Computer Interface Laboratory Experiments
Fourth Year Communications and Electronics Engineering

Experiment 3 – Software Debugger

Objective:

Using UART (Universal Asynchronous Receiver/Transmitter) to create a communication bridge between PC and µC to debug the µC.
In this lab, we will explain how to use the USART module from AVR Atmega16 to receive data. AVR and PC must be able to receive/send multiple bytes in some array. Data frame may include multiple information, it may have a byte representing a specific command (e.g. 'r' for read and 'w' for write) as well as other values representing some integer value.

Introduction:

The Software Debugger is a GUI application that makes it possible to capture state and control AVR µC registers and memory.

The protocol is simple. All we need to do is to send a command from the PC to the ATmega16 microcontroller. The command will be marked with start and end byte "@<CMD>;" and space characters must be ignored.

Start Byte	Command	Address	Data	End Byte	Description
@	r or R	2 Bytes	NA	;	Read
@	w or W	2 Bytes	3 Bytes	;	Write

Test Case:

#	Description
1	Set PORTA as output and then write 0x0F to it
2	Set PORTB as output and then write 0x81 to it

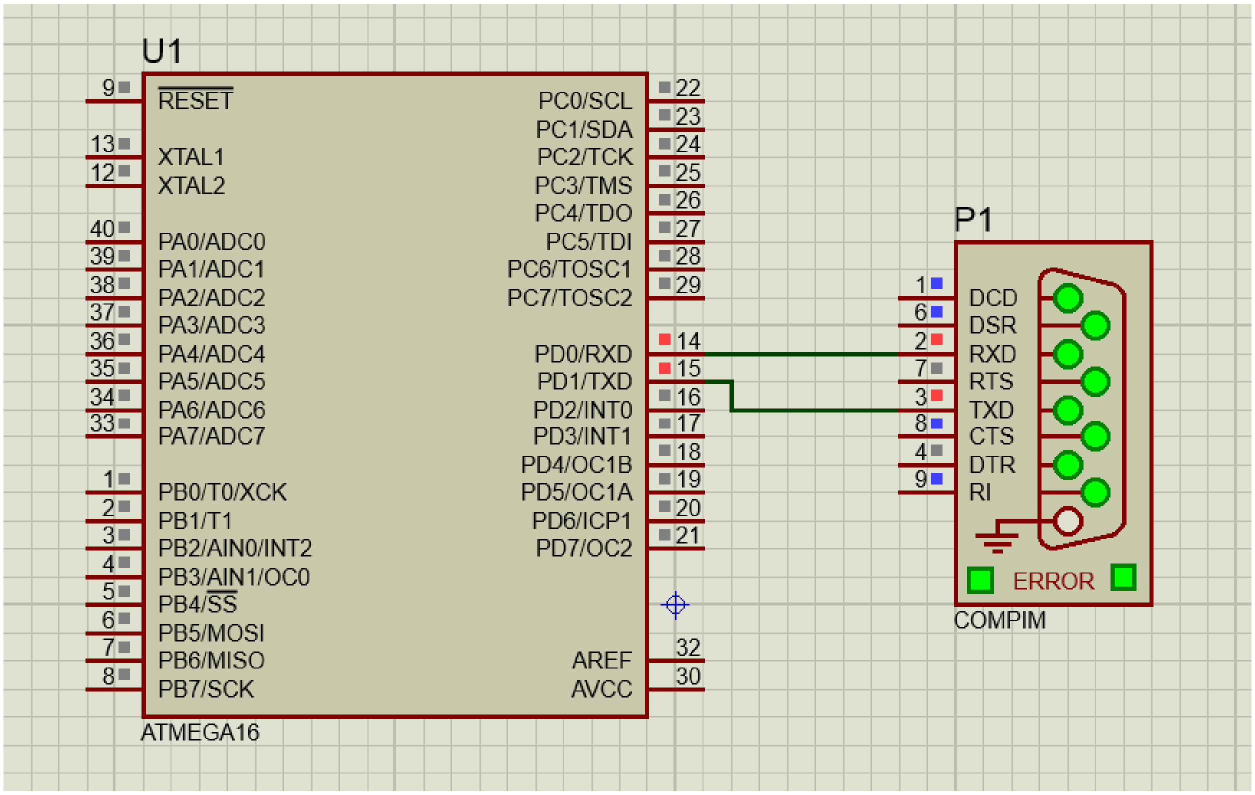
Experiment Procedure:

The goal here is to write values to ATMega16 registers using a GUI application.

