 115,200 baud connection to create an interactive chatbot program
 * Input: User input
 * Output: Chat output
 * Author: Gianna Foti
 *-------------------------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Me doing the bonus colourssss
#define colorReset   "\x1b[0m"
#define colorBot     "\x1b[31m"  // Market Bot in red like example
#define colorUser    "\x1b[35m"  // User in purple

//these are all of my function prototypes
void UART_SETUP();
void UART_sendChar(char bip);
char UART_getChar();
void UART_sendString(char* str);
void UART_getLine(char* buf, int limit);
void main(void) {
   WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog timer
   UART_SETUP();  // Setup UART
   char userName[50];
   char item[10];
   char response[10];
   char message[100];
   int totalPrice = 0;
   while (1) {
      // Ask for User's Name (First Interaction)
      UART_sendString(colorBot "[Market Bot]: " colorReset);
      UART_sendString("\r\nHi, I am Market Bot. What is the name for your order?\r\n");
      UART_sendString(colorUser "[User]: " colorReset);
      UART_getLine(userName, 50);  // Get the user's name
      // Greeting after receiving User's Name
      snprintf(message, sizeof(message),
            "Hello %s! Today we have eggs, chicken, and milk. What would you like to order?\r\n",
            userName);
      UART_sendString(colorBot "[Market Bot]: " colorReset);
      UART_sendString(message);
      totalPrice = 0;  // Reset total price for new order
      do {
         // User selecting an item
         UART_sendString(colorUser "[User]: " colorReset);
         UART_getLine(item, 10);
         while (strcmp(item, "eggs") != 0 && strcmp(item, "chicken") != 0 && strcmp(item, "milk") != 0) {
            snprintf(message, sizeof(message),
```

```
                "%s are not available today. Today we have eggs, chicken, and milk. What would you like to order?\r\n",
                item);
            UART_sendString(colorBot "[Market Bot]: " colorReset);
            UART_sendString(message);
            UART_sendString(colorUser "[User]: " colorReset);
            UART_getLine(item, 10);
        }
        // Handle item price and quantity
        //Ternary Operator
        //
        int PRICE = (strcmp(item, "eggs") == 0) ? 2 :
                (strcmp(item, "chicken") == 0) ? 1 : 2;
        snprintf(message, sizeof(message),
                "%s costs $%d. How many would you like?\r\n", item, PRICE);
        UART_sendString(colorBot "[Market Bot]: " colorReset);
        UART_sendString(message);
        UART_sendString(colorUser "[User]: " colorReset);
        UART_getLine(response, 10);  // Get quantity as input
        int QUANTITY = atoi(response);  // Convert to integer
        totalPrice += PRICE * QUANTITY;  // Update total price
        // Ask if the User Wants to Order More
        UART_sendString(colorBot "[Market Bot]: " colorReset);
        UART_sendString("Do you want to order more (yes/no)?\r\n");
        UART_sendString(colorUser "[User]: " colorReset);
        UART_getLine(response, 10);  // Get yes/no response
        // Validate yes/no response
        while (strcmp(response, "yes") != 0 && strcmp(response, "no") != 0) {
            UART_sendString(colorBot "[Market Bot]: " colorReset);
            UART_sendString("Please respond with 'yes' or 'no'.\r\n");
            UART_sendString(colorUser "[User]: " colorReset);
            UART_getLine(response, 10);
        }
        // If "yes", re-prompt for the next item
        if (strcmp(response, "yes") == 0) {
            UART_sendString(colorBot "[Market Bot]: " colorReset);
            UART_sendString("Today we have eggs, chicken, and milk. What would you like to order?\r\n");
        }
    } while (strcmp(response, "yes") == 0);
    // Show total price and thank the user
    snprintf(message, sizeof(message),
            "The total for all items is $%d. Thank you for shopping with me, %s!\r\n\n",
            totalPrice, userName);
    UART_sendString(colorBot "[Market Bot]: " colorReset);
    UART_sendString(message);
  }
}
// UART Setup
void UART_SETUP() {
  P3SEL |= BIT3 + BIT4;  // Set RXD/TXD pins
  UCA0CTL1 |= UCSWRST;   // Software reset
  UCA0CTL0 = 0;          // Control register
  UCA0CTL1 |= UCSSEL_2;  // Use SMCLK
  UCA0BR0 = 0x09;  // Baud rate 115200
```

