Proiectare cu Microprocesoare

Curs 5

Temporizare la sistemele Arduino

An 3 CTI
Semestrul I

Lector: Răzvan Itu

Temporizare la sistemele Arduino

- **Functii pentru intarziere**
  - **delay**(unsigned long ms) – intarziere pentru un numar specificat de milisecunde
  - **delayMicroseconds**(unsigned int us) – intarziere pentru un numar specificat de microsecunde
- **Functii pentru citirea timpului**
  - unsigned long **millis**() – returneaza timpul, in milisecunde, de la pornirea rularii programului curent. Ajunge la saturatie dupa aproximativ 50 de zile
  - unsigned long **micros**() – returneaza timpul, in microsecunde, de la pornirea rularii programului curent. Ajunge la saturatie dupa aproximativ 70 minute. La placile cu oscilator de 16 MHz (de exemplu, Arduino Mega), aceasta functie are rezolutia de 4 us.

Temporizare la sistemele Arduino

- **Exemplu: temporizare fara a folosi functia delay()**

```
const int ledPin =  13;                                  // pin-ul unde avem atasat un LED

int ledState = LOW;                                      // starea led-ului, initial stins
long previousMillis = 0;                                 // variabila in care stocam timpul ultimei actualizari

long interval = 1000;                                    // intervalul de clipire, in ms

void setup() {
    pinMode(ledPin, OUTPUT);                              // configurare pin ca iesire
}

void loop()
{
  unsigned long currentMillis = millis();                // preluarea timpului curent

  if(currentMillis - previousMillis > interval) {        // daca a trecut mai mult timp decat intervalul prestabilit
    previousMillis = currentMillis;                      // actualizam timpul

    if (ledState == LOW)                                 // comutam starea led-ului
      ledState = HIGH;
    else
      ledState = LOW;

    digitalWrite(ledPin, ledState);                       // scriem starea la iesire
  }
}
```

Sursa: http://arduino.cc/en/Tutorial/BlinkWithoutDelay

Temporizare la sistemele Arduino

- **Exemplu: temporizare fara a folosi functia delay() – pentru multitasking !**
  - **Doua led-uri, fiecare comuta la 1 secunda, cu intarziere de 0.5 sec intre ele**

```
long sequenceDelay = 500;              // intervalul dintre cele doua actiuni
long flashDelay = 1000;

boolean LED13state = false;            // starea celor doua LED-uri, la pinul 13 si la pinul 12
boolean LED12state = false;            // initial ambele sunt stinse
long waitUntil13 = 0;                  // primul LED se aprinde imediat
long waitUntil12 = sequenceDelay;      // al doilea dupa 0.5 sec
void setup() {
   pinMode(13, OUTPUT);                // configurare pini ca iesire
   pinMode(12, OUTPUT);
}
void loop() {
   digitalWrite(13, LED13state);       // la fiecare iteratie, se seteaza valoarea actualizata pe pini
   digitalWrite(12, LED12state);

   if (millis() >= waitUntil13) {      // se verifica timpul pentru primul LED
      LED13state = !(LED13state);      // daca a trecut 1 secunda, se comuta starea
      waitUntil13  += flashDelay;      // noul timp tinta, peste 1 secunda
   }

  if (millis() >= waitUntil12) {       // se verifica timpul pentru al doilea LED
      LED12state = !(LED12state);      // se comuta starea daca a trecut suficient timp
      waitUntil12 += flashDelay;       // noul timp tinta, peste 1 secunda
   }
}
```

