LABORATOR 11 – întreruperi

Dacă este cazul, terminați implementarea pasului 3 – „**c Pitagoreic”** – din laboratorul 10. Testați cu numerele 005, 006, 998 și 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 este ignorată. Toate acestea se întâmplă deoarece pentru valori mari ale lui *c* timpul de execuție a buclei principale devine foarte mare, de ordinul secundelor.

Soluția 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 timerul trebuie să fie suficient de mare 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, adică la 40 ms.

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.

Desfășurarea 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 crea** proiectul cu numele **int.**

b) **Copiați pwm.c**, **IOfn.c** și **defs.h** din folderul **pwm** în folderul **int**.

c) **Adăugați** la proiect fișierele **pwm.c**, **IOfn.c** și **defs.h** din folderul **pwm.**

d) **Eliminați** din proiect fișierul **main.c**

e) **Redenumiți fișierul pwm.c** ca **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.

Alternativ, puteți folosi funcția `sei()` care este echivalentă cu `setbit(…);`

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** şi **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;`~~

și adăugați-o după testarea lui ‘C’ și ‘D’:

```
if(kbhit  && (kbcode == 'C' || kbcode == 'D')){
      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. Mutați** `loop_cnt++` de la sfârșitul lui *main* la sfârșitul ISR.

   Atenţie: 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){
      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 pe durata calculelor apăsăm C sau D 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. După mutarea codului luminozitatea și afișarea luminozității trebuie să se facă imediat ce ați apăsat ‚C' sau ‚D'.

Testați! Dacă *c* pitagoreic, tastatura și luminozitatea funcționează cum a fost descris anterior, chemați profesorul pentru validare.