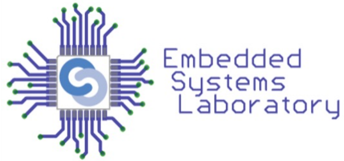


PROIECTAREA CU MICROPROCESOARE

Cursul 2
Debugging & UART

Facultatea de Automatică și Calculatoare
Politehnica București

DON'T NEED TO DEBUG CODE
IF YOU DON'T WRITE BUGS

Hardware is hard

În cine ai încredere când ceva nu merge bine?

- Codul scris de tine?
- Utilizatorii produsului tău?
- Echipamentul de test?
  - Felul în care este configurat și folosit echipamentul de test?
- Proiectantul device-ului?
  - Autorii dataheet-ului?
- Proiectantul PCB-ului?
  - Fabrica care a produs și asamblat dispozitivul?

...

- Fizica universului în care trăim?

Erorile într-un datasheet sunt frecvente

Mistakes in the ATmel datasheet examples and.... solution !

Using Arduino Programming Questions

jptrol May 2019 **post #1**

Hello,

While trying to read asynchronous serial data with the particular 9 bits format I came upon a 'double' mistake in the document I suppose we all use : the ATmegaXX datasheet

It is in chapter 22.7.2 in the C-code example (the assembly code is correct) (p.211)

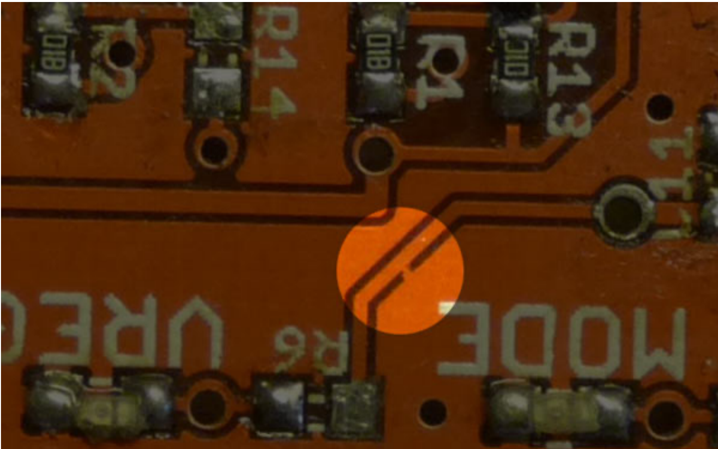
```
unsigned int USART_Receive( void )
{
     unsigned char status, resh, resl;

        while ( !(UCSRnA & (1<<RXCn)) )         /* Wait for data to be receiv
     ;

       status = UCSRnA;    /* Get status and 9th bit, then data from buffer

       resh = UCSRnB;
       resl = UDRn;

       if ( status & (1<<FEn)|(1<<DORn)|(1<<UPEn) ) return -1;      /* If er

      resh = (resh >> 1) & 0x01;            /* Filter the 9th bit, then retu
      return ((resh << 8) | resl);
}
```

With this code you don't stand a chance to get any data since it always returns on error.
In the line :

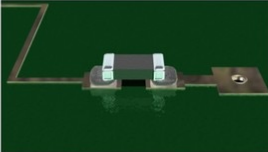
```
if ( status & (1<<FEn)|(1<<DORn)|(1<<UPEn) ) return -1;
```

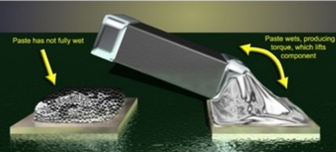
there is a missing level of parenthesis that results in the expression being always true. When there is no mistake the return value is always 12 (or 0xC or 0b1100) since status & (1<<FEn) = 0 but 0 or 1000 or 100 makes 01100.

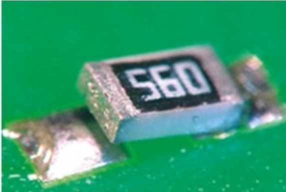
Greșelile de fabricație sunt des întâlnite

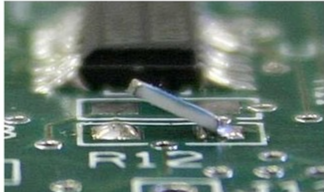


Circuit board manufacturing fault (signal shorted to GND)

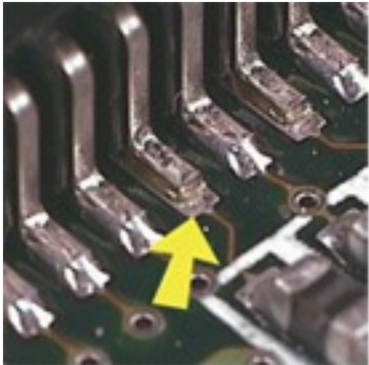


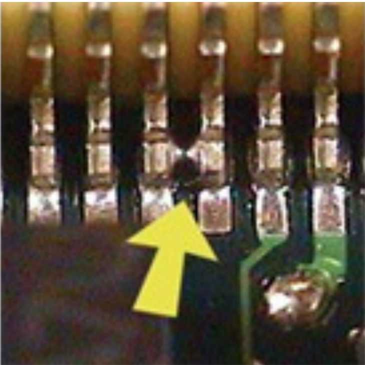


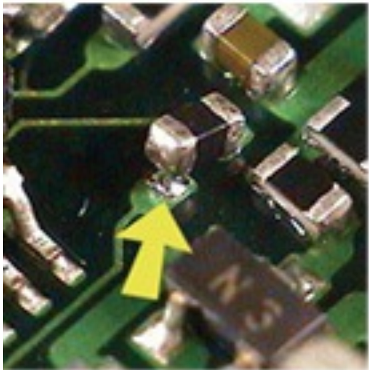




PCB assembly fault (“tombstoning”) resistors







Various soldering faults (unsoldered pins, shorted pins, tombstoning, weak soldering)

When all else fails, the serial port can save the day!

Ce faci dacă software-ul care rulează pe procesorul tău rămâne blocat sau are un crash? Opțiunile tale sunt foarte limitate.

