

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

میکرو کنترلرهای AVR ارتباط سریال

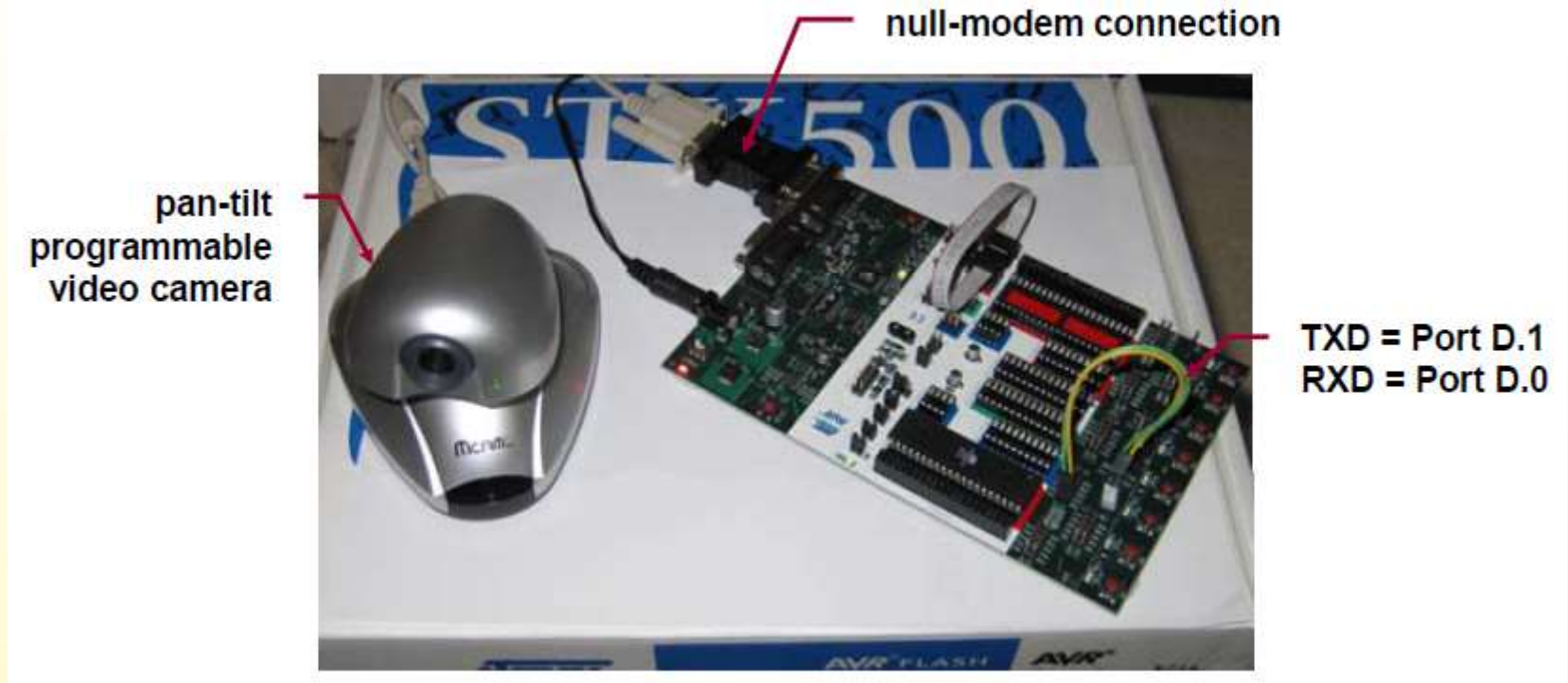
دانشکده برق و رباتیک
دانشگاه صنعتی شاهرود

حسین خسروی

۱۳۹۰-۹۱

An application of serial communications

An STK500 board is programmed to control a pan-tilt video camera, via a serial connection. In this lecture, you'll learn to create such a program.



video link: [avr]/ecte333/pan_tilt_camera.mp4
[avr] = <http://www.elec.uow.edu.au/avr>

Serial communications – The basics

- **Computers transfer data in two ways: parallel and serial.**
 - ❑ **Parallel:** Several data bits are transferred simultaneously, e.g. to printers and hard disks.
 - ❑ **Serial:** A single data bit is transferred at one time.

- **Advantages of serial communications: longer distances, easier to synchronize, fewer IO pins, and lower cost.**

- **Serial communications often require**
 - ❑ **Shift registers:** convert a byte to serial bits and vice versa.
 - ❑ **Modems:** modulate/demodulate serial bits to/from audio tones.

Synchronous versus asynchronous

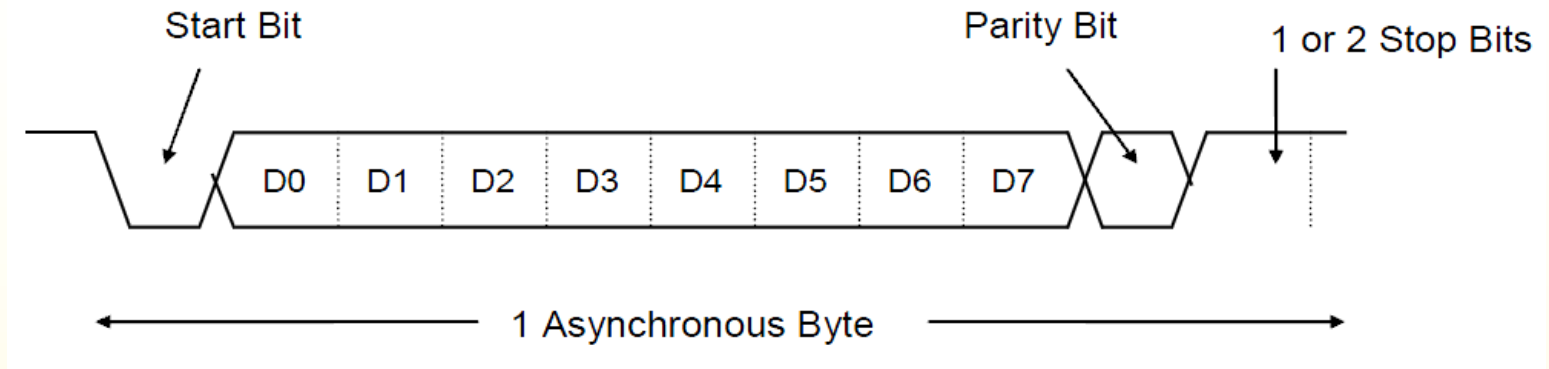
➤ Synchronous serial communications

- ❑ The clocks of the sender and receiver are synchronized (shared).
- ❑ A block of characters, enclosed by synchronizing bytes, is sent at a time.
- ❑ Faster transfer and less overhead.
- ❑ **Examples:** serial peripheral interface (SPI) by Motorola, binary synchronous communication (BISYNC) by IBM.

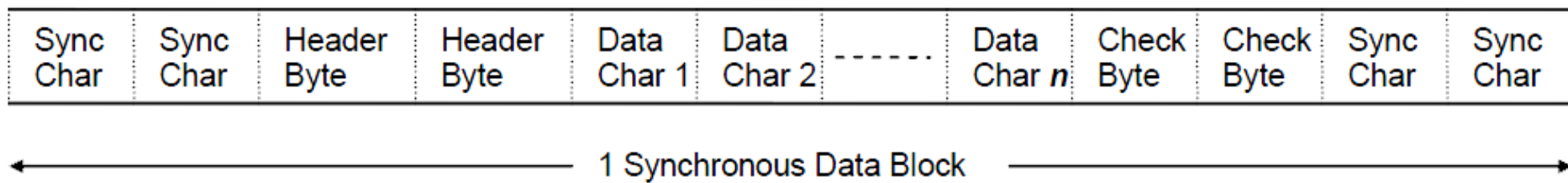
➤ Asynchronous serial communications

- ❑ The transmitter and receiver do not share a common clock.
- ❑ One character (8 or 7 bits) is sent at a time, enclosed between a start bit and one or two stop bits. A parity bit may be included.
- ❑ **Examples:**
RS232 (part of) by Electronic Industry Alliance.
USART of ATmega16

