LABORATOR 11 – întreruperi

Mai întâi terminaţi implementarea pasului 3 – „**c Pitagoreic”** – din laboratorul 10. Testați cu numerele 004, 005, 998 şi 999. **Când funcţionează, chemați profesorul.**

Prezentare

Programul din laboratorul 10 îndeplinește 2 sarcini:

1. Controlează luminozitatea unui LED prin metoda PWM
2. Determină dacă un număr de 3 cifre introdus de la tastatura este cel mai mare număr dintr-o tripletă pitagoreică.

Programul scris anterior prezintă un mare dezavantaj, dezavantaj care îl face inutilizabil: dacă numărul introdus de la tastatură este relativ mare (de exemplu 998), pe durata calculelor făcute pentru a determina dacă este pitagoreic sau nu apăsarea tastelor C și D nu mai este sesizată. Toate acestea se întâmplă deoarece pentru valori mari ale lui *c* timpul de execuție a buclei principale devine foarte mare, de ordinul secundelor.

Soluţia constă în rezolvarea luminozității LED-ului prin întreruperi.

Pentru sarcina 1 – luminozitate –vom scrie o rutină apelabilă pe întrerupere. Ciclarea timerului 0 va genera un apel către această rutina de întrerupere iar aici vom face scanarea tastaturii, calculul luminozității și afișarea acesteia.

Pentru acest laborator parcurgeți obligatoriu prelegerea 6, **în special capitolul 6**.

Afișarea intensității LED-ului în rutina de întrerupere

Pentru ca să putem modifica luminozitatea LED-ului pe durata calculelor programul va trebui să reacționeze cât mai repede, **imediat,** la apăsarea unei taste. **Prima soluție** este să adăugăm un bloc hardware suplimentar care să genereze un semnal care să indice că o tastă, oricare ar fi ea, este apăsată. Acest semnal va fi folosit ca cerere de întrerupere externă (INT0 – INT2) și astfel apăsarea oricărei taste va genera întrerupere.

Cum apăsarea unei taste generează impulsuri parazite (vezi prelegere 4) acest bloc trebuie să facă și deparazitare. Deși nu este complicat, acest bloc este alcătuit din o poartă AND cu patru intrări, un condensator, o rezistență și un inversor trigger Schmitt.

A doua soluție este să **facem scanarea tastaturii într-o rutina de întrerupere de timer** în loc să o facem în bucla `while(1).` Perioada de ciclare a acestui timerului trebuie să fie suficient de mică pentru a elimina impulsurile parazite. Această soluție nu necesită hardware suplimentar dar consumă timp la fiecare ciclare a timerului. Cu o mică optimizare acest timp este foarte mic, aproape insesizabil. Din acest motiv vom alege soluția doi. În continuare vom alege timerul la a cărui ciclare se va face scanarea tastaturii.

În momentul de față timerul 0 este folosit în mod PWM pentru controlul luminozității unui LED. Pentru PWM ciclarea timerului nu este folosită și ar putea fi utilizată pentru scanarea tastaturii. Timerul 0 ciclează la aproximativ 8 ms. Pentru a elimina impulsurile parazite generate de apăsarea sau la eliberarea unei taste vom face scanarea tastaturii din 5 în 5 ciclări.

În mod PWM ambii indicatori de ciclare, TOV0 și OCF0 sunt setați. Deoarece setarea lui TOV0 nu depinde de valoarea înscrisă în OCR0, vom alege acest indicator deoarece are o periodicitate constantă.

