MORSE CODE TRANSLATOR USING MSP432

TERM PROJECT

Submitted by

CB.EN.U4CCE21047	PRANJALI YADAV
CB.EN.U4CCE21048	PUBESH KUMAAR K S
CB.EN.U4CCE21051	RAKESH V
CB.EN.U4CCE21074	TEJASWINI D

B.TECH COMPUTER AND COMMUNICATION ENGINEERING 2025 BATCH

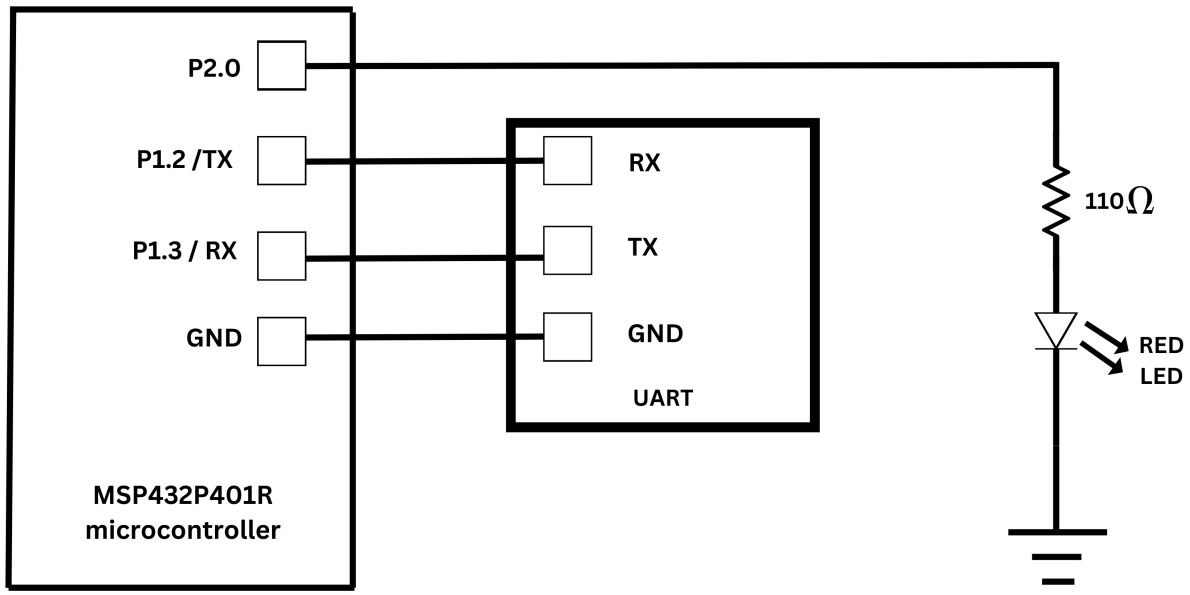
Dept. of Electronics and Communication Engineering

MORSE CODE TRANSLATOR USING MSP432

AIM:

The aim of this project is to implement Morse code communication using the MSP432 microcontroller and the onboard LED. Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks, representing letters, numbers, and punctuation. By utilizing the MSP432 microcontroller and its onboard LED, we can create a simple yet effective Morse code communication system.

DESIGN:



PERIPHERALS AND PINS:

S.NO	PERIPHERALS	PINS	DESCRIPTION
1	UART (Universal synchronous Receiver - Transmitter)	P1.2	Transmitter buffer
2	UART (Universal synchronous Receiver - Transmitter)	P1.3	Receiver buffer
3	GPIO (General Purpose Input and Output)	P2.0	LED output

