ATENTIE:
Comentariile, indicațiile și sugestiile sunt scrise cu verde deschis. Și acest paragraf este scris cu verde deschis. Tot ce este scris cu verde deschis nu face parte din rezolvare, așa cum ar apărea la examen.

1. Tip 1, 8 (ROM-RAM, timer pwm) Problema 40.

I. Să se proiecteze un microsistem bazat pe microcontrolerul ATMega162. Schema bloc a microcontrolerului ce conține latchul de adresă este prezentată în figura 1. Microsistemul va conţine 32KB de memorie SRAM implementaţi cu circuite W24129A. Schema bloc a circuitului W24129A este prezentată în figura 2 iar descrierea pinilor în figura 3.

W24129A 16Kx8 HIGH-SPEED CMOS STATIC RAM

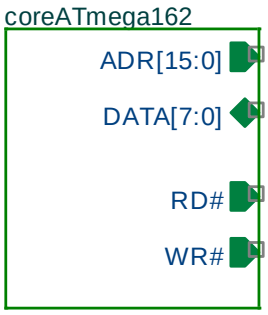


figura 1

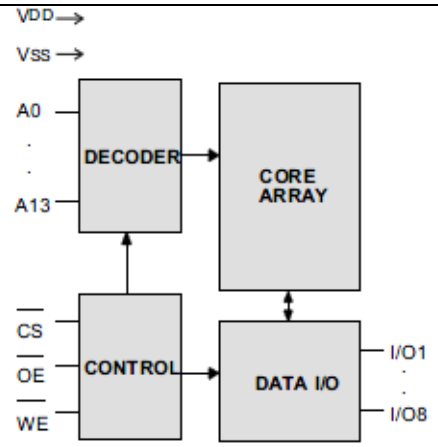


figura 2

PIN DESCRIPTION

SYMBOL	DESCRIPTION
A0–A13	Address Inputs
I/O1–I/O8	Data Inputs/Outputs
$\overline{CS}$	Chip Select Input
$\overline{WE}$	Write Enable Input
$\overline{OE}$	Output Enable Input
VDD	Power Supply
Vss	Ground

figura 3

Cei 32 KB se mapează contiguu începând de la adresa 4000h. Se va folosi decodificarea completă.

Se cere schema simplificată a microsistemului (cu magistrale, ca în prelegerea 1) ce conţine nucleul ATmega162, circuitele RAM şi blocurile de decodificare. Pentru decodificatoarele de adresă se cere doar forma minimă, fără a fi necesară implementarea cu un anumit tip de porţi sau alte dispozitive logice. 2 **puncte**

II. La un microcontroler ATmega16 trebuie conectate **opt** LED-uri notate L0, L1, …, L7 şi **şapte** push-butoane notate B0, B1, …, B6. Fiecare push-buton are montat în interior un LED: L0 este montat în interiorul lui B0, L1 este montat în interiorul lui B1,… şi L6 este montat în interiorul lui B6. **Orice** apăsarea a unui buton va produce schimbarea stării LED-ului interior: dacă acesta era stins se va aprinde iar dacă era aprins se va stinge. Această funcţionalitate trebuie implementată pentru toate cele 7 perechi LED – push-buton. Funcţionarea unei perechi este independentă de funcţionarea celorlalte 6 perechi. De exemplu, dacă se apasă B3 se va schimba numai starea lui L3; starea celorlalte LED-uri va rămâne neschimbată. Push-butoanele prezintă instabilitate.

LED-ul L7, cel fără push-buton, va lumina proporţional cu numărul de leduri L0-L6 aprinse. Intensitatea luminoasă a lui L7 variază după regula:

$$IL7 = \frac{n}{7} IL7_{\max}$$

IL7 este intensitatea curentă a ledului L7, $IL7_{max}$ este intensitatea luminoasă maximă a ledului L7 iar *n* este numărul de leduri L0-L6 aprinse. Intensitatea variabilă a lui L7 se va baza pe tehnica PWM. Durata ciclului timerului trebuie să fie în plaja 10-20ms, cât mai aproape de 10 ms. Frecvenţă ceasului procesor este 5 MHz.

Să se prezinte tabelar modul de conectare a ledurilor şi a push butoanelor la ATmega16. Să se scrie programul C care controlează sistemul ce funcţionează conform descrierii anterioare:

a) Calculele necesare pentru utilizarea timerului. **1 punct**

b) Tabelul de conectare a resurselor, gestionarea push-butoanelor şi a ledurilor L0-L6 **3 puncte**
c) Codul pentru gestiunea ledului L7. Punctajul la acest punct se acordă numai dacă la punctul b) s-a obținut cel puțin jumătate din punctaj. **3 puncte**

Atenţie: programul trebuie să aibă o singură funcţie main şi o singură bucla principală while(1). Codul care nu se integrează în această structură nu se punctează.

Rezolvare:

I.

Deoarece trebuie mapaţi 32 KB iar un modul are capacitatea de memorare de 16 KB, rezulta ca sunt necesare 2 module de memorie (32/16=2). Primul modul se va mapa începând cu adresa 4000h.

Numărul de adrese ale modulului de memorie este 14. Tabelul pentru maparea primului modul de memorie este (se urmărește procedura din prelegerea 1, capitolul 3 - Decodificarea completă (minimă) – exemplu):

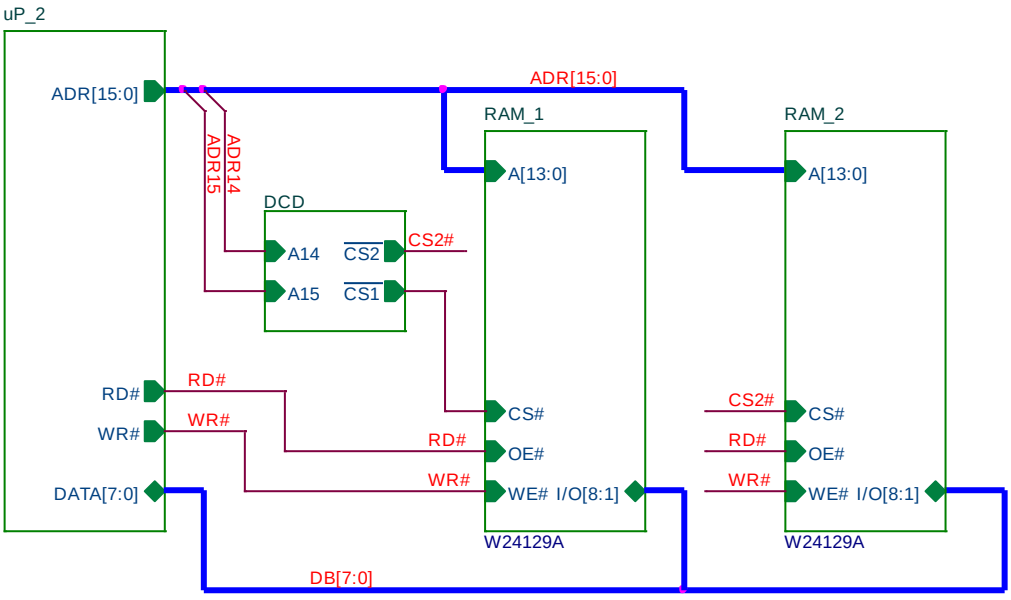
Adresa procesor ADR(15:0)= 4000h = 0100 0000 0000 0000b																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000h
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	...
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7fffh

Al doilea modul se mapează „lipit” de primul modul pentru a crea un spaţiu de contiguu. Adresa care urmează după ultima adresa din primul modul este prima adresa din alocată celui de-al doilea modul. Valoarea acesteia este 7fffh+1=8000h.

Pentru al doilea modul se urmează aceeaşi procedura ca pentru primul modul. Numărul de adrese ale modulului de memorie este tot 14. Tabelul pentru maparea primului modul de memorie este:

Adresa procesor ADR(15:0)= 8000h = 1000 0000 0000 0000b																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000h
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	...
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Bfffh

Schema simplificată a microsistemului:



Funcţiile CS1# şi CS2# se implementează din adresele procesor ADR15 şi ADR14, adrese care nu sunt conectate la pinii de adresă ai memoriei. CS1# este ‚0’ logic când A[15:14]=”01” (vezi primul tabel) iar CS2# este ‚0’ logic când A[15:14]=”10”. Formele logice pentru aceste funcţii sunt:

$$CS1\# = A15 + \overline{A14}$$

$$CS2\# = \overline{A15} + A14$$

Cele două funcții sunt implementate în blocul de decodificare DCD din figura de mai sus.

II.

Tabelul cu modul de conectare a ledurilor şi a push butoanelor:

Push buton	Pin pt. push buton	LED	Pin pt. LED
B0	PC0	L0	PD0
B1	PC1	L1	PD1
B2	PC2	L2	PD2
B3	PC3	L3	PD3
B4	PC4	L4	PD4
B5	PC5	L5	PD5
B6	PC6	L6	PD6
		L7	**PB3/OC0**

În loc de tabel se poate folosi notaţia vectorială din prelegerea 1, pagina 16:

B[6:0] → PC[6:0], L[6:0] → PD[6:0], L7→PB3/OC0

Buton apăsat înseamnă ‚0' logic pe pinul PC corespunzător iar buton neapăsat înseamnă ‚1'.
Orice LED L[0..6] se aprinde cu ‚1' logic pe pinul PD corespunzător şi se stinge cu ‚0'.
L7 este controlat PWM.

a. Calculele necesare pentru utilizarea timerului:

Pentru a controla luminozitatea LED-ului L7 vom alege pentru timer **modul fast PWM**. Pentru modul PWM plecăm de la relaţia 3 din prelegerea 5:

$$f_{CLK\_CPU} = p*N*fcycle = 256*p*fcycle \rightarrow f_{cycle} = \frac{f_{CLK\_CPU}}{256*p},\ T_{cycle} = \frac{256\ p}{f_{CLK\_CPU}}$$

Înlocuim în relaţia anterioară pe f_{CLK_CPU}. Pentru p=256 obţinem:

$$T_{cycle} = \frac{256\ p}{f_{CLK\_CPU}} = \frac{256\ 256}{5MHz} = \frac{2^{16}}{5\ 10^6\ Hz} = \frac{2^{16}}{5\ 10^3} 10^{-3}\frac{1}{Hz} = \frac{2^{16}}{5\ 10^3} ms = \frac{2^{16}}{5\ 2^3\ 5^3} ms = \frac{2^{16}}{2^3 5^4} ms = \frac{2^{13}}{5^4} ms = 13.1ms$$

Pentru p=128 obţinem un T_{cycle} de două ori mai mic, adică 6,5 ms iar pentru p=1024 obţinem un T_{cycle} de 4 ori mai mare, adică aproximativ 52 ms. Singura valoare a lui p pentru care T_{cycle} este în intervalul 10-20 ms este 256.