- Majoritatea procesoarelor embedded sunt programate peste o conexiune serială UART sau USB (care are tot un UART în spate).
- Bootloader-ul este un program rezident în memoria de program care rulează la îndeplinirea unei condiții speciale
  - Poate să fie la RESET sau la un alt tip de întrerupere
  - De obicei bootloader-ul ocupă cât mai puțin spațiu în memorie, suficient pentru a rescrie memoria de program
- Ce se întâmplă dacă și bootloader-ul este defect?
  - Mai ai o singură opțiune: un programator extern (In-System Programmer sau un JTAG)
  - Un programator JTAG este de obicei mai scump și permite și depanarea codului (breakpoints, step-by-step execution, memory inspection etc.)

https://learn.sparkfun.com/tutorials/installing-an-arduino-bootloader/all

Chiar și un LED poate fi esențial pentru debugging

Gândiți-vă la un LED ca la un breakpoint

- Îl aprindeți înainte de execuția unei linii de cod
  - Arată că execuția a ajuns la linia respectivă
- Îl stingeți după execuția liniei respective
  - Arată că linia de cod a fost executată
- Îl faceți să clipească atunci când este apelat un eveniment recurent
  - Vă poate confirma vizual cât de frecvent se execută evenimentul

Configurație pentru debug

JTAG Debugger

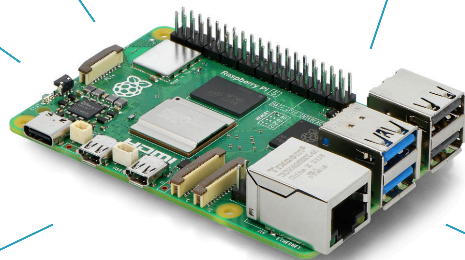
LEDuri!

Multimetru

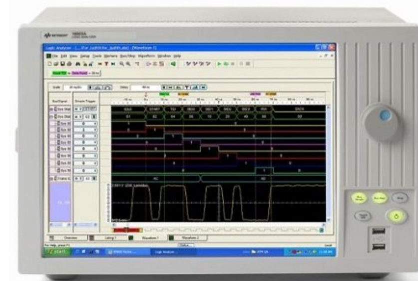
Osciloscop

```
vivek@backup:~$ cu -l /dev/ttyS0 -s 19200
Connected.
0256

Pri Sla  Flash Card                          LBA 996-8-32  127 Mbyte

Slot   Vend Dev  ClassRev Cmd  Stat CL LT HT  Base1     Base2    Int
------------------------------------------------------------------------
0:00:0 1078 0001 06000000 0107 0280 00 00 00 00000000 00000000
0:06:0 100B 0020 02000000 0107 0290 00 3F 00 0000E101 A0000000 10
0:07:0 100B 0020 02000000 0107 0290 00 3F 00 0000E201 A0001000 10
0:08:0 100B 0020 02000000 0107 0290 00 3F 00 0000E301 A0002000 10
0:10:0 10EC 8139 02000010 0107 0290 00 3F 00 0000E401 A0003000 11
0:14:0 168C 0013 02000001 0116 0290 08 3C 00 A0010000 00000000 05
0:18:2 100B 0502 01018001 0005 0280 00 00 00 00000000 00000000
0:19:0 0E11 A0F8 0C031008 0117 0280 08 38 00 A0020000 00000000 09

 1 Seconds to automatic boot.   Press Ctrl-P for entering Monitor.

comBIOS Monitor.   Press ? for help.

>
```

Consolă serială



Device Under Test



Analizor Logic

https://www.edn.com/how-to-debug-embedded-systems/

Tehnici generale pentru debugging

To err is human, but to really foul things up you need a computer.

- Paul R. Erlich

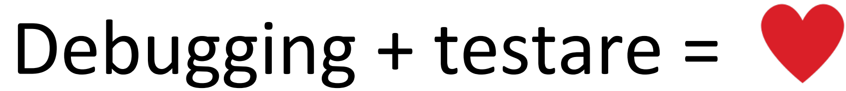
- Debugging-ul este o artă. Ca la orice altă artă, succesul este o combinație de talent, experiență și folosire a uneltelor corecte.
- Secretele debugging-ului nu sunt misterioase. Pașii care trebuie urmați sunt logici și identifică eroarea în majoritatea cazurilor.

Tehnici generale pentru debugging

Cum să eviți bug-uri în cod (câteva sfaturi, nu e o listă exhaustivă)

- Coding style: aderă la un stil și nu devia de la el, folosește un naming standard
- Documentează totul, de la specificațiile arhitecturii și interfețelor până la comentarii în liniile de cod
- Ține sesiuni de code review
- Programează defensiv
- Verifică valorile de retur ale funcțiilor
- Tratează suspicios orice cut/paste
- Folosește un IDE cu verificare dinamică a sintaxei

Tehnici generale pentru debugging



- Testarea găsește erorile, debugging-ul le localizează și le repară
- Împreună, cele două tehnici formează ciclul testare/debugging: testăm, reparăm apoi repetăm
- Orice fel de debugging ar trebui să fie urmat de reaplicarea testelor relevante, în special testele de regresie. Acest lucru reduce posibilitatea de introducere a noi bug-uri în urma debugging-ului
- Testing și debugging nu trebuie să fie făcute de aceeași persoană

De ce este greu să faci debugging?

- Poate să nu fie o relație evidentă între manifestarea externă a unei erori și cauza sa internă.
- Simptomul și cauza pot fi în părți îndepărtate ale programului.
- Schimbările în program pot masca sau modifica bug-urile.
- Simptomul poate fi rezultatul unei greșeli sau neînțelegeri umane dificil de urmărit.
- Bug-ul poate fi declanșat de o secvență de intrare rară sau dificil de reprodus, de sincronizarea programului (fire de execuție) sau de alte cauze externe.
- Bug-ul poate depinde de starea altui software/sistem, de acțiunile altora asupra sistemelor voastre cu săptămâni/luni în urmă.

