Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare

Proiectarea cu Micro-Procesoare

Lector: Mihai Negru

An 3 – Calculatoare și Tehnologia Informației

Seria B

Curs 5: Temporizatoare la sistemele Arduino

http://users.utcluj.ro/~negrum/

Temporizatoare la sistemele Arduino

- **Functii pentru intarziere**
  - **delay**(unsigned long ms) – intârziere pentru un număr specificat de milisecunde
  - **delayMicroseconds**(unsigned int us) – intârziere pentru un număr specificat de µsecunde

- **Functii pentru citirea timpului**
  - unsigned long **millis**() – returnează timpul, în milisecunde, de la pornirea rulării programului curent. Ajunge la saturație după aproximativ 50 de zile
  - unsigned long **micros**() – returnează timpul, în microsecunde, de la pornirea rulării programului curent. Ajunge la saturație după aproximativ 70 minute. La plăcile cu oscilator de 16 MHz (de exemplu, Arduino Mega), această funcție are rezoluția de 4 µs.

Temporizatoare la sistemele Arduino

- **Exemplu: temporizare fără a folosi funcția 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

Temporizatoare la sistemele Arduino

- **Exemplu: temporizare fără a folosi funcția delay() – pentru multitasking !**
  - **Doua led-uri, fiecare comută la 1 secundă, cu intârziere de 0.5 sec între 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/

Temporizatoare la sistemele Arduino

- **Folosirea bibliotecii Timer pentru sincronizare / temporizare**
- **http://playground.arduino.cc/Code/Timer**
- **Metode:**
- **int every(long period, callback)**: rulează funcția ‘callback’ la intervale de ‘period’ milisecunde. Returnează identificatorul evenimentului programat.
- **int every(long period, callback, int repeatCount)**: rulează funcția ‘callback’ la intervale de ‘period’ milisecunde, de ‘repeatCount’ ori.
- **int after(long duration, callback)**: rulează funcția ‘callback’ o singură dată, după un interval de timp de ‘duration’ milisecunde.
- **int oscillate(int pin, long period, int startingValue)**: generare de semnal. Modifică starea pinului ‘pin’ după fiecare ‘period’ milisecunde. Starea inițială a pinului este cea specificată de ‘startingValue’, HIGH sau LOW.
- **int oscillate(int pin, long period, int startingValue, int repeatCount)**: variază starea pinului ‘pin’, la intervale specificate de ‘period’, de ‘repeatCount’ ori.
- **int pulse(int pin, long period, int startingValue)**: schimbă starea lui ‘pin’ o singură dată, după ‘period’ milisecunde. Valoarea inițială este cea specificată in ‘startingValue’.
- **int stop(int id)**: toate funcțiile de mai sus returnează un identificator al evenimentului programat. Această funcție poate fi apelată pentru a opri evenimentul. Doar 10 evenimente pot fi atașate temporizatorului.
- **int update()**: această funcție trebuie apelată in bucla principală (loop) a programului, pentru a actualiza starea obiectului de tip Timer.

Temporizatoare la sistemele Arduino

- **Exemplu: generarea unui puls de durată îndelungată, fără 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

- **Exemplu: apelarea unei funcții la intervale regulate de timp, și 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
}
```

- **Exemplu: stoparea unui proces inițiat**