Sursa: http://www.baldengineer.com/blog/2011/01/06/millis-tutorial/

Temporizare la sistemele Arduino

- **Folosirea bibliotecii Timer pentru sincronizare/temporizare**
- **http://playground.arduino.cc/Code/Timer**
- **Metode:**
- **int every(long period, callback)**: ruleaza functia ‘callback’ la intervale de ‘period’ milisecunde. Returneaza identificatorul evenimentului programat.
- **int every(long period, callback, int repeatCount)**: ruleaza functia ‘callback’ la intervale de ‘period’ milisecunde, de ‘repeatCount’ ori.
- **int after(long duration, callback)**: ruleaza functia ‘callback’ o singura data, dupa un interval de timp de ‘duration’ milisecunde.
- **int oscillate(int pin, long period, int startingValue)**: generare de semnal. Modifica starea pinului ‘pin’ dupa fiecare ‘period’ milisecunde. Starea initiala a pinului este cea specificata de ‘startingValue’, HIGH sau LOW.
- **int oscillate(int pin, long period, int startingValue, int repeatCount)**: variaza starea pinului ‘pin’, la intervale specificate de ‘period’, de ‘repeatCount’ ori.
- **int pulse(int pin, long period, int startingValue)**: schimba starea lui ‘pin’ o singura data, dupa ‘period’ milisecunde. Valoarea initiala este cea specificata in ‘startingValue’.
- **int stop(int id)**: toate functiile de mai sus returneaza un identificator al evenimentului programat. Aceasta functie poate fi apelata pentru a opri evenimentul. Doar 10 evenimente pot fi atasate temporizatorului.
- **int update()**: aceasta functie trebuie apelata in bucla principala (loop) a programului, pentru a actualiza starea obiectului de tip Timer.

Temporizare la sistemele Arduino

- **Exemplu: generarea unui puls de durata indelungata, fara a bloca activitatea sistemului**

```
#include "Timer.h"

Timer t;                                    // declararea obiectului de tip Timer
int pin = 13;

void setup()
{
  pinMode(pin, OUTPUT);
  t.pulse(pin, 10 * 60 * 1000, HIGH);       // puls de 10 minute, cu valoare initiala HIGH
}

void loop()
{
  t.update();                               // necesar pentru functionarea timer-ului
                                            // apelul e de ordinul microsecundelor

  // restul ciclului ramane disponibil pentru alte procesari
}
```

http://www.doctormonk.com/2012/01/arduino-timer-library.html

Temporizare la sistemele Arduino

- **Exemplu: apelarea unei functii la intervale regulate de timp, si generarea unui semnal oscilator.**

```
#include "Timer.h"

Timer t;                          // declararea obiectului de tip Timer
int pin = 13;                     // pin-ul pentru oscilatie

void setup()
{
  Serial.begin(9600);             // initializarea comunicatiei seriale
  pinMode(pin, OUTPUT);           // configurare pin pentru iesire
  t.oscillate(pin, 100, LOW);     // initializarea generarii semnalului oscilatoriu (100 ms)
  t.every(1000, takeReading);     // la fiecare 1000 ms se apeleaza functia takeReading
}

void loop()
{
  t.update();                     // apelul necesar pentru functionarea timerului
}

void takeReading()                // functia care se apeleaza la fiecare 1 secunda
{
  Serial.println(analogRead(0));  // se trimite prin serial starea tensiunii analogice de pe un pin
}
```

http://www.doctormonk.com/2012/01/arduino-timer-library.html

Temporizare la sistemele Arduino

- **Exemplu: stoparea unui proces initiat**

```
#include "Timer.h"

Timer t;

int ledEvent;                                   // identificator eveniment

void setup()
{
  Serial.begin(9600);                           // initializare interfata seriala
  int tickEvent = t.every(2000, doSomething);   // se apeleaza doSomething la fiecare 2 secunde
  Serial.print("2 second tick started id=");    // scrierea prin interfata seriala
  Serial.println(tickEvent);                    // a identificatorului procesului initiat

  pinMode(13, OUTPUT);
  ledEvent = t.oscillate(13, 50, HIGH);         // pornire eveniment oscilare led la 50 ms
  Serial.print("LED event started id=");        // scriere prin interfata seriala
  Serial.println(ledEvent);                     // a identificatorului ledEvent

  int afterEvent = t.after(10000, doAfter);     // programare executie doAfter, dupa 10 secunde
  Serial.print("After event started id=");      // scriere prin interfata seriala
  Serial.println(afterEvent);                   // a identificatorului pentru aceasta programare
  }
```