Desfăşurarea lucrării

Pasul 1: Crearea proiectului

Se va crea un proiect nou cu numele **int** care inițial va fi identic cu proiectul pwm din laboratorul 10. Pentru crearea acestui proiect procedați după cum urmează:

a) **Se va face o copie** a folderului care conține proiectul **pwm**. Numele acestui folder este generat de sistemul de operare. Acest nume este **pwm – Copy**.

b) **Se redenumește folderul pwm – Copy** ca **int**.

c) În folderul **int redenumiți pwm.aps** ca **int.aps.**

d) În folderul **int ștergeți** fișierul **pwm.aws**

e) **Deschideți** (obligatoriu cu NEXT) proiectul **int**.

f) În fereastra **AVR GCC** expandați folderul **Source Files,** faceți click dreapta pe fișierul **pwm.c** și selectați **Rename file...**. Noul nume al fișierului pwm.c va fi **int.c**

Pasul 2: Inițializarea sistemului de întreruperi

În general lucru pe întreruperi presupune o succesiune de pași. Această succesiune trebuie parcursă pentru fiecare cerere de întrerupere (IRQ).

Primii pași aparțin fazei de inițializare. Pentru o cere IRQ generică, notată în continuare IRQ*i*, pașii care trebuie urmați în faza de inițializare sunt:

i. Se configurează blocul care generează cererea de întrerupere IRQ*i*.

ii. Se demaschează cererea de întrerupere IRQ*i*.

iii. Se demaschează întreruperea INT. (IF=1)

Pașii i-iii se execută o singură data, înainte de while(1). În cazul timerului 0 pe baza căruia s-a implementat luminozitatea, acești pași sunt:

i. **Se configurează blocul care generează întreruperea, adică timerul 0.** Setările timerului 0 din laboratorul 10 au fost preluate în acest laborator și sunt deja făcute.

ii. **Se demaschează cererea de întrerupere.** În capitolul „Mascarea cererilor de întrerupere” din curs s-a precizat că orice cerere de întrerupere poate fi mascată. Bitul care maschează cererea TOV0 din TIFR se află în registrul TIMSK:

TIMSK – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
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

Semnificația biților din TIFR și TIMSK se găsește în documentație la pagina 85.

►În *main,* după setările timer-ului 0 și înainte de while(1), demascați cererea de întrerupere TOV0. Documentați (adică scrieți un comentariu) această setare.

iii. **Se demaschează întreruperea INT**. În subcapitolul „Întreruperi mascabile și nemascabile" din curs s-a precizat că mascarea întreruperii se face prin intermediul bistabilului IF. La Atmega16 acest bistabil se află în registrul SREG și se numește I:

SREG – AVR Status Register

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
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

Bistabilul IF din curs este bitul I din SREG.

►În secțiunea de inițializări din *main,* după setarea registrului TIMSK, **configuraţi pe I** pentru a demasca întreruperea procesorului. Documentați această setare. Folosiți macro `setbit(…);` Nu folosiți numele I. Spre deosebire de orice alt identificator din documentația ATmega16 în `io.h` nu exista o definiție pentru I deoarece nu s-ar mai putea defini o variabilă I. În loc de I folosiți propriul identificatorul IFLAG pe care îl definiți cu `#define` în int.c.

Pasul 3: Variabile globale

În prelegerea 4 s-a explicat că pentru a sesiza apăsarea unei taste se fac două scanări ala tastaturii cu *kbscan*(). Dacă prima scanare întoarce NOKEY, adică tastă neapăsată, iar a doua un cod diferit de NOKEY înseamnă că s-a apăsat o tastă între cele două scanări. Cele două scanări se fac la un interval mai mare decât durata instabilității. Programul care sesizează apăsarea unei taste și care generează codului acesteia a fost prezentat în prelegerea 4.