Synchronous versus asynchronous



Asynchronous Serial Data Format



Synchronous Serial Data Format

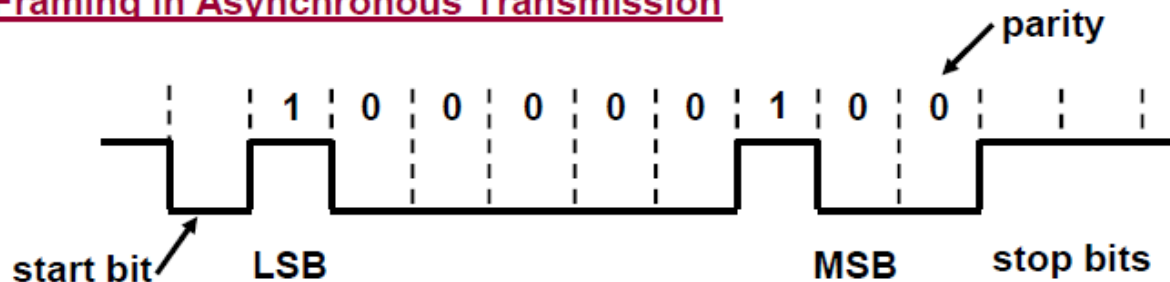
Data framing examples

Data Framing in Synchronous BISYNC



BISYNC Control Characters
SYN (16h): synchronisation
STX (02h): start of text
ETX (03h): end of text
BCC: block checksum char
PAD (FFh): end of frame block

Data Framing in Asynchronous Transmission



Sending character "A" (41h = 0100 0001)
8-bit data, 1 start bit, 2 stop bits, even-parity

Serial communications terminology

- **Baud rate**: the number of bits sent per second (bps). Strictly speaking, baud rate is the number of signal changes per second.
- **Parity bit**: a single bit used for error checking that is sent together with data bits to make the total number of 1's
 - ❑ even (for even parity) or
 - ❑ odd (for odd parity).
- **Start bit**: to indicate the start of a character. Its typical value is **0**.
- **Stop bit**: to indicate the end of a character. Its typical value is **1**.

The RS232 standard

- The RS232 is a widely used standard for serial interfacing.
- The latest revision is RS232E (July 1991).
- The RS232 standard covers four main aspects:
 - ❑ **Electrical**: voltage level, rise and fall time, data rate, distance.
 - ❑ **Functional**: function of each signal
 - ❑ **Mechanical**: number of pins, shape & dimension of connectors.
 - ❑ **Procedural**: sequence of events for transmitting data.

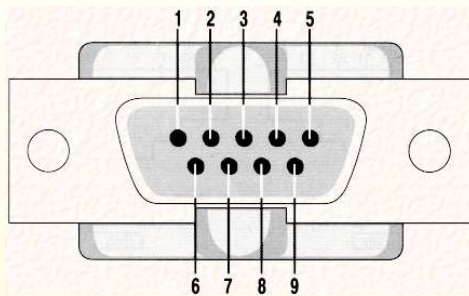
The RS232 standard

- RS232 was set by the Electronics Industries Association (EIA) in 1960
- The standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible
 - ❑ In RS232, a **1** is represented by **-3 ~ -25 V**, while a **0** bit is **+3 ~ +25 V**, making -3 to +3 undefined
- RS232 restricts baud rate to 20 Kbps and cable length to 15m. In practice, it can support **up to 56 Kbps & 30m** of shielded cables.

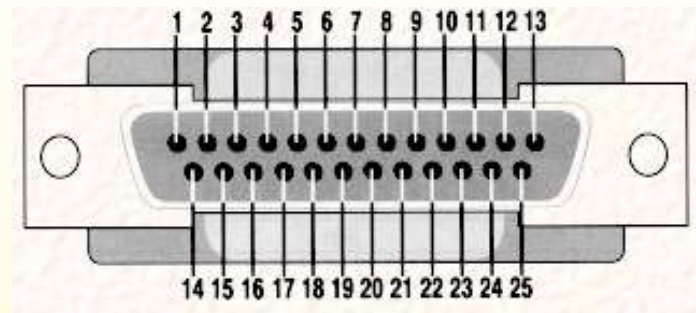
The RS232 standard

- It defines 25-pin D connectors. In many cases, 9-pin connectors are also used.

RS232 Connector DB-9



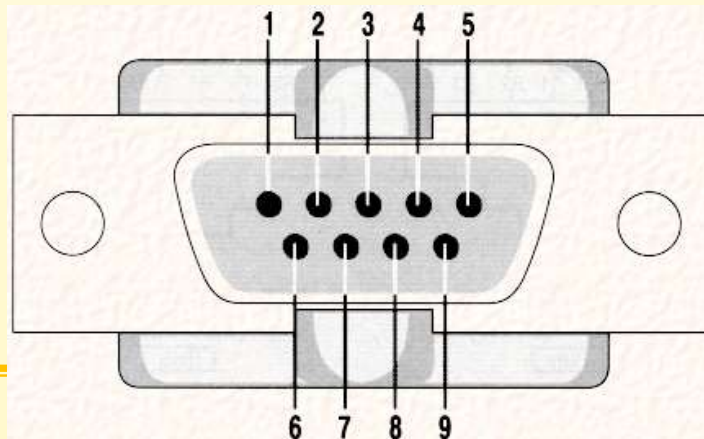
RS232 Connector DB-25



- There are two important terms in the RS232
 - ❑ **Data Terminal Equipment (DTE)** refers to terminal and computers that send and receive data
 - ❑ **Data Communication Equipment (DCE)** refers to communication equipment, such as modems
 - ❑ These definitions are needed to explain the pin functions.

IBM PC 9-pin connector

| Pin | Name | Description |
|-----|-------------------------|--|
| 1 | $\overline{\text{DCD}}$ | Data Carrier Detect: DCE has detected a carrier tone |
| 2 | RXD | Received Data: incoming data from DCE |
| 3 | TXD | Transmit Data: outgoing data to DCE |
| 4 | DTR | Data Terminal Ready: DTE is connected and turned on |
| 5 | GND | Ground |
| 6 | $\overline{\text{DSR}}$ | Data Set Ready: DCE is connected and turned on |
| 7 | $\overline{\text{RTS}}$ | Request To Send: DTE has data to send |
| 8 | $\overline{\text{CTS}}$ | Clear To Send: DCE can receive data |
| 9 | RI | Ring Indicator: synchronized with the phone's ringing tone |



- DTR (data terminal ready)
 - ❑ When terminal is turned on, it sends out signal DTR to indicate that it is ready for communication
- DSR (data set ready)
 - ❑ When DCE is turned on and has gone through the self-test, it assert DSR to indicate that it is ready to communicate
- RTS (request to send)
 - ❑ When the DTE device has byte to transmit, it assert RTS to signal the modem that it has a byte of data to transmit
- CTS (clear to send)
 - ❑ When the modem has room for storing the data it is to receive, it sends out signal CTS to DTE to indicate that it can receive the data now

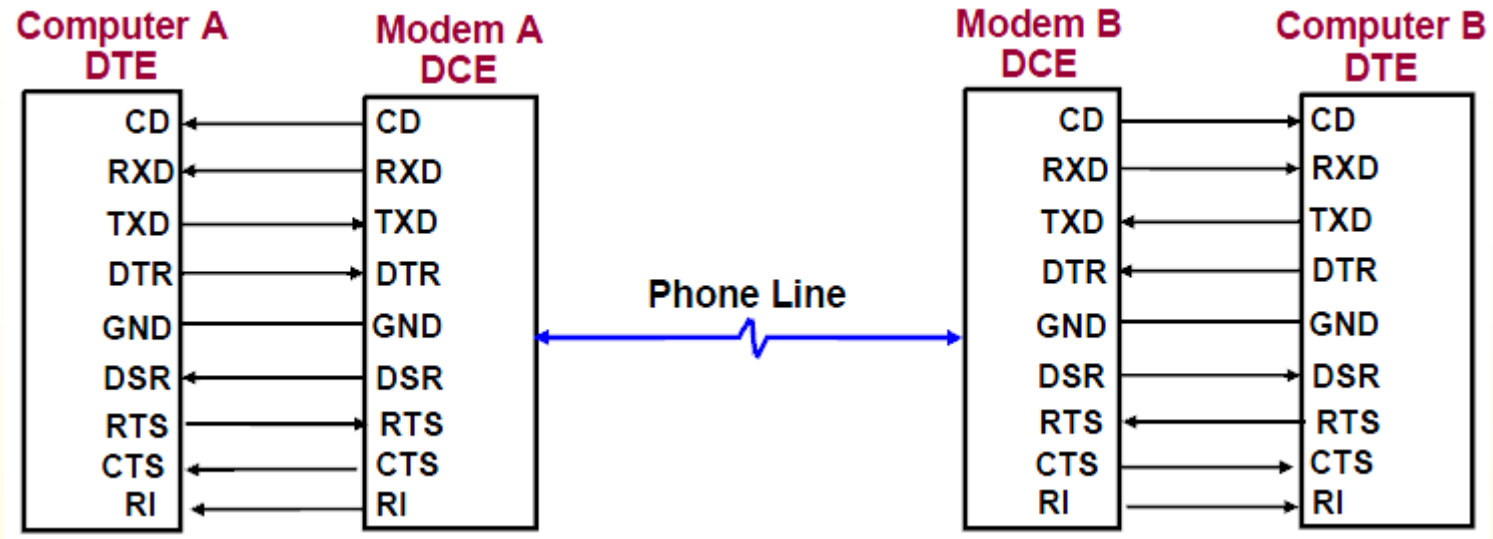
➤ DCD (data carrier detect)

