6. Comunicaţia serială

Comunicația asincronă
Exemplu de utilizare a comunicației seriale asincrone
Aplicație propusă

Sistemul de comunicație serial permite transferul de date bit cu bit între doua sisteme digitale (de exemplu, două microcontrolere sau un microcontroler și un calculator).Fiind un modul necesar în majoritatea aplicațiilor integrate, majoritatea plăcilor de dezvoltare (de exemplu: Arduino Uno) conțin cel puțin un port serial (cunoscut și sub denumirea de UART sau USART). Transmiterea datelor între dispozitive implică existența a minim doua canale (fire). Canalul TX transmite date, respectiv canalul RX primește date. Canalul TX al dispozitivului care transmite se conectează la canalul RX al dispozivului care primeste și vice-versa pentru celalalt canal. În cadrul unui microcontroler dacă acești pini sunt folosiți pentru comunicația serială, ei nu mai pot fi utilizați pentru alte funcții (de exemplu ca pini de intrare/ieșire).

Mediul de dezvoltare Arduino permite utilizarea unui modul predefinit de monitorizare a sistemului de comunicații serial, prin "*Serial Monitor*" și "*Serial Plotter*", din meniul Tools. Descrierea acestor funcționalități poate fi regasită în secțiunea § 1.2.

Modulul de comunicație serială, conținut de microcontrollerul ATMega328P, permite funcționarea într-o manieră sincronă sau asincronă, cu posibilitatea de a se utiliza rate de transfer foarte mari. Transferul de date se poate face cu pachete de date de dimensiuni ce variază de la 5 biți până la 9 biți. La aceștia se adaugă 1 sau 2 biți de stop. La nivel hardware, este implementat un mecanism de detectare a erorilor de transmisie, verificarea realizându-se prin determinarea parității mesajului.

Configurarea modulului de comunicație serială se realizează prin următoarele registre: UDR0, UCSR0A, UCSR0B, UCSR0C. Primul registru, UDR0, reprezintă registrul buffer din care se face transmiterea, respectiv în care se face recepționarea datelor pe serială. Acesta are dimensiunea de 8 biți. Deoarece un mesaj transmis pe serială are o dimensiune de 5-7 biți, partea superioară neutilizată a registrului este ignorată la transmitere și setată cu zero la recepție.

La transmisie, datele din registrul buffer UDR0 sunt încărcate într-un registru de deplasare și apoi sunt transmise serializat pe pinul Tx. La recepție, datele citite pe pinul Rx sunt încărcate într-un buffer FIFO. Dupa umplerea bufferului, datele sunt copiate în registrul UDR0 unde utilizatorul are access pentru copiere.

Structura generală a Registrului de date UDR0 este prezentată în figura 6.1. Biții acestui regitru sunt aceeași cu cei ai buffer-ului de transmisie sau recepție pe serială.

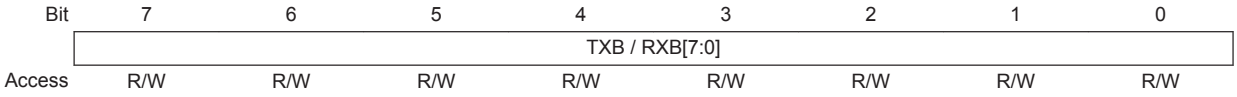


Fig. 6.1 Registrul UDR0.

Registrul de Control și stare UCSR0A conține următorii biți: (figura 6.2):

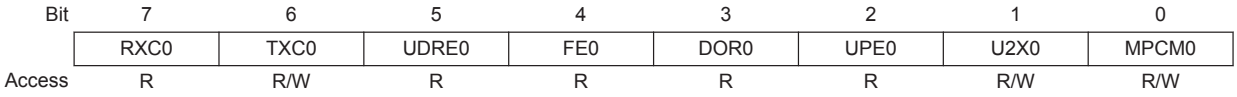
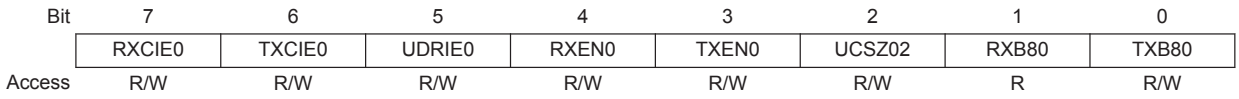


Fig. 6.2 Registrul UCSR0A.

