MICROCONTROLERE – Lucrarea de laborator 2

Scopul lucrarii:

- descrierea unor metode de implementare eficienta a circuitelor logice combinationale (CLC), a circuitelor logice secventiale (CLS) si a proceselor secventiale (PS)
- se va urmari modul de selectare a bitilor de intrare relevanti (metoda mastilor)
- se vor urmari avantajele si dejavantajele metodelor

Desfasurarea lucrarii

1. Se va studia modul de implementare al CLC
2. Se va studia modul de implementare al CLS
3. Se va studia modul de implementare al PS
4. Se vor realiza urmatoarele programe (a-c) folosind metodele indicate in lucrare:

a) Implementeaza un circuit logic combinational descris de urmatoarea functie logica:

$$f(A, B, C) = \overline{(A\bar{B}C)} \cdot \overline{(A\bar{B}\bar{C})} \cdot \overline{(A\bar{B}C)} \cdot \overline{(AB\bar{C})}$$

unde A=SW3; B=SW1; C=SW5; Iesirea este LED5;

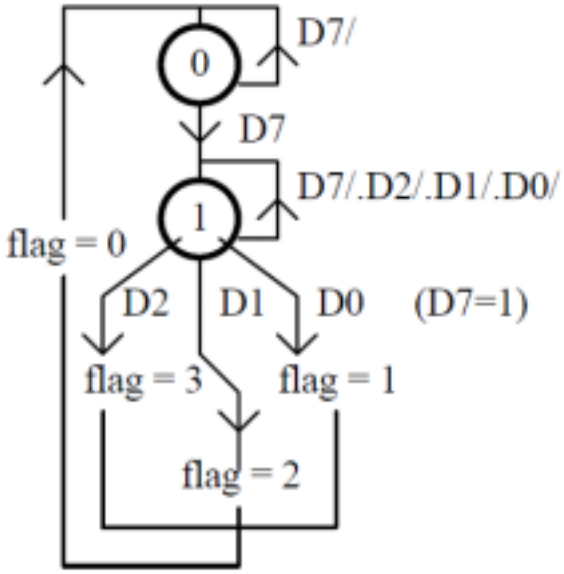
b) Implementeaza un circuit logic secvential cu urmatoarea tabela de tranzitii a starii:

	XY			
Q	00	01	10	11
0	0	1	0	1
1	1	1	0	0
	Q*			

Intrarile sint X=SW0 si Y = SW7, iesirea este identical cu starea Q. Q* reprezinta starea urmatoare.

c) Implementeaza un proces secvential cu urmatorul graf, in care se noteaza:

Di ≡ SWi (i=0…7), flag=LED3-LED2



Toate programele vor fi simulate cu Astudio si rulate in timp real pe STK 500.

Realizarea software a unui circuit logic combinational (CLC)

Un circuit logic combinational, cu *k* intrari si *n* iesiri, poate fi descrise prin urmatoarele metode (se vor nota $\boldsymbol{X} = x_{k-1}\, x_{k-2} \ldots x_1\, x_0$ – intrarea de k biti si $\boldsymbol{Y} = y_{n-1}\, y_{n-2} \ldots y_1\, y_0$ – iesirea de n biti):

1. functie booleana $\boldsymbol{f} : M^k \rightarrow M^n$ cu $M = \{0,1\}$ astfel $\boldsymbol{Y}=\boldsymbol{f(X)}$
2. cu ajutorul unui tablou de adevar, ***TAB***, asociat CLC

