

CPE 325: Intro to Embedded Computer System

FINAL PROJECT

Help Bot with UART, LED, Buzzer, and Switch1

Submitted by: Gianna Foti

Date of Experiment: 14 November 2024

Report Deadline: 22 November 2024

Lab Section: CPE353-02

Demonstration Deadline: 22 November 2024

Introduction

In this lab, we had to design and create our own project. My program was an embedded C program designed for an MSP430 microcontroller that creates an interactive safety communication system, specifically a "Help Bot" that provides emergency assistance through multiple communication channels. This project is inspired by the need for a discrete signaling mechanism that can be used in situations where immediate assistance is required. This system incorporated UART communication, a buzzer, and LED Morse code functionality to create a compact and effective alert system. I will explain how my program works in more detail later in the report.

Theory Topics

1. Watchdog Timer: what is it, how can it be used, different modes, etc.). Give an Example of where each mode could be utilized
 - a. The Watchdog Timer (WDT) in the MSP430 microcontroller is a safeguard designed to reset the system if a software failure occurs. It can prevent the system from hanging by resetting the microcontroller when the software fails to reset the timer within a specified interval. Alternatively, the WDT can be configured in interval mode, where instead of resetting the system, it generates periodic interrupts at specified intervals. This mode is commonly used for regular tasks, such as blinking an LED or executing periodic sensor readings, allowing the system to perform small operations without needing constant CPU intervention. In this configuration, the WDT triggers an Interrupt Service Routine (ISR) at regular intervals, enabling timed actions like toggling an output pin.
2. Timers:
 - a. The Timers A and B in the MSP430 are flexible peripherals used for various time-based operations such as generating Pulse Width Modulation (PWM) signals, counting events, and measuring time intervals.

- i. Timer A is typically used for general timing tasks and PWM generation, making it suitable for most applications that require periodic signals or time delays.
 - ii. Timer B offers higher precision and additional advanced features, making it ideal for more complex tasks that demand finer control, such as precise frequency generation or managing multiple PWM outputs simultaneously.
- b. Both timers allow the microcontroller to perform time-sensitive tasks without constantly engaging the CPU, which enables the system to remain in low-power modes (LPM) while still handling these operations efficiently. This capability is crucial in applications where power conservation is important, such as battery-operated devices.

3. LED and Buzzer

- a. In the MSP430 microcontroller, the LED and buzzer are output devices that provide visual and auditory feedback. The pulse-width modulation (PWM) of the buzzer is controlled using Timer B on the General-Purpose Input/Output (GPIO) pins. In my project, the LED and buzzer are output devices used to deliver visual and auditory feedback in an emergency assistance system. The buzzer, connected to the MSP430 via a GPIO pin, is controlled using Timer B. Timer B is configured to generate a pulse-width modulation (PWM) signal, enabling precise sound production. This is utilized to alert the user when the "help" command is received via UART, signaling a potential danger. The use of Timer B ensures that the buzzer operates efficiently and accurately while allowing the CPU to remain in low-power states during periods of inactivity.

4. Interrupts

- a. Interrupts are a fundamental feature of microcontroller systems, enabling efficient handling of asynchronous events without continuously polling for changes. When an interrupt occurs, the processor temporarily halts its current task, saves the execution state, and executes a predefined function known as an Interrupt Service Routine (ISR). After completing the ISR, the processor resumes the original task, maintaining system

functionality. In this lab, interrupts will be used to respond to button presses for triggering specific tasks. For instance, pressing a button may activate an LED or sound a buzzer.

The ISR will handle these actions, ensuring that the main program flow remains uninterrupted.

Results & Observation

Part 1:

Program Description:

This program implements an interactive chatbot system on the MSP430 microcontroller, combining UART communication with buzzer and LED functionalities to simulate an emergency assistance scenario. The chatbot operates over a serial terminal at a baud rate of 115,200, allowing users to interact by typing commands and receiving responses. When the user types "help," the program activates a buzzer that produces a 2 kHz tone, signaling an alert for assistance. Additionally, the program offers a silent emergency feature where pressing the SW1 button flashes the word "HELP" in Morse code using LED1.