Variabilele **kbhit** și **kbcode** își păstrează semnificația numai că ele vor fi scrise în rutina de întrerupere și citite chiar în rutina de întrerupere pentru luminozitate variabilă și în main pentru *c* pitagoreic. Pentru ca acest lucru să fie posibil variabilele **kbhit** şi **kbcode** vor deveni globale și vor fi utilizate după cum urmează:

- Variabila **kbhit** este setată în rutina de întrerupere dacă s-a detectat o tastă apăsată și este resetată când se folosește codul tastei, în rutina de întrerupere sau în main.
- Variabila **kbcode** conține codul tastei apăsate fiind scrisă în rutina de întrerupere și citită când se folosește codul tastei

►**Mutaţi aceste variabile** înainte de *main* și declarați-le volatile:

```
volatile unsigned char kbhit = 0;
volatile char kbcode;

main(){...
```

Apoi **eliminați** din *main* penultima linie:

```
kbhit=0;
```

Pasul 4: Rutina de întrerupere

Codul de scanare a tastaturii plus codul pentru controlul luminozității, se vor muta în rutina de întrerupere. Se vor executa următorii pași. Urmăriți textul scris cu verde:

1. Mai întâi includeți headerul interrupt.h cu:

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

2. Rutina de întrerupere poate fi scrisă înainte sau după *main*. Din motive de claritate a codului o vom scrie după main(). Declarația rutina de întrerupere cu:

   ```
   ISR(TIMER0_OVF_vect){
   ```

   ISR este definit în interrupt.h iar TIMER0_OVF_vect în io.h.

3. Variabilele `loop_cnt, code_ante` şi `code_now` declarate până acum în *main* vor deveni variabile locale în ISR. **Mutați declararea acestor variabile din *main* în ISR.**

   Atenție: variabilele locale nu își păstrează valoare între apeluri. Detecția apăsării prin compararea a două valorii succesive întoarse de *kbscan* presupune memorarea valorii întoarse la apelul curent. **Modificați** declarația variabilelor locale `code_ante, code_now` şi `loop_cnt` astfel încât acestea să-și păstreze valoarea între apeluri.

4. Codul care are legătură cu detecția apăsării se mută din *main* în rutina de tratare:

   ```
   if(loop_cnt==4){ //4 sau altă valoare
      loop_cnt=0;

      code_ante = code_now;
      code_now  = kbscan();
      if( code_ante == NOKEY && code_now != NOKEY){
         kbhit=1;
         kbcode=code_now;
      }
   }
   ```

5. Terminați rutina de întrerupere:

   ```
   }//end ISR
   ```

În acest moment în rutina de întrerupere se face doar scanarea tastaturii, nu și afișarea, așa că luminozitatea încă nu se poate modifica pe durata calculelor pentru *c* pitagoreic.

Deoarece tastatura se scanează în rutina de întrerupere preluarea celor 3 cifrele pentru *c* pitagoreic trebuie să funcționeze la fel ca până acum. Mai mult, dacă apăsăm C sau D pe durata calculelor vom vedea modificarea luminozității și afișarea aferentă imediat ce calculele se termină, **fără să apăsam încă o tastă**.

În final mutați și eventual modificați codul pentru intensitatea LED-ului în rutina de întrerupere.

Testați! Dacă *c* pitagoreic, tastatura și luminozitatea funcționează cum a fost descris anterior, chemați profesorul pentru validare. Dacă ați ajuns aici aveți nota 5.

Partea opțională

Pasul 5: Adăugarea ceasului.

În acest moment programul îndeplinește două funcțiunii: controlează luminozitatea unui LED prin metoda PWM și determină dacă un număr de 3 cifre introdus de la tastatura este cel mai mare număr dintr-o tripletă pitagoreică. Dorim să adăugăm a treia funcționalitate și anume **ceas digital**.

Mai întâi vom copia variabilele și codul pentru ceas din laboratorul 9. Dacă am adăuga codul pentru ceas în `main()` timpul mare de execuție al lui *c* pitagoreic ar face ca pe durata calculelor ceasul sa „înghețe”. De aceea codul pentru ceas, cu mici modificări, se va scrie într-o nouă rutina de tratare întrerupere ISR.

Observație: implementarea **afișării** ceasului într-o rutină de întrerupere este conceptual greșită. Afișarea unui caracter necesită 37 us iar afișarea timpului înseamnă un `gotoLC` plus afișarea a 8 caractere. În total afișarea timpului necesită 37x8≈300us. Durata mare de afișare face ca rutina de întrerupere să încalce principiul conform căruia rutina de întrerupere trebuie să fie cât mai scurtă. Durata mare de afișare apare din cauza pollingul din `ckbf`.

