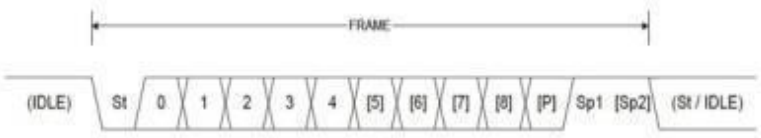
Experiment 2 – PC Bridge

Objective:

Using UART (Universal Asynchronous Receiver/Transmitter) to create a communication bridge between PC and µC.
In this experiment, we will explain how to use the USART module from AVR Atmega16 to send/receive data.

Introduction:

A frame refers to the entire data packet which is being sent/received during a communication. Depending upon the communication protocol, the formats of the frame might vary. For example, TCP/IP has a particular frame format, whereas UDP has another frame format. Similarly in our case, RS232 has a typical frame format as well. A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, a new frame can directly follow it, or the communication line can be set to an idle (high) state. Here is the frame format as mentioned in the AVR datasheet:



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

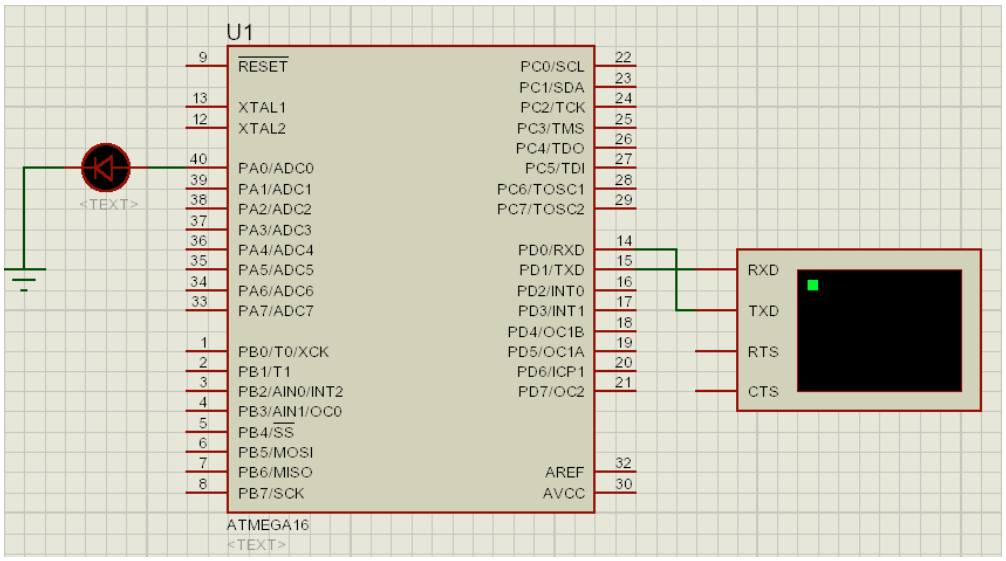
Experiment Procedure:

The goal here is to turn on and off the LED by sending ‘O’ or ‘F’.

Using proteus simulator

- Connect LED on **PIN A0**
- Connect terminal module found in the instruments tab to RXD and TXD pins.

Circuit Diagram:



Code:

main.c file

```
#include <avr/io.h>
#include <avr/interrupt.h>

#include ”UART.h”

#define BAUD_RATE 9600
#define F_CPU 1000000

int main(void) {
	DDRA = 0xFF;
	PORTA = 1;

	static volatile uint8_t cmd_buffer[1];

	/* 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) {

		/* Receive the full buffer command. */
		UART_ReceivePayload(cmd_buffer, 1);

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

		/* Parse command buffer. */
		switch(cmd_buffer[0]) {
			case 'F': {
```

```
                                        // turn off LED.
                                        PORTA = 0;
                                        break;

                                }
                                case 'O': {
                                        // turn on LED.
                                        PORTA = 1;
                                        break;
                                }
                                default: {
                                        /* Do nothing. */
                                }
                        }
                }
                return 0;
}
```

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:

PC Bridge system:
AVR based system to receive 1-byte command via UART module to control a LED:

- Turn on LED when ‘O’ character is received
- Turn off LED when ‘F’ character is received

Extended Feature (Mandatory):
Choose only one of the following features:

- Control DC Motor speed and direction using commands received via UART module.
- Create a GUI Application with two buttons to control the LED.