Se noteaza : $p = 2^k - 1$

TAB

Intrarea **X** (k biti)					Iesirea **Y** (n biti)				
$x_{k-1}^{(0)}$	$x_{k-2}^{(0)}$	…	$x_1^{(0)}$	$x_0^{(0)}$	$y_{n-1}^{(0)}$	$y_{n-2}^{(0)}$	…	$y_1^{(0)}$	$y_0^{(0)}$
$x_{k-1}^{(1)}$	$x_{k-2}^{(1)}$	…	$x_1^{(1)}$	$x_0^{(1)}$	$y_{n-1}^{(1)}$	$y_{n-2}^{(1)}$	…	$y_1^{(1)}$	$y_0^{(1)}$
…	…	…	…	…	…	…	…	…	…
$x_{k-1}^{(p-2)}$	$x_{k-2}^{(p-2)}$	…	$x_1^{(p-2)}$	$x_0^{(p-2)}$	$y_{n-1}^{(p-2)}$	$y_{n-2}^{(p-2)}$	…	$y_1^{(p-2)}$	$y_0^{(p-2)}$
$x_{k-1}^{(p-1)}$	$x_{k-2}^{(p-1)}$	…	$x_1^{(p-1)}$	$x_0^{(p-1)}$	$y_{n-1}^{(p-1)}$	$y_{n-2}^{(p-1)}$	…	$y_1^{(p-1)}$	$y_0^{(p-1)}$

Prelucrarile asociate metodei cu tablou de adevar sunt descries prin urmatoarea organigrama:

```
While (1)
{
Citeste intrarea;
Selecteaza bitii de intrare (variabila X);
Y=TAB(X);
scrie la iesire Y;
}
```

In practica, intrarea se citeste pe un numar de biti mai mare decat numarul de intrari ale circuitului CLC. De exemplu, portul de intrare are 8 biti, iar circuitul CLC are 3 biti de intrare plasati pe diferite pozitii in cuvantul de intrare.

Selectia bitilor de intrare se realizeaza prin operatii logice de tip **AND** la nivel de biti si deplasari stanga / dreapta, astfel incat variabila ***X*** astfel obtinuta sa reprezinte un *index* in tabloul ***TAB***.

Exemplu se selectie a bitilor de intrare:

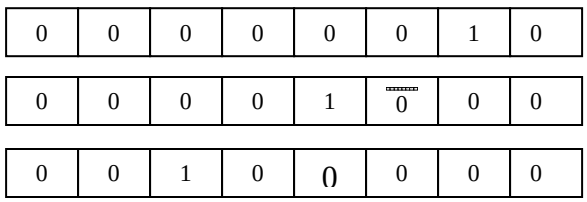
Circuitul CLC are 3 intrari X0,X1 si X2 in pozitiile bitilor 1, 3 si 5 din cuvantul de intrare de 8 biti astfel:

D7	D6	D5	D4	D3	D2	D1	D0
		X2		X1		X0	

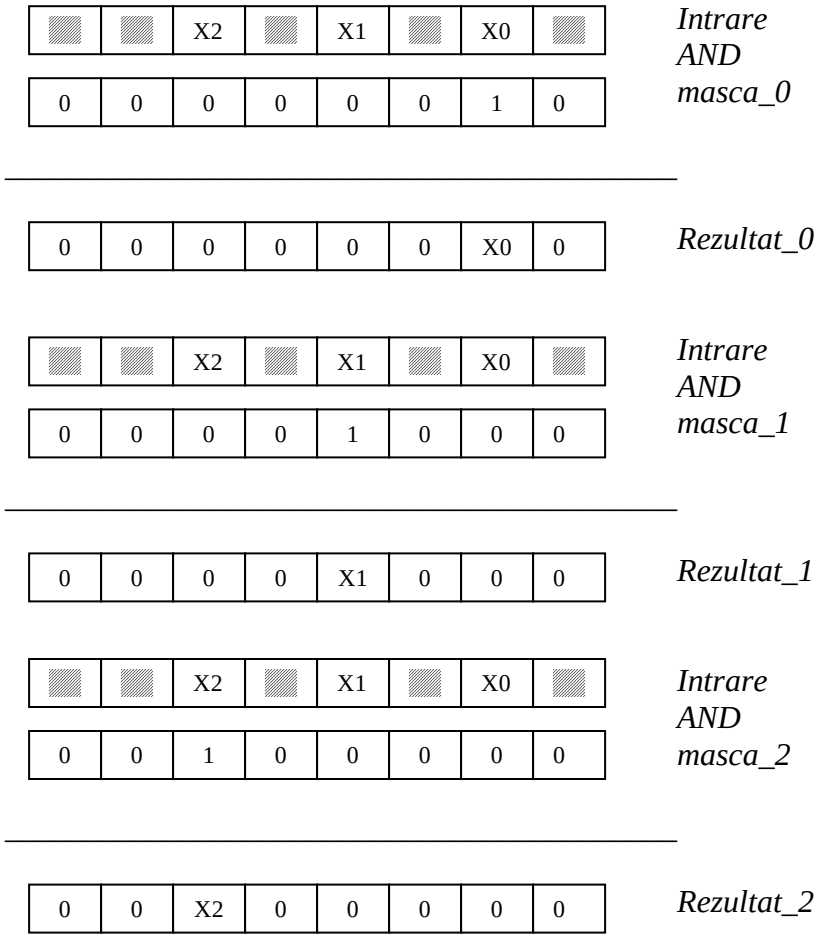
Cel mai semnificativ bit al intrarii circuitului CLC este X2, iar cel mai putin semnificativ este X0.