- ❑ The modem asserts signal DCD to inform the DTE that a valid carrier has been detected and that contact between it and the other modem is established

➤ RI (ring indicator)

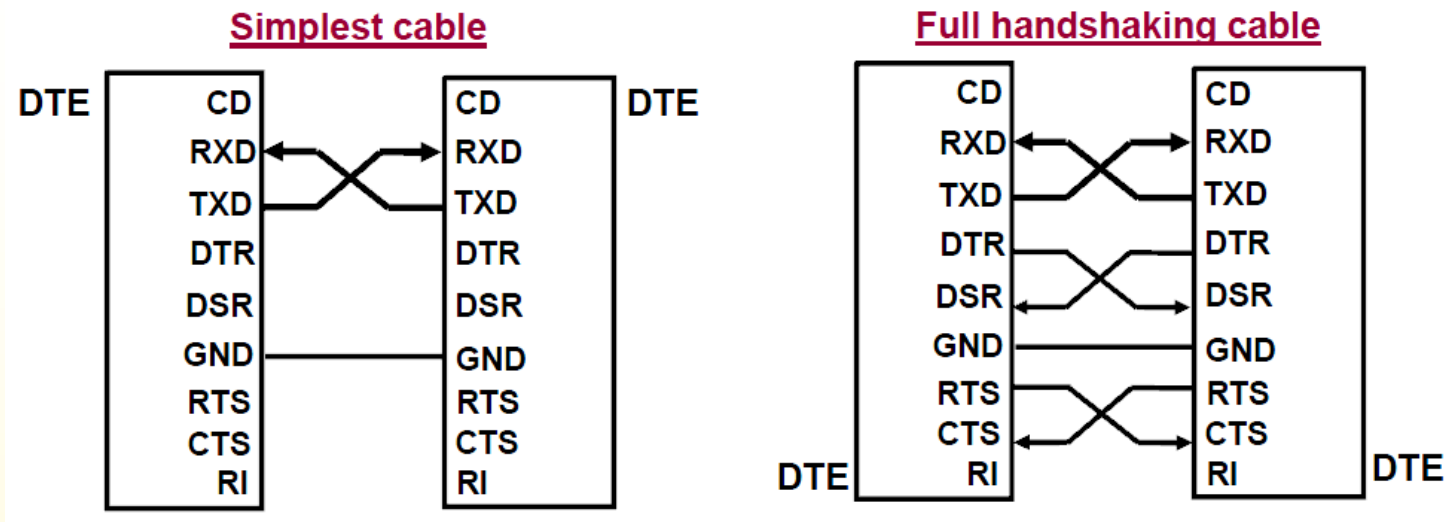
- ❑ An output from the modem and an input to a PC indicates that the telephone is ringing
- ❑ It goes on and off in synchronous with the ringing sound

Modem connection



- RS232 was originally used with modems to connect two PCs over the public phone lines.
- When computer A has data to send, it asserts its RTS pin.
- Modem A will assert its CTS when it is ready to receive.
- Computer A transmits data through its TXD.

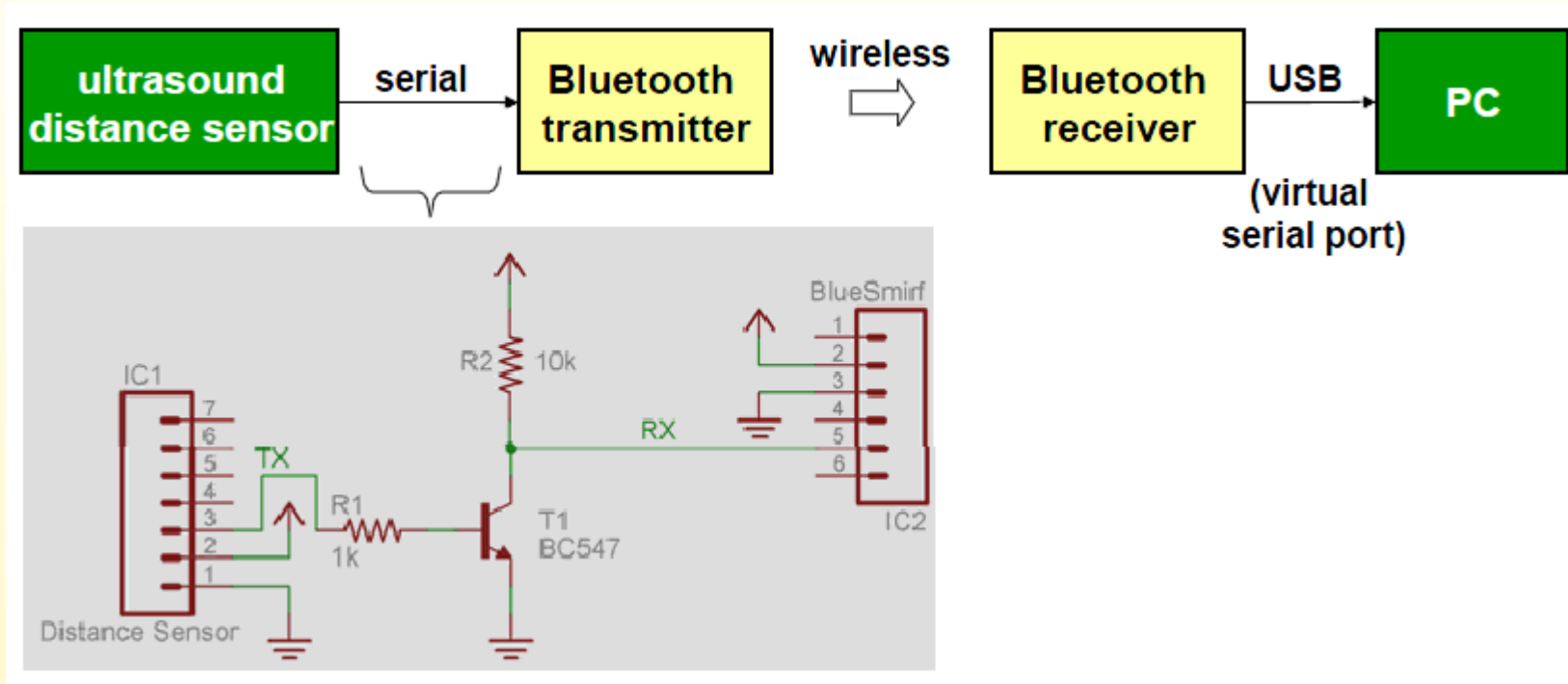
Null-modem connection



- RS232 is now mainly used to connect a microcontroller with PC or peripheral devices (e.g. camera, GPS receiver, infrared range finder).
- This connection configuration is known as null-modem.
- **Key idea:**
 - ❑ Connect pin TXD of a DTE with pin RXD of the other DTE.
 - ❑ Wire other pins to support flow control.

Serial communications — An example

- The sensor sends data via serial interface to Bluetooth transmitter.
- A Bluetooth receiver connected to a PC is configured as a serial port.



