Embedded System Course 2021-22

C4µC

C/C++ Programming for Microcontrollers

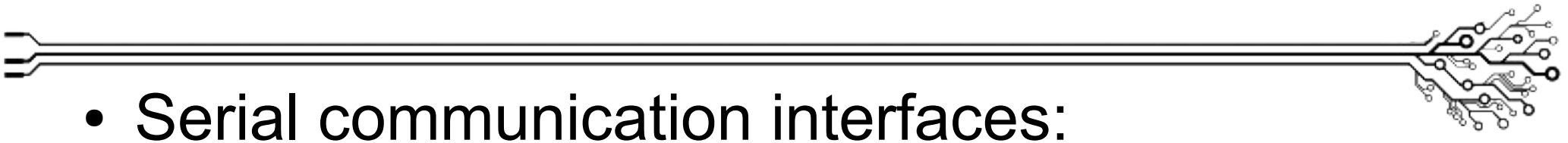
Lecturer: Ing. Marco Santic

marco.santic@univaq.it

C4µC

- Introduction
  - Atmega328P
  - Toolchain
  - IDE
  - Examples
  - C language recap
- Basic functionalities
  - GPIO
  - Timers
  - Interrupt
  - Examples
- **Comm. Interfaces**
  - I2C
  - **UART**
  - SPI
  - Examples
- Other peripherals
  - ADC
  - GPIO bitbanging
  - Examples