Pentru determinarea indexului in tabelul TAB, se parcurg urmatoarele etape:

1. Se creaza un cuvant denumit *"masca"* care are **1** pe pozitiile bitilor relevanti in cuvantul de intrare si **0** in rest:



2. Se efectueaza o operatie ***AND*** logic pe biti intre cuvantul de intrare si si fiecare cuvant *masca*:



Bitii marcati cu  in cuvantul de intrare nu sunt relevanti (valoarea acestora nu conteaza)

3. Se vor deplasa bitii cuvantului *Rezultat* astfel incat bitii X2, X1 si X0 sa fie asezati in pozitiile ponderilor binare coerspunzatoare:

A = Rezultat_0 >> 1

0	0	0	0	0	0	0	X0

B = Rezultat_1 >> 2

0	0	0	0	0	0	X1	0

C = Rezultat_2 >> 3

0	0	0	0	0	X2	0	0

Index = A OR B OR C

0	0	0	0	0	X2	X1	X0

Generarea iesirii se efectueaza in mod similar (prin operatii de deplasare stanga / dreapta).

Exemplu de implementare

Se cere implementarea unui CLC cu urmatorul tabele de adevar (X si Y intrari, Z iesirea) :

TAB

X	Y	Z
0	0	1
0	1	0
1	0	1
1	1	1

X = pinul 2 al PORTD
Y = pinul 5 al PORTD

Z = pinul 3 al PORTB

Programul complet este urmatorul:

```
#define NR 4
// masca pentru X
#define MASKX 0x04
// masca pentru Y
#define MASKY 0x20
```

```
int X,Y,Z;  // intrarile si iesirea
int index; // index in tabela TAB

int TAB[NR]= {1,0,1,1} ; // tabela de adevar

void main(void)
{

// portul PORTB configurat ca iesire
PORTB=0xFF;
DDRB=0xFF;

// portul PORTD configurat ca intrare
PORTD=0xFF;
DDRD=0x00;

// configurarea timer-ului
TCCR0=0x05;
TCNT0=0x4E;

// validarea intreruperilor
#asm("sei")

// bucla de asteptare a intreruperilor
while (1)
      {
      };
}

// Rutina de servire a intreruperilor Timer 0
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
int tmp,tmpx,tmpy,tmpz;
// Reinitializare Timer 0
TCNT0=0x4E;

tmp=PIND; // citeste intrarea
// selecteaza X;
tmpx=tmp&MASKX;
X=tmpx>>2;
// selecteaza Y;
tmpy=tmp&MASKY;
Y=tmpy>>4;
// calculeaza indexul in tabela de adevar TAB
index = X | Y;
```

```
// calculeaza iesirea Z
Z=TAB[index];
// plaseaza bitul de iesire in pozitia 3
tmpz=Z<<3;
// scrie iesirea
PORTB=tmpz;
}
```

Avantajele metodei de implemetare cu tabela de adevar

1. timpul de executie este minim posibil (se executa o citire in tabela de adevar)
2. timpul de executie este constant – nu depinde de combinatia de intare
3. codul este universal; pentru implementarea unui CLC se modifica doar tabela de adevar

Dezvantajele metodei de implemetare cu tabela de adevar

Daca numarul de intrari ale CLC este mare, atunci tabela de adevar are o dimensiune mare, ceea ce creste consumul de memorie necesara stocarii acestei tabele.