Serial communications in ATmega32

- ATmega32 provides three subsystems for serial communications.
 - ❑ Universal Synchronous & Asynchronous Serial Receiver & Transmitter (**USART**)
 - ❑ Serial Peripheral Interface (**SPI**)
 - ❑ Two-wire Serial Interface (**TWI**) also called Inter-Integrated Circuit (**I²C**)

- **USART:**
 - ❑ Supports full-duplex mode between a receiver and transmitter.
 - ❑ Typically used in asynchronous communication.
 - ❑ Start bit and stop bit are used for each byte of data.
 - ❑ **We focus on this subsystem in this lecture.**

Serial communications in ATmega32

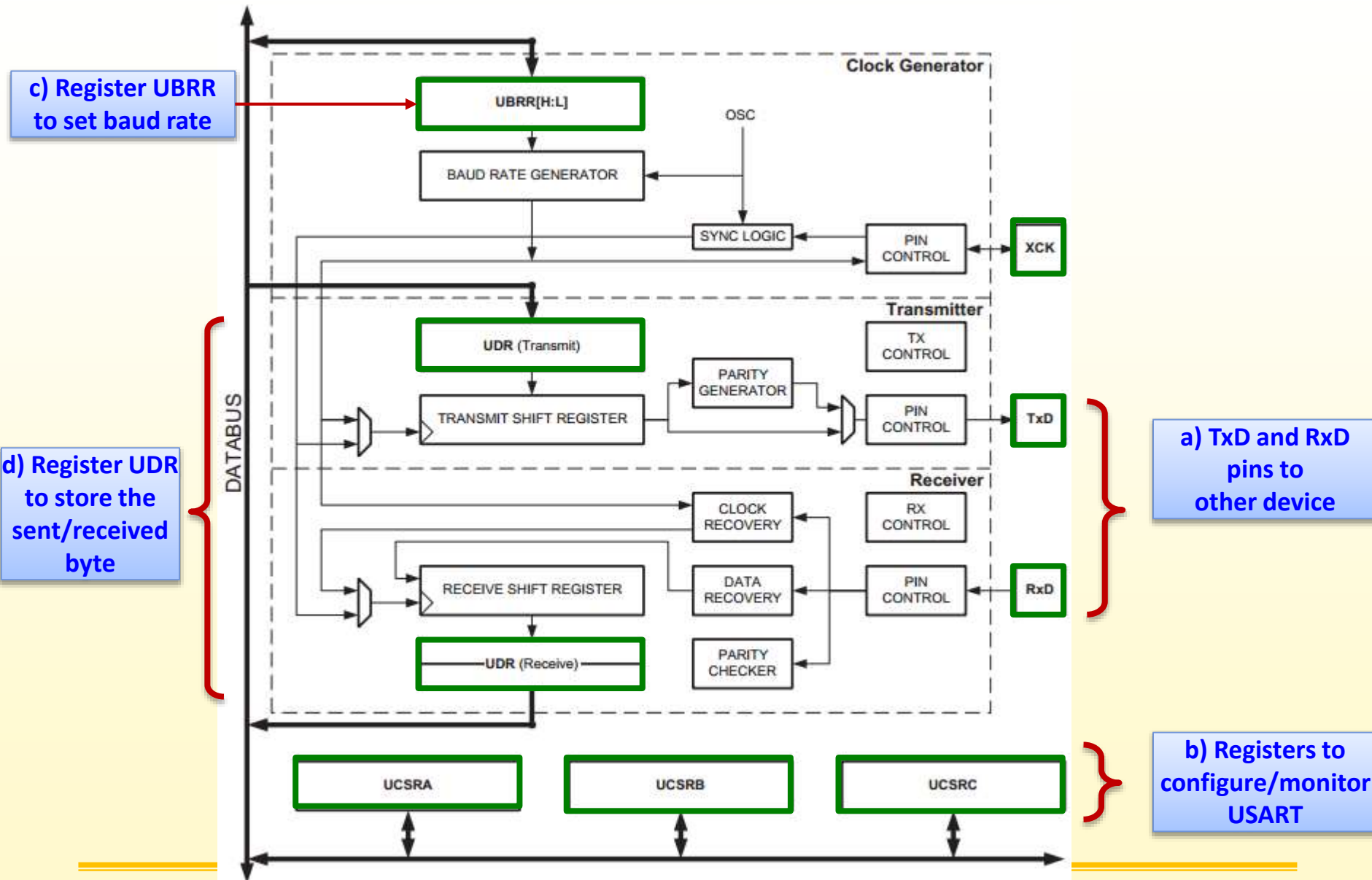
- **Serial Peripheral Interface (SPI)**
 - ❑ The receiver and transmitter share a common clock line.
 - ❑ Supports higher data rates.
 - ❑ The transmitter is designated as the master, the receiver as the slave.
 - ❑ Example of devices using SPI: liquid crystal display, analogue to digital converter.
- **Two-wire Serial Interface (TWI):**
 - ❑ Network several devices such as microcontrollers and display boards, using a two-wire bus.
 - ❑ Up to 128 devices are supported.
 - ❑ Each device has a unique address and can exchange data with other devices in a small network.

Serial USART – An overview

- USART of the ATmega16 supports
 - ❑ baud rates from 960bps up to 57.6kbps,
 - ❑ character size: 5 to 9 bits,
 - ❑ 1 start bit,
 - ❑ 1 or 2 stop bits,
 - ❑ parity bit
 - ❑ (optional: even or odd parity).

| | | | |
|-------------------------|----|----|-------------|
| (XCK/T0) PB0 | 1 | 40 | PA0 (ADC0) |
| (T1) PB1 | 2 | 39 | PA1 (ADC1) |
| (INT2/AIN0) PB2 | 3 | 38 | PA2 (ADC2) |
| (OC0/AIN1) PB3 | 4 | 37 | PA3 (ADC3) |
| (\overline{SS}) PB4 | 5 | 36 | PA4 (ADC4) |
| (MOSI) PB5 | 6 | 35 | PA5 (ADC5) |
| (MISO) PB6 | 7 | 34 | PA6 (ADC6) |
| (SCK) PB7 | 8 | 33 | PA7 (ADC7) |
| \overline{RESET} | 9 | 32 | AREF |
| VCC | 10 | 31 | GND |
| GND | 11 | 30 | AVCC |
| XTAL2 | 12 | 29 | PC7 (TOSC2) |
| XTAL1 | 13 | 28 | PC6 (TOSC1) |
| (RXD) PD0 | 14 | 27 | PC5 (TDI) |
| (TXD) PD1 | 15 | 26 | PC4 (TDO) |
| (INT0) PD2 | 16 | 25 | PC3 (TMS) |
| (INT1) PD3 | 17 | 24 | PC2 (TCK) |
| (OC1B) PD4 | 18 | 23 | PC1 (SDA) |
| (OC1A) PD5 | 19 | 22 | PC0 (SCL) |
| (ICP1) PD6 | 20 | 21 | PD7 (OC2) |

Serial USART – Block diagram



Serial USART – Hardware elements

- **USART Clock Generator:**
 - ❑ to provide clock source.
 - ❑ to set baud rate using UBRR register.

- **USART Transmitter:**
 - ❑ to send a character through **TxD** pin.
 - ❑ to handle start/stop bit framing, parity bit, shift register.

- **USART Receiver:**
 - ❑ to receive a character through **RxD** pin.
 - ❑ to perform the reverse operation of the transmitter.