Using proteus simulator

- Connect compim component to RXD and TXD pins.
- Configure compim settings (Baud rate, Data bits and Stop bits)

Circuit Diagram:



Code:

main.c file

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include ”UART.h”

#define BAUD_RATE 9600
#define F_CPU 8000000

/* Full command will be
 *  @ W/R AA DDD;
 * Spaces will be ignored by the driver
 * address AA and data DDD will be sent in decimal
 */


void SD_MainFunction();

static volatile uint8_t cmd_buffer[8];

int main(void) {
        /* Init UART driver. */
        UART_cfg my_uart_cfg;
        /* Set USART mode. */
        my_uart_cfg.UBRRL_cfg = (BAUD_RATE_VALUE)&0x00FF;
        my_uart_cfg.UBRRH_cfg = (((BAUD_RATE_VALUE)&0xFF00)>>8);
        my_uart_cfg.UCSRA_cfg = 0;
        my_uart_cfg.UCSRB_cfg = (1<<RXEN) | (1<<TXEN) | (1<<TXCIE) | (1<<RXCIE);
        my_uart_cfg.UCSRC_cfg = (1<<URSEL) | (3<<UCSZ0);
        UART_Init(&my_uart_cfg);

        sei();
        while(1) {
                SD_MainFunction();
        }
        return 0;
}

void SD_MainFunction() {
        volatile uint8_t *address;
        uint8_t value;
```

```
    /* Receive the full buffer command. */
    UART_ReceivePayload(cmd_buffer, 8);

    /* Poll until reception is complete. */
    while(0 == UART_IsRxComplete());

    // Parse Address
    char address_buffer[2];

    address_buffer[0] = cmd_buffer[2];
    address_buffer[1] = cmd_buffer[3];

    address = ((volatile uint8_t *)atoi(address_buffer));


    // Parse value
    char value_buffer[3];

    value_buffer[0] = cmd_buffer[4];
    value_buffer[1] = cmd_buffer[5];
    value_buffer[2] = cmd_buffer[6];

    value = (uint8_t)atoi(value_buffer);

    // Parse command buffer
    switch(cmd_buffer[1])
    {
        case 'w':
        case 'W':
        {
            // Write received value to received address.
            *(address) = value;
            break;
        }
        default:
        {
            // Do nothing.
        }
    }
}
```

UART.h file

```
#include <stdio.h>
```

```
#define BAUD_RATE_VALUE (((F_CPU)/(BAUD_RATE*16UL))-1)

typedef struct {
    /* Place here module configuration registers. */
    uint8_t UBRRH_cfg;
    uint8_t UBRRL_cfg;
    uint8_t UCSRA_cfg;
    uint8_t UCSRB_cfg;
    uint8_t UCSRC_cfg;
}UART_cfg;
extern void UART_Init(UART_cfg *my_cfg);
extern void UART_SendPayload(uint8_t *tx_data, uint16_t len);
extern void UART_ReceivePayload(uint8_t *rx_data, uint16_t len);
extern uint8_t UART_IsDataAvaiable(void);
extern uint8_t UART_IsTxComplete(void);
extern uint8_t UART_IsRxComplete(void);
```

UART.c file

```
#include ”UART.h”
#include <avr/interrupt.h>
static volatile uint8_t *tx_buffer;
static volatile uint16_t tx_len;
static volatile uint16_t tx_cnt;
static volatile uint8_t *rx_buffer;
static volatile uint16_t rx_len;
static volatile uint16_t rx_cnt;
ISR(USART_RXC_vect) {
    uint8_t rx_data;
    cli();
    /* Read rx_data. */
    rx_data = UDR;
    /* Ignore spaces */
    if((rx_cnt < rx_len) && (rx_data != ' ')) {
        rx_buffer[rx_cnt] = rx_data;
        rx_cnt++;
    }

    sei();
}
```

```
ISR(USART_TXC_vect) {
        cli();
        tx_cnt++;
        if(tx_cnt < tx_len) {
                /* Send next byte. */
                UDR = tx_buffer[tx_cnt];
        }
        sei();
}
void UART_Init(UART_cfg *my_cfg) {
        /* Set baud rate */
        UBRRH = my_cfg->UBRRH_cfg;
        UBRRL = my_cfg->UBRRL_cfg;
        UCSRA = my_cfg->UCSRA_cfg;
        UCSRB = my_cfg->UCSRB_cfg;
        UCSRC = my_cfg->UCSRC_cfg;
}
void UART_SendPayload(uint8_t *tx_data, uint16_t len) {
        tx_buffer = tx_data;
        tx_len = len;
        tx_cnt = 0;

        /* Wait for UDR is empty. */
        while(0 == (UCSRA & (1 << UDRE)));
        /* Send the first byte to trigger the TxC interrupt. */
        UDR = tx_buffer[0];
}
void UART_ReceivePayload(uint8_t *rx_data, uint16_t len) {
        rx_buffer = rx_data;
        rx_len = len;
        rx_cnt = 0;
}
uint8_t UART_IsTxComplete(void) {
        return ( (tx_cnt >= tx_len) ? 1 : 0 );
}
uint8_t UART_IsRxComplete(void) {
        return ( (rx_cnt >= rx_len) ? 1 : 0 );
}
```

Lab Deliverables:

Software Debugger system:
AVR based system to receive a command frame via UART module from GUI Application to control PORTA and PORTB:

- Set PORTA and PORTB as Output
- Write data to PORTA and PORTB

The GUI based PC application should have a textbox for the user to write the address to write to. It should also have a textbox for the user to write data.

Extended Feature (Mandatory):

Add read functionality to the GUI application.

You should add
- DropDown menu (Explained in Experiment 4 Video) to choose between read and write task
- Label to display the incoming read data (if read task was selected)
- Button to start reading process.

Note: Address textbox used for writing data will be used for reading data.

References:

1. VSPE Program (link)
2. ATmega16 datasheet (link).