Laborator 4 – Utilizarea temporizatoarelor

1. Întreruperi bazate pe temporizatoare

În afara întreruperilor externe, studiate în laboratorul precedent, există şi întreruperi interne, care nu sunt cauzate de evenimente pe pinii exteriori, ci de componente hardware incluse in microcontroler. În aceasta categorie sunt incluse întreruperile generate de temporizatoare (timers).

Prin utilizarea acestor întreruperi, puteţi genera acţiuni la intervale precise de timp, fără a utiliza funcţii de tip delay sau millis. Întreruperile de temporizatoare funcţionează asincron, lucru care permite ca programul sa execute bucla principala, şi doar când s-a ajuns la un anumit prag de timp să execute o rutină specifică. Temporizatorul incrementează un registru numit counter register, şi în momentul în care se atinge valoarea maximă posibilă în acest registru, se setează un bit (flag) ce indică depăşirea acestei valori (overflow). Acest flag poate fi verificat manual, sau poate declanşa o cerere de întrerupere. La execuţia procedurii de tratare a întreruperii (ISR), acest flag va fi pus din nou la zero.

Fiecare temporizator are nevoie de o sursă de semnal de ceas (clock). În general se alege ca sursă oscilatorul plăcii, frecvenţa acestuia putând fi divizată printr-un numărător auxiliar (prescaler).

Arduino Mega are 5 temporizatoare (2 pe 8 biţi si 3 pe16 biţi) care pot fi folosite. La Arduino UNO diferenţa este ca exista doar un timer pe 16 biti). Pentru a folosi aceste temporizatoare, va trebui să setăm valori specifice în registrele de configurare. Două dintre aceste registre sunt TCCRxA şi TCCRxB, unde x este numărul temporizatorului. TCCR vine de la Timer Counter Control Register. Pentru pornirea temporizatorului, cei mai importanţi biţi sunt cei trei biţi care aleg divizarea frecvenţei de intrare, reglând implicit prin aceasta viteza de numărare. In figurile de mai jos se observă structura registrelor A şi B, şi posibile configurări pentru biţii de selecţie a frecvenţei.

TCCR1A – Timer/Counter 1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
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	

TCCR1B – Timer/Counter 1 Control Register B

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	

Table 16-5. Clock Select Bit Description

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.

Implicit cei trei biți sunt 0, ceea ce înseamnă că temporizatorul nu are semnal de ceas, şi deci este oprit. O configurare validă pentru frecvența de intrare înseamnă pornirea temporizatorului. Dacă sursa de ceas selectată este de tip extern, temporizatorul va funcționa doar dacă o astfel de sursă este conectată la placă (ceea ce de obicei nu e cazul, deci astfel de setări se vor evita).

Un sumar al celor mai importante registre pentru lucrul cu temporizatoarele:
- TCCRx - Timer/Counter Control Register. Aici se poate selecta sursa de ceas.
- TCNTx - Timer/Counter Register. Valoarea numărată este stocata aici (valoarea de moment a temporizatorului).
- OCRx - Output Compare Register – valoare scrisă de utilizator, cu care se compară TCNTx, pentru a genera diferite unde sau evenimente.
- ICRx - Input Capture Register (doar pentru numărătoare pe 16 biți) – folosit pentru a măsura timpul dintre evenimente externe.
- TIMSKx - Timer/Counter Interrupt Mask Register. Pentru activarea sau dezactivarea întreruperilor bazate pe temporizator.
- TIFRx - Timer/Counter Interrupt Flag Register. Indică prezența unei cereri de întrerupere.

În exemplul următor vom incrementa o variabila în momentul în care temporizatorul TIMER1 va face overflow. Valoarea variabilei incrementate va fi afişată pe LCD.