- **USART Registers:**
 - ❑ to configure, control and monitor the serial USART

Serial USART – Three groups of registers

➤ **USART Baud Rate Registers**

- ❑ **UBRRH and UBRRL**

➤ **USART Control and Status Registers**

- ❑ **UCSRA**

- ❑ **UCSRB**

- ❑ **UCSRC**

➤ **USART Data Registers**

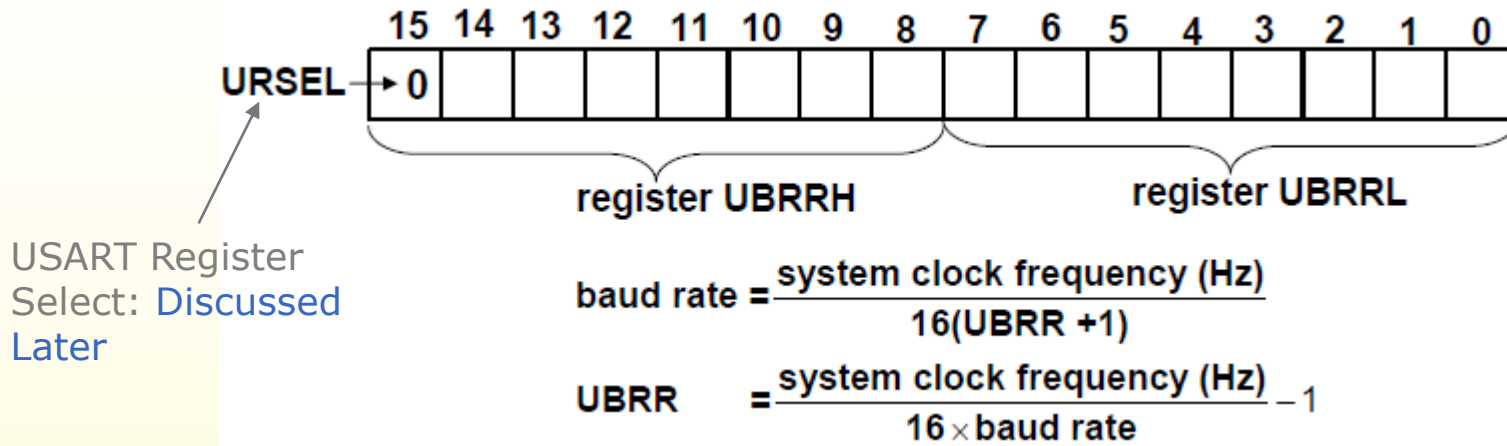
- ❑ **UDR**

➤ **Understanding these registers is essential in using the serial port.**

Therefore, we'll study these registers in depth.

USART Baud Rate Registers

- Two 8-bit registers together define the baud rate.



- **Example:** Find UBRR registers for baud rate of 1200bps, assuming system clock is 1MHz.

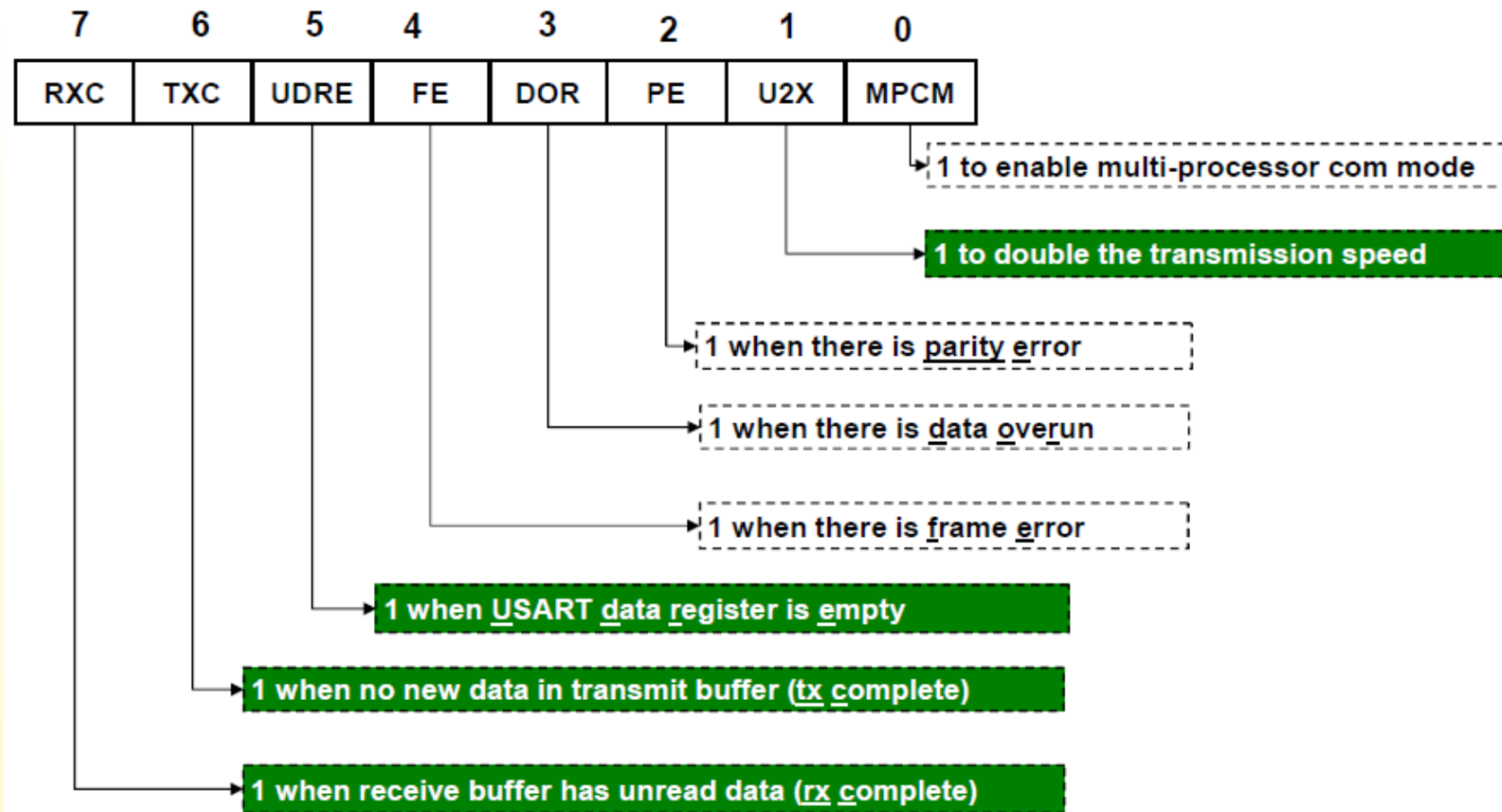
- ☐ $\text{UBRR} = 1000000 / (16 \times 1200) - 1 = 51d = 0033H.$