http://www.doctormonk.com/2012/01/arduino-timer-library.html

Temporizare la sistemele Arduino

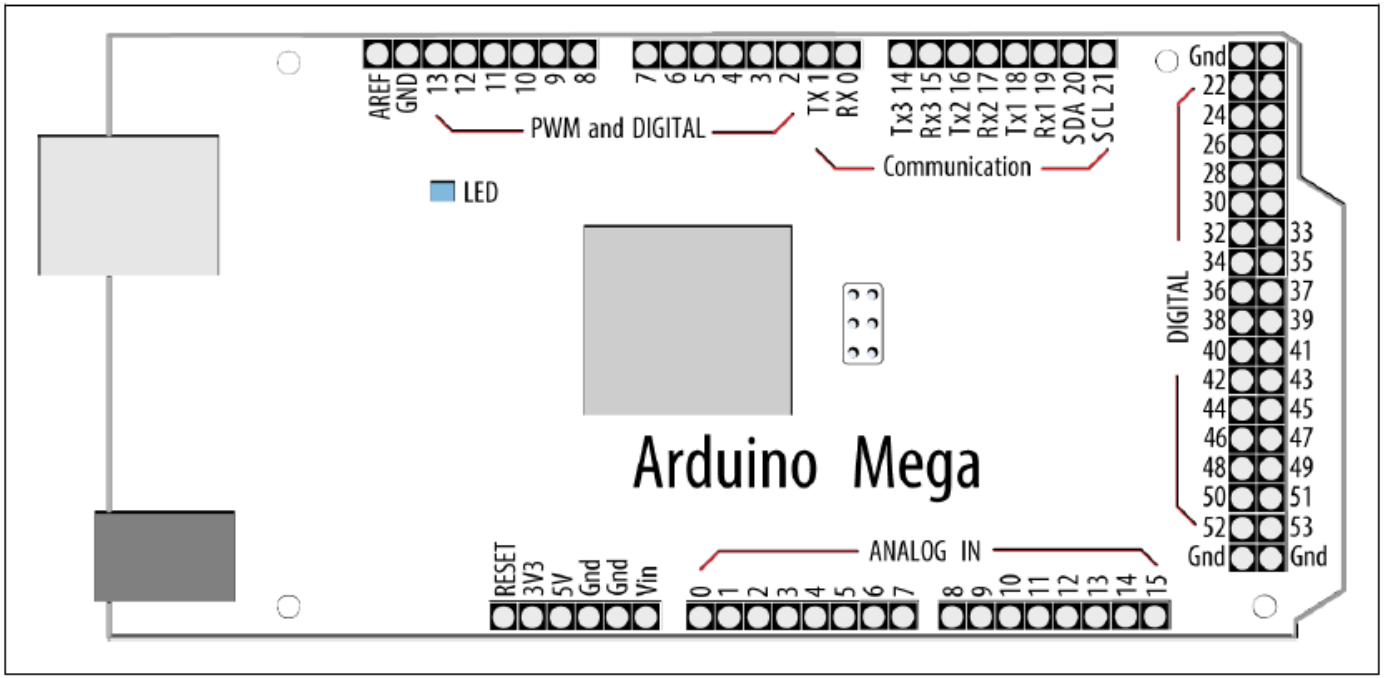
- **Exemplu: stoparea unui proces initiat (continuare)**

```
void loop()
{
  t.update();                                  // actualizare timer
}

void doSomething()                             // functia care se apeleaza la 2 secunde
{
  Serial.print("2 second tick: millis()=");     // transmite numarul de milisecunde curent
  Serial.println(millis());                    // pe interfata seriala
}


void doAfter()                                 // functia care se apeleaza dupa 10 secunde
{                                              // o singura data
  Serial.println("stop the led event");
  t.stop(ledEvent);                            // se opreste oscilarea led-ului
  t.oscillate(13, 500, HIGH, 5);               // si se porneste o alta oscilare, la 500 ms
}                                              // doar de 5 ori
```