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

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
volatile int myVar;

void setup()
{
  myVar = 0;
  //initializarea primului numarator
  cli(); //facem disable la intreruperile globale pentru a face
//modificarile corespunzatoare timerelor
  TCCR1A = 0; // SETAM TCCR1A si B la 0
  TCCR1B = 0; // timer este setat in Normal mode (WGMx3:0 = 0)
```

```
  lcd.begin(16, 2);
  lcd.print("Timere");

  //facem enable la intrerupere de overflow pentru timerul 1
  TIMSK1 = (1 << TOIE1); //timer overflow intrerupt eneble for timer 1
  //setam temporizatorul sa ruleze o frecventa divizata cu 1024
  //DE MENTIONAT CA FRECVENTA PROCESORULUI e de 16 MHZ si timer 1
  //este un temporizator de 16 biti
  //cu un prescaller de 1024  avem incrementare o data la   t = 1024 / (16 * 10^6) secunde
  //temporizatorul va da overflow la fiecare (t * 2^16) = 4.194 secunde

  TCCR1B |= (1 << CS10);
  TCCR1B |= (1 << CS12);

  //activam intreruperile globale
  sei();
}

void loop()
{
 lcd.setCursor(0,1);
 lcd.print(myVar);
 lcd.setCursor(5, 1);
 lcd.print(TCNT1);
}

ISR(TIMER1_OVF_vect)
{
   myVar = myVar + 1;
}
```

Pentru a face ca întreruperea de temporizator să se declanşeze la un anumit moment dorit de utilizator, şi nu doar când capacitatea registrului de numărare este depăşită, vom folosi un alt mod de declanşare a întreruperilor numit CTC (clear on timer compare match). În acest mod de lucru, valoarea registrului de numărare TCNTx se va compara cu registrul OCRx scris de utilizator, şi la egalitate TCNTx va lua din nou valoarea zero.

Pentru a obține o perioadă T între cererile de întrerupere, trebuie să scriem valoarea OCRx rezultată prin aplicarea următoarei ecuații:

$$OCRx + 1 = T / (P / (16 * 10^6)) \qquad (1)$$

În ecuația 1, T reprezintă timpul dorit între evenimente (de exemplu, 1 secundă), iar P reprezintă factorul de diviziune a frecvenței (de exemplu 1024). Se aplică +1 la valoarea OCR deoarece resetarea valorii temporizatorului în momentul în care s-a atins valoarea dorită durează un ciclu de ceas.

Aplicând ecuația pentru o perioadă T de 1 secundă, vom obține pentru OCRx valoarea 15624.

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

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
volatile int myVar;
void setup()
{
   // initialize Timer1
   cli();          // facem disable la intreruperile globale
   TCCR1A = 0;     // setam TCCR1A si B la 0
   TCCR1B = 0;
   lcd.begin(16, 2);
   lcd.print("Timere cu CTC");
   // setam registrul cu valoarea caruia vom compara TCNT
   OCR1A = 15624;
   // activam modul CTC:
   TCCR1B |= (1 << WGM12);
   // divizam ceasul placii cu 1024:
   TCCR1B |= (1 << CS10);
   TCCR1B |= (1 << CS12);
   // facem enable la intreruperea la comparare prin setarea bitului
   //corespunzator din masca
   TIMSK1 |= (1 << OCIE1A);
   // validam sistemul global d eintreruperi
   sei();
}

void loop()
{
 lcd.setCursor(0,1);
 lcd.print(myVar);
 lcd.setCursor(5, 1);
 lcd.print(TCNT1);
}

ISR(TIMER1_COMPA_vect)
{
   myVar = myVar + 1;
}
```

Spre deosebire de exemplul anterior în care incrementam variabila doar la overflow (la aproximativ 4 secunde), în acest exemplu avem mai mult control asupra perioadei de incrementare a variabilei noastre; variabila se incrementează la fiecare secundă.