Designing for Testing & Debugging

- Când scrii cod, gândește-te la modul în care vei testa/depana acel cod.
    - Lipsa de gândire se traduce întotdeauna prin bug-uri.
- Scrie cazuri de test atunci când scrii codul tău.
- Dacă ceva ar trebui să fie adevărat, asigură-te de acest lucru prin funcția assert().
- Creează funcții pentru a ajuta la vizualizarea datelor tale.
- Proiectează cu gândul la testare/depanare încă de la început.
- Testează devreme, testează des.

Tipuri de bug-uri

- La compilare: sintaxă, ortografie, neconcordanță statică a tipurilor - de obicei detectate cu compilatorul.
- Design: algoritmi greșit concepuți - ieșiri incorecte.
- Logica programului (if/else, terminarea buclelor, selectarea cazului etc.) - ieșiri incorecte.
- Absurdități de memorie: pointeri null, limite ale vectorilor, tipuri greșite, memory leaks – runtime exceptions.
- Erori de interfațare între module, fire de execuție, programe (în special cu resurse partajate: socket-uri, fișiere, memorie etc.) – runtime exceptions.
- Condiții neobișnuite: eșecul unei părți a software-ului sau a echipamentului de bază (rețea etc.) - funcționalitate incompletă.
- Blocări: procese multiple care luptă pentru o resursă - deadlocks, procese care nu se termină niciodată.

Procesul ideal de debugging

1. Identifică cazurile de testare care arată în mod fiabil existența defectului (atunci când este posibil).
2. Izolează problema într-un fragment mic al programului.
3. Corelează comportamentul incorect cu eroarea logică/ de cod a programului.
4. Modifică programul (și verifică și alte părți ale programului în care aceeași logică de program similară poate apărea).
5. Efectuează teste de regresie pentru a verifica că eroarea a fost într-adevăr eliminată - fără a introduce noi erori.
6. Actualizează documentația când este necesar.

Ce este un debugger?

- Un debugger nu este un mediu de dezvoltare integrat (IDE) - deși cele două pot fi integrate, sunt entități separate.
- Un debugger încarcă un program (executabil compilat sau cod sursă interpretat) și permite utilizatorului să urmărească execuția.
- De obicei, debuggerele pot realiza dezasamblare, urmărirea stivei, monitorizarea expresiilor și altele.
- Permite verificarea erorilor "live" - fără a fi nevoie să rescrii și să recompilezi când îți dai seama că ar putea apărea un anumit tip de eroare.

Tehnici de debug

- Urmărirea execuției
  - Rularea programului
  - printf
  - Trace utilities
  - Single stepping în debugger
- Verificarea interfeței
  - verificarea numărului/ tipului de parametri procedură (dacă nu este impusă de compilator) și valoare
  - programare defensivă: verificarea intrărilor/ rezultatelor din alte module
  - documentarea presupunerilor despre relațiile dintre module, protocoale de comunicare, etc
- Aserțiuni - includerea range constraints sau a altor informații
- Skipping code - comentarea codului suspect, apoi verificarea dacă eroarea persistă

Sfaturi pentru debugging

- Înțelegeți problema înainte de a o remedia.
- Depanați pe sistemul gazdă cât mai mult posibil.
- Cele mai comune "Heisenbugs" sunt legate de memorie sau de firele de execuție.
- Găsiți problemele de memorie devreme.
- Reproduceti și izolati problema - găsiți cel mai mic exemplu de cod care demonstrează bug-ul.
- Introduceți verificări pentru invariante și opriți totul în program atunci când una este încălcată.
- Verificați fiecare strat cu teste mici și simple.
- Țineți un jurnal al evenimentelor și presupunerilor (git).

Unelte folosite pentru debugging

- Source-level debugger
- Apeluri simple la printf
- In-circuit emulators (ICE) & JTAG debuggers
- Data monitors
- Operating system monitors
- Profilers
- Memory testers
- Execution tracers
- Coverage testers

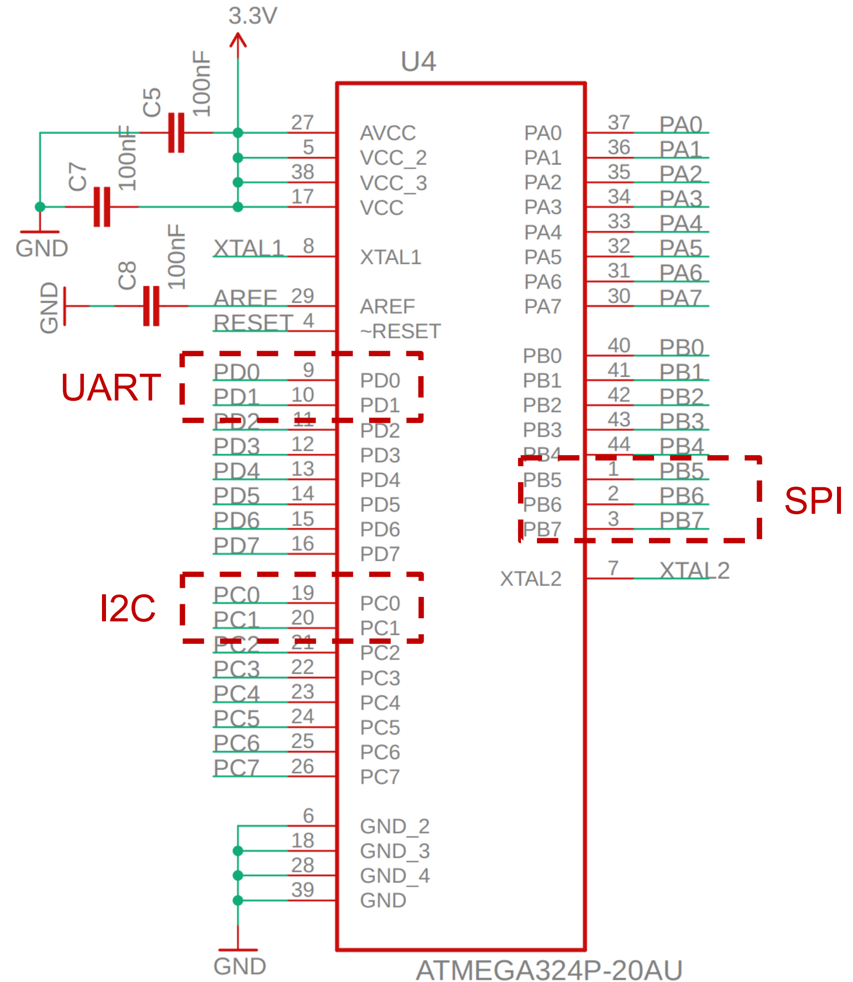
IDE
GDB

printf
Look what they need to
mimic a fraction of our power
imgflip.com

Porturi seriale

Porturi digitale pe care procesorul le folosește pentru a comunica cu perifericele *externe*