Vom folosi timerul 0 în mod fast PWM, cu *p*=256.

b) şi c) gestionarea push-butoanelor şi a ledurilor

```
#include <avr/io.h>
#define DELAY 30

int main(){
   unsigned char sample_ante = 0xff;
   unsigned char sample_now  = 0xff;
   unsigned char loop_cnt=0;

   unsigned char i, val, status =0, mask;

   DDRC=0;        // portul push butoanelor in mod IN

   DDRD=0xff;     // portul LED-urilor L[0..6] in mod OUT
   PORTD=0;       // Stinge toate LED-urile.
   DDRB=0xff;     // portul LED-ului L7 in mod OUT
```

```
  //        ┌──┬──Fast PWM
  TCCR0=0b01101100;
  //        || └┴┴─ clk I/O /256
  //        └┴─────Set OC0 on BOTTOM, Clear on compare.

  OCR0=0;     // L7 stins

  while (1){
     //se foloseşte ca model codul din prelegerea 4, pagina 5
     if(loop_cnt== DELAY){
        loop_cnt=0;
        sample_ante = sample_now;
        sample_now  = PINC;

        val=0;
        for(i=0; i<=6; i++){
           if( (sample_ante & 1<<i)!=0 && (sample_now & 1<<i)==0){
              //daca s-a apasat butonul i, bitul i din val va fi setat.
              val |=1<<i;
           }
        }

        //
        status ^=val; // inverseaza starea LED-urilor pentru care butonul corespunzator
                      // s-a apăsat
        PORTD=status;

        i=0;
        // numără câte din LED-urile L[0..6] sunt aprinse
        for(mask=1; mask<=0b01000000; mask<<=1){
           if(status & mask)
              i++; //in i se calculeaza numarul de LED-uri L[0..6] aprinse
           }

         // se calculează luminozitatea lui L7 cu regula de 3 simplă:
         // 7 LED-uri        255(100%)
         // i LED-uri         x
         // --------------------
         // x=255*i/7

         OCR0=i*255/7;
     }
     loop_cnt++;
  } //end while 1
  return 0;
}
```

2. Tip 2 (funcţii logice), Problema 53.

Să se proiecteze un microsistem bazat pe microcontrolerul ATmega16 care calculează 3 funcţii logice ce depind de 8 variabilele de intrare x_0, x_1,…, x_7. Cele 3 funcţiile logice se definesc astfel:

$$f_0 = x_0 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7$$

$$f_1 = x_0 + x_1 \cdot \overline{x_2} + x_3 \cdot \overline{x_4} + x_5 \cdot \overline{x_6} \cdot \overline{x_7}$$

f_2 este ‚1' când numărul de variabile de intrare care au valoarea ‚1' este mai mare decât numărul de variabile care au valoarea ‚0' şi ‚0' în rest.

Variabilele de intrare sunt generate prin intermediul comutatoarelor K0..K7 iar funcţiile logice sunt afişate prin intermediul LED-urilor L0, L1 şi L2. Când o funcție este ‚1', LED-ul care o afişează trebuie să fie aprins.

La microcontroler se mai conectează LED-ul L3. Intensitatea acestui LED este **direct proporţională** cu numărul X=x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0: dacă X=0b00000000 L3 este stins, dacă X=0b11111111 L3 luminează cu intensitatea maxima I_{max}, dacă X=128, L3 luminează cu jumătate din intensitatea I_{max}, etc.

Specificaţii şi cerinţe obligatorii:

1. Ceasului microcontrolerului se generează cu un cristal de **2,5 MHz**.
2. Intensitatea LED-ului L3 este controlată folosind tehnica PWM.
3. Valoarea lui Tcycle PWM trebuie să fie în intervalul [10ms, 15ms]
4. Programul trebuie să aibă o singură funcţie *main*() şi o singură bucla principală *while*(). Codul care nu se integrează în această structură nu se punctează.

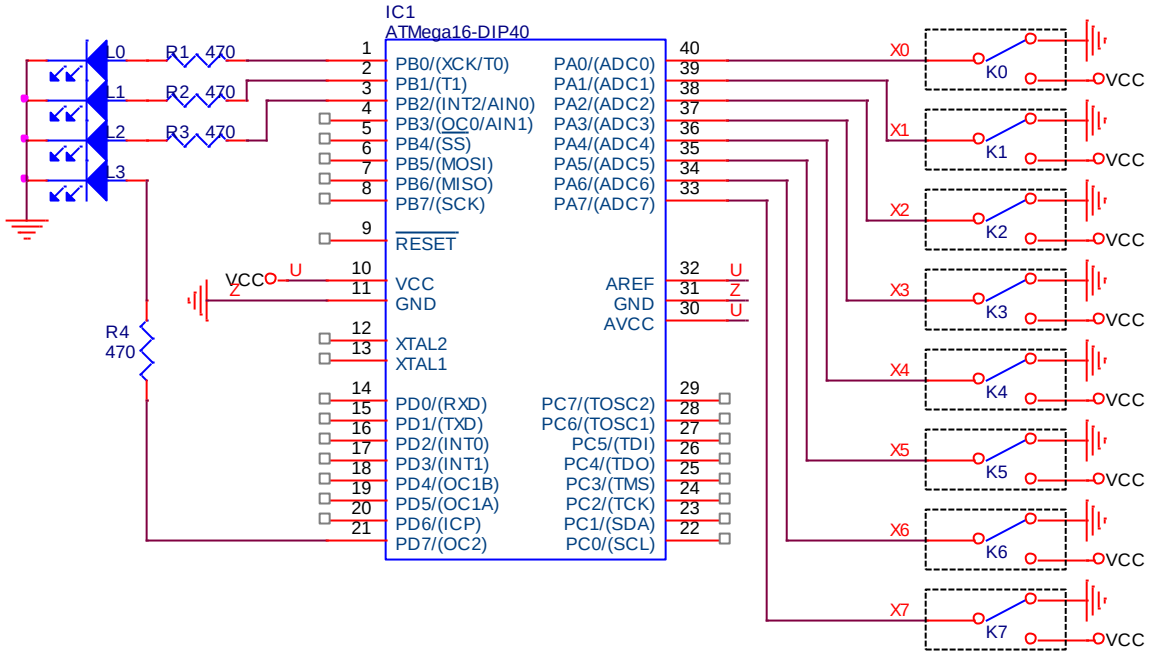
Se cere:

Schema şi programul C care controlează sistemul ce funcționează conform descrierii anterioare:

- **a.** Schema microsistemului, ce conține comutatoarelor K0..K7 şi LED-urile L0-L3 **1 punct.**
- **b.** Implementarea functiilor logice: f_0 – **1 punct**, f_1 – **2 puncte**, f_2 – **1 punct**
- **c.** Calculele necesare pentru stabilirea modului de funcţionare a timerului, calculul lui Tcycle şi iniţializarea timerului. Puteţi utiliza ce timer doriţi. Alegerea şi setările timerului (numărul timerului, modul, *p*, etc.) se justifică. **2 puncte**
- **d.** Setarea intensităţii lui L3. **2 puncte**

Rezolvare:

a) schema este asemănătoare cu schema din figura 2, laboratorul 3



c. Calculele necesare pentru utilizarea timerului:

Pentru a controla luminozitatea LED-ului L3 vom alege pentru timer **modul fast PWM**. Pentru modul PWM plecăm de la relaţia 3 din prelegerea 5:

$$f_{CLK\_CPU} = p*N*fcycle = 256*p*fcycle \rightarrow f_{cycle} = \frac{f_{CLK\_CPU}}{256*p},\ T_{cycle} = \frac{256\ p}{f_{CLK\_CPU}}$$

Înlocuim în relaţia anterioară pe f_{CLK_CPU}. Pentru p=256 obţinem:

$$T_{cycle} = \frac{256\ p}{f_{CLK\_CPU}} = \frac{256\ 256}{2{,}5MHz} = \frac{2^{16}}{2{,}5\ 10^6\ Hz} = \frac{2^{16}}{2{,}5\ 10^3}10^{-3}\frac{1}{Hz} = \frac{2^{16}}{25\ 10^2}ms = \frac{2^{16}}{2^2\ 5^4}ms = \frac{2^{14}}{5^4}ms = 26.2ms$$

Pentru p=128 obţinem un T_{cycle} de două ori mai mic, adică 13,1 ms iar pentru p=1024 obţinem un T_{cycle} de 4 ori mai mare, adică 104,8 ms. Singura valoare a lui p pentru care T_{cycle} este în intervalul 10-15 ms este 128.

Vom folosi timerul 2 deoarece este singurul timer care are p=128. Timerul 2 va fi folosit în mod PWM.

b, d.

```
#include <avr/io.h>
int main(){

   unsigned char inputs, x0, x1, x2, x3, x4, x5, x6, x7;
   unsigned char temp, i, mask, outs;

   DDRA=0;        // portul comutatoarelor K[0..7] in mod IN

   DDRB=0xff;     // portul LED-urilor L[0..2] in mod OUT
   PORTB=0;       // Stinge toate LED-urile.

   DDRD=0xff;     // portul LED-ului L3 in mod OUT

   //         ┌──┬─Fast PWM
   TCCR2=0b01101101;
   //         ||  └┴┴─ clk I/O /128
   //         └┴──────Set OC0 on BOTTOM, Clear on compare.
```

```
  OCR2=0;      // L3 stins

  while (1){
     //se foloseşte ca model laboratorul 3
     inputs =PINA;
     x0= inputs     & 1<<0;
     x1= inputs >>1 & 1<<0;
     x2= inputs >>2 & 1<<0;
     x3= inputs >>3 & 1<<0;
     x4= inputs >>4 & 1<<0;
     x5= inputs >>5 & 1<<0;
     x6= inputs >>5 & 1<<0;
     x7= inputs >>7 & 1<<0;

     outs=0;

     // f0 = x0 · x1 · x2 · x3 · x4 · x5 · x6 · x7
     if(inputs == 0xff){
        outs |=1<<0;
     }

     // f1 = x0 + x1 · x2' + x3 · x4' + x5 · x6' · x7'
     temp=x0 | (x1 & ~x2) | (x3 & ~x4) | (x5 & ~x6 & ~x7 );
     temp &=1;
     if(temp){
        outs |=1<<1;
     }

     // f2
     for(mask=1; mask<=0b10000000; mask<<=1){
        if(inputs & mask)
           i++; //in i calculeaza numarul de ,1'-uri din x
        }
     }
     if(i>4){
        outs |=1<<2;
     }

     PORTB=outs;

     // se calculează luminozitatea lui L7 cu regula de 3 simplă:
     // 255=max(inputs)        255(100%)
     // inputs                 x
     // ------------------------
     // x= inputs *255/255= inputs
     OCR2= inputs;

  } //end while 1
  return 0;
}
```

3. Tip 3,8 (7segmente, timere), Problema 42

Să se proiecteze un microsistem bazat pe microcontrolerul ATmega16 care va controla un zar electronic. Zarul este prevăzut cu un push-buton notat în continuare cu P şi 7 LED-uri notate a, b,…,g. Push-butonul este ideal, adică nu prezintă instabilitate la apăsare sau la eliberare.