Program Setup:

1. Connect UART cable
 - a. White P3.3
 - b. Green P3.4
 - c. Black: GND
2. Connect Buzzer cable
 - a. One pin to P7.4
 - b. Other pin to ground
3. Use 115,200 baud rate
4. Instructions:
 - a. Type help in order to sound the buzzer in the terminal in the UART interface.

- b. Press switch 1 to trigger LED1 to blink HELP in morse code. A dot will have a shorter flash time whereas the line in morse code will have a longer flash time.

Process:

1. Setting Up the Hardware:

- a. The first thing I did was disable the watchdog timer to stop any unexpected resets.
- b. I configured the RX (P3.4) and TX (P3.3) pins and set up UART communication at 115,200 baud to create a smooth conversation between the bot and the user.
- c. Next, I configured the button (SW1) on P2.1 as an input with pull-up resistors and enabled interrupts so it could detect a press.
- d. Finally, I got the LED on P1.0 ready to flash Morse code and prepared the Watchdog Timer to handle periodic tasks like controlling the buzzer.

2. Welcoming the User:

- a. I wanted the bot to feel friendly and helpful, so I added a welcome message that explains how to interact with it.
- b. The bot greets the user, tells them they can type "help" to trigger the buzzer, or press the switch for a quieter Morse code signal.

3. Handling User Commands:

- a. I made sure the bot patiently waits for the user to type a command.
- b. If the user types "help," the bot acknowledges their request, activates the buzzer for assistance, and lets them know help is on its way.
- c. If the input isn't "help," the bot gently lets the user know it didn't understand and encourages them to try again.

4. Responding to the Switch:

- a. For users who might not want the buzzer to sound, I added an option to press SW1 instead.

- b. When the switch is pressed, the bot sends a message over UART explaining that it's signaling for help with Morse code.
 - c. The LED then blinks out "HELP" in Morse code, just like a subtle SOS.
- 5. Blinking Morse Code:
 - a. This part was fun! I mapped out how the LED should blink for dots and dashes, with spaces in between to form the word "HELP."
 - b. I added delays for each symbol and letter to make the signal clear and recognizable.
- 6. Making the Buzzer Work:
 - a. Using the Watchdog Timer, I created a way to control how long the buzzer stays on.
 - b. The bot turns the buzzer on when needed and automatically shuts it off after the appropriate time.

Flowchart:

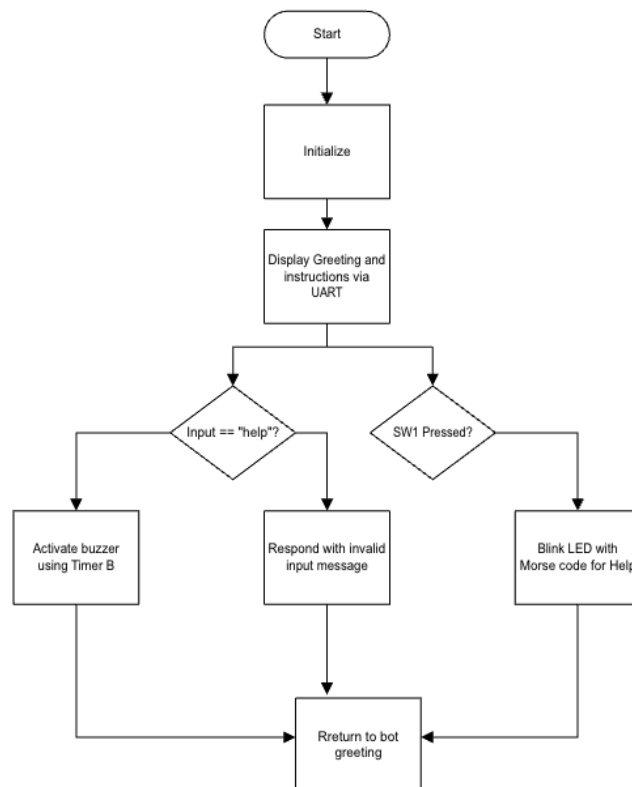
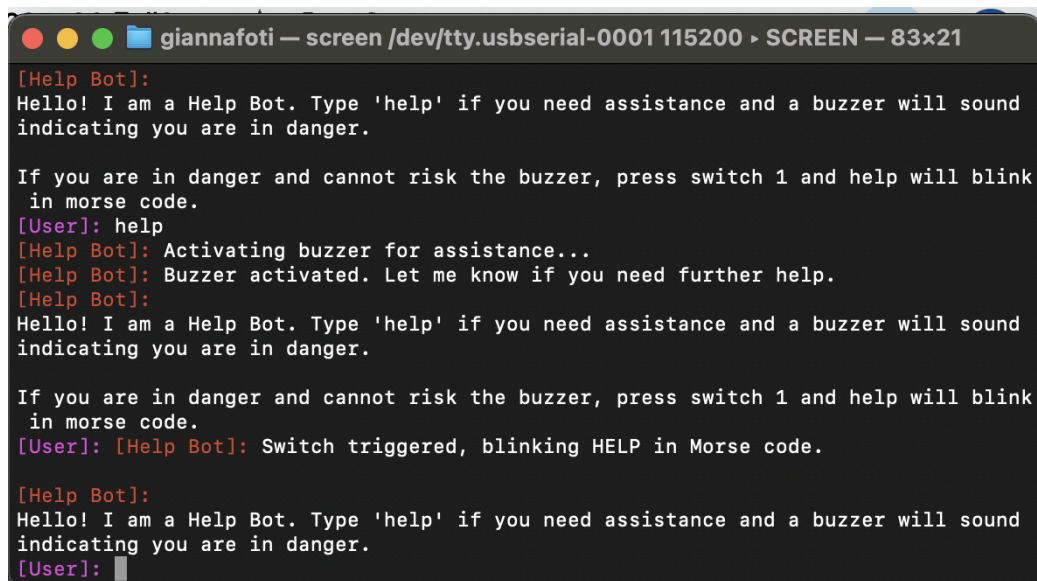


Figure 1: Program 1 Flowchart

Program Output:



```
[Help Bot]:
Hello! I am a Help Bot. Type 'help' if you need assistance and a buzzer will sound
indicating you are in danger.

If you are in danger and cannot risk the buzzer, press switch 1 and help will blink
in morse code.
[User]: help
[Help Bot]: Activating buzzer for assistance...
[Help Bot]: Buzzer activated. Let me know if you need further help.
[Help Bot]:
Hello! I am a Help Bot. Type 'help' if you need assistance and a buzzer will sound
indicating you are in danger.

If you are in danger and cannot risk the buzzer, press switch 1 and help will blink
in morse code.
[User]: [Help Bot]: Switch triggered, blinking HELP in Morse code.

[Help Bot]:
Hello! I am a Help Bot. Type 'help' if you need assistance and a buzzer will sound
indicating you are in danger.
[User]:
```

Figure 2: Program 1 Output

Calculations:

1. $f = \text{SMCLK}/(2(\text{TBCCR0})) \rightarrow 2 \text{ kHz} = 1 \text{ MHz}/(2(\text{TBCCR0})) \rightarrow 2000 = 1000000/(2(\text{TBCCR0}))$
 $\rightarrow 1000000/4000 = 250$

Conclusion

My project fulfills the requirements of the final assignment by integrating multiple MSP430 peripherals and external components in a meaningful way. Specifically, I use Port I/O and interrupts for button handling, TimerB for controlling the buzzer, and the Watchdog Timer for managing periodic tasks like Morse code signaling. I also use communication via UART to create an interactive chatbot interface.

