Module 3 - Serial Port

- Introduction to USART
- USART Register Architecture of ATmega328p
- Implementation and Assembly Code Examples

Introduction to USART

1. USART Definition

USART (Universal Synchronous/Asynchronous Receiver/Transmitter) is a communication protocol used to transfer data between electronic devices, such as microcontrollers, sensors, and other components. This protocol is highly flexible as it supports two main modes:

- **Synchronous**
- **Asynchronous**

2. UART vs. USART

While often used interchangeably, there is a technical difference:

- **UART (Universal Asynchronous Receiver/Transmitter):** Supports only asynchronous communication. It requires no clock signal as it relies on start/stop bits and pre-defined baud rates.
- **USART (Universal Synchronous/Asynchronous Receiver/Transmitter):** A superset of UART. It supports both asynchronous and synchronous modes. In synchronous mode, a dedicated clock pin (XCK) is used to synchronize data.

3. Operating Modes

A. Asynchronous Mode

In this mode, the USART module transmits data without an external clock signal. Synchronization is achieved using data frames consisting of:

- **Start Bit:** Indicates the beginning of transmission.
- **Data Bits:** Contains the primary information (typically 5-9 bits).
- **Parity Bit (Optional):** Used for error detection (Even, Odd, or None).
- **Stop Bit:** Indicates the end of transmission.

> **Characteristics:** Suitable for long-distance communication or between devices that do not share the same clock.

B. Synchronous Mode

Uses a clock signal to synchronize data transfer between the transmitter and the receiver.

- Requires the same clock configuration on both devices.
- Allows for faster and more reliable data transfer compared to asynchronous mode.

> **Characteristics:** Ideal for multimedia applications or high-speed bulk data transfers.

4. Configuration and Baud Rate

To use USART, several parameters must be defined:

1. **Baud Rate:** The speed of data transmission (bits per second).
2. **Data Format:** The number of data bits, parity, and stop bits.
3. **Interrupt:** Enables notifications when data is finished being sent or received.

5. USART and Arduino Serial Monitor

In the Arduino ecosystem (such as the Uno), the USART peripheral is the primary way the microcontroller communicates with your computer:

1. **Hardware Connection:** The ATmega328P uses its USART pins (TX on Pin 1, RX on Pin 0). These are connected to an onboard USB-to-Serial converter chip.
2. **Serial Monitor:** When you open the **Serial Monitor** or **Serial Plotter** in the Arduino IDE, it acts as the "Receiver" (RX) for data sent by the Arduino and the "Transmitter" (TX) for data you type in.
3. **Baud Rate Alignment:** For communication to work, the baud rate selected in the Serial Monitor (e.g., 9600) must match the baud rate configured in your code. If they do not match, you will see "garbage" characters or no data at all.

In Proteus, follow these steps to simulate using a Virtual Terminal to mimic Serial Monitor functionality:

1. Click on **Virtual Instruments Mode** in the left sidebar.

picture 0

2. Select the **Virtual Terminal** component and place it on the schematic.
   picture 1
3. Add an **Arduino Uno** to your schematic. Then, connect the **TX** and **RX** pins of the Virtual Terminal to the **RX** and **TX** pins of the Arduino Uno respectively.
   picture 2
4. Double-click on the Virtual Terminal to set the **Baud Rate** (e.g., 9600) to match your code.
   picture 3

Baud Rate Calculation Formula (UBRR)

The **UBRR (USART Baud Rate Register)** value is calculated based on the CPU clock frequency (F_CPU) and the desired Baud Rate:

> **UBRR = (F_CPU / (16 * BAUD)) - 1**

Calculation Example:
If F_CPU = 16 MHz and the target BAUD = 9600 bps:

1. UBRR = (16,000,000 / (16 * 9600)) - 1
2. UBRR = 104.16 - 1
3. **UBRR ≈ 103** (Hex: 0x67)

USART Register Architecture of ATmega328p

The ATmega328p microcontroller uses several specific registers to control and monitor USART communication.

1. UBRR (USART Baud Rate Register)

picture 1

A 16-bit register that determines the communication speed. It is divided into two 8-bit registers:

- **UBRR0H:** Stores the 8 most significant bits (MSB).
- **UBRR0L:** Stores the 8 least significant bits (LSB).

The following table provides a reference for **UBRR0** (USART Baud Rate Register) settings corresponding to standard baud rates (bps). It details the required register values for three common oscillator frequencies (**f_osc**): 16.0000 MHz, 18.4320 MHz, and 20.0000 MHz. For each frequency, the table accounts for both normal speed (**U2Xn = 0**) and double speed (**U2Xn = 1**) modes, including the resulting percentage error for each configuration.

picture 0

2. UDR (USART Data Register)

picture 2

An 8-bit register that serves a dual purpose:

- **TXB (Transmit Data Buffer):** The location where data to be sent is written.
- **RXB (Receive Data Buffer):** The location where incoming data is read.

3. UCSR0A (USART Control and Status Register A)

picture 3

Used to monitor communication status and configure the speed mode.