Spre deosebire de metoda tabelei de adevar, metoda cu functii booleene necesita un consum de memorie scazut, dar timpul de executie nu este minim, nu este constant (depinde de combinatia aparuta la intrare) si codul nu este universal (implementeaza functia impusa).

Varianta a proiectului AVR anterior

Proiectul implementeaza un circuit logic combinational cu urmatoarea tabela de adevar, folosind un tabel cu doua dimensiuni:

X – intrare (pinul 2 al PORTD)	**Y – intrare (pinul 5 al PORTD)**	**Z – iesire (pinul 3 al PORTB)**
0	0	1
0	1	0
1	0	1
1	1	1

```
/*****************************************
This program was produced by the
CodeWizardAVR V1.23.7a Standard
Automatic Program Generator
© Copyright 1998-2002 HP InfoTech s.r.l.
http://www.hpinfotech.ro
e-mail:office@hpinfotech.ro

Project :
Version :
Date    : 26/09/2005
Author  : Sorin
Company :
Comments:


Chip type           : AT90S8515
Clock frequency     : 4.000000 MHz
Memory model        : Small
Internal SRAM size  : 512
External SRAM size  : 0
Data Stack size     : 128
*****************************************/

#include <90s8515.h>
#define NR 2
#define MASKX 0x04
#define MASKY 0x20

// Declare your global variables here
int X,Y,Z;  // intrarile si iesirea

int TAB[NR][NR]= {1,0,1,1} ; // tabela de adevar

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTA=0x00;
DDRA=0x00;
```

```
// Port B initialization
// Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out Func6=Out Func7=Out
// State0=1 State1=1 State2=1 State3=1 State4=1 State5=1 State6=1 State7=1
PORTB=0xFF;
DDRB=0xFF;

// Port C initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=P State1=P State2=P State3=P State4=P State5=P State6=P State7=P
PORTD=0xFF;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 3.906 kHz
TCCR0=0x05;
TCNT0=0x4E;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
```

```
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x02;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;

// Global enable interrupts
#asm("sei")
 // initializare tabela de adevar

while (1)
      {
      // Place your code here

      };
}

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
int tmp,tmpx,tmpy,tmpz;
// Reinitialize Timer 0 value
TCNT0=0x4E;
// Place your code here
tmp=PIND; // citeste intrarea
// selecteaza X;
tmpx=tmp&MASKX;
X=tmpx>>2;
// selecteaza Y;
tmpy=tmp&MASKY;
Y=tmpy>>5;
// calculeaza iesirea Z
Z=TAB[X][Y];
// scrie iesirea
tmpz=Z<<3;
PORTB=~tmpz;
}
```

Realizarea software a unui circuit logic secvential (CLS)

Un circuit logic secvential este reprezentat prin urmatoarii parametrii (***X,Q,Q***,*f,g*) astfel:

X – intrarea, ***Q*** – starea , ***Q*** – iesirea; f – functia de tranzitie a starii ***Q***=*f*(***X,Q***), *g* – functia de iesire ***Q***=*g(**Q**)*

Functionarea circuitului secvential poate fi descrisa prin doua tabele:

- o tabela de tranzitie a starilor, ***TABQ***
- o tabela de decriere a iesirilor, ***OUT***

TABQ

Intrarea **X** (k biti)					Starea **Q** (n biti)				
					$Q_0^{(0)}$	$Q_0^{(1)}$	…	$Q_0^{(r-2)}$	$Q_0^{(r-1)}$
					$Q_1^{(0)}$	$Q_1^{(1)}$	…	$Q_1^{(r-2)}$	$Q_1^{(r-1)}$
					…	…	…	…	…
					$Q_{n-2}^{(0)}$	$Q_{n-2}^{(1)}$	…	$Q_{n-2}^{(r-2)}$	$Q_{n-2}^{(r-1)}$
					$Q_{n-1}^{(0)}$	$Q_{n-1}^{(1)}$	…	$Q_{n-1}^{(r-2)}$	$Q_{n-1}^{(r-1)}$
$x_{k-1}^{(0)}$	$x_{k-2}^{(0)}$	…	$x_1^{(0)}$	$x_0^{(0)}$			…		
$x_{k-1}^{(1)}$	$x_{k-2}^{(1)}$	…	$x_1^{(1)}$	$x_0^{(1)}$			…	$Q_{(r-2),(1)}$	
…	…	…	…	…			…		
$x_{k-1}^{(p-2)}$	$x_{k-2}^{(p-2)}$	…	$x_1^{(p-2)}$	$x_0^{(p-2)}$			…	$Q_{(r-2),(p-2)}$	
$x_{k-1}^{(p-1)}$	$x_{k-2}^{(p-1)}$	…	$x_1^{(p-1)}$	$x_0^{(p-1)}$			…		