These peripherals are actively utilized throughout the project; for example, the button triggers Morse code signals using interrupts, and timers manage the precise timing of buzzer and LED operations. For external components, I incorporate a switch (SW1) and an LED for Morse code signaling, along with a buzzer for audible alerts. The project is designed to simulate an emergency assistance system, where the chatbot responds to user input and signals danger through sound or light. This purposeful functionality ensures that my project goes beyond simple tasks, addressing a hypothetical real-world problem while meeting the

complexity and practical relevance required by the assignment. This has been a great semester and thank you for being a great TA! Have a great winter break.

Appendix

Table 1: Program 1

```
/*-----
 * File: FINALPROJECT.c
 * Description: Uses a 115,200 baud connection to create an interactive chatbot program
 * Input: User input
 * Output: Chat output, Buzzer sound, LED blinking
 * Author: Gianna Foti
 *-----*/
#include <msp430.h> // Include the MSP430 library for microcontroller functions
#include <string.h> // Include string library for string manipulation functions
// Define the Buzzer Pin
#define BUZZER_PIN BIT4 // Buzzer is connected to P7.4
// Define Colors for UART Output (Optional)
#define colorReset "\x1b[0m" // Reset terminal color formatting
#define colorBot "\x1b[31m" // Bot messages displayed in red
#define colorUser "\x1b[35m" // User messages displayed in purple
// Morse Code Definitions
#define DOT_DURATION 1000000 // Duration of a dot in Morse code (adjusted for clock speed)
#define DASH_DURATION 3000000 // Duration of a dash in Morse code (adjusted for clock speed)
#define SYMBOL_SPACE 1000000 // Space between Morse code symbols
#define LETTER_SPACE 3000000 // Space between Morse code letters
// Function Prototypes
void UART_SETUP(); // Function to set up UART communication
void UART_sendString(char* str); // Function to send a string via UART
void UART_getLine(char* buf, int limit); // Function to receive a line of input via UART
void ConfigureTimerForBuzzer(); // Function to configure the buzzer's timer
void StartBuzzer(); // Function to start the buzzer
void StopBuzzer(); // Function to stop the buzzer
void buttonyButSetup(); // Function to configure the button (SW1)
void watchyWatchSetup(); // Function to configure the Watchdog Timer
void ConfigureTimerForLED(); // Function to configure the LED's timer
void FlashMorseCode(char* message); // Function to flash a message in Morse code
// Global Variables
volatile unsigned int beepyBeep = 0; // Counter for buzzer operation duration
volatile unsigned int debounceCounter = 0; // Counter for button debounce logic
volatile unsigned int useyUse = 1; // Example flag variable
volatile unsigned int switchTriggered = 0; // Flag to indicate switch press
// Add other necessary global variables here
// Morse code for "HELP"
const char* morseHelp = "HELP"; // Morse code message to flash when HELP is needed
// Function Implementations
void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop the watchdog timer to prevent unwanted resets
    UART_SETUP(); // Initialize UART for communication
    watchyWatchSetup(); // Set up the Watchdog Timer for periodic tasks
    buttonyButSetup(); // Configure button (SW1) with interrupts
    lightyLightSetup(); // Configure LED operation
    __enable_interrupt(); // Enable global interrupts
    char input[50]; // Buffer to store user input
    char message[100]; // Buffer to store bot response messages
    while (1) { // Main infinite loop
```

```

// Bot greets the user
UART_sendString(colorBot "[Help Bot]: " colorReset); // Send bot prompt
UART_sendString("\nHello! I am a Help Bot. Type 'help' if you need assistance and a buzzer will sound indicating you
are in danger.\n");
UART_sendString("\nIf you are in danger and cannot risk the buzzer, press switch 1 and help will blink in morse code.
\n");
// User's turn to type a command
UART_sendString(colorUser "[User]: " colorReset); // Prompt user for input
UART_getLine(input, sizeof(input)); // Receive input from the user
// Check for "help" keyword
if (strcmp(input, "help") == 0) { // Compare input with "help"
    // Bot acknowledges and activates the buzzer
    UART_sendString(colorBot "[Help Bot]: " colorReset);
    UART_sendString("Activating buzzer for assistance...\n");
    beepyBeep = 32; // Set buzzer duration
    __delay_cycles(1000000); // Delay for ~1 second (assuming 1MHz SMCLK)
    // Bot confirms buzzer action
    UART_sendString(colorBot "[Help Bot]: " colorReset);
    UART_sendString("Buzzer activated. Let me know if you need further help.\n");
} else {
    // Bot informs the user of invalid input
    snprintf(message, sizeof(message), // Format the invalid input message
        "I didn't recognize '%s'. Please type 'help' if you need assistance.\n",
        input);
    UART_sendString(colorBot "[Help Bot]: " colorReset);
    UART_sendString(message); // Send the formatted message
}
}
}

void buttonyButSetup() {
    // Configure SW1 (P2.1) as input with pull-up resistor and enable interrupt
    P2DIR &= ~BIT1; // Set P2.1 as input
    P2REN |= BIT1; // Enable pull-up resistor on P2.1
    P2OUT |= BIT1; // Set pull-up resistor to default high state
    P2IE |= BIT1; // Enable interrupt on P2.1
    P2IES |= BIT1; // Trigger interrupt on high-to-low transition
    P2IFG &= ~BIT1; // Clear interrupt flag for P2.1
}

void UART_SETUP() {
    P3SEL |= BIT3 + BIT4; // Set P3.3 (TXD) and P3.4 (RXD) for UART functionality
    UCA0CTL1 |= UCSWRST; // Enable software reset to configure UART
    UCA0CTL1 |= UCSSEL_2; // Select SMCLK as the clock source
    UCA0BR0 = 9; // Configure baud rate to 115200 (assuming 1MHz clock)
    UCA0BR1 = 0; // Upper byte of baud rate is 0
    UCA0MCTL = 2; // Set modulation pattern for baud rate
    UCA0CTL1 &= ~UCSWRST; // Release software reset, enabling UART
}

// Rest of the functions and ISRs follow the same commenting style
// Send a string via UART
void UART_sendString(char* str) {
    while (*str) {
        while (!(UCA0IFG & UCTXIFG));
        UCA0TXBUF = *str++;
    }
}