Pentru ca rezultatul aruncării să fie aleator acesta trebuie generat pornind de la un eveniment aleatoriu. Acest eveniment este timpul cât P este apăsat. Cât P este apăsat se va incrementa o variabilă software. Fie *ta* (tastă apăsată) numele acestei variabile. Înainte de apăsare *ta* este iniţializat cu zero. Pe durata apăsării variabila *ta* este incrementată cât mai repede cu putinţă. Imediat după eliberarea lui P incrementarea lui *ta* se opreşte.

Valoarea lui *ta* obţinută la eliberarea lui P se împarte la 6, se determină restul şi în final la rest se adună o unitate. Restul+1 este rezultatul aruncării. De exemplu, dacă la eliberarea lui P valoarea variabilei este 60002, rezultatul aruncării este 60002 % 6+1=3.

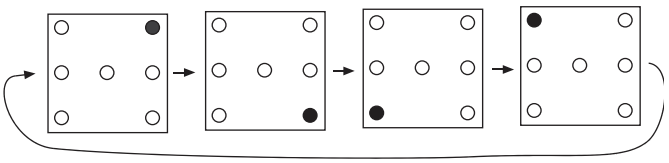


figura 4

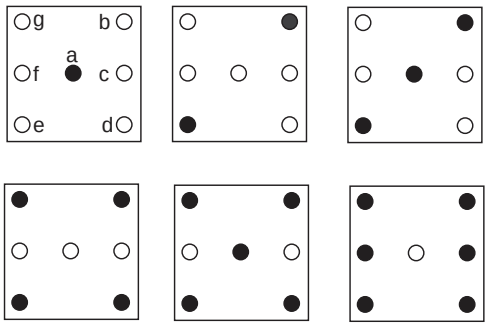


figura 5

Zarul electronic funcţionează după cum urmează:

1. La apăsarea push-butonului zarul începe să se „rotească”. Pentru a da senzaţia de rotire se afişează în continuu succesiunea din figura 4. Rotirea **începe întotdeauna** cu configuraţia cea mai din dreapta din figura 4.
2. Cât timp P este apăsat, oricare din configuraţiile pentru rotire din figura 4 se va afișa **exact** 0,2 secunde.
3. În momentul în care P este eliberat se afişează rezultatul aruncării. Rezultatul aruncării este un număr între 1 şi 6. Acest număr este afişat ca în figura 5.

Specificaţii şi cerinţe obligatorii:

1. Ceasului microcontrolerului se generează cu un cristal de **16 MHz**.
2. **Nu se admit** întârzieri implementate cu bucle soft de tip for(i=0;i<DELAY;i++){}
3. Programul trebuie să aibă o singură funcţie *main*() şi o singură bucla principală *while*(). Codul care nu se integrează în această structură nu se punctează.

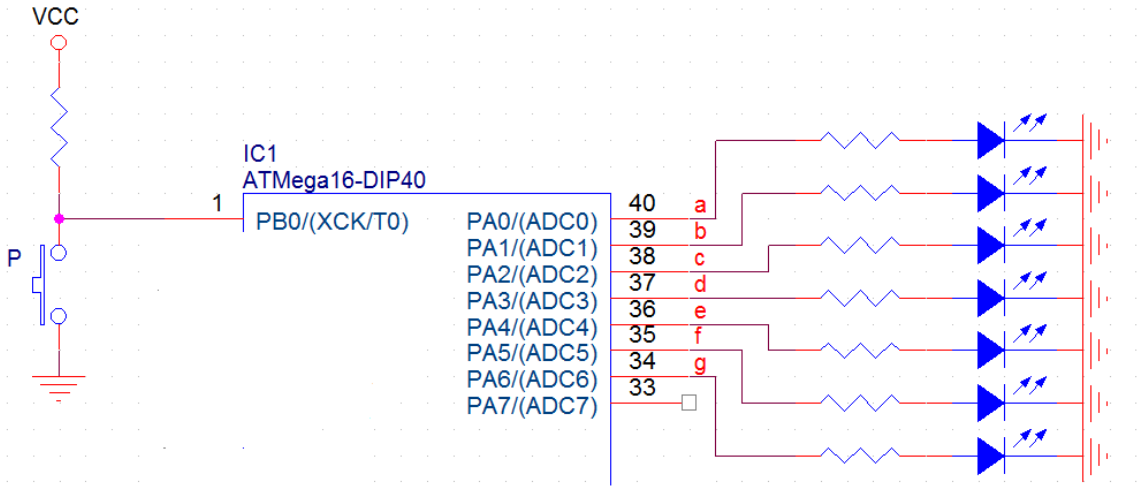
Implementarea care nu respecta cerinţele şi specificaţiile de mai sus **nu se punctează**.

Se cere:

1. Schema de conectarea al LED-urile a, b, …., g şi al push-butoanul P la pinii ATMega16 si configurarea porturilor. **1 punct**
2. Stabilirea modului de funcţionare a timerului, calculul lui Tcycle, iniţializarea timerului. Puteţi utiliza ce timer doriţi. Alegerea şi setările timerului (numărul timerului, modul, *p*, ieşirea OC) se justifică. Tcycle trebuie să fie mai mare sau egal cu 5ms. **2 puncte**
3. Utilizarea timerului pentru rotire şi afişarea „rotirii”. **3 puncte**
4. Gestiunea push-butonului, calcularea şi afişarea rezultatului aruncării. **3 puncte**

Rezolvare:

1. Schema de conectarea al LED-urile a, b, …., g şi al push-butoanul:



Schema de conectare a unui LED este prezentată în laboratorul 3, pagina 5. Schema de conectare a push-butonului este prezentată în prelegere 4, pagina 1.

Configurarea porturilor: DDRA=0xFF; DDRB=0;

2. Se aplică (sau se deduce) relaţia 5 din prelegere 5:

$$\frac{f_{CLK\_CPU}}{f_r} = kpN \qquad \textbf{Rel. 1}$$

f_{CLK_CPU}=16MHz, Tr=0,2s => f_r=1/T_r=1/0,2s=5Hz

$$\frac{f_{CLK\_CPU}}{f_r} = kpN, \frac{1610^6 Hz}{5Hz} = \frac{2^4 2^6 5^6}{5} = 2^{10}5^5$$

Pentru timerele 0 şi 1 *p* este una din valorile =1, 8, 64, 256, 1024. Pentru timerul 0 şi 2 N<256 iar pentru timerul 1 N < 65536.

Întâi se alge **cel mai mare *p*** cu condiția ca *p* = 1, 8, 64, 256 sau 1024. Alegem p=1024=2^{10}. Rămâne *k* N=5^5.

Din *k* N=5^5 **alegem cel mai mare N**. N=5^5=3125. Deoarece 256 < N < 65536 ar trebui să alegem timerul 1.

Ce rămâne este k, adică k=1.

Sau putem **alege cel mai mare N, N<256**. N=5^3=125 și rămâne k=5^2=25,. Deoarece N < 256 putem alege timerul 0.

Calculam T_{cycle} în fiecare caz:

Din prelegere 5, pagina 16: $T_r = k\,T_{cycle}$, $T_{cycle} = \frac{T_r}{k}$

- Pentru k=1 => T_{cycle}=0,2s/1=0,2 s = 200ms
- Pentru k=25 => T_{cycle}=0,2s/25 = 8 ms

Vom alege timerul 1 deoarece oferă cel mai mare T_{cycle}, adică 200ms. În concluzie **k=1, p=1024, N=3125**

Pentru timerul 1 alegem modul CTC deoarece este singurul mod modulo programabil. Registrele de control şi semnificaţia acestora este prezentată în atmega16a.pdf începând de la pagina 109. Pentru modul CTC ales modul este 0100 ia valoarea top este în OCR1A; ieşirile OC nu sunt utilizate. Valorile registrelor sunt:

```
//            ┌┐ CTC Mode
TCCR1B=0b00001101;
//            └┘└┘ clk I/O /1024
//              ┌┐CTC Mode
TCCR1A=0b00000000;
OCR1A=3125-1;
```

3,4. Valorile care se vor afişa se stochează în vectorul dots. În dots[0] se memorează configuraţia pentru ‚1', în dots[1] se memorează configuraţia pentru ‚2'etc. Valorile pentru rotire se memorează în dots[6..9]. Afişarea punctelor zarului este se face asemănător cu afişarea 7 segmente din laboratorul 4.

Din analiza cerinţelor rezultă că există 4 cazuri mutual exclusive:

- Acţiuni care se execută când P este neapăsat. Nu există acţiuni în această categorie.
- Acţiuni care se execută când P trece din neapăsat în apăsat, adică pe frontul coborâtor. Se resetează *ta* şi timerul 1 şi se iniţializează *val* (valoarea care se va afişa) cu dots[6] (prima configuraţie a rotirii. Detecţia frontului coborâtor este prezentată în prelegere 4, pagina 2.
- Acţiuni care se execută când P este apăsat. Se incrementează variabila *ta* şi de fiecare dată când timerul 1 ciclează, se schimba configuraţia pentru rotire.
- Acţiuni care se execută când P trece din apăsat în neapăsat, adică pe frontul ridicător. Se calculează valoarea care se va afişa şi se afişează. Detecţia frontului ridicător NU este prezentată dar se face asemănător cu detecţia frontului coborâtor. Se testează capacitatea de adaptare la situaţii noi, dar asemănătoare cu cele întâlnite în prealabil.

```
#define P 0
int main(){
   unsigned char sample_ante = 1, sample_now  = 1, val;
   unsigned long int ta=0;
                        //gfedcba
   unsigned char dots[]={0b0000001,  //1
                         0b0010010,  //2
                         0b0010011,  //3
                         0b1011010,  //4
                         0b1011011,  //5
                         0b1111110,  //6
                         0b0000010,  //I
                         0b0001000,  //II
                         0b0010000,  //III
                         0b1000000   //IV
                         };

   DDRA=0xff;
   DDRB=0;

   //            ┌┐ CTC Mode
   TCCR1B=0b00001101;
   //            └┘└┘ clk I/O /1024
   //              ┌┐CTC Mode
   TCCR1A=0b00000000;
   OCR1A=3125-1;

   while(1){

      sample_ante = sample_now;
      sample_now  = PINB;

      //front coborator=apăsare: cerinta 3
      if( (sample_ante & 1<<P)!=0 && (sample_now & 1<<P)==0 ){
         ta=0;
```

```
         TCNT1=0;
         val=6;
      }

      //push butonul P apasat: cerinta 3
      else if ( (PINB&1)=0 ){
         ta++;
         PORTA=dots[val];
         if(TIFR & 1<< OCF1A){
            TIFR |= 1<< OCF1A;
            val++;
            if(val == 10)
               val=6;
         }
      }

      //front ridicator = eliberare. Cerinta 4
      else if ((sample_ante & 1<<P)==0 && (sample_now & 1<<P)!=0) {
         val = ta%6;
         PORTA = dots[val];
      }

   }//end while
}//end main
```