```
#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

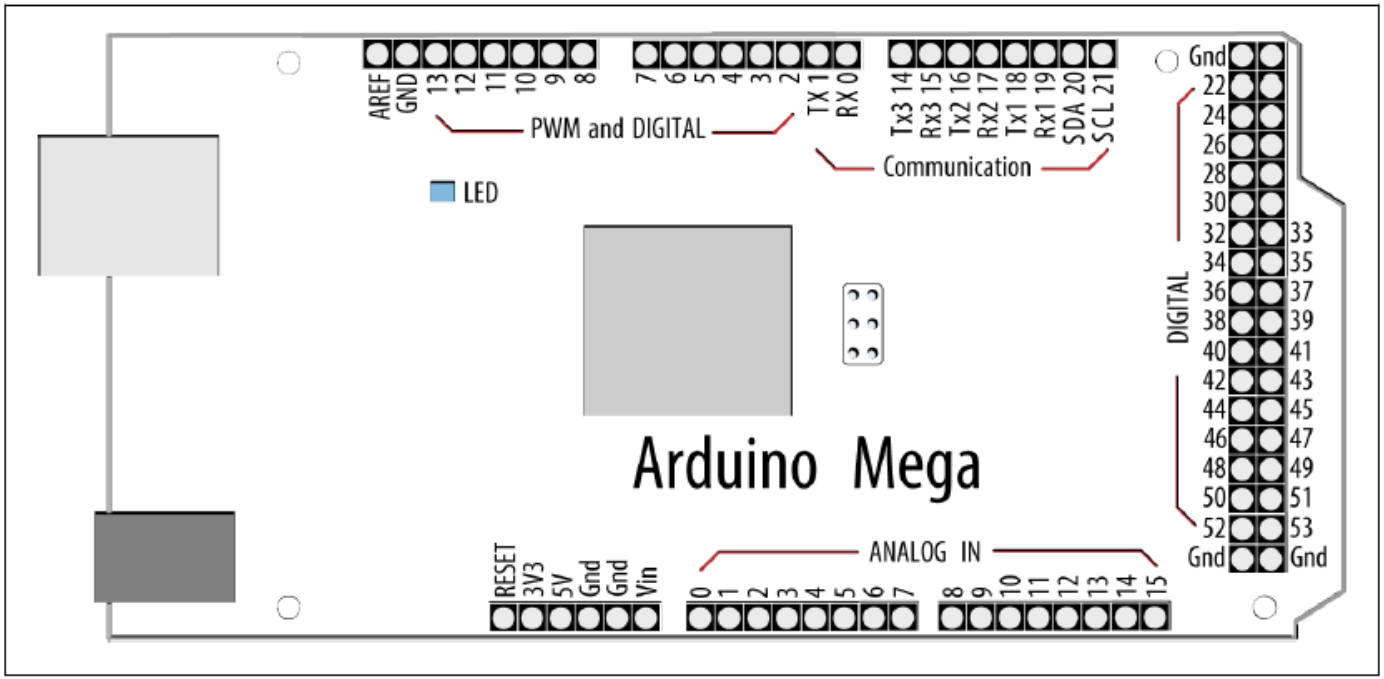
- **Exemplu: stoparea unui proces inițiat**

```
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

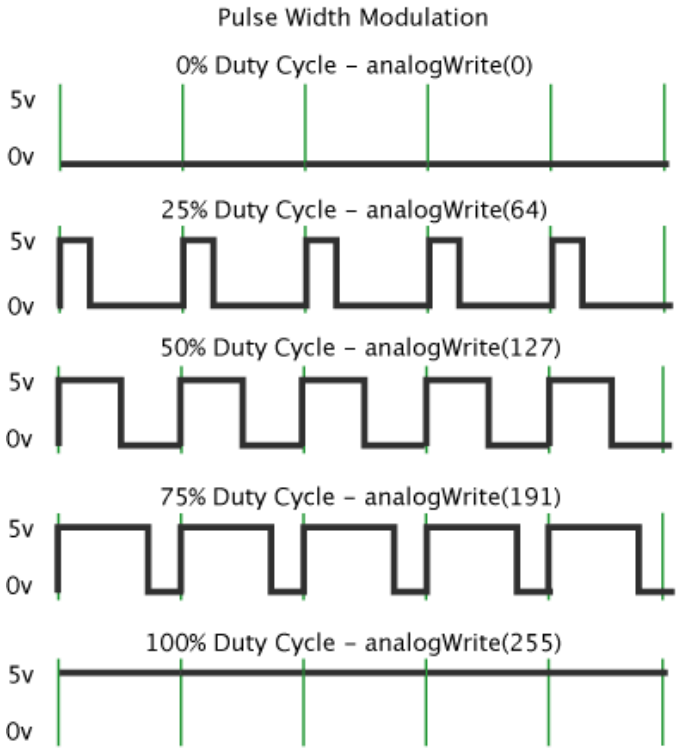
Generare de semnale PWM

- **Pulse Width Modulation (PWM) la Arduino:** o parte din pinii Arduino suportă funcția PWM, realizată intern prin intermediul temporizatoarelor
- La Arduino Mega, pinii 2-13 suporta funcția PWM
- Frecvența semnalului PWM este fixă, aproximativ 500 Hz