C4µC - U(S)ART

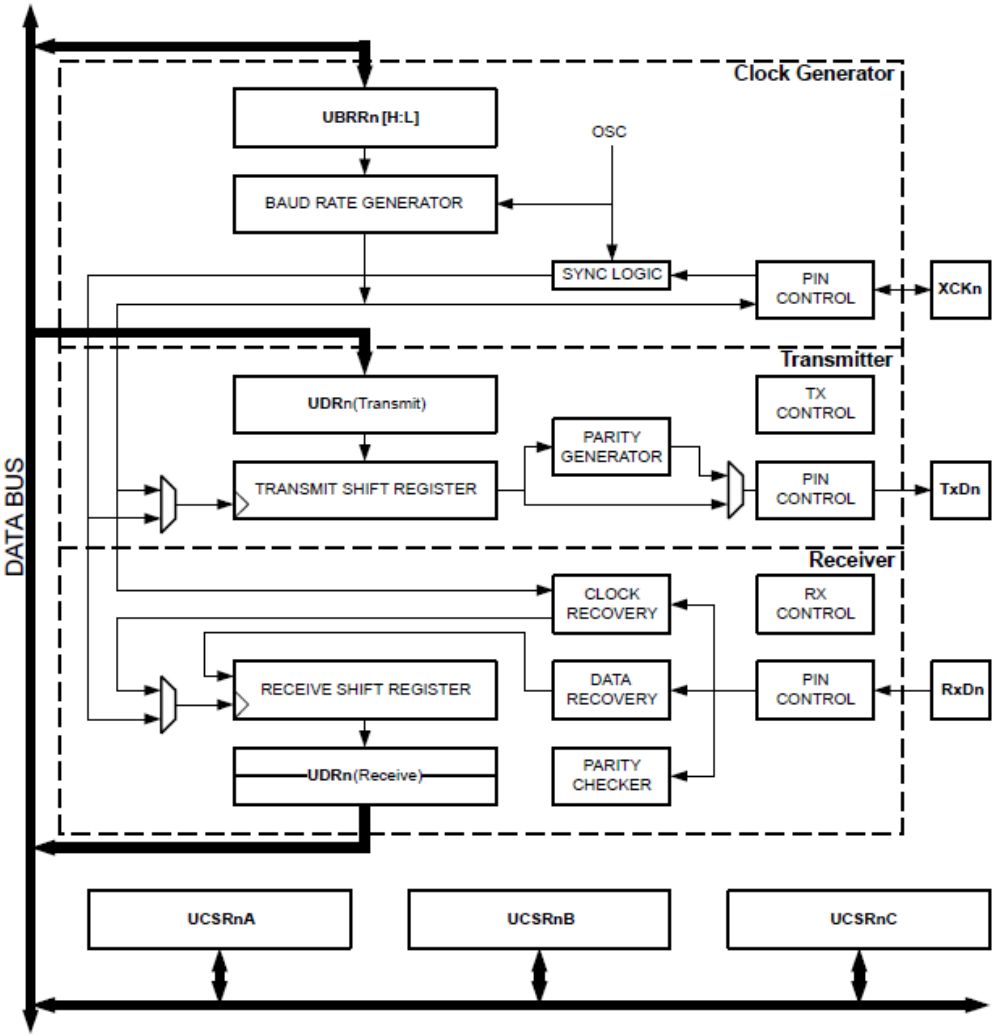


- Serial communication interfaces:
  - U(S)ART:
    - HW device available on Atmega328
      - Full duplex
      - Synch – asynch
      - Hi resolution baudrate generator
      - Data-bit 5 to 9, stop bit 1-2
      - Parity generator
      - Frame error detection
      - INT: Tx complete, TX dataReg empty, Rx complete

C4µC - U(S)ART

- Block diagram
  - Regs:
    - UCSRnA
    - UCSRnB
    - UCSRnC
    - UBRRn
    - UDRn (Tx e Rx)

Figure 24-1. USART Block Diagram

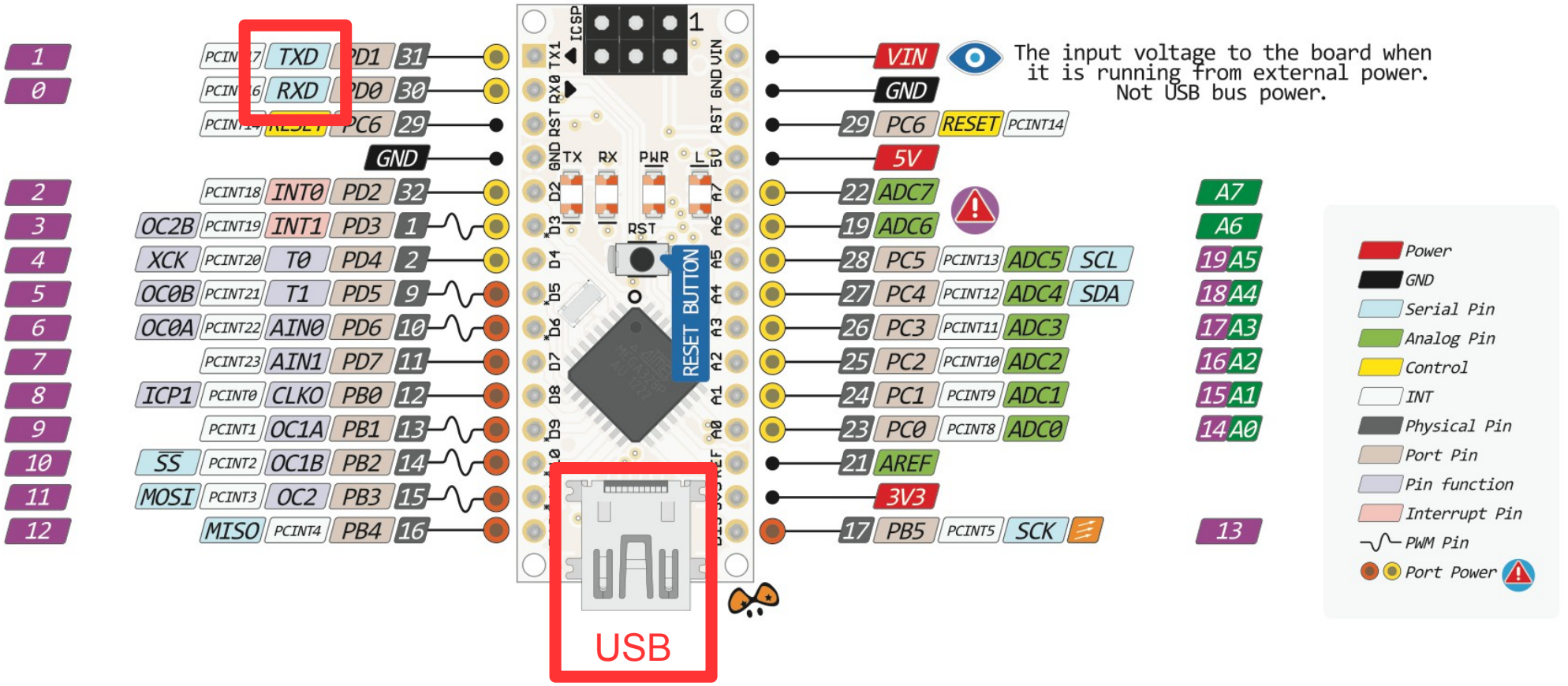


TxDn, RxDn, XCKn,...
Where are the other lines of RS-232?