4. Tip 4 (extensie porturi), Problema 64.

Să se proiecteze un microsistem bazat pe microcontrolerul ATMega16 care controlează o ghirlandă luminoasă cu următoarele specificaţii:

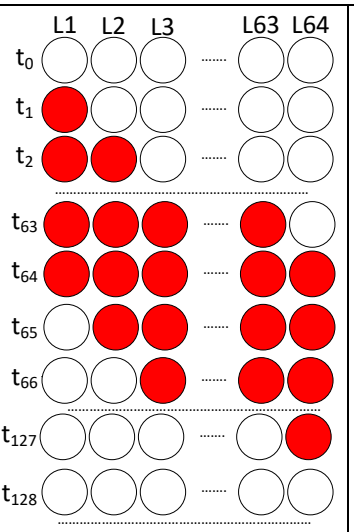


figura 6

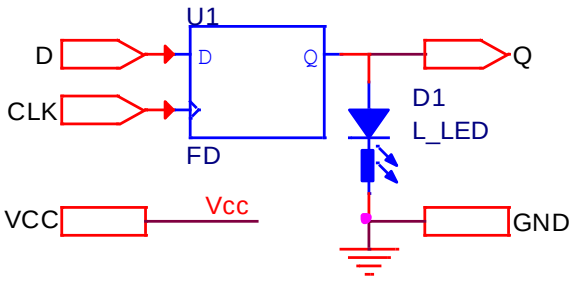


figura 7

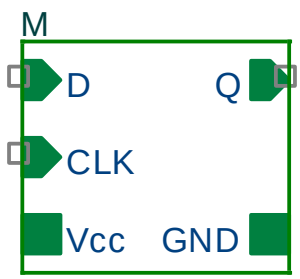


figura 8

a. Ghirlanda este alcătuit din 64 LED-uri. Acestea vor lumina ca în figura 6 (numărător Johnson). În figură pe verticală s-a figurat timpul. La momentul t_0 nu este nici un LED aprins. Apoi LED-urile se aprind unul câte unul, la rând, până când sunt aprinse toate: la momentul t_1 se aprinde L1, la t_2 sunt aprinse L1 şi L2, etc. La t_{64} sunt aprinse toate. În continuare LED-urile se sting pe rând până când la t_{127} este aprins numai L64. Următoarea configuraţie, la t_{128}, are toate LED-urile stinse şi este identică cu configuraţia de la t_0. Secvenţă t_0-t_{127} se repetă la nesfârşit.

b. Fiecare configuraţie din figura 6 este menţinută o perioadă de timp de 0,1 secunde.

c. Oricare LED al ghirlandei este controlat de un bistabil D **comutabil pe front ridicător**. Bistabilul D şi LED-ul sunt conectate ca în figura 7. Schema din figura 7 este implementată pe un circuit (cablaj) şi formează un modul independent numit în continuare modul M (figura 8). Ghirlanda este alcătuit din 64 de module M interconectate astfel încât sa formeze un **registru de deplasare**.

d. Se garantează că la punerea sub tensiune toate bistabilele sunt în starea ‚0'.

Specificaţii şi cerinţe obligatorii:

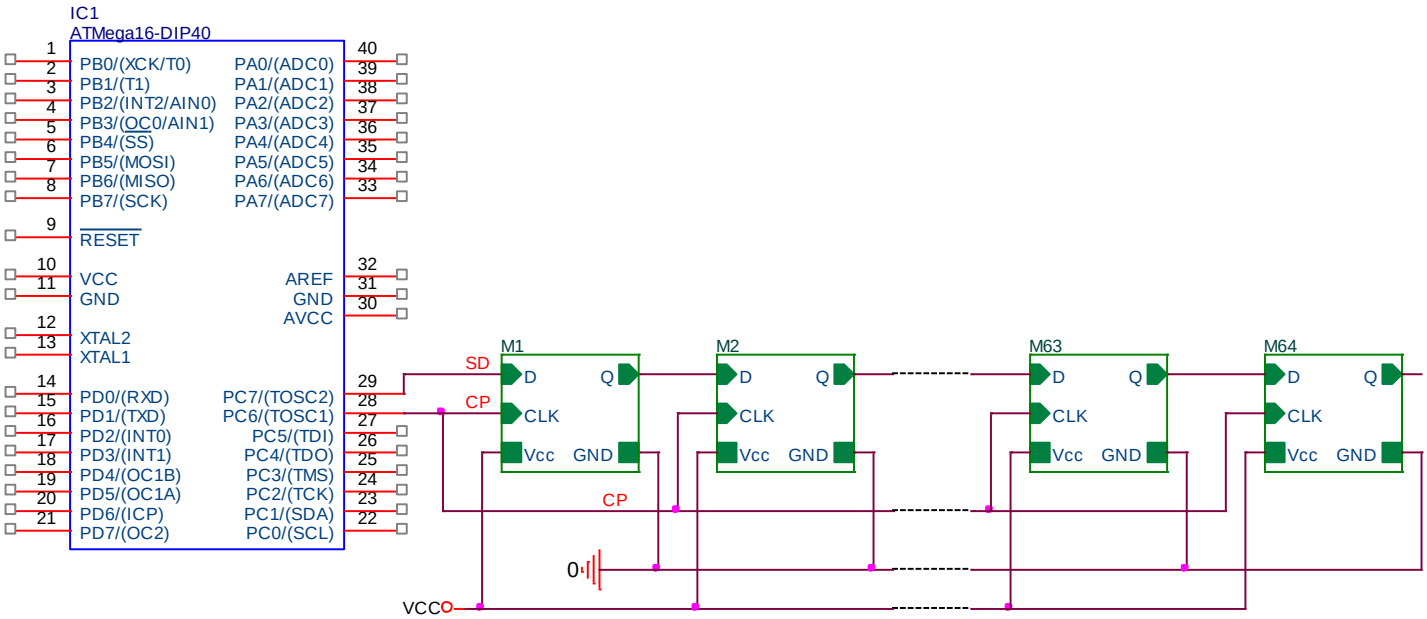
1. Ceasului microcontrolerului se generează cu un cristal de **8 MHz**.
2. Tcycle este **obligatoriu 0,1s**.
3. **Nu se admit** întârzieri implementate cu bucle soft de tip for(i=0;i<DELAY;i++){}
4. Programul trebuie să aibă o singură funcţie *main*(), o singură bucla principală *while*() şi ISR-uri (dacă este cazul). Codul care nu respectă această structură **nu se punctează**.

Se cere:

1. Schema microsistemului format din microcontrolerul ATMega16, la care se conectează modulele M. Se vor figura doar primul modul, al doilea modul, penultimul şi ultimul modul. Restul modulelor se vor figura cu …(puncte). Pe schemă modulele se vor reprezenta prin intermediul schemei bloc din figura 8. **2 puncte**
   Atenţie: dacă în loc de schema bloc se foloseşte schema detaliată sau dacă se desenează mai mult de 4 module M, nu se acordă punctajul.
2. Stabilirea modului de funcţionare a timerului. Puteţi utiliza ce timer doriţi. Alegerea şi setările timerului (numărul timerului, modul, *p*, eventual ieşirea OC) se justifică. **Implementarea fără justificare sau care nu respectă condiţia asupra lui Tcycle nu se punctează. 2 puncte**
3. Programul C pentru ATMega16:
   1. Setări porturi, iniţializarea timerului **1 punct**
   2. Controlul LED-urilor conform figurii 1. **4 puncte**

Rezolvare:

1. registrele de deplasare sunt tratate în cursul de BLPC (LDDC) (http://www.cs.ucv.ro/staff/edumitrascu/DSD/LD2_05.pdf)



2.
Ştim că

T_{cycle}=0.1 s f_{CLK_CPU}=8MHz T=1/f

Aplicăm relaţia 2 din prelegere 5:

$T_{cycle} = p * N * T_{CLK\_CPU}$

$T_{cycle}\ f_{CLK\_CPU} = pN$

$T_{cycle}\ f_{CLK\_CPU} = 0{,}1s\ 8MHz = 10^{-1}s\ 8\ 10^{6}Hz = 8\ 10^{5} = pN \qquad 2^3 2^5 5^5 = pN \qquad 2^8 5^5 = pN$

p este evident 2^8 iar $N=5^5=3125$. Deoarece N=3125>255, vom folosi timerul 1.

În concluzie vom folosi timerul 1 în mod CTC cu p=256 şi N=3125.

3.

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

#define clrbit(var,bit) var &= ~(1<<bit)
#define setbit(var,bit) var |= 1<<bit

#define SD 7
#define CP 6
#define N  64

int main(){
   unsigned char i;
   //3.1
   // data serială (SD)pe PC7 iar ceasul de serializare (CP) pe PC6
   // soluţia este aproape identică cu exemplul din prelegerea 3, capitolul
   // „Extensie de porturi de ieşire cu registre serie-paralel”, pagina 11
   PORTC=0;
   DDRC=0xFF;
```

```
    //           ┌┌── CTC Mode
    TCCR1B=0b00001100;
    //             └└└── clk I/O /256
    //              ┌┌──CTC Mode
    TCCR1A=0b00000000;

    OCR1A = 3125-1;

    //3.2
    while(1){
       for(i=1;i<=2*N;i++){
          while((TIFR & 1<<OCF1A)==0){}
          TIFR |= 1<<OCF1A;

          //genereaza bitul i
          // în exemplul amintit anterior se trimiteau biţii variabilei data.
          // Acum se trimit 64 de ‚1' urmaţi de 64 de ‚0'
          if(i<=N)
             setbit(PORTC, SD);
          else
             clrbit(PORTC, SD);