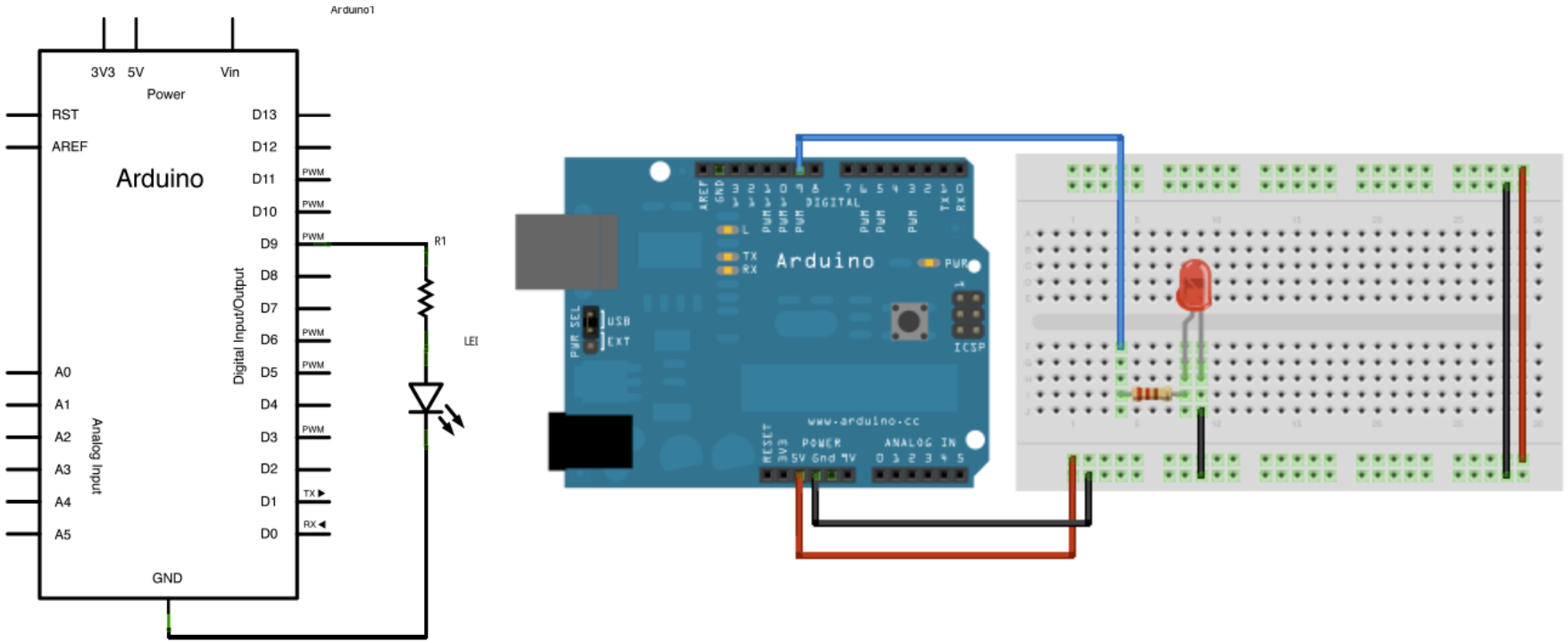
Generare de semnale PWM

- Funcția **analogWrite (pin, value)** determină 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 umplere de la 0% la 100%
- Pinul pe care se dorește generarea semnalului PWM trebuie configurat ca ieșire.



Generare de semnale PWM

- **Exemplu: fade-in, fade-out cu un led extern**



http://arduino.cc/en/Tutorial/Fade

Generare de semnale PWM

- **Exemplu: fade-in, fade-out cu un led extern**