http://www.doctormonk.com/2012/01/arduino-timer-library.html

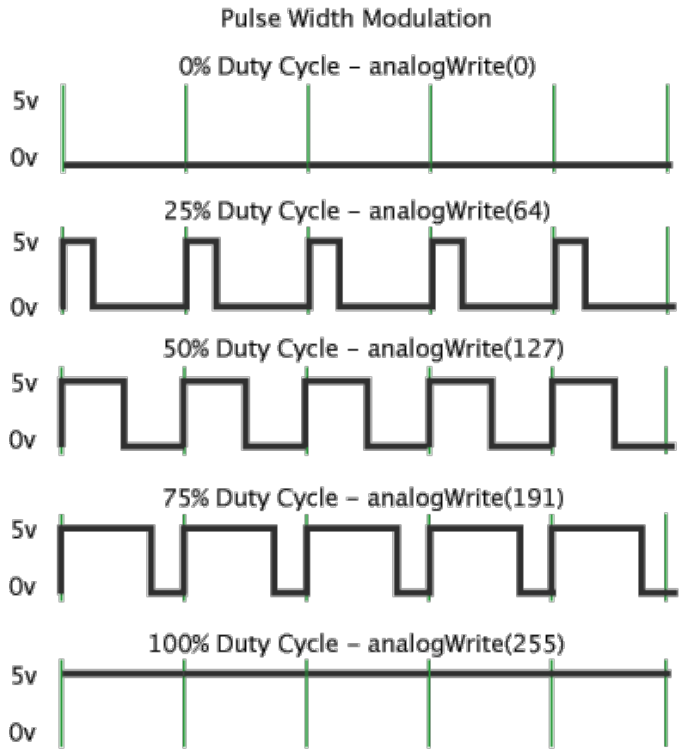
Generare de semnale

- **Pulse Width Modulation (PWM) la Arduino: o parte din pinii Arduino suporta functia PWM, realizata intern prin intermediul temporizatoarelor.**
- **La Arduino Mega, pinii 2-13 suporta functia PWM**
- **Frecventa semnalului PWM este fixa, aproximativ 500 Hz**