- Bitul 7 – RXC0: Recepție completă de date: acest bit este setat când sunt date necitite în buffer-ul de recepție. Valoarea zero a acestui registru înseamnă că nu sunt date necitite;
- Bitul 6 – TXC0: Transmisie completă: valoarea acestui bit este 1 atunci când întreg mesajul din registrul de deplasare a fost transmis în exterior. Bitul este automat șters când este finalizată întreruperea de transmisie completă, atașată;
- Bitul 5 – UDRE0: Registru de date gol: dacă acest bit este 1, bufferul de date este gol și poate fi scris;
- Bitul 4 – FE0: eroare mesaj: acest bit este setat în cazul unei erori de transmisie;
- Bitul 3 – DOR0: este setat în cazul unei condiții de rulare peste date existente. O astfel de condiție poate fi când buffer-ul de recepție este plin, un nou caracter așteaptă în registrul de deplasare și este detectat un bit nou de start;
- Bitul 2 – UPE0: Eroare de paritate: acest bit este setat dacă următorul caracter în buffer-ul de recepție are o eroare de paritate;
- Bitul 1 – U2X0: dublare viteză transmisie pe serială: în mod asincron, prin setarea acestui bit se reduce factorul de divizare al ratei de transfer de la 16 la 8, astfel realizându-se dublarea vitezei de comunicație;
- Bitul 0 – MPCM0: funcționare în mod multi-procesor: activează funcționarea în mod multi-procesor.

Al treilea registru important pentru realizarea comunicației seriale este registrul de Control și stare UCSR0B figura 6.3. Biții acestui registru sunt următorii:

- bitul 7 – RXCIE0: Activare întrerupere recepție completă: setarea acestui bit va activa mecanismul de întrerupere pentru recepție;

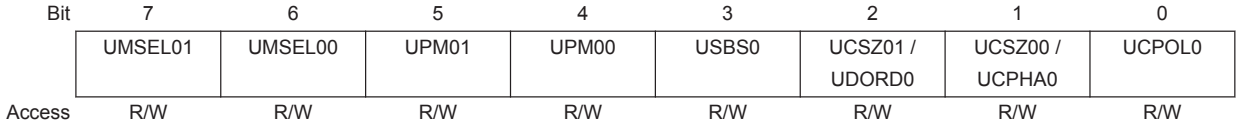


Bit	7	6	5	4	3	2	1	0
	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W

Fig. 6.3 Registrul UCSR0B.

- bitul 6 – TXCIE0: Activare întrerupere transmisie completă: setarea acestui bit va activa mecanismul de întrerupere pentru transmisie;
- bitul 5 – UDRIE0: Activare întrerupere registru de date gol: activarea acestui bit activează întreruperea de registru gol;
- bitul 4 – RXEN0: Activare recepție: setarea acestui bit conduce la activarea mecanismului de recepție;
- bitul 3 – TXEN0: Activare transmisie: setarea acestui bit conduce la activarea mecanismului de transmisie;
- bitul 2 – UCSZ02: dimensiune caracter: În combinație cu biții UCSZ0[1:0] din registrul UCSR0C, permite setarea numărului de biți de date transmiși sau recepţionaţi într-un mesaj;
- bitul 1 – RXB80: Bitul 9 al unui mesaj pe 9 biți la recepție. Trebuie citit înaintea citirii biților din UDR0;
- bitul 0 – TXB80: Bitul 9 al unui mesaj pe 9 biți la transmisie. Trebuie scris înaintea scrierii biților din UDR0.

Registrul UCSR0C permite configurarea formei mesajului ce urmează a fi transmis sau recepționat. Următorii doi biți fac parte din acest registru (figura 6.4):



Bit	7	6	5	4	3	2	1	0
	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 / UDORD0	UCSZ00 / UCPHA0	UCPOL0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Fig. 6.4 Registrul UCSR0C.