- UART
- I2C
- SPI

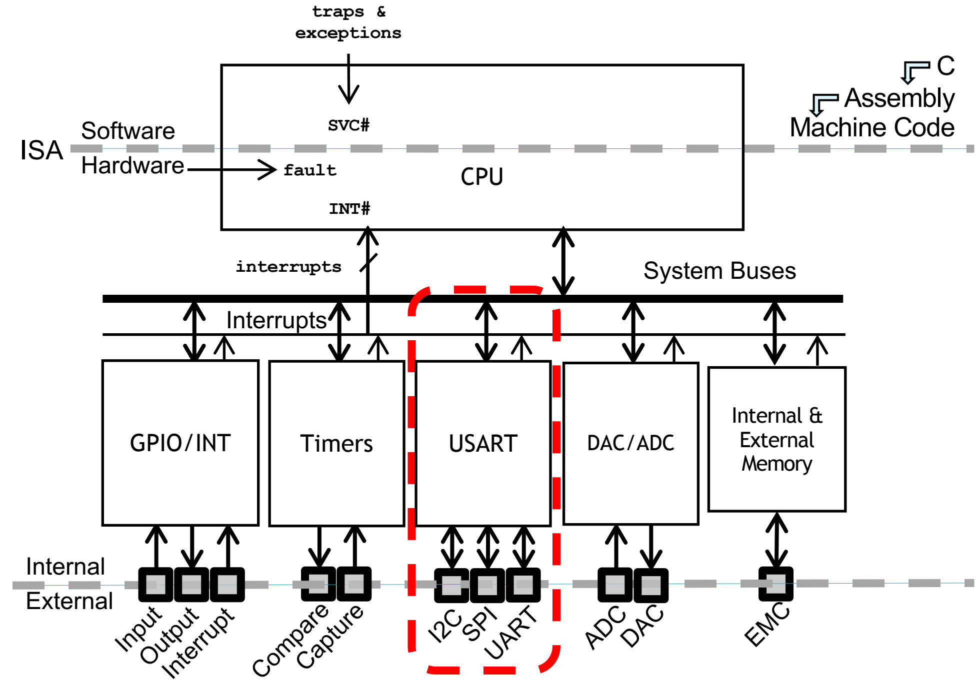


Porturi seriale pe placa de la laborator

- **UART** (port serial) pentru a programa prin bootloader placa sau pentru a comunica cu calculatorul pe care faceți dezvoltarea codului
- **I2C** pentru comunicația cu senzorii: accelerometrul, giroscopul, magnetometrul, senzorul barometric de presiune
- **SPI** pentru comunicația cu cardul SD și cu ecranul LCD grafic

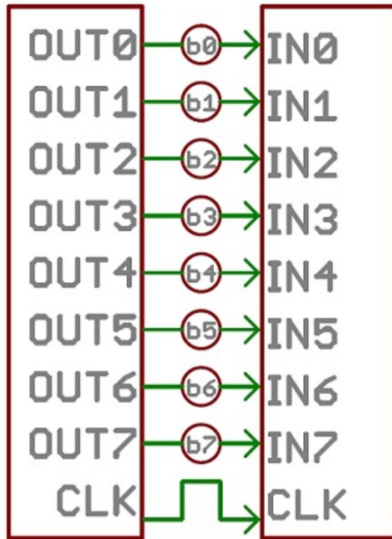
Periferic intern vs. extern

- Procesorul nu poate comunica direct cu un *periferic extern*
- Are nevoie de un *periferic intern* pe care să-l inițializeze și care să efectueze transferurile de date
- Comunicația cu perifericul intern se poate face prin *întreruperi* sau prin *polling*