```
    UCA0BR1 = 0x00;
    UCA0MCTL = 0x02;
    UCA0CTL1 &= ~UCSWRST;  // Initialize UART state machine
}
// Send a character via UART
void UART_sendChar(char bip) {
    while (!(UCA0IFG & UCTXIFG));  // Wait for TX buffer to be ready
    UCA0TXBUF = bip;
}
// Receive a character via UART
char UART_getChar() {
    while (!(UCA0IFG & UCRXIFG));  // Wait for RX buffer to be ready
    return UCA0RXBUF;
}
// Send a string via UART
void UART_sendString(char* str) {
    while (*str) {
        UART_sendChar(*str++);
    }
}
// Receive a line of input via UART with backspace handling
void UART_getLine(char* buf, int limit) {
    int i = 0;
    char ch;
    while (1) {
        ch = UART_getChar();
        if (ch == '\r' || ch == '\n') {  // Enter key pressed
            buf[i] = '\0';  // Null-terminate the string
            UART_sendString("\r\n");
            break;
        } else if (ch == '\b' || ch == 127) {  // Backspace pressed
            if (i > 0) {
                i--;  // Move buffer pointer back
                UART_sendString("\b \b");  // Clear character on screen
            }
        } else if (i < limit - 1) {  // Normal character input
            buf[i++] = ch;
            UART_sendChar(ch);  // Echo the character back
        }
    }
}
```

Table 2: Program 2 source code

```
/*-------------------------------------------------------------------------------------------- *
 *  File: Lab08_p1.c
 * Description: This program generates a triangular waveform with an amplitude of 8 units and a frequency of 2.5 Hz using
 * the MSP430 microcontroller. The waveform data is transmitted over UART at 57,600 baud to an external application (like
 * UAH Serial App). The waveform completes 4 full periods (4 rising and 4 falling phases).
 * Author: Gianna Foti
*--------------------------------------------------------------------------------------------*/
```

```
#include <msp430.h>
#include <stdint.h>
#include <stdbool.h>

// Global variables
volatile float myData = 0.0;  // Stores the current value of the waveform
int i = 0;  // Counter to track the number of completed periods (full wave cycles)
bool peak = true;  // Flag to indicate if the waveform is rising (true) or falling (false)

// Function to set up UART communication
void UART_setup(void) {
   P3SEL |= BIT3 | BIT4;       // Configure P3.3 (TXD) and P3.4 (RXD) for UART functionality
   UCA0CTL1 |= UCSWRST;        // Put USCI_A0 in reset mode to configure settings
   UCA0CTL1 |= UCSSEL_2;       // Use SMCLK (1 MHz) as the clock source for UART
   UCA0BR0 = 18;               // Set the lower byte of the baud rate divider (1 MHz / 57600)
   UCA0BR1 = 0;                // Set the upper byte of the baud rate divider to 0
   UCA0MCTL = UCBRF_0 | UCBRS_2;  // Configure modulation for accurate baud rate timing
   UCA0CTL1 &= ~UCSWRST;       // Release USCI_A0 from reset to begin operation
}

// Function to transmit a character over UART
void sendChar(char c) {
   while (!(UCA0IFG & UCTXIFG));  // Wait until the UART transmit buffer is ready
   UCA0TXBUF = c;                 // Load the character into the transmit buffer to send it
}

// Main function to set up the system and enter low power mode
int main() {
   WDTCTL = WDT_MDLY_32;  // Configure the Watchdog Timer for 32 ms interval interrupts
   UART_setup();  // Initialize UART for communication
   SFRIE1 |= WDTIE;  // Enable interrupts for the Watchdog Timer

   myData = 0.0;  // Initialize waveform value to 0 (start of rising phase)
   __bis_SR_register(LPM0_bits + GIE);  // Enter Low Power Mode 0 with global interrupts enabled
}

// Watchdog Timer ISR - Handles the generation and transmission of the triangular wave
#pragma vector = WDT_VECTOR
__interrupt void watchdog_timer(void) {
   char *dataPtr = (char*)&myData;  // Pointer to access the bytes of the floating-point waveform value
   sendChar(0x55);  // Send a start byte (optional) to indicate the beginning of the packet

   // Loop to send each byte of the floating-point value over UART
   int j;
   for (j = 0; j < 4; j++) {
      sendChar(dataPtr[j]);  // Send one byte of the floating-point value
   }

   // Update the waveform value based on the current phase (rising or falling)
   if (peak) {
      myData += 0.04;  // Increase the value by 0.04 units in the rising phase
      if (myData >= 8.0) {  // Check if the peak amplitude is reached
```

```
            peak = false;  // Switch to the falling phase
        }
    } else {
        myData -= 0.04;  // Decrease the value by 0.04 units in the falling phase
        if (myData <= 0.0) {  // Check if the waveform has reached the minimum value
            peak = true;  // Switch back to the rising phase
            i++;  // Increment the period counter after a complete up/down cycle
        }
    }

    // Stop the waveform generation after completing 4 full periods
    if (i == 4) {
        SFRIE1 &= ~WDTIE;  // Disable Watchdog Timer interrupts to stop waveform generation
    }
}
```