Implementarea corectă se face cu coadă: orice operație cu LCD-ul se scrie sub forma unei comenzi într-o coada. În celălalt capăt al cozii se face afișarea propriu-zisă: se verifică dacă LCD este liber și dacă da, se extrage o comandă și se execută. Dacă LCD nu este liber, nu se așteaptă ci se continuă în secvență. Evident secțiunea de afișare propriu-zisă nu poate fi rezidentă în bucla principală pentru că în unele cazuri (cum ar fi *c* pitagoreic) timpul de execuție al acestei bucle devine foarte mare ci între-o rutină de întrerupere a unui timer ce ciclează la aproximativ 10ms.

Pentru că implementarea afișării cu coadă ar lua prea mult timp, în acest laborator vom face afișarea timpului în rutina de întrerupere chiar dacă este conceptual greșit.

Succesiunea de pași pentru ceas este aceeași ca cea pentru luminozitate. Pentru scrierea rutinei de întrerupere se vor executa următorii pași. Urmăriţi textul scris cu verde:

a) Mai întâi se execută cei trei pași ce aparțin fazei de inițializare, așa cum s-a procedat la pasul 2. La pasul i) copiați setarea timerului 2 din laboratorul 9. Inserați această setare după setarea timrului 0. La pasul ii) **demascați cererea de întrerupere** corespunzătoare flagului folosit pentru ceas. Bitul care maschează această cerere se află în registrul TIMSK. Pasul iii), setarea lui I, este deja făcut.

b) Rutina de întrerupere va fi scrisă după *main*. Declaraţia rutina de întrerupere cu:

```
ISR(TIMER???){
```

Numele vectorului asociat indicatorului (flag-ul) folosit pentru ceas se găsește în tabelul următor. Numele vectorilor sunt definiți în fișierul `iom16.h`. Înlocuiți `TIMER???` cu numele din tabel corespunzător flag-ului folosit de timerul ceasului!

Eveniment	**Denumire vector în io.h**	**Număr vector**
External Interrupt Request 0	#define INT0_vect	_VECTOR(1)
External Interrupt Request 1	#define INT1_vect	_VECTOR(2)
Timer/Counter2 Compare Match	#define TIMER2_COMP_vect	_VECTOR(3)
Timer/Counter2 Overflow	#define TIMER2_OVF_vect	_VECTOR(4)
Timer/Counter1 Capture Event	#define TIMER1_CAPT_vect	_VECTOR(5)
Timer/Counter1 Compare Match A	#define TIMER1_COMPA_vect	_VECTOR(6)
Timer/Counter1 Compare Match B	#define TIMER1_COMPB_vect	_VECTOR(7)
Timer/Counter1 Overflow	#define TIMER1_OVF_vect	_VECTOR(8)
Timer/Counter0 Overflow	#define TIMER0_OVF_vect	_VECTOR(9)
Serial Transfer Complete	#define SPI_STC_vect	_VECTOR(10)
USART, Rx Complete	#define USART_RXC_vect	_VECTOR(11)
USART Data Register Empty	#define USART_UDRE_vect	_VECTOR(12)
USART, Tx Complete	#define USART_TXC_vect	_VECTOR(13)
ADC Conversion Complete	#define ADC_vect	_VECTOR(14)
EEPROM Ready	#define EE_RDY_vect	_VECTOR(15)
Analog Comparator	#define ANA_COMP_vect	_VECTOR(16)

Two-wire Serial Interface	#define TWI_vect	_VECTOR(17)
External Interrupt Request 2	#define INT2_vect	_VECTOR(18)
Timer/Counter0 Compare Match	#define TIMER0_COMP_vect	_VECTOR(19)
Store Program Memory Ready	#define SPM_RDY_vect	VECTOR(20)

c) Mutați variabila `cycles` în rutina de întrerupere.