- ☐ Therefore, $\text{UBRRH} = 00H$ and $\text{UBRRL} = 33H.$

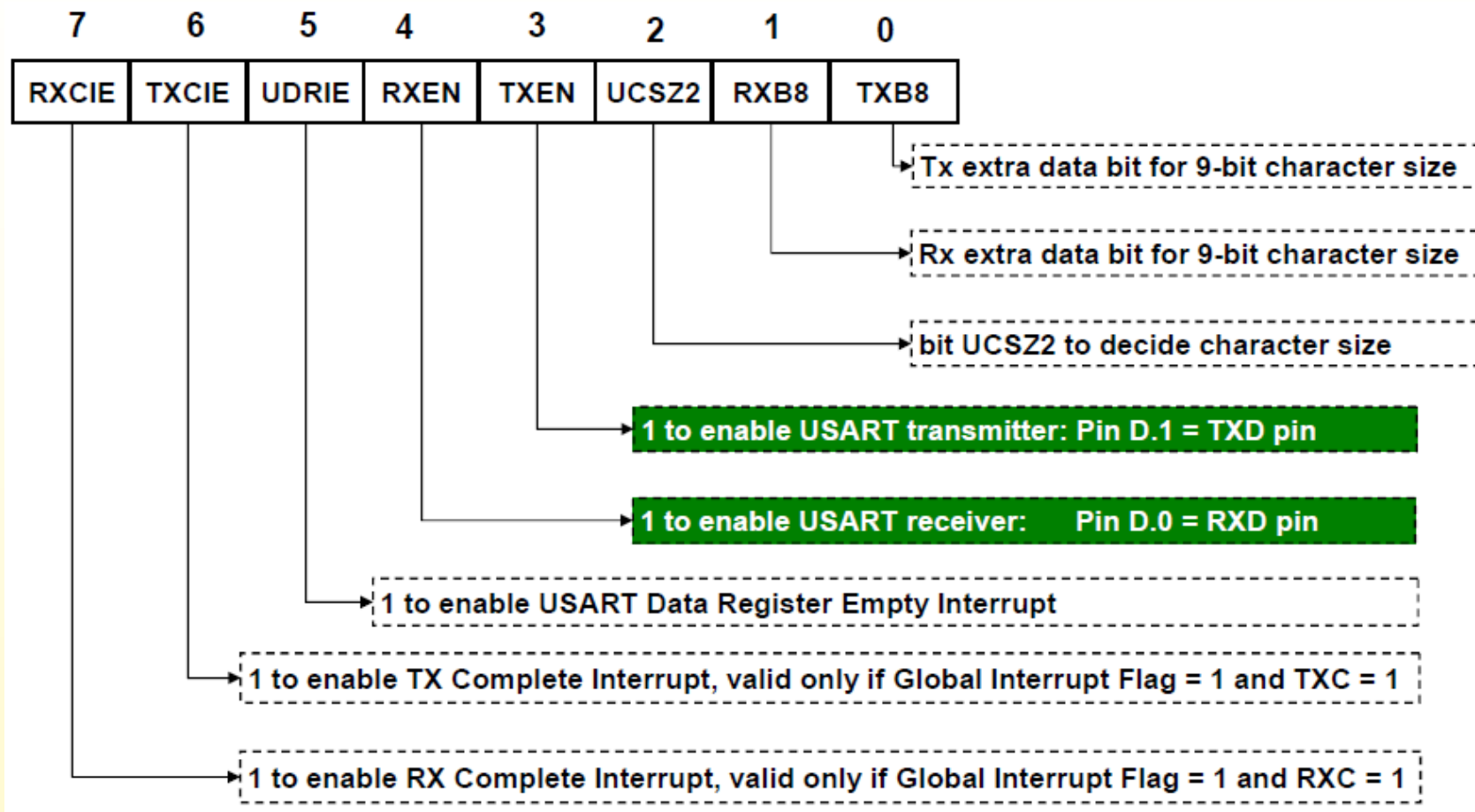
- ☐ C code

- ☐ $\text{UBRRH} = 0x00; \text{UBRRL} = 0x33;$

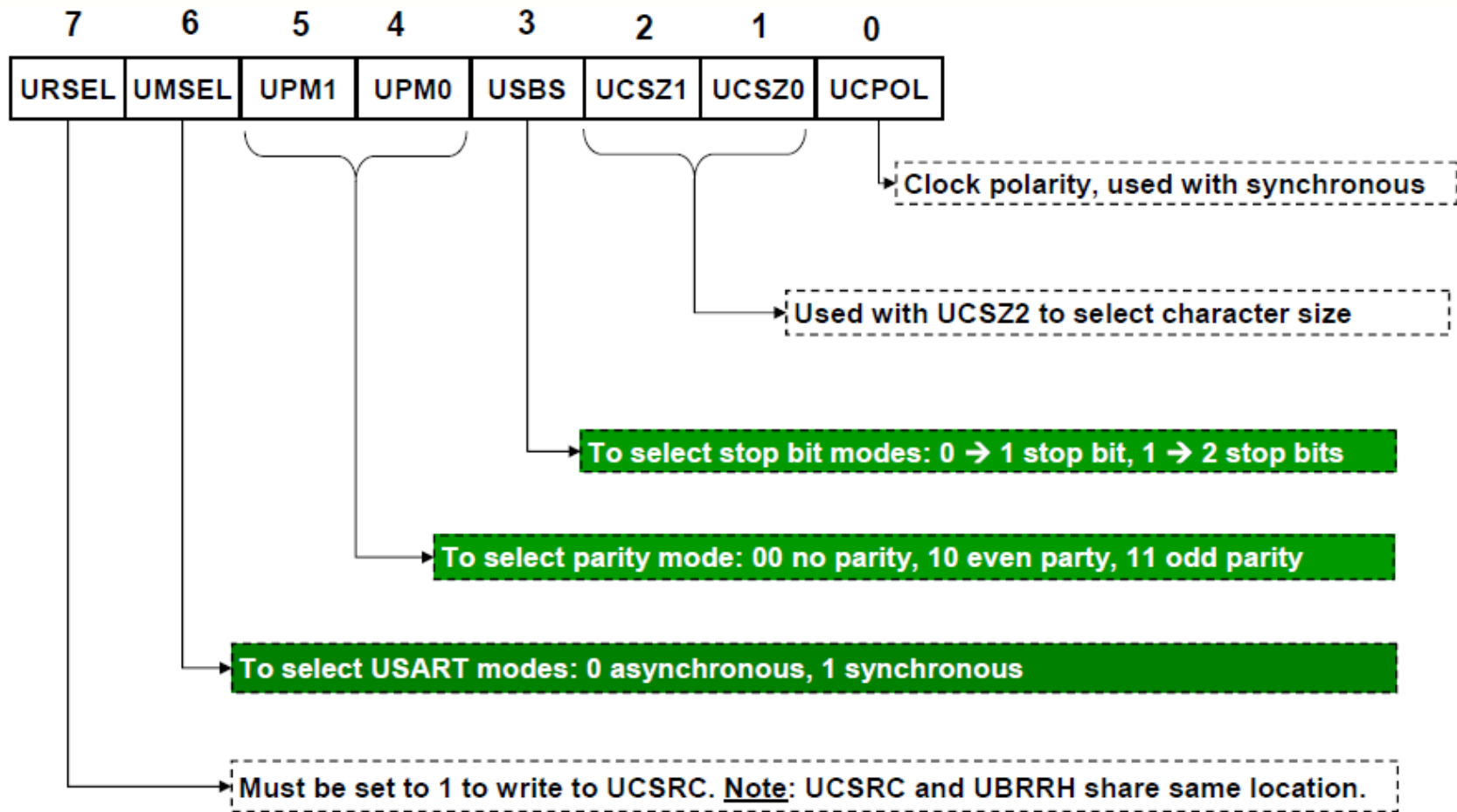
USART Control and Status Register A (UCSRA)



USART Control and Status Register B (UCSRB)



USART Control and Status Register C (UCSRC)



Setting character size

- Character size (5, 6, 7, 8, 9) is determined by three bits
 - ❑ bit **UCSZ2** (in register **UCSRB**),
 - ❑ bit **UCSZ1** and bit **UCSZ0** (in register **UCSRC**).
- **Example:** For a character size of 8 bits, we set
 - ❑ **UCSZ2 = 0, UCSZ1 = 1, and UCSZ0 = 1.**

Table 66. UCSZ Bits Settings

| UCSZ2 | UCSZ1 | UCSZ0 | Character Size |
|--------------|--------------|--------------|-----------------------|
| 0 | 0 | 0 | 5-bit |
| 0 | 0 | 1 | 6-bit |
| 0 | 1 | 0 | 7-bit |
| 0 | 1 | 1 | 8-bit |
| 1 | 0 | 0 | Reserved |
| 1 | 0 | 1 | Reserved |
| 1 | 1 | 0 | Reserved |
| 1 | 1 | 1 | 9-bit |

