**Page 1: Basics of Serial Communication**

**What is Serial Communication?**

* When a microcontroller (like the ATmega328P) needs to "talk" to another device (e.g., a computer or another microcontroller), it sends data in **bytes** (8-bit chunks).
* There are two ways to send data:
  + **Parallel**: All 8 bits of a byte are sent at once over 8 separate wires. This is fast but only works over short distances because long cables can distort signals.
  + **Serial**: Bits are sent one at a time over a single wire, making it suitable for long-distance communication (from hundreds of feet to millions of miles).
* The document includes a figure (not provided in the text) comparing **serial vs. parallel data transfer**.

**How Serial Communication Works**

* To send a byte serially, the microcontroller uses a **parallel-in-serial-out shift register** to convert the 8-bit byte into a stream of individual bits.
* At the receiving end, a **serial-in-parallel-out shift register** collects the bits and reassembles them into a byte.
* Serial communication can be:
  + **Synchronous**: A separate clock signal is sent along with the data to synchronize the sender and receiver.
  + **Asynchronous**: No clock signal is sent; instead, the sender and receiver agree on a **baud rate** (speed of data transfer, measured in bits per second or **bps**).
* The lab focuses on **asynchronous communication**, which is simpler and doesn’t require a clock wire.

**Baud Rate**

* The **baud rate** determines how fast data is sent or received (e.g., 9600 bps means 9600 bits per second).
* It’s also called the number of **signal changes per second**.
* The baud rate depends on the communication ports of the devices involved, and the ATmega328P allows you to set this rate programmatically.

**USART Chip**

* To make serial communication easier, microcontrollers like the ATmega328P have a built-in **USART (Universal Synchronous-Asynchronous Receiver/Transmitter)** chip.
* The USART handles the complex task of converting data between parallel and serial formats, so you don’t have to write lengthy code to manage it.

**Page 2: USART in ATmega328P**

**USART Overview**

* The ATmega328P’s **USART** module supports both synchronous and asynchronous modes, but this lab uses **asynchronous mode** for **full-duplex communication** (sending and receiving data at the same time).
* This allows the microcontroller to connect to a PC’s serial port for data exchange.

**TX and RX Pins**

* The ATmega328P has two specific pins for serial communication:
  + **TX (Transmit, PD1)**: Sends data from the microcontroller.
  + **RX (Receive, PD0)**: Receives data sent to the microcontroller.
* These pins use **TTL logic (Transister-Transister Logic)** (0V for low, 5V for high), which is compatible with the microcontroller but not directly with older **RS232 serial ports** (which use different voltage levels).
* To connect to a PC, a **USB-to-RS232 converter** is used. This device converts the microcontroller’s TTL signals to USB signals (and vice versa) so the ATmega328P can communicate with a PC’s USB port.

**Setting the Baud Rate with UBRR**

* The **baud rate** (speed of data transfer) is set using a 16-bit register called **UBRR (USART Baud Rate Register)**, which is split into two 8-bit registers:
  + **UBRRnH** (high byte): Holds the upper 4 bits.
  + **UBRRnL** (low byte): Holds the lower 8 bits.
* Only 12 bits of the UBRR are used to determine the baud rate.

A screenshot of a computer

AI-generated content may be incorrect.



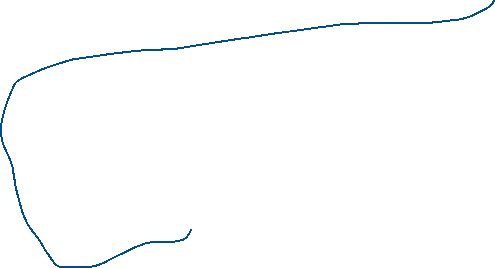
A white rectangular with blue border and black text

AI-generated content may be incorrect.



**Where, X is the value to be loaded to UVRR register.**

* The UBRR value is calculated based on the microcontroller’s **system clock (Fosc)** and the desired baud rate. For example:



A white background with black text

AI-generated content may be incorrect.



* + You load this value (103) into the UBRR registers to set the baud rate to 9600 bps.

**Page 3: UBRR and USART Data Register (UDR)**

**A diagram of a computer flow

AI-generated content may be incorrect.**

**How UBRR (USART Baud rate Register ) generates the Baud Rate**

* The UBRR works with a **down-counter** (a hardware component) to generate the baud rate. A screenshot of a computer

  AI-generated content may be incorrect.