Bit	Name	Description
RXC0	Receive Complete	Set to 1 if there is new unread data in the UDR.
TXC0	Transmit Complete	Set to 1 if all data has been transmitted.
UDRE0	UDR Empty	Set to 1 if the UDR register is empty and ready for new data.
FE0	Frame Error	Occurs when there is an error in the stop bit.
DOR0	Data Overrun	Occurs when new data arrives before old data is read.
UPE0	Parity Error	Occurs when there is a parity error in the received data.
U2X0	Double Speed	If set to 1, the transmission speed is doubled.
MPCM0	Multi-processor	Enables multi-processor communication mode.

4. UCSR0B (USART Control and Status Register B)

picture 4

Used to enable the module and interrupts.

- **RXCIE0:** Enables the receive complete interrupt.
- **TXCIE0:** Enables the transmit complete interrupt.
- **UDRIE0:** Enables the data register empty interrupt.
- **RXEN0:** Enables the Receiver.
- **TXEN0:** Enables the Transmitter.
- **UCSZ02:** Additional bit (along with UCSR0C) to determine data size (5-9 bits).
- **RXB80 / TXB80:** Holds the 9th data bit (if using 9-bit format).

5. UCSR0C (USART Control and Status Register C)

picture 5

Used for frame format configuration and operating mode.

- **UMSEL01:0:** Selects the mode (Asynchronous or Synchronous).
- **UPM01:0:** Selects the Parity mode (None, Even, or Odd).
- **USBS0:** Selects the number of Stop Bits (1 or 2).
- **UCSZ01:0:** Determines the data size (paired with UCSZ02 in UCSR0B).
- **UCPOL0:** Clock polarity for synchronous mode.

> **Note:** When writing to UCSR0C, ensure bit configurations are performed carefully according to the communication protocol requirements of the target device.

Implementation and Assembly Code Examples

This page contains basic implementation examples of USART serial communication using the Assembly programming language on an AVR Microcontroller (ATmega328p).

1. Printing Text to Serial Monitor

This code is used to repeatedly send the string "Programming Serial Interface!" to the Serial Monitor via the USART port.

```
;------------------------
; Assembly Code - Print Text
;------------------------
#define __SFR_OFFSET 0x00
#include "avr/io.h"
;------------------------
.global main

main:
    CLR   R24
    STS   UCSR0A, R24               ; Clear UCSR0A register
    STS   UBRR0H, R24               ; Clear UBRR0H
    LDI   R24, 103                  ; Set UBRR value = 103 (9600 Baud Rate)
    STS   UBRR0L, R24

    LDI   R24, (1<<RXEN0) | (1<<TXEN0) ; Enable RX and TX
    STS   UCSR0B, R24

    LDI   R24, (1<<UCSZ00) | (1<<UCSZ01) ; Mode: 8-bit data, 1 stop bit, No Parity
    STS   UCSR0C, R24

print_msg:
    LDI   R30, lo8(message)
    LDI   R31, hi8(message)         ; Z points to string message
agn:
```

```
    LPM   R18, Z+                   ; Load character into R18
    CPI   R18, 0                    ; Check if end of string (null)
    BREQ  ext                       ; If yes, exit loop

l1:
    LDS   R17, UCSR0A
    SBRS  R17, UDRE0                ; Wait until buffer is empty (UDRE0=1)
    RJMP  l1
    STS   UDR0, R18                 ; Send character to Serial Monitor
    RJMP  agn                       ; Loop to next character

ext:
    RCALL delay_sec                 ; Wait for a moment
    RJMP  print_msg                 ; Repeat string transmission

message:
    .ascii "Programming Serial Interface!"
    .byte 10, 13, 0

delay_sec:                          ; Delay Subroutine (~3 seconds)
    LDI   R20, 255
l4: LDI   R21, 255
l5: LDI   R22, 255
l6: DEC   R22
    BRNE  l6
    DEC   R21
    BRNE  l5
    DEC   R20
    BRNE  l4
    RET
```

---

2. Reading Input from Serial Monitor

This code reads characters sent from the Serial Monitor and controls an LED. If the character 'H' is received, the LED turns ON; if 'L' is received, the LED turns OFF.

```
;------------------------
; Assembly Code - Input Text and Control LED
;------------------------
```

```
#define __SFR_OFFSET 0x00
#include "avr/io.h"
;------------------------
.global main

main:
    CLR   R24
    STS   UBRR0H, R24
    LDI   R24, 103
    STS   UBRR0L, R24
    LDI   R24, (1<<RXEN0 | 1<<TXEN0)
    STS   UCSR0B, R24
    LDI   R24, (1<<UCSZ01 | 1<<UCSZ00)
    STS   UCSR0C, R24

    SBI   DDRB, 5               ; Set PB5 as output

wait_input:
    ; 1. Check if a byte arrived
    LDS   R17, UCSR0A
    SBRS  R17, RXC0             ; Wait for Receive Complete
    RJMP  wait_input

    ; 2. Read the character into R18
    LDS   R18, UDR0

    ; 3. Check if character is 'H'
    CPI   R18, 'H'
    BREQ  led_on

    ; 4. Check if character is 'L'
    CPI   R18, 'L'
    BREQ  led_off

    RJMP  wait_input

led_on:
    SBI   PORTB, 5              ; Turn LED ON
    RJMP  wait_input

led_off:
```

```
    CBI   PORTB, 5              ; Turn LED OFF
    RJMP  wait_input
```