Lab 3 Interrupt Subroutines and UART Serial Communications

Simon Hu

yh4970

1. Background

Communication is essential in embedded systems because it enables the exchange of information between devices. One common method is UART (Universal Asynchronous Receiver-Transmitter), which allows two microcontrollers to send and receive serial data. In this lab, the Atmel ATmega32U4 microcontroller's built-in UART is used to create a communication link between two boards.

To make the communication responsive and efficient, interrupts are used instead of constantly checking for data in loop function. Interrupts allow the microcontroller to pause its main program when a certain event happens—like receiving data through UART—and immediately handle it using an interrupt service routine (ISR).

In this experiment, two microcontrollers form a group that has Team A and Team B. When a button is pressed on Team A's board, it sends a signal to Team B via UART. Team B receives this signal through an interrupt and lights up its onboard LED accordingly. It then sends a confirmation back to Team A, which also lights up its own LED to match the state through an interrupt.

2. Experimental Procedure

2.1 Team A and Team B UART setup

For this lab, the UART is available on the board with TX and RX. By controlling the UCSRA, UCSRB, UCSRC and UBRR registers, the UART can be set up. The UART serial bus needs to operate on 9600 bit rate, with 8 data bit, no parity, and has one stop bit. It should work in asynchronous and normal speed. Finally, because the other function such as lighting LED is based on interrupt, the RX interrupt is enabled. This part of code is shown here:

```
UCSR1A = 0b00000000; // normal speed

UCSR1C = 0b00000110; // 8 bit
```

```
 UBRR1 = 51;

 UCSR1B = 0b10011000; // RX Interrupt, receiver, transmitter

 sei(); // enable global interrupts
```

2.2 Team A and Team B button and LED setup

For this lab, the left button is on PD4, and led is on PC7. First, the constants are defined and then corresponding DDR should be set and then specific port can be set which enabling the control of button and LED.

The set up code for team A is

```
DDRC |= (1 << ledPin); // LED DDRD, output on

DDRD &= ~(1 << buttonPin); // logic 0 as input

PORTD |= (1 << buttonPin); // using pull up resistor to prevent floating when unconnected
```

The set up code for team B is

```
 DDRC |= (1 << ledPin); // LED DDRD, output on
```

2.3 Team A button information loop

Team A needs to handle the sending of button condition. When it is pressed, the UART should send 1, and it should send 0 when the button is not pressed. The button is read by PIND and data is send by UDR1. So the part of code is shown here:

```
void loop() {

 if ((PIND & ( 1 << buttonPin)) != 0) {

  while (!(UCSR1A & (1 << UDRE1))) {}

  uint8_t d = '1';

  UDR1 = d;

 } else {

  while (!(UCSR1A & (1 << UDRE1))) {}
```

```
    uint8_t d = '0';
    UDR1 = d;
  }
}
```

2.4 LED handled by Interrupt

To make sure the communication between boards does not affect other program's running, the interrupt is often used to read the data. The board's loop function is always running until it heard an interrupt called USART1_RX_vect. In this lab, the Team's B loop is empty since it only needs to hear data from team A and send it back. When this interrupt is called, which means one data is received, if data is 0, the LED turns off, and else, if the data is 1, the LED turns on. This is accomplished by the following code:

```
// Team A
volatile uint8_t rx = 0;
ISR (USART1_RX_vect) {
  rx = UDR1;
  if (rx == 0x31){
    PORTC |= (1 << ledPin); // Turn ON LED
  }
  else if (rx == 0x30){
    PORTC &= ~(1 << ledPin); // Turn OFF LED
  }
}
// Team B
volatile uint8_t rx = 0;
ISR (USART1_RX_vect) {
  rx = UDR1;
```

```
 if (rx == 0x31){
   PORTC |= (1 << ledPin); // Turn ON LED
 }
 else if (rx == 0x30) {
   PORTC &= ~(1 << ledPin); // Turn OFF LED
 }


 // send info back to team A
 while (!(UCSR1A & (1 << UDRE1))) {}
 UDR1 = rx;
}
```

2.5 Final Code Upload Testing

The Rx on Team A should connect to Tx on Team B, and Team B's Rx should connect to Team A's Tx. This way, the data can be sent and received correctly through UART. In the testing, when no button is pressed initially, the light on both boards should be OFF. When the left button on Team A board is pressed, the light on both boards should be ON; when the left button is released, both lights should be OFF. If A Tx - B Rx is not connected and the other line is connected, when the left button is pressed, all lights should be OFF. At this moment, the data cannot be transfer from A to B, and no data transfer from B to A. Thus, no LED lights up. If A Rx – B Tx is not connected while the other line is connected, when the left button on Team A board is pressed, only the LED on the Team B board will be ON. This is because the data was transmitted from A to B successfully, while not able to be transmitted back to A, then LED on board A would never be light up.

3. Conclusion

In this lab, my group successfully implemented UART-based serial communication between two microcontrollers using interrupts. By setting up the proper UART configuration (UCSRnA/B/C, UBRR1, etc.) and writing interrupt service routines, we enabled real-time communication where one team could control and respond to the LED status of the other. This lab demonstrates importance of efficient data exchange in embedded system applications.

4. Complete Code Listing

```
#include <Arduino.h>

const int ledPin = PC7;     // LED connected to PB0
const int buttonPin = PD4; // Right button connected to PD4


// my code, team A

void setup() {

  UCSR1A = 0b00000000; // normal speed

  UCSR1C = 0b00000110; // 8 bit
  UBRR1 = 51;
  UCSR1B = 0b10011000; // RX Interrupt, receiver, transmitter

  DDRC |= (1 << ledPin); // LED DDRD, output on

  DDRD &= ~(1 << buttonPin); // logic 0 as input
  PORTD |= (1 << buttonPin); // using pull up resistor to prevent floating
when unconnected

  sei(); // enable global interrupts
}


void loop() {
  if ((PIND & ( 1 << buttonPin)) != 0) {
    while (!(UCSR1A & (1 << UDRE1))) {}
    uint8_t d = '1';
    UDR1 = d;
  } else {
    while (!(UCSR1A & (1 << UDRE1))) {}
    uint8_t d = '0';
    UDR1 = d;
  }
}
```

```
volatile uint8_t rx = 0;
ISR (USART1_RX_vect) {
  rx = UDR1;
  if (rx == 0x31){
    PORTC |= (1 << ledPin); // Turn ON LED
  }
  else if (rx == 0x30){
    PORTC &= ~(1 << ledPin); // Turn OFF LED
  }
}



// my team B code

void setup() {

  UCSR1A = 0b00000000; // normal speed

  UCSR1C = 0b00000110; // 8 bit
  UBRR1 = 51;
  UCSR1B = 0b10011000; // RX Interrupt,  receiver, transmitter

  DDRC |= (1 << ledPin); // LED DDRD, output on

  sei(); // enable global interrupts
}

volatile uint8_t rx = 0;

ISR (USART1_RX_vect) {
  rx = UDR1;
  if (rx == 0x31){
    PORTC |= (1 << ledPin); // Turn ON LED
  }
  else if (rx == 0x30) {
    PORTC &= ~(1 << ledPin); // Turn OFF LED
  }

  // send info back to team A
  while (!(UCSR1A & (1 << UDRE1))) {}
  UDR1 = rx;
}
```

```
void loop(){

}
```

5. Picture of Two Boards Connection