```

```

    }
}
// Receive a line of input via UART
void UART_getLine(char* buf, int limit) {
    int i = 0;
    char ch;
    while (1) {
        while (!(UCA0IFG & UCRXIFG));
        ch = UCA0RXBUF;
        if (ch == '\r' || ch == '\n') {
            buf[i] = '\0'; // Null-terminate
            UART_sendString("\r\n");
            break;
        } else if (ch == '\b' || ch == 127) {
            if (i > 0) {
                i--;
                UART_sendString("\b\b");
            }
        } else if (i < limit - 1) {
            buf[i++] = ch;
            while (!(UCA0IFG & UCTXIFG));
            UCA0TXBUF = ch; // Echo
        }
    }
}

//// Configure Timer for Buzzer
void watchyWatchSetup()
{
    WDTCTL = WDT_MDLY_32; // Interval timer mode with ~32ms
    SFRIE1 |= WDTIE; // Enable Watchdog Timer interrupts
    P7DIR |= BUZZER_PIN; // Set P7.4 as output
    P7SEL |= BUZZER_PIN; // Select peripheral function (TimerB output)
    TB0CCTL2 = OUTMOD_0; // Initialize with output off
    TB0CTL = TBSSSEL_2 | MC_1; // SMCLK, up mode
    TB0CCR0 = 500 - 1; // 2 kHz frequency
    TB0CCR2 = 250; // 50% duty cycle
}

void lightyLightSetup()
{
    // Assuming LED1 is connected to P1.0
    P1DIR |= BIT0; // Set P1.0 as output
    P1OUT &= ~BIT0; // Ensure LED is off
}

void FlashMorseCode(char* message)
{
    int i; // Declare 'i' at the beginning for C89 compatibility
    while (*message) {
        if (*message == 'H') {
            // H: ....
            for(i = 0; i < 4; i++) {
                P1OUT |= BIT0; // LED on
                delay_cycles(DOT_DURATION); // Delay for dot
                P1OUT &= ~BIT0; // LED off
            }
        }
        // ... (rest of the function)
    }
}