- Biții 7:6 – UMSEL0n: selectare mod [n=1:0]: acești biți permit setarea modului de funcționare USART0, astfel:
  - 00: comunicație serială asincronă;
  - 01: comunicație serială sincronă;
  - 10: rezervat;
  - 11: Master SPI.
- Biții 5:4 – UPM0n: mod paritate [n=1:0]: Acești biți activează și setează tipul de paritate generat și verificat. Dacă este activat, transmițătorul va genera și va trimite automat paritatea mesajului curent. Receptorul va genera la rândul său paritatea mesajului recepționat. Următoarele combinații sunt posibile:
  - 00: dezactivat;
  - 01: rezervat;
  - 10: activat, paritate pară;

  - 11: activat, paritate impară.
- Bitul 3 – USBS0: selecție număr de biți de stop: acest bit permite selectarea numărului de biți de stop. Valoarea zero reprezintă un bit de stop, iar valoarea unu corespunde pentru doi biți de stop;
- Biții 2:1 – UCSZ0n: dimensiune mesaj/ordinea datelor [n=1:0]: biții UCSZ0[1:0] combinați cu bitul UCSZ02 din registrul UCSR0B permit setarea numărului de biți de date (dimensiunea mesajului) la recepție și transmisie. Următoarele combinații pot fi posibile:
  - 000: 5 biți de date;
  - 001: 6 biți de date;
  - 010: 7 biți de date;
  - 011: 8 biți de date;
  - 100: rezervat;
  - 101: rezervat;
  - 110: rezervat;
  - 111: 9 biți de date.
- Bitul 0 – UCPOL0: Polaritate semnal ceas.

6.1 Exemplu de aplicaţie

Să se elaboreze și simuleze, utilizându-se Tinkercad, un program care transmite pe serială mesajul "Laborator SMC". Pentru aceasta se va utiliza un modul Arduino Uno și funcționalitatea de Serial Monitor.

Programul care realizează cerinţele impuse este următorul:

Algoritmul 6.1 Utilizarea sistemului de comunicații seriale.

```
1  /* UART SERIAL DEFINES */
2  #define BAUD 9600
3  #define MYUBRR F_CPU/16/BAUD-1
4
5  /* configurare UART */
6  void USART_Init( unsigned int ubrr)
7  {
8     /*Setare baud rate */
9     UBRR0H = (unsigned char)(ubrr>>8);
10    UBRR0L = (unsigned char)ubrr;
11
12   /*activare emitor si receptor */
13    UCSR0B = (1<<RXEN0)|(1<<TXEN0);
14
15    /* setare format: 8data, 2stop bit */
16    UCSR0C = (1<<USBS0)|(3<<UCSZ00);
```

```
17 }
18
19 /* Functia de transmitere pe seriala*/
20 void USART_Transmit( unsigned char data )
21 {
22     while(!(UCSR0A & (1<<UDRE0)));
23     UDR0 = data;
24 }
25
26 //Functia de receive date poe seriala
27 unsigned char USART_Receive( void )
28 {
29    /* Wait for data to be received */
30     while ( !(UCSR0A & (1<<RXC0)) )
31
32     /* Preliare si returnare a datelor receptionate din buffer */
33     return UDR0;
34 }
35
36 //Functie pentru transmiterea unui sir de caractere pe seriala
37 void SendString(char *StringPtr)
38 {
39     while(*StringPtr != 0x00)
40     {
41     USART_Transmit(*StringPtr);
42     StringPtr++;
43     }
44 }
45
46 void setup ()
47 {
48     //Setare comunicatie Seriala
49     USART_Init(MYUBRR);
50 }
51
52 unsigned char c;
53 void loop(){
54         SendString("Laborator SMC\n");
55         delay(1000);
56 }
```

În programul anterior este prezentat un exemplu de implementare a funcţiilor de scriere și citire folosindu-se modulul de comunicație serială. Acest program conține, pe lângă funcțiile de scriere și citire, și o metodă prin care se poate transfera un șir de caractere pe serială.

Programul conține la început două directive de preprocesare. Acestea permit definirea frecvenței la care se dorește realizarea transferului de date pe modulul serial.

Funcția "USART_Int" permite configurarea modulului de comunicație serială. Aceasta presupune setarea ratei de transfer între emitor și receptor, activarea modulului de emitor și receptor, configurarea pachetului transmis. Funcțiile de transmitere și recepționare date permit transferul sau recepția unui byte, folosindu-se registrele descrise anterior. Funcția "SendString" permite trimiterea unui șir de caractere pe serială. Principiul constă în transferul byte cu byte al mesajului.

În funcția de inițializare este facută setarea modulului. În bucla infinită, la fiecare secundă, este transferat mesajul dorit pe serială.

6.2 Aplicaţie propusă

Să se implementeze un mecanism care citește pe serială o comandă pentru un modul PWM. Comanda PWM este atașată unui motor de cc. Comanda este reprezentată de un șir de caractere, format din 2 bytes. După ce comanda a fost trimisă către modulul de PWM, este trimis pe serială un mesaj de succes.