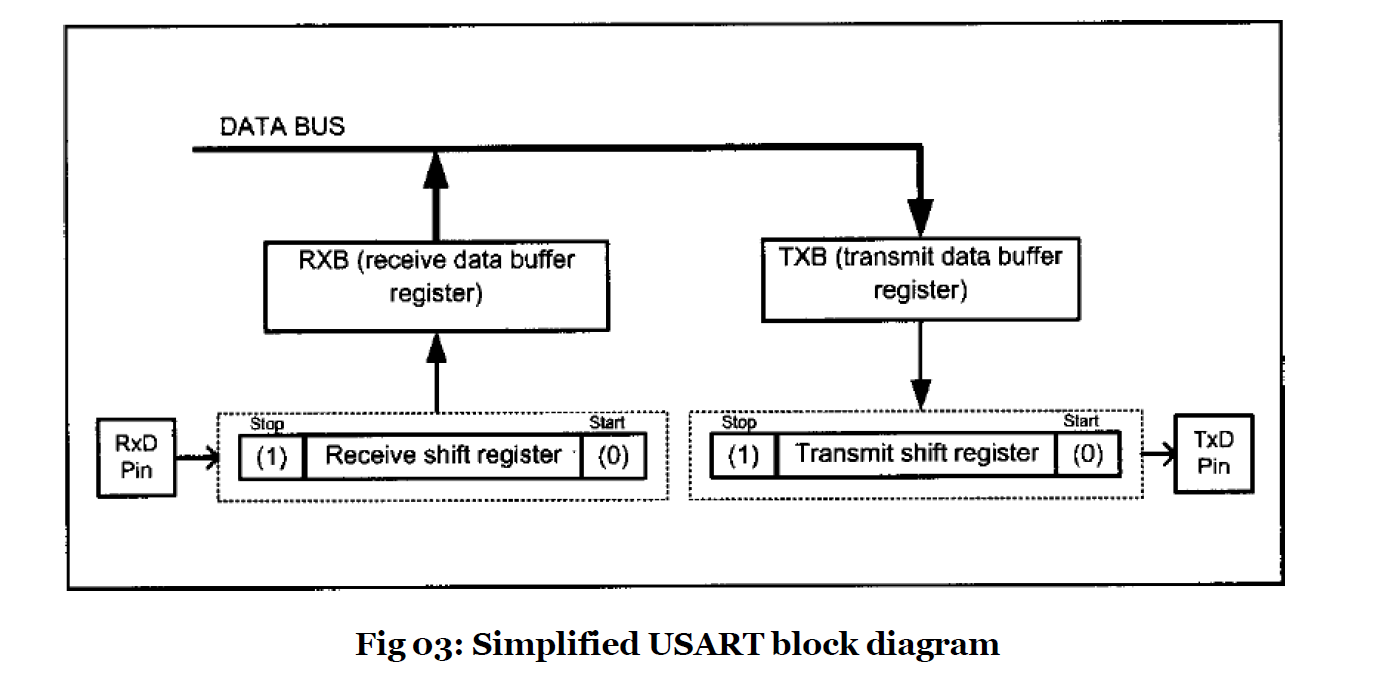
* The microcontroller’s **system clock (Fosc)** feeds into the down-counter, which divides the clock frequency based on the UBRR value.
* When the counter reaches zero, it generates a clock pulse for serial communication and reloads the UBRR value to start counting down again.



* The resulting frequency is further divided (by 2, 4, or 2, depending on the mode) to achieve the final baud rate.

**USART Data Register (UDRo)**

* The **UDRo (USART Data Register)** is used to send and receive data.
* The ATmega328P has two **shift registers** for full-duplex communication:
  + **Transmit Shift Register**: Converts a byte into a serial bit stream for sending.
  + **Receive Shift Register**: Converts a received serial bit stream back into a byte.
* Each shift register has a **buffer**:
  + **Transmit Data Buffer**: Holds the byte you want to send.
  + **Receive Data Buffer**: Holds the byte received.
* Both buffers share the same **UDR I/O address**:
  + Writing to UDR sends data to the **Transmit Data Buffer (TXB)**.
  + Reading from UDR retrieves data from the **Receive Data Buffer (RXB)**.



* This shared address simplifies programming, as you use the same register (UDR) for both sending and receiving.