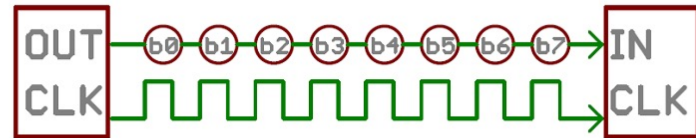


Parallel Bus vs. Serial Bus

Parallel

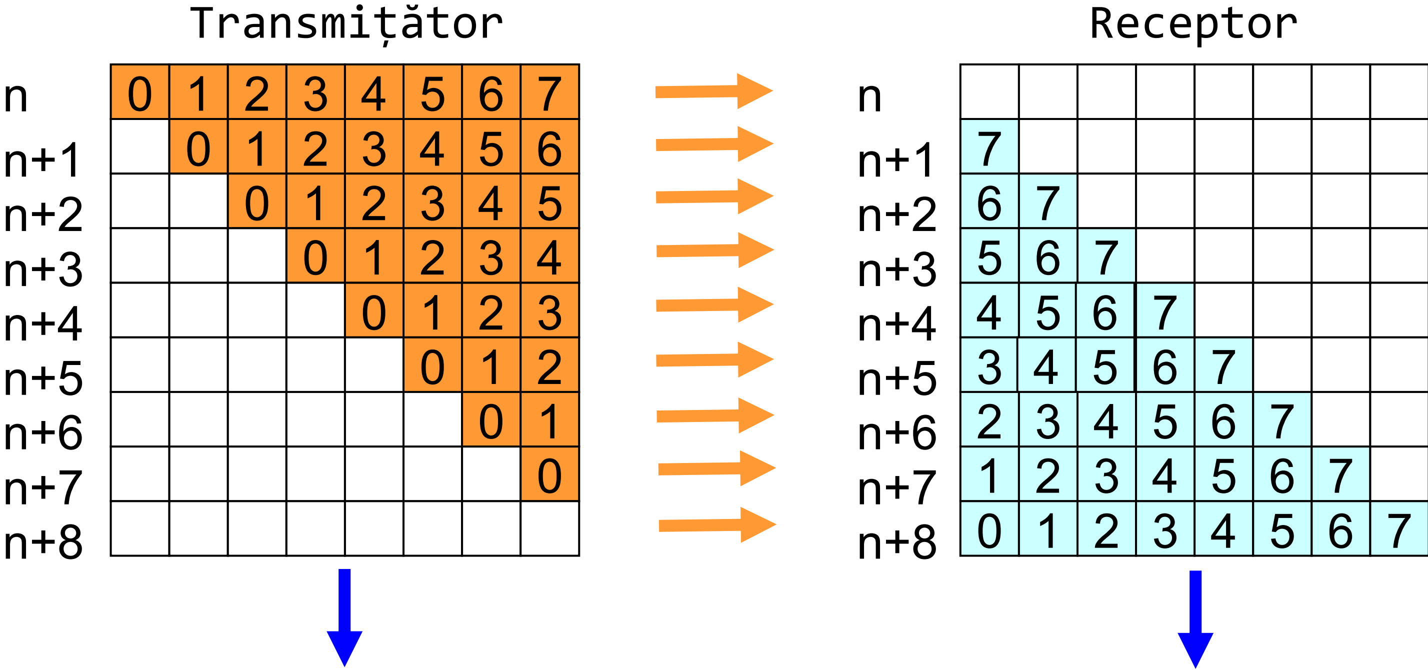


Serial



Care sunt beneficiile unei magistrale seriale față de una paralelă (și vice-versa)?

Diagramă (ultra)simplificată a comunicației seriale



Întrerupere generată când buffer-ul Transmițătorului (Tx) este gol
Octetul a fost transmis și următorul octet este gata pentru a fi încărcat

Întrerupere generată când buffer-ul Receptorului (Rx) este plin
Octetul a fost recepționat și este gata pentru a fi citit

De ce folosim magistrale seriale?

- Nu folosesc multe linii de date
  - Liniile de date au nevoie de pini de IO care costă
  - Necesită spațiu pe PCB care costă
- Pot conecta mai multe sisteme complet diferite prin aceeași magistrală
  - De ex. senzori, display și touch capacitiv pe același bus I2C
  - Un PC și un sistem embedded prin UART
  - Două sau mai multe microcontrolere prin aceeași magistrală SPI sau I2C
- Viteza de transfer este deseori scăzută
  - Dar nu întotdeauna (USB, HDMI, WiFi, SATA etc.)

Serial Bus Design Space

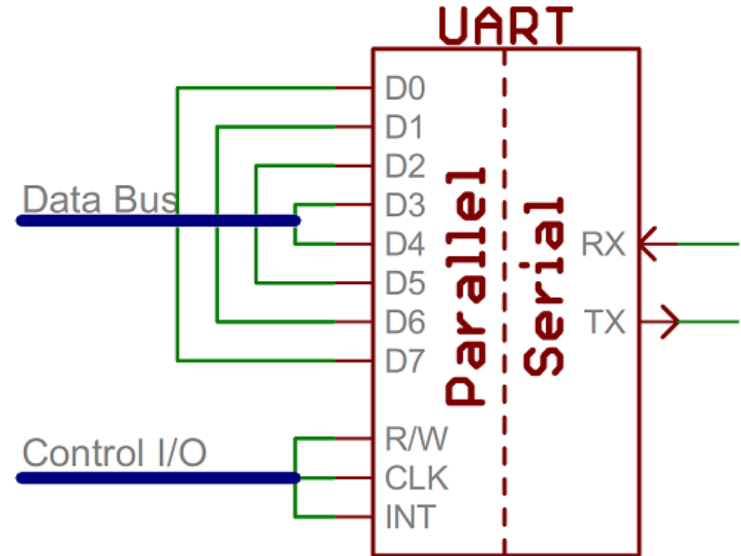
- Câte linii de legătură sunt necesare?
- Sincron sau asincron?
- Care este rata de transfer necesară?
- Este suficientă o conexiune punct la punct?
- Trebuie să aibă mai mult de un master?
- Cum tratăm flow control?
- Cum tratăm erorile și zgomotul?
- Cât de departe trebuie să transmitem datele?

Exemple de interfețe seriale

	S/A	Tip	Duplex	Dispozitive	Viteză (kbps)	Distanță (m)	Linii de date
RS232	A	Peer	Full	2	20	10	2+
RS422	A	Multi-drop	Half	10	10000	1200	1+
RS485	A	Multi-point	Half	32	10000	1200	2
I2C	S	Multi-master	Half	127+	3400	<2	2
SPI	S	Multi-master	Full	?	>1000	<2	3+
Microwire	S	Master/slave	Full	?	>625	<2	3+
1-Wire	A	Master/slave	half	?	16	200	1+

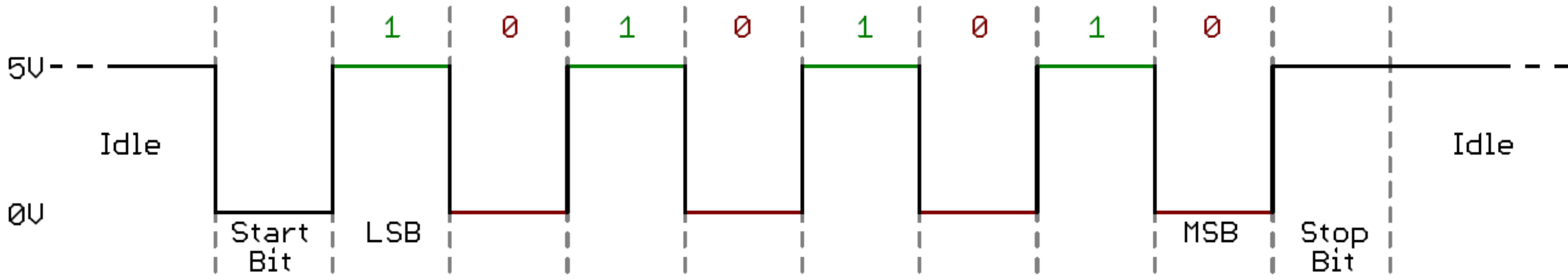
U(S)ART

- Universal (Synchronous) Asynchronous Receiver/Transmitter
- Circuit care face translatarea dintre o reprezentare paralelă a datelor și una secvențială
- Folosit des împreună cu standarde de comunicație precum EIA, RS-232, RS-422 or RS-485, USB, SATA etc.
- *Universal*
  - Formatul datelor și viteza de transfer sunt configurabile
  - Semnalizarea este standardizată