```

```

    delay_cycles(SYMBOL_SPACE); // Space between symbols
}
    delay_cycles(LETTER_SPACE); // Space between letters
}
else if (*message == 'E') {
    // E: .
    P1OUT |= BIT0;
    delay_cycles(DOT_DURATION);
    P1OUT &= ~BIT0;
    delay_cycles(SYMBOL_SPACE);
    delay_cycles(LETTER_SPACE);
}
else if (*message == 'L') {
    // L: .-..
    // .
    P1OUT |= BIT0;
    delay_cycles(DOT_DURATION);
    P1OUT &= ~BIT0;
    delay_cycles(SYMBOL_SPACE);
    // -
    P1OUT |= BIT0;
    delay_cycles(DASH_DURATION);
    P1OUT &= ~BIT0;
    delay_cycles(SYMBOL_SPACE);
    // ..
    for(i = 0; i < 2; i++) {
        P1OUT |= BIT0;
        delay_cycles(DOT_DURATION);
        P1OUT &= ~BIT0;
        delay_cycles(SYMBOL_SPACE);
    }
    delay_cycles(LETTER_SPACE);
}
else if (*message == 'P') {
    // P: .--.
    // .
    P1OUT |= BIT0;
    delay_cycles(DOT_DURATION);
    P1OUT &= ~BIT0;
    delay_cycles(SYMBOL_SPACE);
    // --
    for(i = 0; i < 2; i++) {
        P1OUT |= BIT0;
        delay_cycles(DASH_DURATION);
        P1OUT &= ~BIT0;
        delay_cycles(SYMBOL_SPACE);
    }
    // .
    P1OUT |= BIT0;
    delay_cycles(DOT_DURATION);
    P1OUT &= ~BIT0;
    delay_cycles(LETTER_SPACE);
}
}

```

```

    message++;
}
}
// UART RX ISR to handle incoming data if needed
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    // You can handle UART RX interrupts here if you prefer
    // Currently, UART_getLine() is polling, so this ISR is optional
}
// ISR for SW1 ("No" response)
#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
    if (P2IFG & BIT1)
    {
        volatile unsigned long k; // Declare 'k' for debounce
        for(k = 25000; k > 0; k--); // Debounce delay
        useyUse = 0; // Set "no" response (ensure 'useyUse' is declared)
        P2IFG &= ~BIT1; // Clear interrupt flag
        // Display message over UART
        UART_sendString(colorBot "[Help Bot]: " colorReset);
        UART_sendString("Switch triggered, blinking HELP in Morse code.\r\n");
        // Flash Morse code for "HELP"
        FlashMorseCode("HELP");
        // Return to Help Bot prompt
        UART_sendString(colorBot "\r\n[Help Bot]: " colorReset);
        UART_sendString("\r\nHello! I am a Help Bot. Type 'help' if you need assistance and a buzzer will sound indicating you are in danger.\r\n");
        UART_sendString(colorUser "[User]: " colorReset);
    }
}
// Watchdog Timer ISR
#pragma vector = WDT_VECTOR
__interrupt void WDT_ISR(void)
{
    if (beepyBeep > 0) {
        beepyBeep--;
        TB0CCTL2 = OUTMOD_7; // Start PWM (buzzer on)
        if (beepyBeep == 0) {
            TB0CCTL2 = OUTMOD_0; // Stop PWM (buzzer off)
        }
    } else {
        TB0CCTL2 = OUTMOD_0; // Ensure buzzer remains off
    }
}

```