NOTE: do not confuse UART and RS-232...

C4µC - U(S)ART

- Arduino Nano v3 pinout

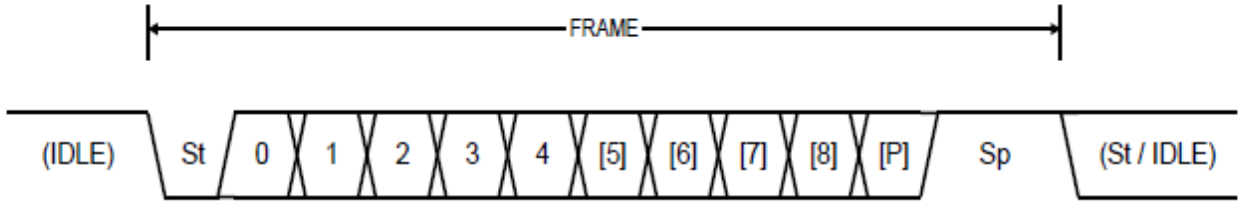


UART is avaiable because there are a UART-USB converter (board side) and the proper driver installed (host-side) → Serial port (COMn, ttyUSBn)

C4µC - U(S)ART

- General frame format

Figure 24-4. Frame Formats



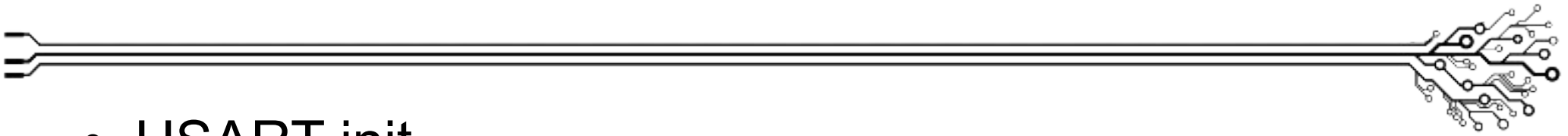
St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

Test with oscilloscope!?

C4µC - U(S)ART

- UART settings: commonly used 8N1
  - It means: 8 (data bits), None (parity), 1 (stop bit)
- Baudrate consideration (9600: what it means?)
  - 9600 bit/s @ 8N1 -> (1+8+1 = 10bit per TxByte)
  - 960 Byte/s effective data rate
- Control flow considerations (Sw,Hw)
  - How transmitter and receiver agree when and who is ready? (Xon-Xoff, RTS, CTS,...)
  - Device Tx and Rx buffers ... are enought?
    - Overrun and underrun conditions

C4µC - U(S)ART



- USART init

```
#define FOSC 1843200 // Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
int main( void ){
  ...
  USART_Init(MYUBRR)
  ...
}

//NOTE "=" and not "|=" or "&="
void USART_Init( unsigned int ubrr){
  /*Set baud rate */
  UBRR0H = (unsigned char)(ubrr>>8);
  UBRR0L = (unsigned char)ubrr;
  /*Enable receiver and transmitter */
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);
  /* Set frame format: 8data, 1stop bit */
  UCSR0C = (3<<UCSZ00);
}
```