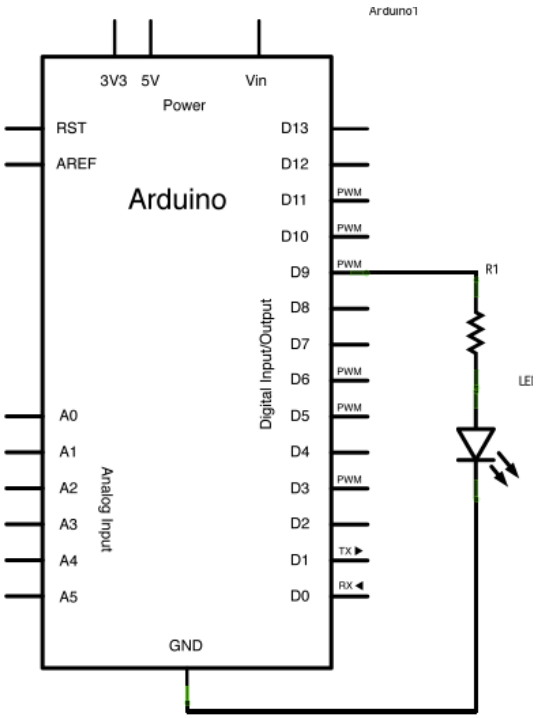
Generare de semnale

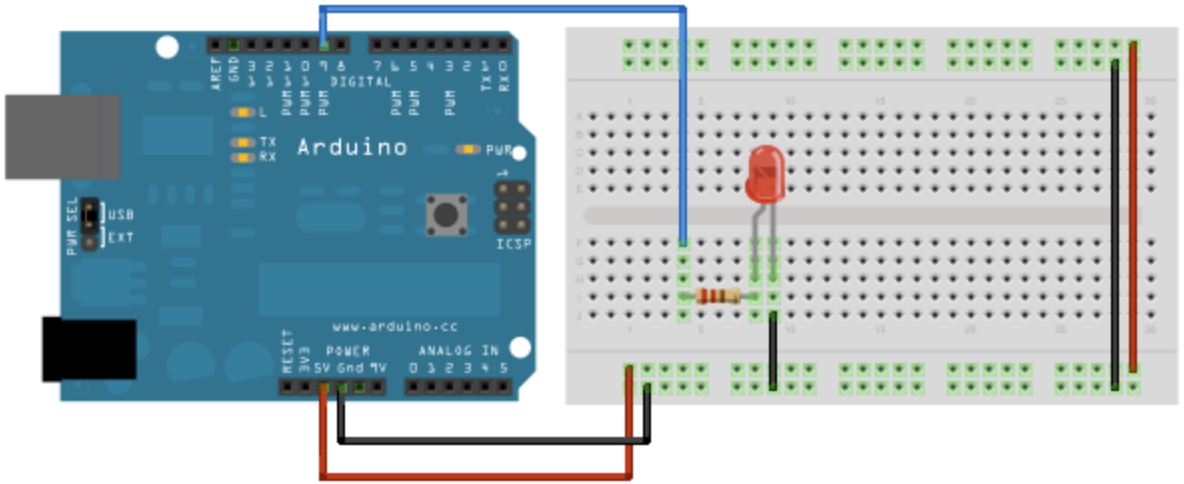
- Functia **analogWrite (pin, value)** determina generarea unui semnal PWM pe pinul ‘pin’, cu factorul de umplere specificat de ‘value’.
- ‘value’ poate lua valori de la 0 la 255, corespunzatoare factorilor de umplede de la 0% la 100%
- Pinul pe care se doreste generarea semnalului PWM trebuie configurat ca iesire.



Generare de semnale

- **Exemplu: fade-in, fade-out cu un led extern**
- **Sursa: http://arduino.cc/en/Tutorial/Fade**





Generare de semnale

- **Exemplu: fade-in, fade-out cu un led extern**
- **Sursa: http://arduino.cc/en/Tutorial/Fade**

```
int led = 9;
int brightness = 0;                    // starea curenta a led-ului, initial stins
int fadeAmount = 5;                    // incrementul starii led-ului


void setup()  {
  pinMode(led, OUTPUT);                // pin 9, iesire
}


void loop()  {

  analogWrite(led, brightness);        // configureaza factorul de umplere al PWM

  brightness = brightness + fadeAmount; // modifica starea curenta prin adaugarea incrementului

  if (brightness == 0 || brightness == 255) { // la capatul intervalului, schimba semnul incrementului
    fadeAmount = -fadeAmount ;
  }

  delay(30);                           // intarziere
}
```

Generare de semnale

- Functia **tone ()** cauzeaza producerea de pulsuri cu factor de umplere 50%, si frecventa variabila. Exista doua moduri de apelare:
- **tone(pin, frequency)** – produce un semnal de frecventa ‘frequency’ pe pinul ‘pin’, de durata nedefinita
- **tone(pin, frequency, duration)** – produce un semnal de frecventa ‘frequency’, pe pinul ‘pin’, cu durata de timp ‘duration’ in milisecunde.

- Functia **noTone(pin)** opreste generarea semnalului pentru pinul ‘pin’.
- Doar un singur pin poate genera ton la un moment dat. Daca se doreste generarea de ton pe alt pin, in timp ce un ton este activ, trebuie apelata functia **noTone()** pentru oprirea tonului activ.
- La unele placi Arduino, generarea tonului poate afecta generarea semnalelor PWM.