          //genereaza puls CP
          setbit(PORTC, CP);
          clrbit(PORTC, CP);
       }
    }//end while 1
    return 0;
}
```

5. Tip 6, 9, 11 (tastatură, intreruperi, protocol), Problema 48.

Să se proiecteze un microsistem bazat pe microcontrolerul ATmega16 care va îndeplini funcţia de controler de tastatură. Microcontrolerul va genera codului tastei apăsate şi va transmite în exterior acest cod serial. Matricea de taste din care este alcătuită tastatura este prezentată în figura 9. Schema bloc a tastaturii este prezentată în figura 10.

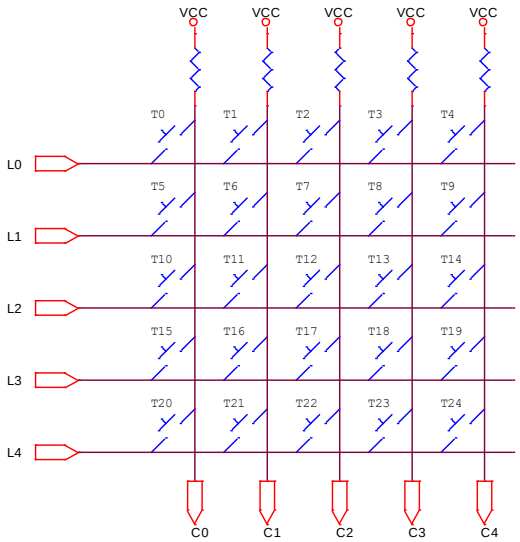


figura 9

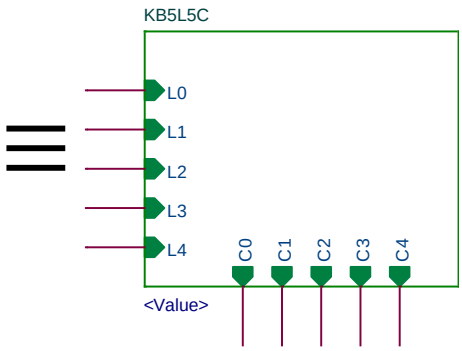


figura 10

Codul tastei este chiar indexul tastei, adică este codul de scanare:

	C_4	C_3	C_2	C_1	C_0
T0 →	0	0	0	0	0
T1 →	0	0	0	0	1
T2 →	0	0	0	1	0
T3 →	0	0	0	1	1
.....					
T23→	1	0	1	1	1
T24→	1	1	0	0	0

c_4 este MSb iar c_0 este LSb.

Biţii $c_4c_3c_2c_1c_0$ se vor emite serial prin intermediul unui pin IO. Semnalul generat prin intermediul acestui pin se numeşte **Data**. Emisia serială începe cu c_0 şi se termină cu c_4. Fiecărui bit îi corespunde un puls cu durata de **100µs**. Factorul de umplere este 80% pentru un bit de ‚1' şi 20% pentru un bit de ‚0'. Când nu este nimic de transmis linia Data este ‚0'. Ca exemplu, în figura următoare este prezentat timingul emisiei codului 11001:

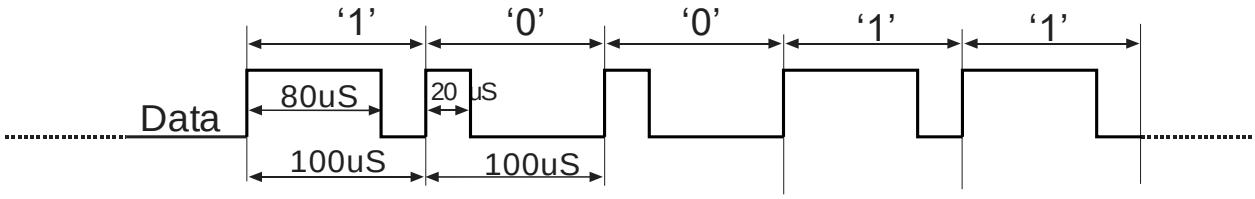


figura 11

Specificaţii şi cerinţe obligatorii:

1. Durata instabilității pentru pushbuton mai mica de 10ms.
2. Ceasului microcontrolerului se generează cu un cristal de **16 MHz**.
3. **Nu se admit** întârzieri implementate cu bucle soft.
4. Programul trebuie să aibă o singură funcţie *main*() şi o singură bucla principală *while*(). Codul care nu se integrează în această structură nu se punctează.

Implementarea care nu respecta cerinţele de mai sus **nu se punctează**.

Se cere:

a. Se cere schema simplificată a microsistemului format din microcontrolerul ATMega16, tastatura **KB5L5C** şi semnalul **Data**. Pe schemă tastatură se va reprezenta ca în figura 10. **1 punct**

b. Să se scrie funcţia care face scanarea tastaturii şi întoarce codului tastei. Această funcţie se va numi *kbscan5x5*(). Pentru funcţie + secvenţa de iniţializare **3 puncte**

c. Calculele necesare pentru stabilirea modului de funcţionare a timerului, iniţializarea timerului şi utilizarea timerului. Puteţi utiliza ce timer doriţi. Alegerea şi setările timerului (numărul timerului, modul, *p*, N) se justifică. **2 puncte**

d. Detecţia apăsării unei taste şi emisia conform protocolului serial specificat anterior. **3 puncte**

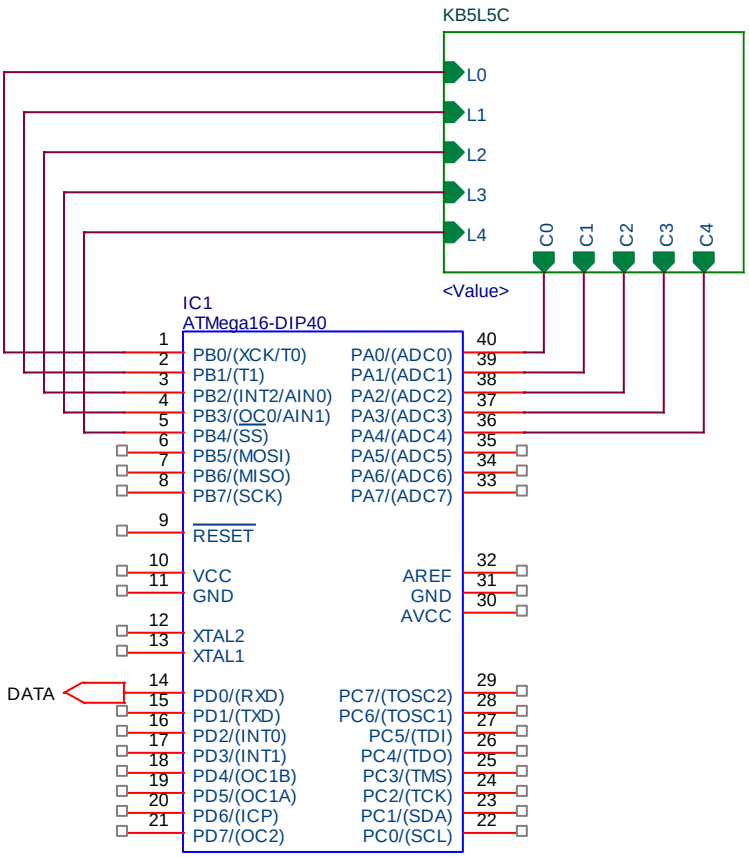
Rezolvare:

a. În figura alăturată este prezentată schema simplificată a microsistemului.

Schema este asemănătoare cu schema de cuplare a tastaturii din laboratorul 7, figura 4. Spre deosebire de laboratorul 7, acum avem de cuplat 10 semnale: 5 linii şi 5 coloane. Cum un port are 8 biţii rezultă ca este nevoie de cel puţin două porturi.

Deoarece în enunţul problemei nu există nici o restricţie privitor la utilizarea porturilor, cel mai simplu este să folosim un port pentru linii şi un port pentru coloane.

Vom folosi biţii PB4-PB0 pentru linii şi PA4-PA0 pentru coloane.



Funcţia kbscan5x5() este foarte asemănătoare cu kbscan() implementă în laboratorul 7. În continuare este prezentată o varianta de implementare, dar orice implementare care funcţionează se punctează:

b.

```
#define nop asm("nop"::);

unsigned char kbscan5x5(){
   unsigned char cols, temp, la, ca;

   for(DDRB=1; DDRB<=0b10000; DDRB<<=1){
      nop; /*nop ...numarul de nopuri se determina experimental  */ nop;
      cols =~PINA;
      cols &= 0x1f;
      if(cols){
         break;
      }
   }

   if(cols == 0)
      return 0x7f;

   la=0;
   temp=DDRB;
   while((temp&1) == 0){
      la++;
      temp>>=1;
   }

   ca=0;
   while((cols&1) == 0){
      ca++;
      cols>>=1;
   }

   return la*5+ca;
}
```

c. Durata necesara pentru transmiterea unui bit este 100 us. Această durată se va numi în continuare **celulă**. La începutul oricărei celule semnalul **Data** va primi valoarea '1'. După 20 us pentru un bit ‚0' sau după 80 us pentru un bit ‚1' **Data** va deveni '0'. Indiferent de bit trebuie aştept să treacă 100 us pentru a transmite următorul bit. Cel mai mare divizor comun a lui 20, 80 şi 100 us este 20 us. Aceasta este rezoluţia minimă a timerului. În concluzie **Tr** din prelegerea 5 este 20 us.

Pentru aplicațiile de tip ceas se aplică relaţia 4 din prelegerea 5. Transmisia unui cod se bazează pe un ceas de tip 20 us – 5 – 5. Un ceas obișnuit este de tip 1s – 60 – 60.

Conform relaţiei 4 din prelegerea 5 avem $T_r = kT_{cycle}$ iar conform relaţiei 2 avem $T_{cycle} = p * N * T_{CLK\_CPU}$. Din cele două relații rezultă:

$$T_r = k * p * N * T_{CLK\_CPU}$$

După cum s-a explicat în prelegerea 5 se doreşte cel mai mic k posibil, adică k=1.

Pentru **Tr**=20us şi T_{CLK_CPU} =1/16MHz avem:

$T_r = k * p * N * T_{CLK\_CPU}$, $T=1/f \Rightarrow T_r * f_{CLK\_CPU} = k * p * N$, **20us *16MHz =k*p*N,**

20 ~~10^-6~~ *16 ~~10^6~~ = k*p*N, 20*16 =k * p * N,

Pentru timerele 0 şi 1 prescalerul *p* poate fi 1, 8, 64, 256, 1024

20*16 = 8*40 = k * p * N => k=1, p=8, N=40

Vom folosi timerul 0 în mod CTC, cu prescaler 8 iar N va fi înscris în registrul OCR0:

```
//          ┌──┌CTC mode
TCCR0=0b00001010;
//          ||  └┘└─P=8
//          └──────Normal port operation, OC0 disconnected.
OCR0=40-1; // N=40
```

d. La începutul prelegerii 6 s-au explicat noţiunile de eveniment urgent şi timp de răspuns. Revedeți aceste noţiuni!

Respectarea cât mai exactă a timingului din figura 11 este esenţială în aplicațiile de transmisii de date. Dacă durata unui bit de ‚0' creşte mult peste 20 us sau durata unui bit de ‚1' scade mult sub 80 us, există posibilitatea ca celulele să fie interpretate greşit.

Semnalul Data va primi valoarea ‚0' sau ‚1' atunci când ciclează timerul 0. Ciclarea timerului este evenimentul urgent. Pentru ca între momentul în care timerul ciclează şi momentul în care semnalul data primeşte o nouă valoare să treacă cât mai puţin timp, ciclarea timerului se va trata prin mecanismul de întreruperi. Un exemplu de transmisie serială se găseşte în prelegerea 7, capitolul Emiţătorul PS/2.

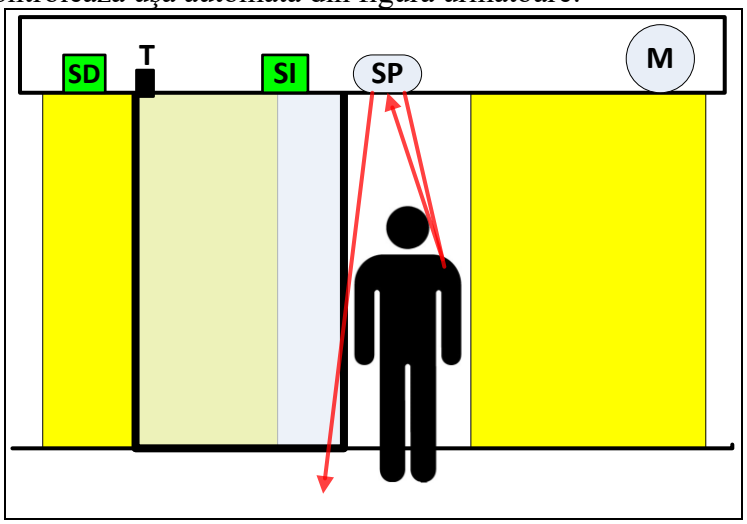
Codul pentru preluarea unei taste şi transmisia sa seriala este prezentat în continuare:

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

#define DELAY  10

volatile unsigned char t20=0;
volatile unsigned char kbhit=0, kbcode, mask;

int main(){
   unsigned char sample_ante = 1;
   unsigned char sample_now  = 1;
   unsigned char loop_cnt=0;

   PORTB = 0; //coloane
   DDRB  = 0; //nu este neaparat necesara. Valoarea la reset este oricum 00
```

```
   DDRA=0;  // linii
   DDRD =1; // semnalul data
   PORTD=0;

   TCCR0 = 0b00001010;
   OCR0 = 40-1;
   sei();

   while(1){ //bucla principala
      if(loop_cnt >= DELAY){
       loop_cnt=0;
       sample_ante = sample_now;
       sample_now  = kbscan5x5();

       if( sample_ante == 0x7f && sample_now !=0x7f){
          kbhit=1;
          kbcode=sample_now;
          //pana aici citirea unei taste este la fel ca în prelegerea 4, pagina 10

          // mask selectează bitul din cod care se va transmite.
          // primul bit care se transmite este c0, pentru c0 mask este 0b00001
          // pentru c1 mask se deplaseaza stanga si devine ob00010, ş.a.m.d.
          mask=1;

          //t20 numără intervalele de 20 us.
          //Este nevoie de 5 intervale pentru a transmite un bit.
          t20=0;

          // Timerul numără permanent, dar tastele se apasă rar, cel mult 10 pe secundă.
          // Când se detectează o noua apăsare OCF0 este sigur ‚1'.
          // Pentru ca primul interval de 20 us sa fie întreg, trebuie ca sa resetăm OCF0
          TIFR|=1<<OCF0;

          // demascăm întreruperea de la timer doar când este ceva de transmis.
          TIMSK |= 1<<OCIE0;
       }
      }
      loop_cnt++;
   }//end while(1)
}

ISR(TIMER0_COMP_vect){
   if(t20==0)
      PORTD=1;
   else if(t20==1 && ((kbcode & mask)==0))
      PORTD=0;
   else if(t20==4)
      PORTD=0;

   t20++;
   if(t20==5){

      //au trecut 5 intervale de 20 us
      t20=0;
      mask<<=1;
      if(mask==0b100000){

         // s-a transmis tot codul. Maschează IRQ de la timer pentru ca nu mai avem
         // nimic de transmis
         TIMSK &= ~(1<<OCIE0);

         // de fapt nu este nevoie de kbhit, dar a fost păstrat ca implementarea
         // să semene cu cea din curs
         kbhit=0;
      }
   }
}
```

6. Tip 7, 8 (maşini de stare, timere pwm), Problema 68.

Să se proiecteze un microsistem cu ATMega16 care controlează uşa automată din figura următoare:



Uşa este prevăzută cu:

- **SP** (senzor prezenţă). Oferă la ieşire un semnal care este ‚1' când în faţă uşii sau în dreptul uşii se află o persoană.
- **SD** (senzor deschis) şi **SI** (senzor închis) sunt doi senzori magnetici. Senzorul magnetic oferă la ieşire un semnal care este ‚1' când în dreptul senzorului se află un magnet.
- Magnetul **T** este lipit de colţul uşii, ca în figură. Când uşa este deschisă la maxim, T se află în dreptul senzorului SD. Când uşa este închisă, T se află în dreptul senzorului SI.
- Motorul **M** mişcă uşa stânga-dreaptă cu două viteze. Viteza motorului este controlată PWM de semnalul **MV** (motor viteză). Factorul de umplere (**FU**) al lui MV controlează viteza motorului astfel : dacă FU(MV) = 100% motorul mişcă uşa cu viteză mare, dacă FU(MV)= 10% motorul mişcă uşa cu viteză mica iar dacă FU(MV)=0% motorul este oprit. Motorul mai primeşte şi semnalul de comandă **MS** (motor sens). Dacă MS= ‚0' motorul mişcă uşa astfel încât aceasta să se închidă iar dacă MS= ‚1' motorul mişcă uşa astfel încât aceasta să se deschidă.

Uşa va funcţiona după cum urmează:

P1. În mod normal uşa este închisă. Când uşa este închisă semnalul de la senzorul SI este ‚1'.
P2. Dacă senzorul SP detectează o persoană, uşa începe să se deschidă cu viteză mare. Mişcarea cu viteză mare durează 4 secunde.
P3. În continuare uşa se mişcă cu viteză mică până se deschide la maxim, adică până când semnalul de la senzorul SD devine ‚1'.
P4. Uşa rămâne deschisă la maxim până când semnalul de la senzorul SP devine ‚0', adică nu mai este nimeni în dreptul uşii. În continuare uşa mai rămâne deschisă încă 5 secunde.
P5. Apoi uşa începe să se închidă cu viteză mare. Mişcarea cu viteză mare durează 4 secunde.
P6. În final uşa se mişcă cu viteză mică până se închide, adică până când semnalul de la senzorul SI este ‚1'.
P7. Dacă în procesul de închidere a uşii semnalul de la SP devine ‚1', sari la P3.
P8. La punerea sub tensiune uşa se va închide cu viteză mică până când SI='1'.

Specificaţii şi cerinţe obligatorii:

1. Ceasului microcontrolerului se generează cu un cristal de **2,048 MHz** (sau 3MHZ).
2. **Nu se admite** gestiunea timpului cu bucle soft de tip for(i=0;i<DELAY;i++){}
3. Se va utiliza un singur timer/counter.
4. $T_{cycle} \in [8\text{ ms}, \ 12\text{ ms}]$
5. Când uşa este închisă sau deschisă la maxim este obligatoriu ca FU(MV)=0%.
6. Programul trebuie să aibă o singură funcţie *main*(), o singură bucla principală *while*() şi ISR-uri (dacă este cazul). Codul care nu respectă această structură **nu se punctează**.

Se cere:

1. Să se precizeze pe ce pin al microcontrolerului se conectează fiecare semnal. Dacă această cerinţă nu este îndeplinită, nu se acordă punctaj la c).
2. Stabilirea modului de funcţionare a timerului şi calculul lui Tcycle. Puteţi utiliza ce timer doriţi. Alegerea şi setările timerului (numărul timerului, modul, *p*, eventual valoarea OCR, eventual ieşirea OC) se justifică. **Implementarea fără justificare sau care nu respectă condiția asupra lui Tcycle nu se punctează. 2 puncte**
3. Programul C pentru ATMega16:
   a. Setarea porturilor şi a timerului. **1 punct**
   b. Maşina de stare. **4 puncte**
   c. Gestiunea timpului. **2 puncte**

Rezolvare:

1. SP → PA0, SI → PA1, SD → PA2, MV→PB3/OC0, MS → PB0

2. Rezolvăm varianta cu f_{CLK_CPU} = 2,048MHz.

Pentru a controla viteaza motorului prin tehnica PWM, vom alege pentru timer **modul fast PWM**.

Pentru modul PWM plecăm de la relaţia 3 din prelegerea 5:

$$f_{CLK\_CPU} = p*N*fcycle = 256*p*\ fcycle \quad \rightarrow \quad p = \frac{f_{CLK\_CPU}}{256*\ f_{cycle}}$$

Deoarece se impune $T_{cycle} \in [8\ ms,\ \ 12\ ms]$, încercăm un T_{cycle} la mijlocul intervalului, adică 10ms. f=1/T => f_{cycle}=1/10ms=100Hz.

Înlocuim în relaţia anterioară pe f_{cycle} şi pe f_{CLK_CPU}:

$$p = \frac{2{,}048MHZ}{256*100} = \frac{2048\ \ 10^3}{256*100} = \frac{2^{11}\ \ 1000}{2^8*100} = 2^3\ 10 = 80$$

Cel mai apropiat p de 80 este 64. Pentru $\boldsymbol{p}$**=64** obţinem:

$$T_{cycle} = \frac{pN}{f_{CLK\_CPU}} = \frac{64\ \ 256}{2048\ 1000} = \frac{2^6\ \ 2^8}{2^{11}} 10^{-3} = 2^3\ 10^{-3} = 8ms,\ 8ms \in [8ms,\ \ 12ms]$$

Vom folosi timerul 0 în mod fast PWM, cu *p*=64.

Trebuie obţinuţi 3 factori de umplere: 100% pentru viteză mare, 0% pentru viteză 0 şi 10% pentru viteză mică. FU=100% se obţine cu OCR0=255, viteza 0 cu OCR0=0 iar valoarea care se va înscrie în OCR0 pentru FU=10% se calculează cu regula de trei simplă:

100 …… 255
10 …… x

x=255 10 / 100 ≈ 26

Cerinţă 3 impune utilizarea unui singur timer. Timerul 0 în mod fast PWM se va utiliza şi pentru implementarea unui ceas. Deoarece pasul P2 durează 4s, pasul P4 durează 5s iar pasul P5, 4s, rezoluţia acestui ceas este 1s. Şi 4s şi 5s sunt multiplii secundei.

Deoarece rezoluţia este 1s iar Tcycle este 8ms, vom aplica relaţia 4 - $T_r = kT_{cycle}$ - din prelegerea 5:

1s=k * 8ms → k= 1s/8ms= 1000ms/8ms=125

3.

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

#define clrbit(var,bit) var &= ~(1<<bit)
#define setbit(var,bit) var |= 1<<bit

//senzor prezenţă SRP, senzor închis SRI, senzor deschis SRD
#define SRP    0
#define SRI    1
#define SRD    2

//Motor sens
#define MS    0

//stările FSM. Semnificaţia stărilor va fi explicată pe măsură ce acestea vor fi folosite
#define START  0
#define TOPEN  1
#define OPENF  2
```

```
#define OPENS  3
#define OPEN   4
#define CLOSEF 5
#define CLOSES 6

void init_time();

unsigned char sec, cycles;

int main(){
   unsigned char status = START;
   // cerinţă 3a
   DDRA=0;   // portul A este configurat ca port de intrare pentru a citi informaţiile
             // de la senzori

   DDRB=0xFF; // portul B este configurat ca port de ieşire pentru a controla motorul

   //configurarea timerului se face după modelul din prelegere 5, pagina 22
   //        ┌──┬──Fast PWM
   TCCR0=0b01101011;
   //        ||  └┴┴─ clk I/O /64
   //        └┴──────Set OC0 on BOTTOM, Clear on compare.