Se noteaza : $p = 2^k - 1$si $r = 2^n - 1$. Cu $\boldsymbol{Q_{(i),(j)}}$ s-a notat starea CLS daca starea anterioara este $\boldsymbol{Q_{(i)}}$ si intrarea este $\boldsymbol{X_{(j)}}$.

OUT

Starea **Q** (n biti)					Iesirea **Y** (m biti)				
$x_{k-1}^{(0)}$	$x_{k-2}^{(0)}$	…	$x_1^{(0)}$	$x_0^{(0)}$	$y_{m-1}^{(0)}$	$y_{m-2}^{(0)}$	…	$y_1^{(0)}$	$y_0^{(0)}$
$x_{k-1}^{(1)}$	$x_{k-2}^{(1)}$	…	$x_1^{(1)}$	$x_0^{(1)}$	$y_{m-1}^{(1)}$	$y_{m-2}^{(1)}$	…	$y_1^{(1)}$	$y_0^{(1)}$
…	…	…	…	…	…	…	…	…	…
$x_{k-1}^{(p-2)}$	$x_{k-2}^{(p-2)}$	…	$x_1^{(p-2)}$	$x_0^{(p-2)}$	$y_{m-1}^{(p-2)}$	$y_{m-2}^{(p-2)}$	…	$y_1^{(p-2)}$	$y_0^{(p-2)}$
$x_{k-1}^{(p-1)}$	$x_{k-2}^{(p-1)}$	…	$x_1^{(p-1)}$	$x_0^{(p-1)}$	$y_{m-1}^{(p-1)}$	$y_{m-2}^{(p-1)}$	…	$y_1^{(p-1)}$	$y_0^{(p-1)}$

Metoda este asemnatoare metodei de implementare cu tabela de adevar a circuitelor combinationale. Vom avea aceleasi avantaje. Totusi, daca numarul de intrsri si de stari este mare atunci dimensiunea tabelelor **TABQ** si **TAB** este foarte mare si metoda devine ineficienta din punctual de vedere al consumului de memorie.

In continuare se va prezenta o metoda mai eficienta din acest punct de vedere, numita metoda tabelelor de semnale relevante.

Implementarea unui circuit logic secvential cu metoda tabelelor de semnale relevante

Se va inlocui tabela de tranzitie a starilor TABQ cu mai multe tabele, dupa cum urmeaza:

- **TAB** – tabela de adrese a tabelelor de semnale relevante asociate fiecarei stari
- A_i – tabela asociata starii i

Se noteaza: T – terminator de tablou (semnal de intrare care nu apare niciodata)

TAB

Stare	**Adresa tabelei de semnale relevante**
$Q_{(0)}$	A_0
$Q_{(1)}$	A_1
…	…
$Q_{(r-2)}$	A_{r-2}
$Q_{(r-1)}$	A_{r-1}

Ai

X_{i0}
Q_{i0}
X_{i0}
Q_{i0}
…
T
i

In tabela de semnale relevante asociata starii i, se trec perechi (X_{iq}, Q_{iq})= (semnalul relevant *q* in starea *i*, starea urmatoare starii *i* daca la intarea a aparut semnalul relevant *q*)**.**

Atunci cind nu mai exista semnale relevante pentru o stare se va trece in tabela de semnale relevante perechea (**T, i**) = (terminator de tablou, starea asociata tabelei de semnale relevante)

Metoda tabelelor de semnale relevante are avantajul ca reduce consumul de memorie datorita faptului ca se vor memora doar semnalele de intrare care produc tranzitii (sint relevante pentru o stare data).