În continuare vom vedea o metodă mai simplă pentru utilizarea temporizatoarelor, folosind biblioteca TimerOne. Exemplul de mai jos va activa o rutină la fiecare secunda. Biblioteca Timer One poate fi descărcată de pe site-ul Arduino, de la adresa

http://playground.arduino.cc/Code/Timer1 , unde sunt date şi explicaţii detaliate, precum şi instrucţiunile de instalare.

```
#include <TimerOne.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

volatile int myVar;

void setup(void)
{
Timer1.initialize(1000000); //se initializeaza intervalul de timp la care temporizatorul va
// declanşa evenimente (in microsecunde, 1000000 microsecunde = 1 secunda)

  Timer1.attachInterrupt(ShowMessage); // funcţia ShowMessage se va apela la intevalul
                                       // stabilit
}

void ShowMessage(void)
{
  lcd.setCursor(0,0);
  lcd.print(myVar);
  myVar++;
}

void loop(void)
{
}
```

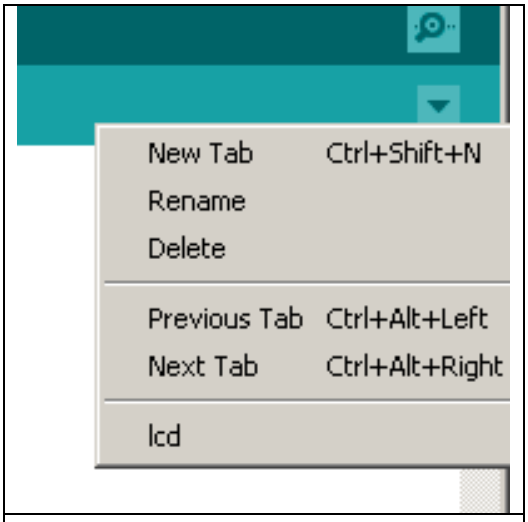
2. Generarea de tonuri de frecvenţă dată

Pentru generarea de tonuri Arduino pune la dispoziţia utilizatorului funcţia **tone**, care generează un puls dreptunghiular de frecvenţă specificată, şi factor de umplere 50%. La apelul acestei funcţii trebuie specificata durata semnalului, altfel tonul va continua să sune până la apelarea funcţiei **noTone()**. Nu este posibil să generaţi tonuri mai joase de 31 Hz. Daca vreţi să generaţi tonuri pe mai mulţi pini, va trebui să apelaţi funcţia **noTone()** înainte de a genera tonul pe un alt pin. Sintaxa funcţiei este

tone(pinul, frecventa, durata)
sau
tone(pin, frecventa)

Pentru a putea rula exemplul următor, realizaţi următoarea conexiune: conectaţi **pinul rosu** (de semnal) de la difuzor la **pinul digital 8 de pe placa Arduino**. **Pinul negru** al difuzorului trebuie conectat la masă (**GND**).

În continuare, creaţi un fişier nou numit **pitches.h**. Pentru a crea un fişier nou pentru un proiect, apăsaţi butonul din partea dreapta sus a ferestrei mediului de dezvoltare, şi apăsaţi pe butonul “New Tab”, ca în figura următoare.



Creare fişier nou

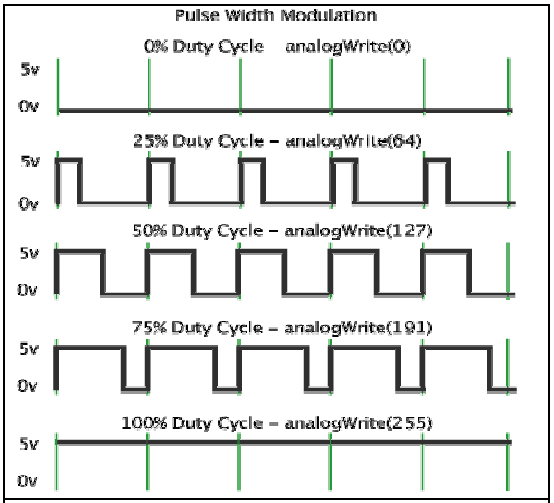
Numiţi acest fişier pitches.h, şi introduceţi în el tonurile definite in **anexa A** a acestui document. Salvaţi documentul, şi reveniţi la tab-ul principal, unde veţi introduce următorul program:

```
//includem fisierul cu definițiile pentru tonuri
#include "pitches.h"

// melodia ca lista de note
int melody[] = {
  NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};

//introducem durata pentru fiecare nota din melodie
int noteDurations[] = {4, 8, 8, 4,4,4,4,4 };

void setup() {

//pentru fiecare nota din vectorul melody
 for (int thisNote = 0; thisNote < 8; thisNote++) {
        //calculam durata de afisare a notei
         int noteDuration = 1000/noteDurations[thisNote];
        //apelam functia de tone pentru difuzorul atasat la pinul 8 si durata specificata
        tone(8, melody[thisNote],noteDuration);
        int pauseBetweenNotes = noteDuration * 1.30;
        delay(pauseBetweenNotes);
        noTone(8);
        }

}

void loop()
{
}
```