CODE:

```
#include "msp.h"
void UART0_init(void);  // Function for UART Transmission
void delayMs(int n);    // Delay Function
void dot(void);
void dash(void);
unsigned char c;
unsigned char message[] = "Ready\n";
int i;

// Main Function:
int main(void)
{
```

```
UART0_init();  // Call UART Transmission Function

for (i = 0; i < 7; i++)
{
   while (!(EUSCI_A0->IFG & 0x02))
   {
   }                    // Wait until transmitter buffer is empty
   EUSCI_A0->TXBUF = message[i];  // Send a character
}

// Infinite Loop (An embedded program does not stop):
while (1)
{
   while (!(EUSCI_A0->IFG & 0x02))
   {
   }                    // Wait until transmitter buffer is empty
   c = EUSCI_A0->RXBUF;      // Receive a character

   while (!(EUSCI_A0->IFG & 0x02))
   {
   }                    // Wait until transmitter buffer is empty
   EUSCI_A0->TXBUF = c;      // Display the character

                                P2 -> SEL0 &= ~1;
                                P2 -> SEL1 &= ~1;
                                P2 -> DIR |= 1;
   if (c == 'a' || c == 'A') // Check if the received character is 'a' or 'A'
   {
```

```
    dot();
                                        dash();
                                            delayMs(1500);
                                            // Delay to create a blinking effect
}
                                else if(c == 'b' || c == 'B')
                                {
    dash();
                                            dot();
                                            dot();
                                            dot();
                                        delayMs(1500);// Delay to create a blinking effect
}
                                else if(c == 'c' || c == 'C')
                                {
                                        dash();
                                        dot();
                                        dash();
                                        dot();
                                        delayMs(1500);
                                }
                                else if(c == 'd' || c == 'D')
                                {dash();
                                        dot();
                                        dot();
                                        delayMs(1500);
                                }
                                else if(c == 'e' || c == 'E')
```

```
{dot();
        delayMs(1500);
}
else if(c == 'f' || c == 'F')
{dot();
        dot();
        dash();
        dot();
        delayMs(1500);

}
else if(c == 'g' || c == 'G')
{dash();
 dash();
        dot();
        delayMs(1500);
}
else if(c == 'h' || c == 'H'){
        dot();
        dot();
        dot();
        dot();
        delayMs(1500);
}
else if(c == 'i' || c == 'I'){
        dot();
        dot();
        delayMs(1500);
```

```
}
else if(c == 'j' || c == 'J'){
        dot();
        dash();
        dash();
        dash();
        delayMs(1500);
}
else if(c == 'k' || c == 'K'){
        dash();
        dot();
        dash();
        delayMs(1500);
}
else if(c == 'l' || c == 'L'){
        dot();
        dash();
        dot();
        dot();
        delayMs(1500);
}
else if(c == 'm' || c == 'M'){
        dash();
        dash();
        delayMs(1500);
}
else if(c == 'n' || c == 'N'){
        dash();
```

```
            dot();
            delayMs(1500);
        }
        else if(c == 'o' || c == 'O'){
            dash();
            dash();
            dash();
            delayMs(1500);
        }
        else if(c == 'p' || c == 'P'){
            dot();
            dash();
            dash();
            dot();
            delayMs(1500);
        }
        else if(c == 'q' || c == 'Q'){
            dash();
            dash();
            dot();
            dash();
            delayMs(1500);
        }
        else if(c == 'r' || c == 'R'){
            dot();
            dash();
            dot();
            delayMs(1500);
```

```
}
else if(c == 's' || c == 'S'){
        dot();
        dot();
        dot();
        delayMs(1500);
}
else if(c == 't' || c == 'T'){
        dash();
        delayMs(1500);
}
else if(c == 'u' || c == 'U'){
        dot();
        dot();
        dash();
        delayMs(1500);
}
else if(c == 'v' || c == 'V'){
        dot();
        dot();
        dot();
        dash();
        delayMs(1500);
}
else if(c == 'w' || c == 'W'){
        dot();
        dash();
        dash();
```

```
                delayMs(1500);
            }
            else if(c == 'x' || c == 'X'){
                dash();
                dot();
                dot();
                dash();
                delayMs(1500);
            }
            else if(c == 'y' || c == 'Y'){
                dash();
                dot();
                dash();
                dash();
                delayMs(1500);
            }
            else if(c == 'z' || c == 'Z'){
                dash();
                dash();
                dot();
                dot();
                delayMs(1500);
            }
    }
}

// Function for UART Transmission:
void UART0_init(void)
```

```
{
   EUSCI_A0->CTLW0 |= 1;        // Put in reset mode to configure UART
   EUSCI_A0->MCTLW = 0;         // Disable oversampling
   EUSCI_A0->CTLW0 = 0x0081;   // 00 - 1 stop bit, No Parity, 8-bits data, Asynchronous
Mode, First LSB Then MSB, SMCLK
                                          // 81 -  Enabled EUSCI_A0 logic is held in reset state
   EUSCI_A0->BRW = 26;          // 3000000/115200 = 26
   P1->SEL0 |= 0x0C;            // Configure functionality of P1.2, P1.3 as UART Pins
   P1->SEL1 &= ~0x0C;
   EUSCI_A0->CTLW0 &= ~1;       // Take UART out of reset mode
}
void dot(void) {
   P2->OUT |= 1;    // Turn on LED
   delayMs(250);
   P2->OUT &= ~1;   // Turn off LED
   delayMs(500);
}

void dash(void) {
   P2->OUT |= 1;    // Turn on LED
   delayMs(1000);
   P2->OUT &= ~1;   // Turn off LED
   delayMs(500);
}
// Delay milliseconds when system clock is at 3 MHz for Rev C MCU:
void delayMs(int n)
{
   int i, j;
   for (j = 0; j < n; j++)
```

```
    for (i = 750; i > 0; i--)
        ; // Delay of 1 ms
}
```

IMPLEMENTATION STATUS:

Implementing Morse code using the MSP432P401R microcontroller, The UART communication and LED control functionalities have been successfully implemented. Mapping UART characters to Morse code patterns has been defined, allowing the microcontroller to toggle the LED accordingly.

INFERENCE:

The MSP432P401R microcontroller can be utilized to implement Morse code communication. By mapping UART characters to Morse code patterns, the microcontroller can control the blinking of an LED based on received characters. The project can be expanded with additional features like user interfaces or wireless communication.

CONCLUSION:

This project is to develop a Morse code translator using the MSP432 microcontroller. We programmed the microcontroller in embedded C using the Keil µVision IDE. The input for translator will come from UART peripherals, which will receive a character. The goal is to translate this text characters into Morse code signals and output them through the inbuilt LED in the microcontroller. By implementing this algorithm in firmware, we will be able to read the input stream, detect the duration of dots and dashes, and map them to their respective characters. The final result will be a working Morse code translator that can convert text input into Morse code output.