Tabela iesirilor, **OUT**, ramaine nemodificata.

Metoda implica prelucrari mai complexe care presupun o cautare a semnalului relevant in tabele. Organigrama prelucrarilor este prezentata in continuare.

Variabila ***Q*** semnifica starea CLS, variabila ***Adr*** indica adresa de inceput a tabelei de semnale relevante associate starii ***Q***, ***i*** reprezinta indexul current in tabela de semnale relevante, ***ready*** indica terminarea procesului de cautare la gasirea unui semnal relevant.

```
Q = 0;
While (1)
{
Citeste intrarea;
Selecteaza bitii de intrare (variabila X);
Adr=TAB(Q);

i=0
ready=0
while (!ready)
        {
        if  (X==Adr(i)) {Q=Adr(i+1); ready=1;}
        else
                if (X==T) ready=1;
                        else
                                i=i+2;
        }
scrie la iesire OUT(Q);
}
```

Avantajele metodei de implemetare cu tabele de semnale relevante

1. Se reduce consumul de memorie.
2. Codul este universal; pentru implementarea unui CLS se modifica doar tabelele de semnale relevante

Dezvantaje ale metodei de implemetare cu tabela de adevar

4. timpul de executie este mai mare in raport cu metoda tabelei de tranzitii a starilor
5. timpul de executie **nu** este constant – depinde de pozitia semnalului relevant in tabela de semnale relevante

Exemplu de proiect CAVR

Se implemneteaza un CLS cu urmatoarele tabele de semnale relevante:

TAB

Stare	Adresa tabelei de semnale relevante
0	A_0
1	A_1

A_0

0
1
1
0
2
0

A_1

0
1
1
0
2
1

Intrarea X este pe bitul 0 al PORTD, iar iesirea Y≡Q pe bitul 0 al PORTB.

Codul este prezentat in continuare:

```
/*****************************************
This program was produced by the
CodeWizardAVR V1.23.7a Standard
Automatic Program Generator
© Copyright 1998-2002 HP InfoTech s.r.l.
http://www.hpinfotech.ro
e-mail:office@hpinfotech.ro

Project :
Version :
Date    : 26/09/2005
Author  : Sorin
Company :
Comments:


Chip type           : AT90S8515
Clock frequency     : 4.000000 MHz
Memory model        : Small
Internal SRAM size  : 512
External SRAM size  : 0
Data Stack size     : 128
*****************************************/

#include <90s8515.h>
#define NR 2
```

```
#define MASK 0x01

// Declare your global variables here
int S; // starea CLS
int X; //intrarea CLS
int Y; //iesirea CLS

int *TAB[NR]; //tabela de adrese , NR - numar de stari
int A0[]={0,1,1,0,2,0};
int A1[]={0,1,1,0,2,1}; // tabelele pentru starile 0 si 1
int out[NR]={0,1};// tabela de iesire

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T
State7=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out
Func6=Out Func7=Out
// State0=1 State1=1 State2=1 State3=1 State4=1 State5=1 State6=1
State7=1
PORTB=0xFF;
DDRB=0xFF;

// Port C initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T
State7=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
Func7=In
// State0=P State1=P State2=P State3=P State4=P State5=P State6=P
State7=P
```

```
PORTD=0xFF;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 3.906 kHz
TCCR0=0x05;
TCNT0=0x4E;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x02;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;

// Global enable interrupts
TAB[0]=A0;
TAB[1]=A1;
S=0;
#asm("sei")
```

```
while (1)
      {
      // Place your code here

      };
}

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
int tmp,i;
int *adr;
int ready;

// Reinitialize Timer 0 value
TCNT0=0x4E;
// Place your code here
tmp=PIND; // citeste intrarea
X=tmp&MASK;
adr=TAB[S];

i=0;
ready=0;

while (!ready)
{
if (X==*(adr+i)) {S=*(adr+i+1); ready=1;}
else if (*(adr+i)==2) ready=1;
          else i=i+2;
}
// scrie iesirea
Y=out[S];
PORTB=~Y;
}
```