   //la iniţializare uşa trebuie să se închidă cu viteză mică: FU=10%, MS=0
   OCR0=26;            //FU=10%

   clrbit(PORTB, MS);  //MS=0 (direcţia inchidere)

   while(1){
      //Cerinta 3c
      // ceas cu rezolutie de o secundă asemănător cu exemplul din prelegere 5,
      // pagina 17 şi cu exemplul din laboratorul 9
      if(TIFR & 1<< TOV0){
         TIFR |= 1<< TOV0;
         cycles++;
         if(cycles == 125){
            cycles=0;
            sec++;
         }
      }

      //Cerinţă 3b - FSM. Vezi laboratorul 8 şi exemplele din prelegerea 7
      switch (status){
       // motorul se roteşte cu viteză mică şi sens închidere până când semnalul
       // de la senzorul SI devine ‚1'
       case START:
          if(PINA & 1<<SRI){
             OCR0=0;
             status = TOPEN;
          }
          break;

       //TOPEN înseamnă TEST OPEN. În această stare se testează semnalul de la
       // senzorul de prezenţă SP. Când se detectează ‚1', se setează FU=100% si
       // sens închidere.
       case TOPEN:
          if(PINA & 1<<SRP){
             // Se iniţializează la zero componentele ceasulu. Componentele ceasului
             // sunt secundele sec, numărul de cicluri cycles, valoarea timerului şi
             // indicatorul de ciclare TOV0. Toate componentele se înscriu cu zero de
             // către funcţia init_time(), mai puţin valoarea timerului. Valoarea
             // timerului nu se modifică deoarece modificarea ar afecta ieşirea PWM,
             // ceea ce ar putea afecta viteza de rotire a motorului. Din această cauză
             // acurateţea cu care se măsoară timpul este de un Tcycle, adică 8 ms, ceea
             // ce este admisibil în cazul unei uşi automate.
             init_time();
             setbit(PORTB, MS); // sens rotire pentru închidere
```

```
            OCR0=255;           // viteza mare
            status = OPENF;
        }
        break;

    // OPENF vine de la OPEN FAST - deschide repede. Viteza mare şi sensul stabilite
    // în starea TOPEN se menţin 4 secunde.
    case OPENF:
        if(sec==4){
            OCR0=26;   //dupa 4 secunde setează viteza mică
            status = OPENS;
        }
        break;

    // OPENS vine de la OPEN SLOW - deschide încet. Viteza mică şi sensul stabilite
    // anterior se menţin până când semnalul de la senzorul SD devine ‚1'.
    case OPENS:
        if(PINA & 1<<SRD){
            init_time();
            OCR0=0;    //viteza zero pentru starea următoare
            status = OPEN;
        }
        break;

    //usa este deschisă. Stai în această stare 5 secunde
    case OPEN:
        if(sec==5){
            //după 5s setează vitează mare, sens deschidere, timp=0
            init_time();
            clrbit(PORTB, MS);
            OCR0=255;
            status = CLOSEF;
        }
        break;

    // CLOSEF vine de la CLOSE FAST - închide repede. Viteza mare şi sensul stabilite
    // în starea OPEN se menţin 4 secunde.
    case CLOSEF:
        // dacă pe durata închiderii senzorul de prezenţă se activează, setează vitează
        // mică şi sens deschidere şi treci în starea OPENS
        if(PINA & 1<<SRP){
            setbit(PORTB, MS);
            OCR0=26;
            status = OPENS;
        }
        else if(sec==4){
            OCR0=26;
            status = CLOSES;
        }
        break;

    // CLOSES vine de la CLOSE SLOW - închide încet. Viteza mică şi sensul stabilite
    // anterior se menţin până când semnalul de la senzorul SI devine ‚1'.
    case CLOSES:
        // dacă pe durata închiderii senzorul de prezenţă se activează, setează vitează
        // mică şi sens deschidere şi treci în starea OPENS
        if(PINA & 1<<SRP){
            setbit(PORTB, MS);
            OCR0=26;
            status = OPENS;
        }
        else if(PINA & 1<<SRI){
            OCR0=0;
            status = TOPEN;
        }
        break;
}//end switch
```