Generare de semnale

- **Exemplu: reproducerea unei melodii dupa note.**

```
const int speakerPin = 9;  // Difuzorul se conecteaza la pin-ul 9

char noteNames[] =     {'C','D','E','F','G','a','b'};  // Numele notelor
unsigned int frequencies[] = {262,294,330,349,392,440,494};   // Frecventele asociate
const byte noteCount = sizeof(noteNames);  // Numarul de note in tabela

// Partitura melodiei, spatiul reprezinta pauza
char score[] = "CCGGaaGFFEEDDC GGFFEEDGGFFEED CCGGaaGFFEEDDC ";
const byte scoreLen = sizeof(score);  // Numarul de note in melodie

void setup()
{
}

void loop()
{
  for (int i = 0; i < scoreLen; i++)  // Se parcurge partitura
  {
    int duration = 333;     // Fiecare nota va fi cantata 0.33 secunde
    playNote(score[i], duration);  // Apel functie cantare nota (slide urmator)
  }

  delay(4000);   // Pauza inainte de repetarea melodiei
}
```

Generare de semnale

- **Exemplu: reproducerea unei melodii dupa note (continuare).**

```
void playNote(char note, int duration)
{
   // Se parcurge lista de note
  for (int i = 0; i < noteCount; i++)
  {
     // Se cauta nota curenta in lista
    if (noteNames[i] == note)   // Daca a fost gasita
      tone(speakerPin, frequencies[i], duration);  // Se genereaza tonul corespunzator, cu frecventa notei
  }
   // Daca nota nu a fost gasita, se face pauza oricum
  delay(duration);
}
```

Utilizare avansata a temporizatoarelor

- Uneori este nevoie de mai multe optiuni pentru configurarea temporizatoarelor.
- Doua optiuni: folosirea unor biblioteci dedicate, sau accesul direct la registrii de configurare ai temporizatoarelor de pe microcontrollerul AVR
- Biblioteca Timer1: http://playground.arduino.cc/Code/Timer1
  - Functii pentru folosirea temporizatorului pe 16 biti Timer 1.
  - La Arduino Mega, nu se pot folosi toti pinii de generare de semnal ai Timer 1, si se recomanda folosirea Timer 3, http://playground.arduino.cc/uploads/Code/TimerThree.zip, cu aceleasi functii.
- Cele mai importante metode ale clasei Timer:
- **initialize(period)** – initializarea temporizatorului, cu perioada ‘period’, in microsecunde. Perioada este intervalul in care temporizatorul efectueaza o numaratoare completa.
- **setPeriod(period)** – modifica perioada unui temporizator deja initializat.
- **pwm(pin, duty, period)** – genereaza un semnal PWM pe pinul ‘pin’, cu factorul de umplere ‘duty’ avand valori de la 0 la 1023, si cu perioada specificata (optional) de ‘period’, in microsecunde. Pentru ‘pin’ se accepta doar valorile la care temporizatorul este conectat fizic (Timer 1 conectat la pinii 9 si 10, Timer 3 conectat la pinii 2, 3 si 5 la Arduino Mega).
- **attachInterrupt(function, period)** – ataseaza functia ‘function’ pentru a fi apelata de fiecare data cand temporizatorul isi termina un ciclu, sau la intervale specificate de parametrul optional ‘period’.

Utilizare avansata a temporizatoarelor

- **detachInterrupt()** – dezactiveaza intreruperea si detaseaza functia ISR specificata de attachInterrupt().
- **disablePwm(pin)** – dezactiveaza generarea semnalului PWM pe pin-ul specificat.
- **read()** – returneaza intervalul trecut de la ultima saturare a numaratorului temporizatorului, in microsecunde.

Relatia dintre perioade si prescaller, pentru sisteme cu oscilator de 16MHz:

Prescaler	**Timpul dintre doua incrementari**	**Perioada maxima**
1	0.0625 uS	8.192 mS
8	0.5 uS	65.536 mS
64	4 uS	524.288 mS
256	16 uS	2097.152 mS
1024	64uS	8388.608mS