Atenție: variabilele locale nu își păstrează valoare între apeluri. **Modificați** declarația lui cycles astfel încât aceasta să-şi păstreze valoarea între apeluri.

d) Variabilele `sec, min` și `hrs` declarate până acum în *main* **vor deveni variabile globale** pentru ca timpul să poată fi setat în main (nu se va face în acest laborator). Numele ales de fiecare echipă pentru variabilele secunde, minute şi ore pot să difere. Nu redenumiţi variabilele, folosiţi numele alese iniţial.

e) Codul pentru ceas este scris în corpul unui if:

```
~~if( TIFR...){~~
~~    TIFR=...;~~
    cycles++;
    if(cycles...){
        ...
    }
~~}~~
```

`if`-ul care testează OCF2 din TIFR **nu mai este necesar** deoarece setarea lui OCF2 ne duce automat în rutina de întrerupere asociată.

Linia următoare este pentru ștergerea lui OCF2 din TIFR. Nici această linie **nu mai este necesară** deoarece OCF2 este şters automat de hardware la intrarea în ISRul asociat. **Mutați restul codului pentru ceas în ISR**

f) Afișați ceasul începând din linia 2 coloana 1 și mutați luminozitatea în linia 2 coloana 11.

g) Terminați rutina de întrerupere:

```
}//end ISR
```

Testaţi! Dacă pe durata afișării mesajului **Busy…** ceasul se evoluează corect, chemați profesorul pentru validare. Dacă aţi ajuns aici aveți între 1 și 4 puncte.

Pasul 6: Reentranţă

Testați repetat, repede, 005. Una din încercări ar trebui să ducă la o situație asemănătoare cu cea din figura următoare:

CL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	5					B	u				P	Y	T	H
2	1	0	:	1	5	:	0	7	s	y	.	.	.	0	%	

Mesajul **Busy...** apare scris eronat, o parte pe linia 1 și o parte pe linia 2, după ceas.

Dacă nu apare comportamentul descris mai sus, modificați modalitatea de afișarea a mesajului „Busy…”: în loc de `putsLCD` folosiți secvența:

```
char msg_busy[] = "Busy... ";
....
for(i=0; msg_busy[i] != 0; i++){
      putchLCD(msg_busy[i]);
      wait(50000);
}
```

Astfel afișarea eronată va apare sigur.

De ce apare afișarea eronată? Afișarea eronată are legătură cu noțiunea de reentranță. Dacă nu știți ce este reentranța citiți explicația următoare.

Fie o funcție numită *foo*. Pentru a înţelege ce este o funcție reentrantă vom considera două scenarii:

Scenariul 1:

1. Din main sau din altă funcție se execută un apel către *foo*.
2. Execuția lui *foo* se termina și se revine în *main* sau în funcția din care a fost apelat *foo*.

Scenariul 2:

3. Din *main* sau din altă funcție se execută primul apel către *foo*.
4. Pe durata primei execuții a lui *foo* apare întrerupere și se intră în rutina de întrerupere.
5. În rutina de întrerupere se execută al doilea apel către *foo* şi *foo* se execută.
6. Se revine din rutina de întrerupere şi se continuă prima execuție a lui *foo*. Şi această execuție se termină. Se revine în *main* sau în funcția din care a fost apelat *foo*.

Dacă prima execuția a lui *foo* din scenariul 2 este identică cu execuția lui *foo* din scenariul 1, atunci spunem că *foo* este reentrantă. Adică o funcție întreruptă de ea însăși se execută corect!

După ce face modificările, testați. De ce apare afișarea eronată?

Dacă apare funcționarea eronată, chemați profesorul pentru validare. Ar fi extraordinar dacă ați ști de ce uneori apare afișarea eronată. Dacă ați ajuns aici aveţi 1 punct.