```
    }//end while 1
    return 0;
}

void init_time(){
    TIFR |= 1<< TOV0;
    cycles=0;
    sec=0;
}
```

7. Tip 8, 9 (timer - turometru, intreruperi), Problema 61

Să se proiecteze un computer de bicicletă bazat pe microcontrolerul ATMega16. Computerul primeşte semnalul REV de la perechea **magnet** permanent– **traductor** magnetic de proximitate. Magnetul şi traductorul sunt montate ca în figura alăturată.

Atunci când magnetul este în dreptul senzorului semnalul REV are valoarea ‚1'; în caz contrar REV este ‚0'.

La fiecare rotaţie a roţii traductorul generează un puls pozitiv al semnalului REV. Lăţimea pulsului depinde de turaţia roţii: cu cât turaţia roţii este mai mare cu atât pulsul va avea o durată mai mică.

Computerul trebuie să calculeze şi să afişeze **3 mărimi**: **viteza** instantanee, **distanţa** parcursă şi **timpul** în care a fost parcursă această distanţă, adică timpul activ. Aceste 3 mărimi se calculează în următoarele condiţii:

a. Circumferinţă roţii este 2 metri.

b. Se garantează că lăţimea minimă a pulsului pozitiv al semnalului REV este 30 microsecunde.

c. Viteza maximă a bicicletei este 72 km/h. Se garantează că bicicleta nu poate depăşi această viteză.

d. Dacă viteza este mai mică de 3,6 km/h, viteza afişată va fi zero şi nici distanţă parcursă, nici timpul nu se vor modifica. De exemplu, atunci când bicicleta stă viteaza este zero, distanţa parcursă rămâne neschimbată, la fel şi timpul activ.

e. Afişarea celor trei mărimi se face pe un LCD cu organizarea 2 linii şi 16 coloane. Pentru afişare se vor folosi funcţiile *putchLCD*(*char ch*), *putsLCD*(*char *ch*), *clrLCD*() şi *gotoLC*(*unsigned char line, unsigned char col*) cu semnificaţia de la laborator. Codul pentru aceste funcţii este deja disponibil **şi nu mai trebuie scris**. Durata de execuţie pentru aceste funcţii este minim 40 µs şi maxim 1 ms.

f. Ceasul microcontrolerului este generat cu un cristal cu frecvenţa de **1,024 MHz**.

g. La reset viteza, distanţa şi timpul activ se vor iniţializa cu zero.

Cerinţe obligatorii:

1. Durata mare de execuţie a funcţiilor de afişare în raport cu durata impulsului de la senzor face imposibilă detecţia prin pooling. Detecţia impulsului REV în pooling prin metoda celor două eşantioane **nu se punctează**.
2. Pentru oricare mărime calculată se va descrie metoda de calcul, inclusiv formulele matematice necesare (dacă este cazul). Se vor preciza toate resursele folosite (porturi, timere, sistem de întreruperi) şi modul în care acestea sunt folosite. Se va implementa conform descrierii. **Implementarea fără descriere nu se punctează**.
3. Programul trebuie să aibă o singură funcţie *main*(), o singură bucla principală *while*() şi ISR-uri (dacă este cazul). Codul care nu se integrează în această structură **nu se punctează**.

Se cere:

1. Să se specifice pe ce pin se conectează semnalul REV. În lipsa acestei specificaţii nu se acorda nici in punct. Conectarea LCD-ul la microcontroler se va rezolva ulterior şi nu se punctează.
2. Calcularea şi afişarea vitezei instantanee. Viteza se afişează în kilometrii pe oră, fără zecimale, în plaja 3,6-72 km/h (de exemplu 35 km/h). În intervalul 3,6-72 km/h viteza se va calcula cu precizie de minim 1%. Implementarea care nu respectă precizia de 1% se depunctează. Descrierea metodei + formule **2 puncte,** implementare **2 puncte**
3. Calcularea şi afişarea distanţei parcurse. Distanţă se afişează în kilometrii pe două cifre la partea întreagă şi o cifră la partea zecimală (de exemplu 25,7 km). **3 puncte**
4. Calcularea şi afişarea timpului activ (timpul în care s-a parcurs distanţă calculată la punctul 2). Timpul activ se afişează în ore şi minute (de exemplu 2h35). **2 puncte**

Rezolvare:

1. REV se conectează pe pinul **(ICP1) PD6** pentru a putea folosi blocul de captură din timerul 1. Vezi turometru din prelegere 5, capitolele 5.1, 5.2. Timerul 1 se va folosi modul normal.

2. Viteza se calculează cu formula : $v = \frac{spațiu}{timp} = \frac{circumferință\ roții}{T_{rev}}$. T_{rev} este timpul în care roata execută o rotaţie completă şi se calculează prin metoda celor două amprente de timp. Timpul este măsurat cu timerul 1. Timpul necesar pentru o rotaţie completă se măsoară ca diferenţă între timpii frontului ridicător a două impulsuri succesive ale semnalului REV.

   Notăm s2 valoarea timerului 1 capturată pe frontul ridicător cel mai recent al impulsului REV şi cu s1 valoarea timerului 1 capturată pe frontul ridicător precedent al impulsului REV. Dacă înlocuim pe T_r cu $(s2-s1)T_{CLK\_CNT}$ obţinem:

   $$v = \frac{circumferință\ roții}{T_{rev}} = \frac{circumferință\ roții}{(s2-s1)T_{CLK\_CNT}} = \frac{2m}{(s2-s1)\ p\ T_{CLK\_CPU}} = \frac{2m}{(s2-s1)\ p\ T_{CLK\_CPU}} =$$

   $$= \frac{2m\ f_{CLK_{CPU}}}{(s2-s1)\ p} = \frac{2m\ 1{,}024MHz}{(s2-s1)\ p} = \frac{2m\ 1024\ 1000\ Hz}{(s2-s1)\ p} = \frac{2m\ 2^{10}\ 1000\ \frac{1}{s}}{(s2-s1)\ p} = \mathbf{\frac{2^{11}\ 1000}{(s2-s1)\ p}\ \frac{m}{s}}$$

   În continuare se va alege p. Pentru a alege pe p vom considera două situaţii: viteză minimă şi viteză maximă.

   a. **Viteza minimă**: la viteza minimă **T_{cycle} al timerul 1 trebuie să fie mai mare ca T_{rev}**. Mai întâi calculăm viteaza minimă în metri pe secundă: $v_{min} = 3{,}6km/h = \frac{3600m}{3600s} = 1m/s$. Apoi calculăm cât timp durează o rotaţie a roţii la viteaza minimă: $T_{rev\_min} = \frac{spațiu}{viteza} = \frac{circumferință\ roții}{v_{min}} = \frac{2m}{1m/s} = 2s$. Pentru ca să putem măsura prin metoda amprentei de timp trebuie ca $T_{cycle} > T_{rev\_min}$.

   Ştim (conform relaţiei 2, pagina 8, prelegere 5 sau deducem dacă nu ştim) că $T_{cycle} = pNT_{CLK\_CPU}$. N pentru timerul 1 în modul normal, este 2^{16} iar T_{CLK_CPU} se dă.

   Aflăm pentru ce p este adevărată inegalitatea $T_{cycle} > T_{rev\_min}$:

   $$T_{cycle} = pNT_{CLK_{CPU}} > T_{rev\_min},\ p > \frac{T_{rev\_min}}{NT_{CLK_{CPU}}} = \frac{T_{rev\_min}\ f_{CLK_{CPU}}}{N} = \frac{2s\ 1024\ 10^3 Hz}{2^{16}} = \frac{2\ 2^{10}\ 2^3\ 5^3}{2^{16}} = \frac{2^{14}\ 5^3}{2^{16}}$$

   $$= \frac{125}{4} = 31{,}25. \quad \textbf{Deci } \boldsymbol{p>31{,}25}$$

   b. **Viteza maximă**: la viteza maximă precizia de măsurare trebuie să fie de minim 1%.

   Durata unei rotaţii la viteza maximă este

   $$v_{max} = \frac{circumferință\ roții}{T_{rev\_max}},\quad v_{max} = \frac{72km}{h} = \frac{72000m}{3600s} = \frac{20m}{s} \rightarrow \quad T_{rev\_max} = \frac{circumferință\ roții}{v_{max}} =$$

   $$= \frac{2m}{20m/s} = 0{,}1\ s$$

   Durata unei rotaţii la o viteză cu 1% mai mică decât viteza maximă este:

   $$v_{max1} = \frac{72\left(1-\frac{1}{100}\right)km}{h} = \frac{72000\quad 99\ m}{3600s\quad 100} = 20\ \ 0{,}99\ \frac{m}{s} \rightarrow \quad T_{rev\_max1} = \frac{circumferință\ roții}{v_{max}} =$$

   $$= \frac{2m}{0{,}99\ 20m/s} = 0{,}1\ \ \frac{1}{0{,}99}\ s = 0.10101s$$

   Din diferența $T_{rev\_max1} - T_{rev\_max} = 0{,}10101 - 0{,}1 = 0{,}00101s = 1{,}01$ ms rezultă perioada minimă a ceasului de numărare:

T_{CLK_CNT} < 1,01 ms, $p\ T_{CLK\_CPU}$ < 1,01 ms , $p < \frac{1{,}01ms}{T_{CLK\_CPU}} = 1{,}01ms\ f_{CLK\_CPU} =$ $1{,}01\ \ 10^{-3}\ \ 1024\ \ 10^{3} = 1{,}01\ \ \ 1024$

La punctul a) a rezultat că p>31,25 iar mai sus a rezultat că p< 1,01 1024. Vom alege din valorile posibile (1, 8, 64, 256, 1024) o valoare la mijlocul intervalului: vom alege p=256.

Știind pe p revenim la calculul vitezei:

$$v = \frac{2^{11}\ 1000}{(s2 - s1)\ p}\ \frac{m}{s} = \frac{2^{11}\ 1000}{(s2 - s1)\ 2^{8}}\ \frac{m}{s} = \frac{2^{3}\ 1000}{(s2 - s1)}\ \frac{m}{s} = \frac{2^{3}\ 1000}{(s2 - s1)}\ \frac{\frac{km}{1000}}{\frac{h}{3600}} = \frac{2^{3}\ 3600}{(s2 - s1)}\ \frac{km}{h} = \frac{28800}{(s2 - s1)}\ \frac{km}{h}$$

3. Exprimă durata unei rotaţii la viteaza minimă în impulsuri de ceas numărător. Pornim de la T_{rev_min} de la punctul 2a:

$$N_{rev\_min} = \frac{T_{rev\_min}}{T_{CLK\_CNT}} = \frac{2s}{pT_{CLK\_CPU}} = \frac{2\ f_{CLK\_CPU}}{p} = \frac{2\ \ 1024\ \ 1000}{256} = \frac{2\ \ 2^{10}1000}{2^{8}} = 8000$$

N_{rev_min} este necesar pentru a decide când calculăm şi afişăm viteza, distanța şi timpul activ.

4. Pentru p=256 vom determina T_{cycle}:

$$T_{cycle} = p\mathrm{N}T_{\mathrm{CLK\_CPU}} = 256\ \ 2^{16}\frac{1}{1024\ \ 10^{3}} = \frac{2^{8}\ \ 2^{16}}{2^{10}2^{3}5^{3}} = \frac{2^{24}}{2^{13}5^{3}} = \frac{2^{11}}{5^{3}} = \frac{2048}{125} \cong 16{,}3\ s$$

Precizări privind implementarea:

a. Vom relua analogia cu ceasul şi vom analiza două cazuri. În primul caz o activitate începe la ora 10 şi se termină la ora 2. Respectiva activitate a durat 4 ore şi ceasul a ciclat deoarece a trecut prin 12. În al doilea caz activitatea începe tot la ora 10 şi durează 16 ore, terminându-se tot la ora 2. Şi în acest caz ceasul a ciclat deoarece a trecut prin 12, numai că acum a trecut de 2 ori! Dacă durata maximă permisă pentru activitate este 8 ore, cum ne dam seama că în primul caz nu există depăşire şi în al doilea există? În ambele cazuri avem începutul la 10, sfârşitul la 2 şi ştim că ceasul a ciclat. Evident trebuie să numărăm de câte ori a ciclat! În primul caz avem **1ciclu +2-10=12+2-10=4** iar în al doilea caz avem **2cicluri +2-10=24+2-10=16**.

   Dacă durata maximă permisă este de un ciclu (12 ore) nu mai are sens să sa calculam durata activităţii pentru a vedea dacă există depăşire atunci când numărul de ciclări este mai mare sau egal cu 2. Pentru a evita depăşirea formatului de reprezentare numărul de ciclări se va incrementa cu saturare la 2. În implementarea următoare variabila care memorează numărul de cicli de numărare se numeşte *cycles*.

b. În continuare vom analiza al treilea caz. În acest caz o activitate începe la ora 9 şi se termină la ora 11. Deoarece suntem ocupaţi, ne uitam la ceas la ora 1. Când activitatea s-a sfârşit timpul de terminare este înregistrat automat de ceas, aşa că şi dacă ora este 1 ştim că sfârşitul activităţii a fost la 11. Dar în acest caz apare ciclare şi calculul duratei activităţii conform procedurii anterioare conduce la un rezultat eronat: **1ciclu +11-9=12+11-9=14**, ceea ce înseamnă depăşire. Depăşirea apare pentru ca o ciclare din ciclul următor de măsurare a fost contorizată în ciclul curent. Contorizarea incorectă se poate determina dacă se analizează timpul curent şi timpul la care s-a terminat activitatea, 1 şi 11 în acest caz: dacă timpul curent este mai mic decât timpul la care s-a terminat activitatea, atunci avem această situaţie. Evident, detecţia funcţionează dacă nu întârziem mai mult de un ciclu de numărare. Cum un ciclu de numărare durează aproximativ 16 secunde iar execuţia buclei principale durează maxim 5 ms (1clr LCD x 1 ms + 32 scrieri LCD x 40 us + calcule diverse) întârzierea în tratarea capturii nu poate depăşi un ciclu de numărare.

c. A treia precizare privind implementarea se va face după prezentarea codului C.

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

int main(){
   unsigned int s2, s1, cnt_now, v=0, t_min, t_hrs;
   unsigned long int t=0, d=0, delta, t_sec;
   unsigned char cycles=2, cycles_cpy;
   char buf[17];

   //           ┌┌─Normal Mode
   TCCR1B=0b01000100;
   //         |   └└└─ clk I/O /256
   //         └──────ICES1=1: capture on rising edge.
   //              ┌┌─Normal Mode
   TCCR1A=0b00000000;

   while(1){
      //contorizează ciclările numărătorului
      if(TIFR & 1<<TOV1){
         cycles++;
         TIFR |= 1<<TOV1;
         if(cycles>2)
            cycles=2;
      }
//cazul 4. Aici se face captura; timerul este in starea 0000 şi simultan apare şi TOV1
      // detectează ca exista o captură
      if( (TIFR & 1<<ICF1) && TCNT1 != 0 && TCNT1 != 0xFFFF){
         cnt_now = TCNT1; // cnt_now este folosit pentru detectia situaţiei b.
         TIFR |= 1<<ICF1;
         cycles_cpy = cycles;
         s1=s2;
         s2=ICR1;

         // tratează situaţia b.
         if (cnt_now < s2){
            cycles=1;
            cycles_cpy--;
         } else{
            cycles=0;
         }

         //calculează durata rotatiei roţii prin metoda celor două amprente de timp
         delta = (unsigned long int)cycles_cpy * 65536UL + s2 - s1;

         if(delta <=8000 ){ //dacă viteza este mai mare decat viteaza minimă
            v=28800UL/delta;
            d+=2; //in metri
            t+=delta;  //in pulsuri CLK NUM
         }
         else{
            v=0;
         }

         t_sec = t/4000UL;
         t_min = t_sec/60;
         t_hrs = t_min/60;

         sprintf(buf, "%2d km/h  %2dh%dm", v, t_hrs, t_min);
         gotoLC(1,1);
         putsLCD(buf);

         sprintf(buf, "%2ld,%1ld km",d/1000, (d-d/1000)/100);
         gotoLC(2,1);
         putsLCD(buf);
      }
   } //end while 1
}
```

A treia precizare:

Vom considera al patrulea caz. Acest caz apare foarte rar si netratarea lui nu va scădea nota. În acest al patrulea caz trebuie să aibă loc simultan trei evenimente:

1. Execuţia programului să fie la începutul testului de detecție a capturii, loc marcat cu „`//cazul 4"` în implementarea de mai sus.
2. Numărătorul să fie în starea 0000, imediat după tranziţia din starea 0xffff. (Nu uitaţi, o stare a numărătorului durează *p* impulsuri de ceas, adică 256.) În acest moment apare TOV1.
3. Captura să se facă exact în starea 0000, imediat după tranziţia din starea 0xffff, simultan cu TOV1.

În acest caz testul de captură reuşeşte, dar suntem în starea 0000 şi ciclarea care a apărut după testarea lui TOV1 nu a fost contorizată. În concluzie avem cu un TOV1 mai puţin!

Pentru a evita apariţia acestei situaţii nu vom face calculele dacă numărătorul este în starea 0000 sau 0xFFFF. Această soluţie introduce o întârziere a afişării de 512 ceasuri procesor, ceea ce nu constituie o problemă.