**UCSR0 registers (USART Control Status Registers) and USART configurations in the AVR**

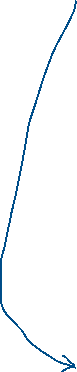
**Page 4: Programming and Exercises**

**Programming the AVR for Serial Transmission (Polling Method)**

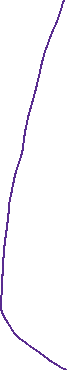


To **send data** serially:

1. **Enable the Transmitter**: Set the **TXEN0 bit** in the **UCSR0B register** to activate the transmitter. This configures the TX pin (PD1) for serial output.
2. **Set Asynchronous Mode**: Configure the **UCSR0C register** for an 8-bit data frame, no parity, and one stop bit (standard settings for serial communication).
3. **Set Baud Rate**: Load the **UBRR0 register** with the calculated value for your desired baud rate (e.g., 103 for 9600 bps with a 16 MHz clock).
4. **Write Data**: Place the byte you want to send into the **UDR0 register**.



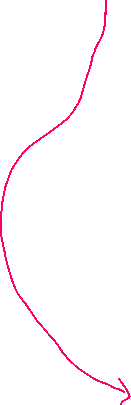
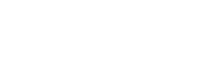
1. **Check Readiness**: Monitor the **UDRE0 bit** in the **UCSR0A register**. When it’s high, the UDR0 is ready to accept the next byte.



1. **Repeat**: Go back to step 4 to send the next byte.

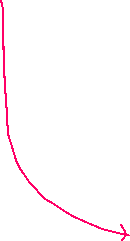
A computer screen with text

AI-generated content may be incorrect.



A screen shot of a computer code

AI-generated content may be incorrect.



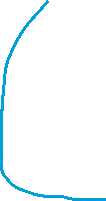
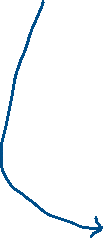
**Programming the AVR for Serial Reception (Polling Method)**



To **receive data** serially:

1. **Enable the Receiver**: Set the **RXEN0 bit** in the **UCSR0B register** to activate the receiver. This configures the RX pin (PD0) for serial input.
2. **Set Asynchronous Mode**: Same as for transmission (8-bit data, no parity, one stop bit in UCSR0C).
3. **Set Baud Rate**: Load the UBRR0 register with the same baud rate value used for transmission.
4. **Check for Data**: Monitor the **RXC0 bit** in the **UCSR0A register**. When it’s high, a complete byte has been received and is available in UDR0.
5. **Read Data**: Read the byte from the **UDR0 register**.
6. A screen shot of a computer code

   AI-generated content may be incorrect.**Repeat**: Go back to step 4 to receive the next byte.



A screenshot of a computer error

AI-generated content may be incorrect.

**Exercises explanation**

The lab includes three programming tasks to practice serial communication:

1. **Write Basic USART Functions**:
   * **usart\_init()**: Initializes the USART for 9600 baud, 8-bit data, no parity, and 1 stop bit.
   * **usart\_send(char)**: Sends a single character via the USART.
   * **usart\_receive()**: Returns a single character received via the USART.
   * These functions are building blocks for the next tasks.
2. **Transmit Names and B-Numbers**:
   * Write a C program to send the names and B-numbers (likely student IDs) of your group mates to the PC via serial communication.
3. **Caesar Cipher Program**:
   * Write a program that:
     + **Receives a sentence** from the PC (using ASCII characters), detecting the end of the sentence with a **carriage return** (Enter key, ASCII 13).
     + **Encrypts the sentence** using a **Caesar cipher** with a key of 3 (shifts each alphabetic character by 3 positions, e.g., A → D, B → E, ..., Z → C). Non-alphabetic characters (like spaces or punctuation) remain unchanged.
     + **Sends the encrypted sentence** back to the PC via serial communication.

**Key Concepts in Simple Terms**

* **Serial Communication**: Sending data one bit at a time over a single wire, ideal for long-distance communication.
* **USART**: A hardware module in the ATmega328P that handles serial communication, making it easy to send and receive data.
* **TX and RX Pins**: TX sends data, RX receives data. They need a USB-to-RS232 converter to connect to a PC’s USB port.
* **Baud Rate**: The speed of data transfer, set using the UBRR register based on the microcontroller’s clock.
* **UDR Register**: Used to send data (write to UDR) or receive data (read from UDR).
* **Polling Method**: The program continuously checks (polls) if the USART is ready to send or receive data, rather than using interrupts.
* **Exercises**: Practical tasks to program the ATmega328P to send/receive data and perform a simple encryption (Caesar cipher).

**Why This Matters**

This lab teaches you how to use the ATmega328P’s USART to communicate with a PC, which is a fundamental skill in embedded systems. Serial communication is widely used for debugging (e.g., printing messages to a terminal), connecting to sensors, or interfacing with other devices. The exercises help you practice initializing the USART, sending/receiving data, and applying a simple algorithm (Caesar cipher) to manipulate data.

more detail, let me know!