```
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
}
```

http://arduino.cc/en/Tutorial/Fade

Generare de semnale PWM

- Funcția **tone ()** cauzează producerea de pulsuri cu factor de umplere 50%, și frecvență variabilă. Există două moduri de apelare:
- **tone(pin, frequency)** – produce un semnal de frecvența 'frequency' pe pinul 'pin', de durată nedefinita
- **tone(pin, frequency, duration)** – produce un semnal de frecvența 'frequency', pe pinul 'pin', cu durata de timp 'duration' in milisecunde.

- Funcția **noTone(pin)** oprește generarea semnalului pentru pinul 'pin'.
- Doar un singur pin poate genera ton la un moment dat. Dacă se dorește generarea de ton pe alt pin, în timp ce un ton este activ, trebuie apelată funcția **noTone()** pentru oprirea tonului activ.
- La unele placi Arduino, generarea tonului poate afecta generarea semnalelor PWM.

Generare de semnale PWM

- **Exemplu: reproducerea unei melodii după 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 PWM

- **Exemplu: reproducerea unei melodii după note**

```
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);
}
```

Utilizarea avansată a temporizatoarelor

- Uneori este nevoie de mai multe opțiuni pentru configurarea temporizatoarelor.
- Două opțiuni: folosirea unor biblioteci dedicate, sau accesul direct la regiștrii de configurare ai temporizatoarelor de pe microcontrollerul AVR
- Biblioteca Timer1: http://playground.arduino.cc/Code/Timer1
  - Funcții pentru folosirea temporizatorului de 16 biți Timer 1.
  - La Arduino Mega, nu se pot folosi toți pinii de generare de semnal ai Timer 1, și se recomandă folosirea Timer 3

    http://playground.arduino.cc/uploads/Code/TimerThree.zip, cu aceleași functii.
- Cele mai importante metode ale clasei Timer:
- **initialize(period)** – inițializarea temporizatorului, cu perioada 'period', în microsecunde. Perioada este intervalul în care temporizatorul efectuează o număratoare completă.
- **setPeriod(period)** – modifică perioada unui temporizator deja inițializat.
- **pwm(pin, duty, period)** – generează un semnal PWM pe pinul 'pin', cu factorul de umplere 'duty' avand valori de la 0 la 1023, și cu perioada specificată (opțional) de 'period', în microsecunde. Pentru 'pin' se acceptă doar valorile la care temporizatorul este conectat fizic (Timer 1 conectat la pinii 9 și 10, Timer 3 conectat la pinii 2, 3 și 5 la Arduino Mega).

Utilizarea avansată a temporizatoarelor

- **attachInterrupt(function, period)** – atașeaza funcția ‘function’ pentru a fi apelată de fiecare dată cand temporizatorul își termină un ciclu, sau la intervale specificate de parametrul opțional ‘period’.
- **detachInterrupt()** – dezactivează întreruperea și detașează funcția ISR specificată de attachInterrupt().
- **disablePwm(pin)** – dezactivează generarea semnalului PWM pe pin-ul specificat.
- **read()** – returnează intervalul trecut de la ultima saturare a număratorului temporizatorului, in microsecunde.

Relația dintre perioade și prescaller, pentru sisteme cu oscilator de 16MHz:

Prescaler	Timpul dintre două incrementari	Perioada maxima
1	0.0625 µs	8.192 ms
8	0.5 µs	65.536 ms
64	4 µs	524.288 ms
256	16 µs	2097.152 ms
1024	64 µs	8388.608 ms

- **Exemplu de utilizare a bibliotecii Timer1**
- Activează generarea unui semnal PWM pe pinul 9, cu factor de umplere 50%, și activează o intrerupere care modifică starea pinului 10 la fiecare 0.5 secunde

```
#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
}
```

Utilizarea avansată a temporizatoarelor

- **Folosirea regiștrilor de configurare**
- **Exemplu:** funcția 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 cicluri 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

}
```

Utilizarea avansată a temporizatoarelor

- **Folosirea regiștrilor de configurare**

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 biți, împărțit in ICR1H și ICR1L. Utilizat pentru specificarea valorii prag TOP pana la care numărătorul se incrementează. După atingerea TOP, numărătorul se intoarce la 0, cand se termină 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	

Exerciții

- Presupunând că nu există funcțiile delay() și millis(), implementați-le folosind temporizatoare

- Având un afișor cu două cifre de 7 segmente, realizați afișarea unui număr dat folosind temporizatoare

- Realizați un sistem de control al iluminării. Fiecare oră din cele 24 ale unei zile este definită de utilizator (printr-un LUT) ca zi, seară sau noapte. In funcție de tipul orei, lumina va fi stinsă, aprinsă mediu, sau aprinsă maxim

Referințe

1. Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V datasheet
2. Atmel Atmega64 datasheet
3. http://arduino.cc/en/
4. http://www.baldengineer.com/blog/2011/01/06/millis-tutorial/
5. http://www.doctormonk.com/2012/01/arduino-timer-library.html
6. http://arduino.cc/en/Tutorial/Fade