Protocolul serial

- Fiecare caracter este trimis pe linia serială după cum urmează
  - Un bit de **start** (nivel logic 0)
  - Un număr configurabil de **biți de date** (de la 5 la 9, de obicei 8)
  - Un bit opțional de **paritate** (rezultatul operației XOR pe biții de date)
  - Unul sau doi **biți de stop** (nivel logic 1)
  - Intervalul de timp dintre doi biți dă viteza de transfer (numită și **baud rate**)
- Exemple
  - <9600-N-8-1= <baudrate><parity><databits><stopbits>
  - <9600-8-N-1= <baudrate><databits><parity><stopbits>



Variațiuni

- USART este un termen generic care include un număr mare de dispozitive și standarde
- RS-232 este un standard
- Specifică caracteristicile și temporizarea semnalelor, semnificația semnalelor și dimensiunea fizica precum și pinout-ul conectorilor

Cele mai frecvent întâlnite semnale

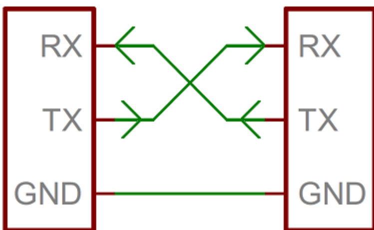
- Definiții
  - **DTE**: **D**ata **T**erminal **E**quipment
  - **DCE**: **D**ata **C**ircuit-terminating **E**quipment
- Flow Control
  - **RTS** (**R**equest **T**o **S**end): DTE cere DCE să trimită date
  - **CTS** (**C**lear **T**o **S**end): DCE comunică DTE că e gata să accepte date
- Linii de date
  - **Rx** (Receive): linia de intrare a datelor
  - **Tx** (Transmit): linia de ieșire a datelor

TX-RX
RX-TX
TX-TX
RX-RX

JAKE-CLARK.TUMBLR
imgflip.com

DB9 legacy stuff

- DTE vs. DCE
- Pinout al DCE?
- Masă comună?
- Care sunt efectele zgomotului?



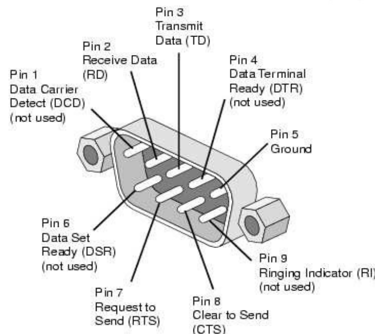
Cea mai simplă conexiune posibilă (fară flow control)

Modem to Modem Cable - **Crossover Cable DB9 to DB9**

DCE Device (Modem) **DB9** | **DCE to DCE** Connections | DCE Device (Modem) **DB9**

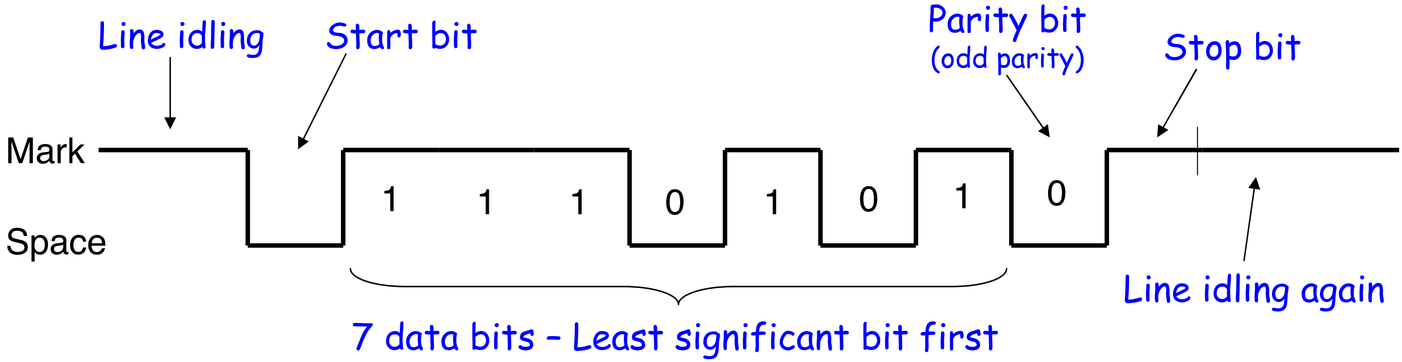
Pin# DB9 RS-232 Signal Names		Signal Direction	Pin# DB9 RS-232 Signal Names	
#1 Carrier Detector (DCD)	CD		#1 Carrier Detector (DCD)	CD
#2 Receive Data (Rx)	RD		#2 Receive Data (Rx)	RD
#3 Transmit Data (Tx)	TD		#3 Transmit Data (Tx)	TD
#4 Data Terminal Ready	DTR		#4 Data Terminal Ready	DTR
#5 Signal Ground/Common (SG)	GND		#5 Signal Ground/Common (SG)	GND
#6 Data Set Ready	DSR		#6 Data Set Ready	DSR
#7 Request to Send	RTS		#7 Request to Send	RTS
#8 Clear to Send	CTS		#8 Clear to Send	CTS
#9 Ring Indicator	RI		#9 Ring Indicator	RI
Soldered to DB9 Metal - Shield	FGND		Soldered to DB9 Metal - Shield	FGND

Note: Signal directions reversed if devices are DTE to DTE - "Null Modem" cable for DTE devices also connects pins #1 & #6 on each side to simulate Carrier (CD) which is required by some Terminal program software.