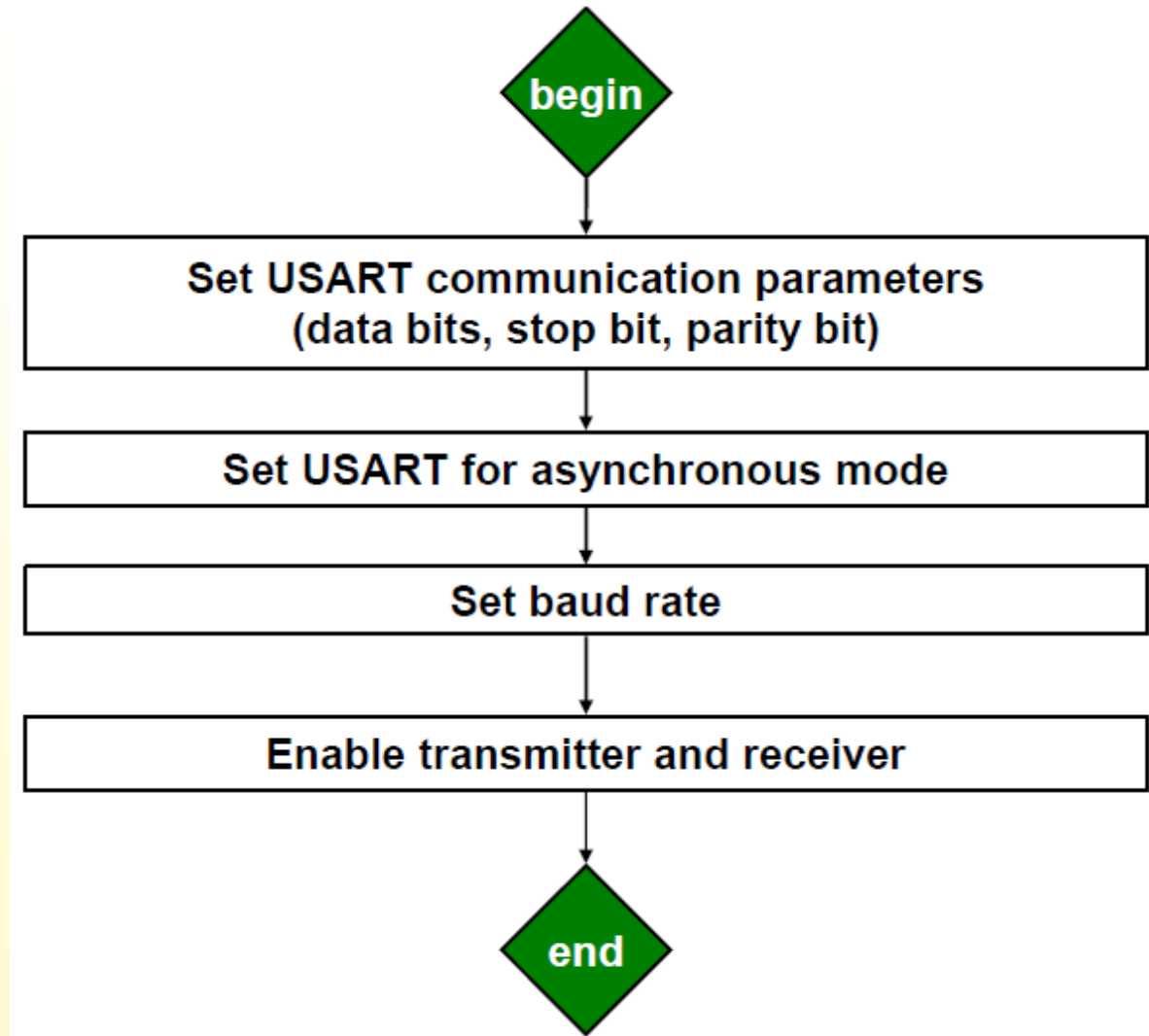
USART Data Register

- Register **UDR** is the buffer for characters sent or received through the serial port.
- To start sending a character, we write it to UDR.
`unsigned char data;`
`data = 'a';`
`UDR = data; // start sending character`
- To check a received character, we read it from UDR.
`unsigned char data;`
`data = UDR; // this will clear UDR`

Serial USART – Main tasks

- The main tasks involved in using the serial port are:
 - ❑ Initializing the serial port.
 - ❑ Sending a character.
 - ❑ Receiving a character.
 - ❑ Sending/receiving formatted strings.

Initializing serial port

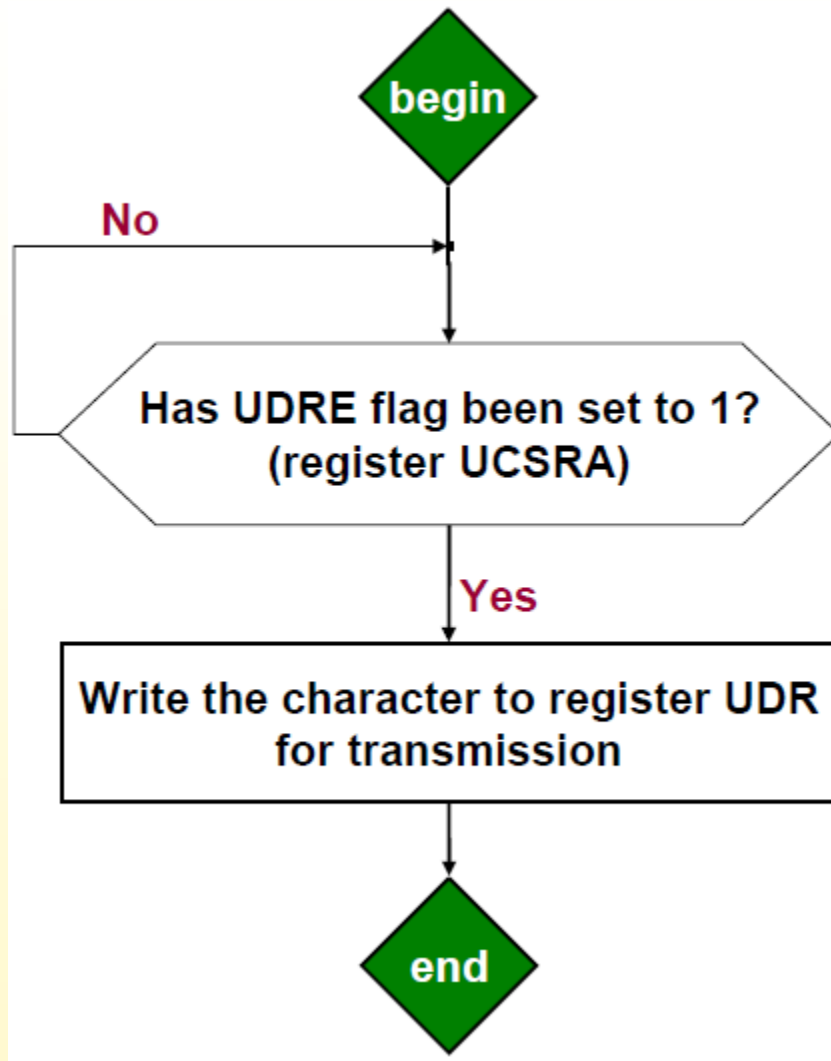


Initializing serial port – Example

Initialize serial port to baud rate 1200 bps, no parity, 1 stop bit, 8 data bits. Assume a clock speed of 1MHz.

```
void USART_init(void) {  
    // Normal speed, disable multi-processor  
    UCSRA = 0b00000000;  
    // Enable Tx and Rx, disable interrupts  
    UCSRB = 0b00011000;  
    // Asynchronous mode, no parity, 1 stop bit, 8 data bits  
    UCSRC = 0b10000110;  
    // Baud rate 1200bps, assuming 1MHz clock  
    UBRRL = 0x33;  
    UBRRH = 0x00;  
}
```

Sending a character



UDRE:
Data Register Empty
in UCSRA

Sending a character – Example

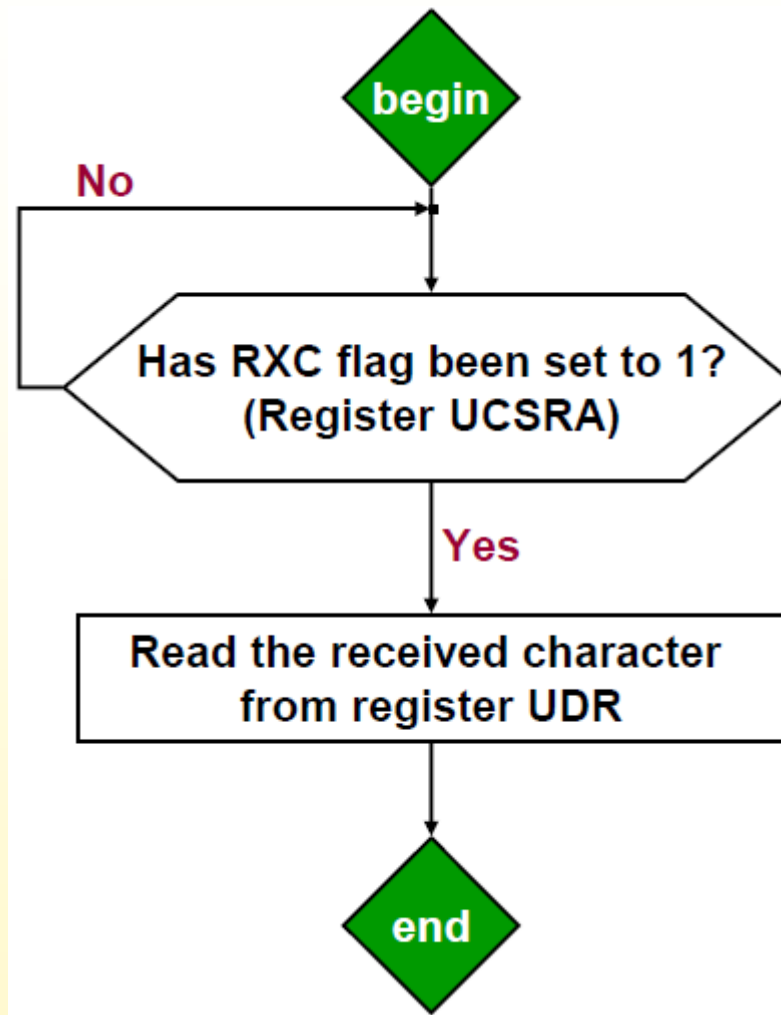
Write a C function to send a character through the serial port of ATmega32.

```
void USART_send(unsigned char data){  
    // wait until UDRE flag is set to logic 1  
    while ((UCSRA & (1 << UDRE)) == 0x00){;}  
  
    UDR = data; // Write character to UDR for  
    transmission  
}
```