3. Generare de semnale PWM cu Arduino

PWM (pulse width modulation, modulație prin lățimea pulsului) este o metoda de a obține un semnal analogic prin intermediul unui semnal digital, alternând periodic ieşirea între nivelul logic 1 şi nivelul logic 0. Fracțiunea de perioadă cât semnalul este activ (1 logic) se numeşte factor de umplere, sau, în engleză, duty cycle. Cu Arduino se poate genera PWM în trei moduri: fie folosind direct modurile de lucru PWM ale temporizatoarelor, fie folosind funcția **analogWrite,** fie variind prin program durata cat un pin este 1 logic (software PWM).

În continuare vom folosi funcția **analogWrite (factor_de_umplere)**. Valorile posibile pentru argumentul funcției analogWrite sunt de 0, care reprezintă 0 Volți, la 255, echivalentul Vcc (semnal permanent 1). Factorul de umplere de 50% se realizează pentru argumentul 127. În figura următoare se pot observa mai multe exemple de PWM.



Exemple PWM cu factor de umplere diferit

In exemplul care va urma vom genera un semnal PWM pentru generatorul de sunete piezoelectric, şi un PWM, cu acelaşi factor de umplere, pentru LED-ul conectat pe placă. Pinul de semnal al difuzorului (firul roşu) se conectează la pinul digital 8, iar firul negru se conectează la GND.

```
int buzerPin = 8; //pinul la care atasam generatorul de sunete
int puls = 0;    // factorul de umplere, initial 0
int pas = 10;    // pasul de incrementare al factorului de umplere
int ledPin = 13; //ledul de pe placa

void setup()  {
  // declararea pinilor ca iesire
  pinMode(buzerPin, OUTPUT);
  pinMode(ledPin, OUTPUT);
}
```

```
void loop()  {
  // setam factorul de umplere al buzerului si al ledului
  analogWrite(buzerPin, puls);
  analogWrite(ledPin, puls);
  // modificam pwm-ul pentru urmatoarea iteratie
  puls = puls + pas;

  // schimbam directia la capetele intervalului: din incrementare devine decrementare, si invers
  if (puls <= 0 || puls >= 255) {
    pas = -pas ;
  }
  // un mic delay pentru a vedea efectul
  delay(30);
}
```

Lucru Individual

1. Implementați toate exemplele din laborator. Întrebați cadrul didactic pentru orice nedumerire legata de conceptele din laborator sau conectivitatea cu placa.
2. Folosind întreruperile, realizați rularea melodiei în fundal, în mod repetat, în timp ce programul principal va rula o animație pe afişorul LCD. Animația poate fi simplă (gen: deplasarea unui caracter pe ecran). Viteza animației va varia în timp, pentru a demonstra independența melodiei de programul principal.
3. Folosind blocul de butoane, realizați un mini-pian cu patru note. Asigurați-va ca sunetul generat se va opri la ridicarea butonului.
4. Folosind PWM şi un bloc de led-uri, realizați o animație prin care intensitatea fiecărui LED este variată în mod continuu.

Referinte

1.Datasheet ATMega 2560 http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf
2. Biblioteca Timer 1 http://playground.arduino.cc/Code/Timer1
3. Functia analogWrite https://www.arduino.cc/en/Reference/AnalogWrite
4. Generarea de tonuri: https://www.arduino.cc/en/Reference/Tone

Anexa A: continutul fisierului pitches.h

```
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
```

```
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```