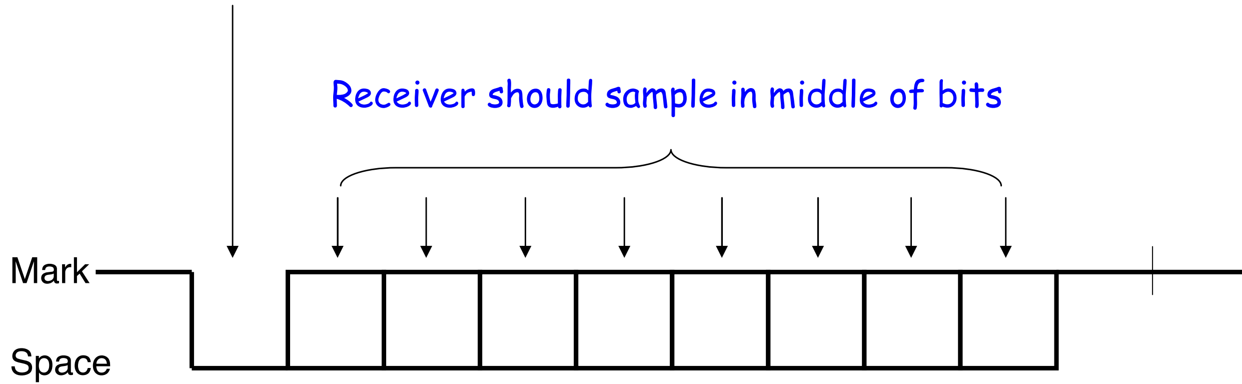


Exemplu UART

Transmisia caracterului ASCII ‘W’ (1010111)



Recepția unui caracter pe UART



Receiver uses a timer (counter) to time when it samples.
Transmission rate (i.e., bit width) must be known!

Recepția unui caracter pe UART

- Receptorul se resincronizează la fiecare bit de start
  - Trebuie să fie suficient de precis doar pentru următorul data frame (maxim 9 biți)
  - Dacă ceasul receptorului este mai lent sau mai rapid, se recepționează gunoi
- Receptorul verifică de asemenea ca bitul de stop este ‘1’
  - Dacă nu, raportează un “framing error” procesorului
- Noul bit de start poate să apară imediat după bitul de stop
  - Receptorul se va resincroniza la fiecare bit de start

AVR USART

Emițător-receptor USART la ATmega324P

UCSRnA (status)

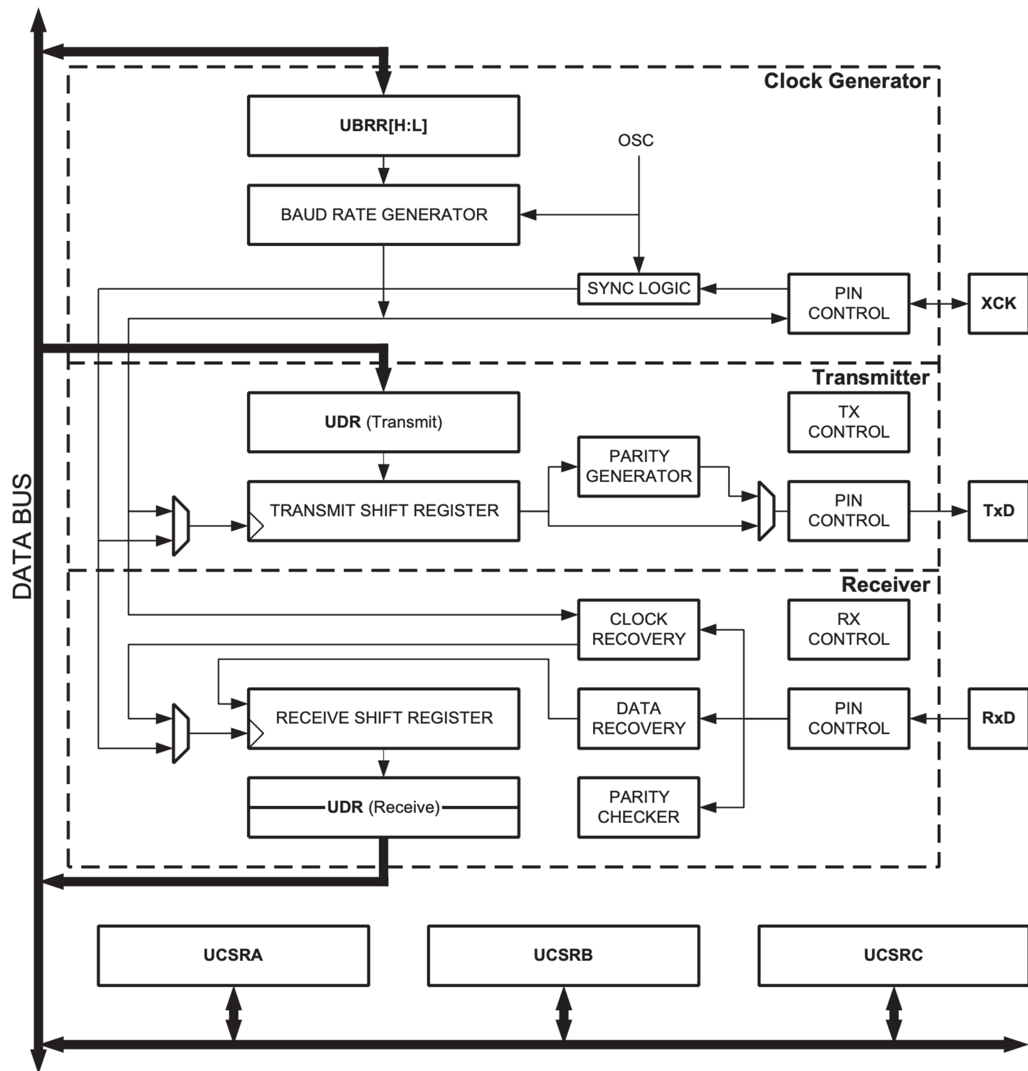
- UDRE - 1 când buffer-ul de ieșire este liber
- RXC - 1 când am primit un frame nou
- U2X - 1 dacă micșorăm prescalerul la 8
- FE - 1 dacă a fost Frame Error
- UPE - 1 dacă a fost Parity Error