➤ The constant UDRE has been defined in <avr/io.h>

```
#define UDRE 5
```

Receiving a character



RxC:
Receive Complete
in UCSRA

Receiving a character – Example

Write a C function to receive a character through the serial port of ATmega32.

```
unsigned char USART_receive(void) {  
    // Wait until RXC flag is set to logic 1  
    while ((UCSRA & (1 << RXC)) == 0x00) {;}  
  
    return UDR; // Read the received character from UDR  
}
```

- The constant RXC has been defined in <avr/io.h>

```
#define RXC 7
```

Sending/receiving formatted strings

- In ANSI C, the header file `<stdio.h>` has two functions for formatted strings: `printf` and `scanf`.
- Function `printf` sends a formatted string to the standard output device, which is usually the video display.

```
unsigned char a, b;  
a = 2; b = 3;  
printf("a = %d, b = %d, sum = %d", a, b, a+b);
```

- Function `scanf` reads a formatted string from the standard input device, which is usually the keyboard.

```
unsigned char a, b;  
scanf("%d %d", &a, &b); //get integers a, b from input string
```

Sending/receiving formatted strings

- Being able to send/receive formatted strings through a serial port is useful in microcontroller applications.
- To do so, we need to configure the serial port as the standard input and output device.
- **General steps:**
 1. Write two functions to send and receive a character through serial port.
 2. In main program, call **fdevopen()** to designate the two functions as the handlers for standard output and standard input device.
 3. Use printf/scanf as usual. Formatted strings will be sent/received through serial port.

Sending/receiving formatted strings – Example

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

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

```
int USART_send(char c, FILE *stream){  
    while ((UCSRA & (1<<UDRE)) == 0x00){;}  
    UDR = c; // Write character to UDR for transmission  
}
```

```
int USART_receive(FILE *stream){  
    while ((UCSRA & (1<<RXC)) == 0x00){;}  
    return (UDR); // Read the received character from UDR  
}
```

```
int main(void){  
    unsigned char a;  
    //Code to initialise baudrate, TXD, RXD, and so on is not shown here  
    // Initialise the standard IO handlers  
    stdout = fdevopen(USART_send, NULL);  
    stdin = fdevopen(NULL, USART_receive);  
    // Start using printf, scanf as usual  
    while (1){  
        printf("\n\rEnter a = ");  
        scanf("%d", &a); printf("%d", a);  
    }  
}
```

Example application

- The MCAM100 is a programmable pan-tilt video camera.
- It can be controlled through a serial connection: **8 data bit, 1 stop bit, no parity bit**, baud rate **9600bps**.
- Sending character **'4'** or **'6'** will turn the camera left or right, respectively.
- We'll write a program to rotate the camera repeatedly.



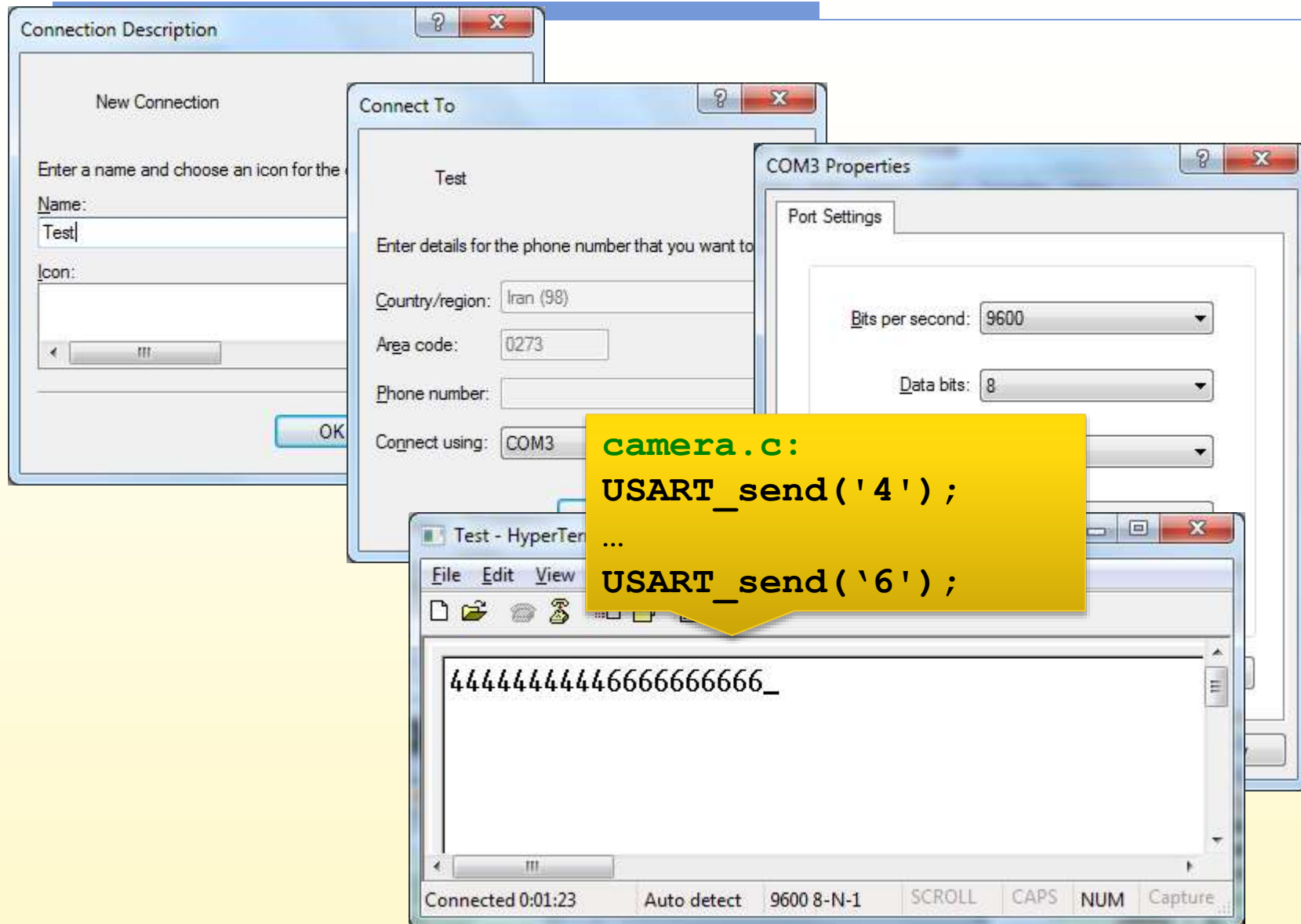
camera.c

```
#include <util/delay.h>
#include <avr/io.h>
void USART_init(void){
    UCSRA = 0b00000010; // double speed, disable multi-proc
    UCSRB = 0b00011000; // Enable Tx and Rx, disable interrupts
    UCSRC = 0b10000110; // Asyn mode, no parity, 1 stop bit, 8 data bits
    // in double-speed mode, UBRR = Fclock/(8 x baud rate) - 1
    UBRRH = 0; UBRL = 12; // Baud rate 9600bps, assuming 1MHz clock
}
void USART_send(unsigned char data){
    while ((UCSRA & (1<<UDRE)) == 0x00){;}
    UDR = data; // Write character to UDR for transmission
}
int main(void) {
    unsigned char i;
    USART_init(); // initialize USART
    while (1) {
        for (i = 0; i < 10; i++){ // rotate left 10 times
            USART_send('4');
            _delay_ms(500);
        }
        for (i = 0; i < 10; i++){ // rotate right 10 times
            USART_send('6');
            _delay_ms(500);
        }
    }
}
```


Debugging tool: Hyper Terminal

- Sending and receiving data through serial port are a useful debugging technique when writing a microcontroller program.
- A tool for monitoring serial data is Hyper Terminal program, which is part of the Windows OS (until XP).
- Hyper Terminal can be used to
 - ❑ establish a serial connection between the PC and the microcontroller.
 - ❑ send a text string to the microcontroller.
 - ❑ receive a text string sent from the microcontroller.

Hyper Terminal



Summary

- **What we learnt in this lecture:**
 - ❑ Basics of serial communications.
 - ❑ Serial communications subsystems in ATmega32.
 - ❑ Using serial port to send/receive characters and formatted strings.