Utilizare avansata a temporizatoarelor

- **Exemplu de utilizare a bibliotecii timer 1**
- Activeaza generarea unui semnal PWM pe pinul 9, cu factor de umplere 50%, si activeaza o intrerupere care modifica starea pinului 10 la fiecare jumatate de secunda.

```
#include "TimerOne.h"

void setup()
{
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);                // initializeaza timer 1, cu perioada de 0.5 secunde
  Timer1.pwm(9, 512);                       // activeaza pwm pe pinul 9, cu factor de umplere 50%
  Timer1.attachInterrupt(callback);         // ataseaza functia callback() pentru tratarea intreruperii
}                                           // declansata de saturarea temporizatorului

void callback()
{
  digitalWrite(10, digitalRead(10) ^ 1);
}

void loop()
{
  // liber pentru alte procesari
}
```

Utilizare avansata a temporizatoarelor

- **Folosirea registrilor de configurare**
- **Exemplu:** functia setPeriod din libraria Timer1

```
#define RESOLUTION 65536    // Temporizatorul este pe 16 biti

void TimerOne::setPeriod(long microseconds)
{
long cycles = (F_CPU / 2000000) * microseconds;              // modul PWM phase correct, numara in sus si in jos pentru 1  perioada

// verifica daca numarul de cicli se potriveste in valoarea maxima a numaratorului
if(cycles < RESOLUTION)             clockSelectBits = _BV(CS10);              // fara divizare
else if((cycles >>= 3) < RESOLUTION) clockSelectBits = _BV(CS11);             // divizare cu 8
else if((cycles >>= 3) < RESOLUTION) clockSelectBits = _BV(CS11) | _BV(CS10);  // divizare cu 64
else if((cycles >>= 2) < RESOLUTION) clockSelectBits = _BV(CS12);             // divizare cu 256
else if((cycles >>= 2) < RESOLUTION) clockSelectBits = _BV(CS12) | _BV(CS10);  // divizare cu 1024
else        cycles = RESOLUTION - 1, clockSelectBits = _BV(CS12) | _BV(CS10);  // cerere imposibila, divizare maxima si setare
                                                                                // numarul de cicli la maximul posibil

  oldSREG = SREG;                                   // salvare registru stare
  cli();                                            // dezactivare intreruperi
  ICR1 = pwmPeriod = cycles;                        // configurar registru ICR1
  SREG = oldSREG;                                   // refacere stare SREG (alterata de CLI)
  TCCR1B &= ~(_BV(CS10) | _BV(CS11) | _BV(CS12));   // stergere biti de clock select pentru Timer 1
  TCCR1B |= clockSelectBits;                                // configurare biti clock select pentru Timer 1, calculati mai sus

}
```

Utilizare avansata a temporizatoarelor

- **Folosirea registrilor de configurare**
- **Registrii implicati (sursa: http://www.atmel.com/images/doc2545.pdf):**

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

CS12	CS11	CS10	Description
0	0	0	No clock source (timer/counter stopped)
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Registrul ICR1 este pe 16 biti, impartit in ICR1H si ICR1L. Utilizat pentru specificarea valorii prag TOP pana la care numaratorul se incrementeaza. Dupa atingerea TOP, numaratorul se intoarce la 0, cand se termina perioada.

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Utilizare avansata a temporizatoarelor

- **Moduri de configurare pentru temporizatoare de 16 biti**

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Exercitii

- **Presupunand ca nu exista functiile delay() si millis(), implementati-le folosind temporizatoare**
- **Avand un afisor cu doua cifre de 7 segmente, realizati afisarea unui numar dat folosind temporizatoare**
- **Realizati un sistem de control al iluminarii. Fiecare ora din cele 24 ale unei zile este definita de utilizator (printr-un LUT) ca zi, seara sau noapte. In functie de tipul orei, lumina va fi stinsa, aprinsa mediu, sau aprinsa maxim.**
- **Presupunem că avem un convertor Digital/Analog (DAC) extern conectat la portul A, care convertește instantaneu numărul de 8 biți primit într-o tensiune analogică, 0 V pentru valoarea 0, și 3.3 V pentru valoarea 255. Scrieți programul Arduino pentru a genera o funcție sinusoidală cu valoarea medie 1.5V, amplitudinea de 1.5V, și frecvența de 50 Hz.**