UCSRnB (bit,enable)

- TXEN - Activarea transmițătorului
- RXEN - Activarea receptorului
- RXCIE, TXCIE, UDRIE - Activarea întreruperilor pentru RX complete, TX complete, buffer empty

UCSRnC (parametri frame)

- USBS - numărul de biți de stop
- UCSZ - dimensiunea unui frame
- UPM - configurarea bitului de paritate



AVR USART

UBRRn (setare baud rate)

- Valoarea care se scrie în acest registru va seta baud rate-ul
- Valoarea depinde de frecvența de ceas a procesorului
- Poate fi calculată printr-o formulă simplă
  - De ex. pentru a seta baud 115200 la o frecvență a ceasului de 12MHz, trebuie să scriem în UBRR valoarea 6
- Putem dubla viteza de transfer dacă setăm bitul U2X din UCSRnA
  - De ex. pentru baud 1Mbps, 12MHz, setăm U2X și scriem UBRR = 1

$$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$$

$$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$$

Cu Busy Waiting

- Implementare cu busy waiting pentru ATmega324P pe USART0
    - Pentru USART1, înlocuiți 0 din numele tuturor registrelor cu 1 (de ex. UDR0 – UDR1)
- Baud rate setat prin setbaud.h
    - Pentru a calcula corect trebuie neapărat să definiți F_CPU!
- Busy waiting cu loop_until_bit_is_set(reg, bit)

```c
#include <avr/io.h>
#define F_CPU 12000000UL
#define BAUD 115200
#include <util/setbaud.h>

void initUSART(void)                              /* requires BAUD */
{
  UBRR0H = UBRRH_VALUE;                           /* defined in setbaud.h */
  UBRR0L = UBRRL_VALUE;
#if USE_2X
  UCSR0A |= (1 << U2X0);
#else
  UCSR0A &= ~(1 << U2X0);
#endif

  UCSR0B = (1 << TXEN0) | (1 << RXEN0);   /* Enable USART transmitter/receiver */
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); /* 8 data bits, 1 stop bit */
}


void transmitByte(uint8_t data)
{
  loop_until_bit_is_set(UCSR0A, UDRE0);   /* Wait for empty transmit buffer */
  UDR0 = data;                            /* send data */
}

uint8_t receiveByte(void)
{
  loop_until_bit_is_set(UCSR0A, RXC0);    /* Wait for incoming data */
  return UDR0;                            /* return register value */
}

int main()
{
    uint8_t serialChar;
    while(1)
    {
        serialChar = receiveByte();
        transmitByte(serialChar);         /* Loopback incoming serial data */
    }
    return 0;
}
```

Cu întreruperi

- Eliminăm busy waiting, dar acum datele se pot pierde din buffer-ul perifericului (UDR0) dacă nu îl citim la timp
  - UDR poate reține doar un singur caracter!
- Introducem un buffer mai mare în software care să adreseze problema

```c
#include <avr/io.h>
#define F_CPU 12000000UL
#define BAUD 115200
#define BUFSIZE 100
#include <util/setbaud.h>

void initUSART(void)                                   /* requires BAUD */
{
  UBRR0H = UBRRH_VALUE;                                /* defined in setbaud.h */
  UBRR0L = UBRRL_VALUE;
#if USE_2X
  UCSR0A |= (1 << U2X0);
#else
  UCSR0A &= ~(1 << U2X0);
#endif

  UCSR0B = (1 << TXEN0) | (1 << RXEN0) | (1 << RXCIE0); /* Enable USART */
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);               /* 8N1 */
}

volatile uint8_t rxBuffer[BUFSIZE];
volatile uint8_t readIndex = 0, writeIndex = 0;

ISR(USART0_RX_vect){
    uint8_t nextWrite = (writeIndex + 1) % BUFSIZE;
    if(nextWrite != readIndex) /* If buffer is not full */
    {
        rxBuffer[writeIndex] = UDR0;
        writeIndex = nextWrite;
    }
}

int main()
{
    initUSART(); /* Init with RXCIE flag enabled! */
    sei();
    while(1)
    {
        while(readIndex != writeIndex)
        { /* As long as there are characters in the buffer */
            LCD_putchar(rxBuffer[readIndex]);
            readIndex = (readIndex + 1) % BUFSIZE;
        }
    }
    return 0;
}
```

printf()

- Putem să legăm *stdout* la ieșirea portului serial
- Permite folosirea printf() pentru a afișa date direct în consola serială
- Similar, putem lega *stdin* și folosi scanf()
- Implicit printf() și scanf() lucrează doar cu întregi. Dacă aveți nevoie de floats, trebuie să adăugați linker flags:

```
$ avr-gcc -std=gnu99 -Wl,-u,vfprintf -lprintf_flt -Wl,-u,vfscanf -lscanf_flt -lm -mmcu=atmega324p main.o -o main.elf
```

https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html
https://www.youtube.com/watch?app=desktop&v=JrsaKc2hVac

```c
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>
static int uart_putchar(char c, FILE *stream);
static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL,
                                         _FDEV_SETUP_WRITE);

static int uart_putchar(char c, FILE *stream)
{
  if (c == '\n') uart_putchar('\r', stream);

  loop_until_bit_is_set(UCSR0A, UDRE0);
  UDR0 = c;

  return 0;
}

int main()
{
  uint32_t count = 0;

  initUSART();
  stdout = &mystdout;

  printf("Hello, world!\n");

  while(1)
  {
    printf("Counter value: %d \n", count++);
    _delay_ms(1000);
  }
  return 0;
}
```

Subiecte de discuție

- Cât de rapid putem opera un UART?
- Care sunt limitările?
- De ce avem nevoie de biți de start/stop?
- Câți biți de date putem transmite?
  - De ex. 115200 baud, parity even, 8 data bits, 2 stop bits