Code from datasheet

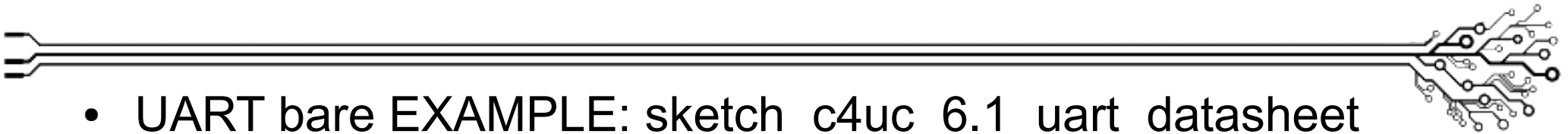
C4µC - U(S)ART

- USART transmit and receive

```
void USART_Transmit( unsigned char data ){
  /* Wait for empty transmit buffer */
  while ( !( UCSR0A & (1<<UDRE)) )
  ;
  /* Put data into buffer, sends the data */
  UDR0 = data;
}


unsigned char USART_Receive( void )
{
  /* Wait for data to be received */
  while ( !(UCSR0A & (1<<RXC0)) )
  ;
  /* Get and return received data from buffer */
  return UDR0;
}
```

Code from datasheet

C4µC - U(S)ART



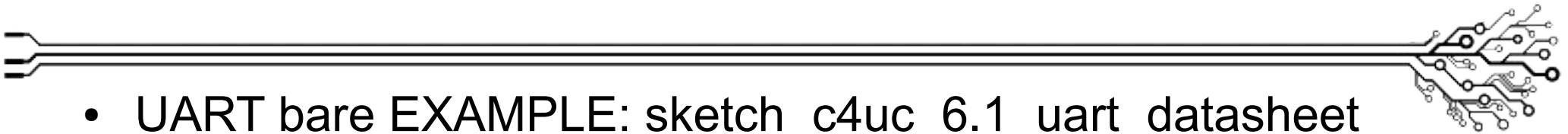
- UART bare EXAMPLE: sketch_c4uc_6.1_uart_datasheet

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

#define FOSC 16000000L // Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1  //See table 24-1 datasheet p.227

void USART_Init( unsigned int ubrr);
void USART_Transmit( unsigned char data );
unsigned char USART_Receive( void );
bool USART_Overrun( void );
bool USART_ClearOverrun( void );

uint8_t test = 1;

int main( void ){

  DDRB |= _BV(DDB5); // Led
  USART_Init(MYUBRR);
  unsigned char msg[] = "Hello UART!";
  unsigned char msg2[] = "You wrote ";
  uint8_t msg_len = 11;
  uint8_t msg2_len = 10;
```

Added 2 functions to control overrun condition

Note variable: test = 1

C4µC - U(S)ART



- UART bare EXAMPLE: sketch_c4uc_6.1_uart_datasheet

```
unsigned char input = 0;

bool overrunOccurred = 0;

switch(test){
  case 1:
    for(uint8_t i=0; i<msg_len;i++){
      USART_Transmit(msg[i]);
    }
    break;
  case 2:
    while(1){
      input = USART_Receive();
      USART_Transmit(input);
    }
    break;
```

Test 1 just sends hello message

Test 2 receives a char and sends it back, it is an echo

C4µC - U(S)ART

- UART bare EXAMPLE: sketch_c4uc_6.1_uart_datasheet

```
    default:
      while(1){
        input = USART_Receive();

        if(USART_Overrun()) overrunOccurred = 1;
        if(input == 'c') overrunOccurred = USART_ClearOverrun();

        if (overrunOccurred){ PORTB |= _BV(PORTB5);}
        else { PORTB &= ~_BV(PORTB5);}

        for(uint8_t i = 0; i< msg2_len; i++){
          USART_Transmit(msg2[i]);
        }
        USART_Transmit(input);
        USART_Transmit(0x0A); //0x0A = 10 = '\n'
      }
  }
}
```

Test 3 receives a char and sends it back after a “You wrote “ message

C4µC - U(S)ART

- If we send a character in the serial monitor ("1")...
  - we will see back: "You wrote 1"
- If we send 3 characters in the serial monitor ("123")...
  - we will see back: "You wrote 1"

    "You wrote 2"

    "You wrote 3"

- If we send a sequence of 5 characters ("12345")
  - We will see that we have lost some chars
  - And that the LED on the board is on

We have an overrun situation

C4µC - U(S)ART

- The two functions we have added are:

```
bool USART_Overrun( void ){
  return (bool) (UCSR0A & (1<<DOR0));
}

bool USART_ClearOverrun( void ){
  unsigned char dummy;
  UCSR0A &= ~(1<<DOR0); //clear overrun flag, needed?!?!
  while ( UCSR0A & (1<<RXC0) ) dummy = UDR0; //flushing the buffer
  return 0;
}
```

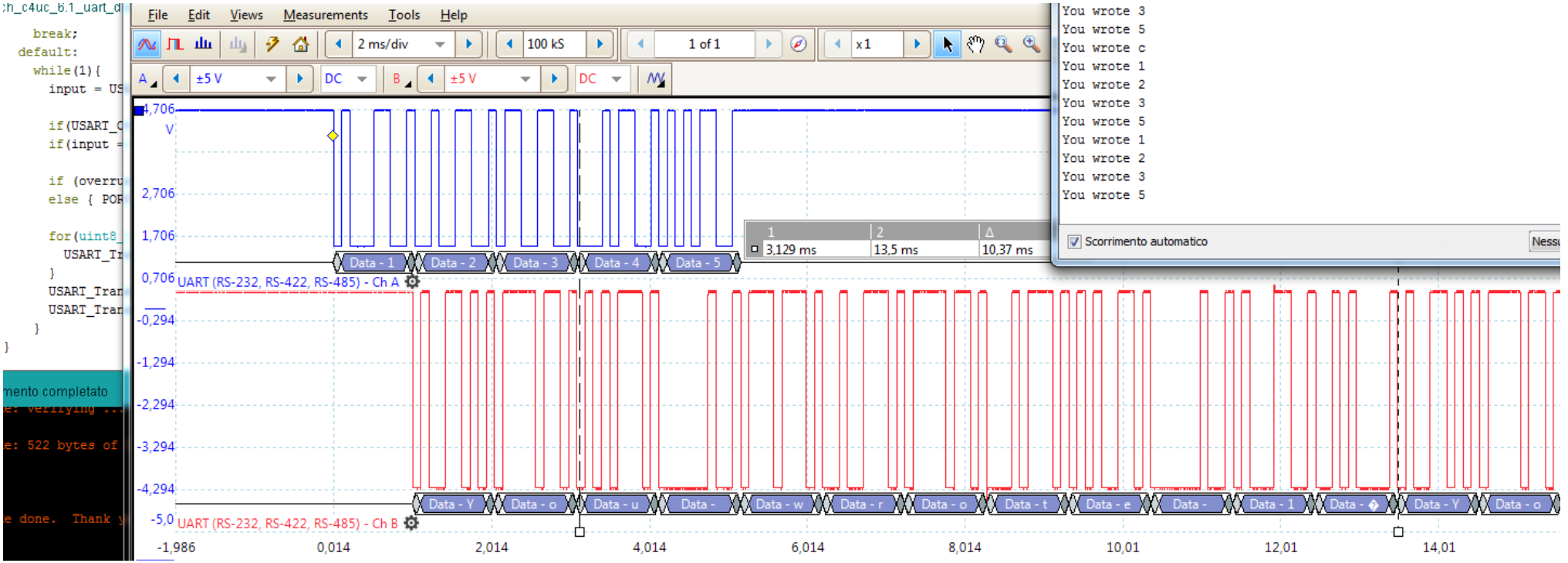
- With the first we verify if an overrun occurred
  - Read datasheet at p.245:

Bit 3 – DOR0:
Data OverRun This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDR0) is read. Always set this bit to zero when writing to UCSR0A.

The receive buffer has only 2 bytes!!!

C4µC - U(S)ART

- This is what happens:



- 1st byte ("1") is received and read, it starts the transmission of messages
- 2nd and 3rd are received and are in buffer, 4th is received and is in shift register
- 5th byte's start bit sets DOR0 flag (overrun) and 5th byte overrides 4th in shift register

C4µC - U(S)ART

- U(S)ART summary:
  - We have seen a lot of examples using serial comms also in previous lessons
  - The implementation in the Arduino API of *Serial* keeps us in a "safe user zone"
  - We have seen the registers involved in UART comms
  - When we need to use a UART at lower level, we need to take into account the hardware logic and limitations

Ex.: try to discover out the buffering mechanism in API impl. of Serial

  - When also software buffer is not enough, we need to apply flow control (RTS, CTS, ...)

Questions?