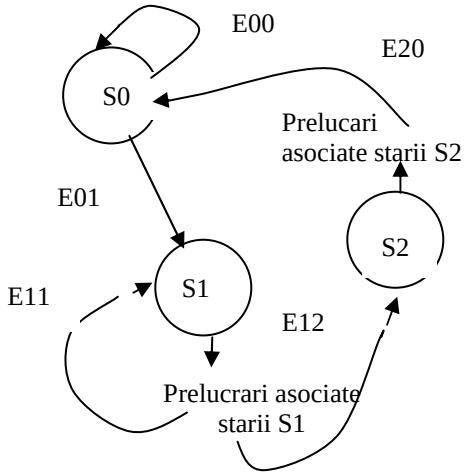
Implementarea unui proces secvential descris prin grafuri hibride de tranzitii

Un *proces* reprezinta o secventa de instructiuni (program) ce opereaza pe un set de date. Procesul are mai multe etape care sint parcurse in mod secvential, motiv pentru care acesta se mai numeste *proces secvential*. Etapele in functionarea procesului sint reprezentate ca *stari* ale procesului. Procesul secvential va fi reprezentat printr-un *graf de tranzitii*.

Tranzitia intre stari se efectuaza la aparitia unor evenimente (conditii logice, semnale de intrare) sau spoantan.

O data cu efectuarea tranzitiei se pot efectua anumite prelucrari associate cu evenimenrul care determina tranzitia. In aceasta situatie procesul secvential este descris printr-un *graf hibrid de tranzitii (GHT)*.

Structura generala a unui graf hibrid de tranzitii este urmatoarea:



Starile au fost notate cu S0, S1 si S2, iar evenimentele cu E00, E01, E11, E12, E20.

Implementarea procesului secvential se realizeaza astfel:

1. Se genereaza intreruperi periodice cu perioada T
2. Testarea evenimentelor care pot produce tranzitii in GHT si executia prelucrarilor din fiecare stare se vor efectua in rutina de servire a intreruperii periodice.

Organigrama generala a prelucrarilor este prezentata in continuare. S-a notat cu **Q** – variabila de stare asociata procesului secvential.

Q = S0; // S0 – starea initiala
Initializari alte variabile utilizate in prelucrarile associate fiecarei stari

Valideaza intreruperi periodice cu perioada T
Asteapta intreuperi

Rutina de servire a intreruperilor periodice

Selecteaza dupa valoarea variabilei de stare, Q :

S0: citeste intrarea X
daca X=E00 atunci Q=S0
daca X=E01 atunci Q=S1

S1: citeste intrarea X
Prelucrari asociate starii S1
daca X=E11 atunci Q=S1
daca X=E12 atunci Q=S2

S2: citeste intrarea X
Prelucrari asociate starii S2
daca X=E20 atunci Q=S0

Avantajele descrierii si implementarii prin GHT sint:

- procesele sint implementate printr-o structura de prelucrare unica (pentru diferite procese se vor modifica numai numarul de stari asociate si prelucrarile aferente lor)
- modificarile se realizeaza simplu, doar pentru starea dorita
- se pot introduce stari suplimentare sau elimina stari existente foarte usor

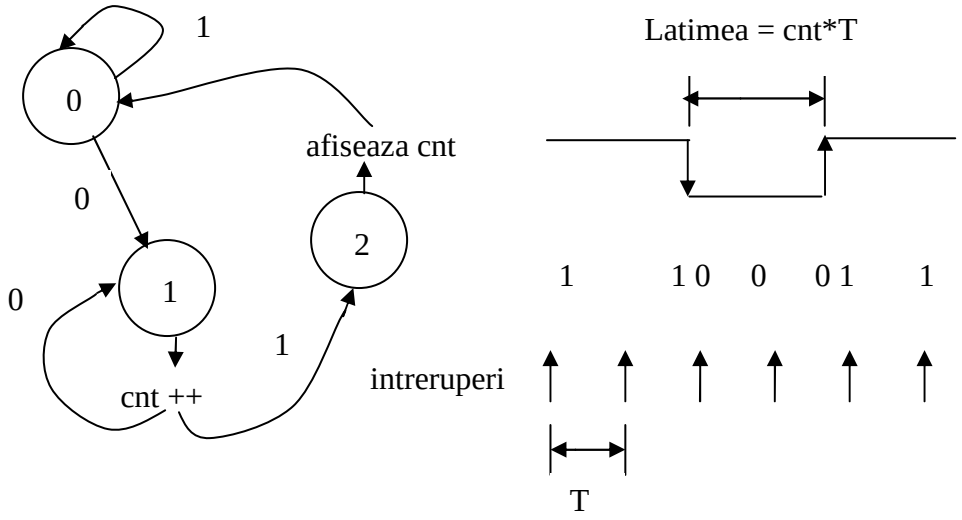
Dezavantajele implementarii prin GHT:

- nu toate procesele secventiale pot fi descrise prin GHT
- implica existenta unui mecanism de intreruperi

Exemplu: Masurarea latimii unui puls negativ

Se considera un process secvential care primeste la intrare (bitul 0) un semnal binar. Procesul asteapta aparitia frontului negativ (1->0) al semnalului de intrare; dupa aparitia acestui front se asteapta frontul pozitiv (0->1) si se numara perioadele intreruperilor periodice pina la aparitia acestuia. Dupa detectarea frontului pozitiv se afiseaza numarul de perioade de intrerupere determinat anterior (acest numar reprezinta o masura a duratei pulsului negative ce apare pe intrare).

Procesul secvential poate fi reprezentat prin urmatorul graf hibrid de tranzitii (GHT):



Organigrama prelucrarilor

Q = 0; // starea procesului secvential
Cnt = 0; // contor de intreruperi

Valideaza intreruperi periodice cu perioada T
Asteapta intreuperi

Rutina de servire a intreruperilor periodice

Selecteaza dupa valoarea variabile de stare, Q:

0: citeste intrarea X
daca X=0 atunci Q=1

1: cnt ++
citeste intrarea X

```
        daca X=1 atunci Q=2

2:      afiseaza cnt
        Q=0
        cnt=0
```

Codul programului este urmatorul:

```
/*****************************************
This program was produced by the
CodeWizardAVR V1.23.7a Standard
Automatic Program Generator
© Copyright 1998-2002 HP InfoTech s.r.l.
http://www.hpinfotech.ro
e-mail:office@hpinfotech.ro

Project :
Version :
Date    : 26/09/2005
Author  : Sorin
Company :
Comments:


Chip type           : AT90S8515
Clock frequency     : 4.000000 MHz
Memory model        : Small
Internal SRAM size  : 512
External SRAM size  : 0
Data Stack size     : 128
*****************************************/

#include <90s8515.h>


// Declare your global variables here
#define T  50

int Q; // starea procesului
int X; //intrarea procesului - bitul 0
int Y; //iesirea procesului = 1 daca latimea pulsului este > 1 sec

int cnt;
int cnt1;
int cnt2;
```

```
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out Func6=Out Func7=Out
// State0=1 State1=1 State2=1 State3=1 State4=1 State5=1 State6=1 State7=1
PORTB=0xFF;
DDRB=0xFF;

// Port C initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=P State1=P State2=P State3=P State4=P State5=P State6=P State7=P
PORTD=0xFF;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 3.906 kHz
TCCR0=0x05;
TCNT0=0x4E;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
```

```
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x02;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;


Q=0;
cnt=0;
cnt1=0;
cnt2=0;

// Global enable interrupts
#asm("sei")


while (1)
      {
      // Place your code here

      };
}

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
```

```
int tmp,tmp1,tmp2;

// Reinitialize Timer 0 value
TCNT0=0x4E;
cnt1=(cnt1+1)%T;
if (cnt1<T/2) cnt2=0; else cnt2=1;
tmp1=cnt2<<7;
tmp2=Q<<4;
PORTB=~(Y+tmp1+tmp2);

// Place your code here
switch (Q)
{
case 0:
        tmp=PIND;
        X=tmp&0x01;
        if (X==0) Q=1;
break;
case 1:
        cnt++;
        tmp=PIND;
        X=tmp&0x01;
        if (X==1) Q=2;
break;
case 2:
        if (cnt>=T) Y=1; else Y=0;
        Q=0;
        cnt=0;
break;

}
}
```