LABORATOR 2 – Prima aplicaţie

Implementările practice la laboratorul disciplinei „Proiectarea sistemelor cu microprocesor” se fac folosind simulatorul **simulIDE** şi microcontrolerul **ATmega16A**. Datele de catalog pentru microcontroler se afla pe pagina cursului, în secțiunea **ATMega16A Data Sheet**.

Chestiuni teoretice:
Dezvoltarea aplicațiilor pentru microcontrolere cu componente reale.

Dezvoltarea aplicaţiilor pe PC presupune o succesiune de faze. Acestea sunt:

1. **Editarea surselor.**
2. **Compilarea acestora**
3. **Editarea legăturilor**
4. **Depanarea**
5. **Obţinerea fişierului executabil (care funcţionează corect).**

Toate aceste programe au o interfaţă grafică comună care se numește **IDE** (integrated development environment - ***mediu integrat de dezvoltare***). Un exemplu de IDE este **Visual Sudio** de la Microsoft. Toate programele care constituie IDE rulează pe aceeaşi maşină (calculator).

Dezvoltarea aplicaţiilor pentru microcontrolere sau microprocesoare cu forţă redusă de calcul nu se poate face în acest fel deoarece forţa de calcul a acestor circuite este mică sau foarte mică. De aceea dezvoltarea aplicaţiilor care vor rula pe microcontrolere presupune folosirea a trei componente:

- un calculator puternic de tip **PC**,
- un montaj hardware – **target board** (montajul țintă) – ce conţine microcontrolerului (microcontroler se va abrevia µC) plus hardware-ul necesar aplicaţiei ce se dezvoltă: LED-uri, afișoare 7 segmente, LCD, minitastauri, etc.
- un pachet de programe tip **IDE** care rulează distribuit: o parte pe PC şi o altă parte pe µC.

Ansamblul celor trei componente funcţionează după cum urmează:

1. **Se obţine codul executabil pentru µC folosind un calculator puternic (tip PC).** Calculatorul pe care se face dezvoltarea aplicaţiei se numeşte calculator gazdă sau maşina gazdă (**host** machine). Acesta este un calculator puternic, de obicei un PC. Partea de IDE care rulează pe maşina gazdă conţine editor de texte, asamblor, compilator (de obicei de C), linkeditor și interfață grafică pentru rularea comodă a acestor programe. În urma compilării și linkeditării se obține **cod maşină** pentru µC. Evident, acest cod ce nu poate fi executat pe maşina gazdă. Modul de operare prezentat mai sus se numește cross-compilare.
2. **Codul maşină rezultat la pasul anterior se transferă pe microcontroler.** În funcţie de tipul memoriei în care va fi rezident codului maşină există două variante majore: ROM sau RAM. În continuare se va prezenta numai varianta ROM deoarece µC de tip **ATmega** şi implicit montajele construite pe baza acestora aparţin acestei categorii. În varianta ROM µC dispune de resurse hardware dedicate pentru:
    - comunicaţia cu maşina gazdă.

- înscrierea codului maşină generat la punctul 1.
- depanare (debug). Depanarea înseamnă breakpoint, watches, step-by-step, etc.

Comunicaţia între maşina gazdă şi µC se face prin intermediul unui hardware special. În cazul anumitor µC AVR o variantă a acestui hardware se numeşte **ATMEL ICE**. ATMEL ICE se conectează cu gazda prin intermediul unei interfețe USB și cu microcontrolerul AVR prin interfața JTAG, ca în figura 1:

figura 1

3. **Se depanează programul de aplicaţie**. Pentru depanare mediul de dezvoltare este prevăzut cu depanator simbolic.
4. După ce programul de aplicaţie funcţionează corect pe montajul ce conține µC, acesta **se deconectează** de maşina gazda şi va funcţiona de sine stătător, independent.

Dezvoltarea aplicațiilor pentru microcontrolere în mediu simulat.

In cazul laboratorului online dezvoltarea aplicațiilor se desfășură asemănător cu cea din mediul real:

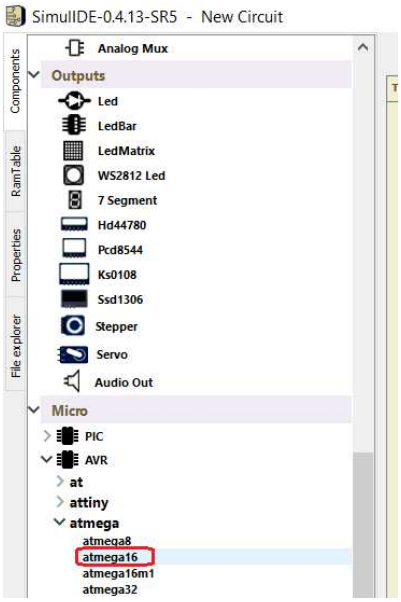
1. La fel ca în cazul anterior, **se obține codul executabil pentru µC folosind un calculator puternic (tip PC).** În urma compilării și linkeditării se obține **cod mașină** pentru µC. Codul mașină este stocat într-un fișier cu extensia **.hex**.
2. **Codul maşină (.hex) rezultat la pasul anterior se transferă pe microcontrolerul din simulIDE.** Ulterior, in acest laborator se va explica cum.
3. **Se depanează programul de aplicație**. Depanarea in simulIDE este dificilă. Se face prin oprirea programului și inspectarea memoriei, prin mesaje transmise serial sau prin mesaje afișate pe LED-uri.

Scopul lucrării.

- **Se va realiza un montaj** bazat pe ATmega16 la care se va conecta un LED. LED-ul se va conecta la bitul 0 al portului A.
- **Se va scrie programul C care face să clipească** acest LED aproximativ o dată pe secundă.

Pasul 1: Crearea montajului in simulIDE

Pe planșa de desenare adăugați componenta atmega16 din categoria micro. Schema electrică se va crea împreună cu profesorul. In final schema trebuie să arate ca in figura 2. Led-urile trebuie sa fie **verzi**. Salvați schema cu numele **blink_sch**.



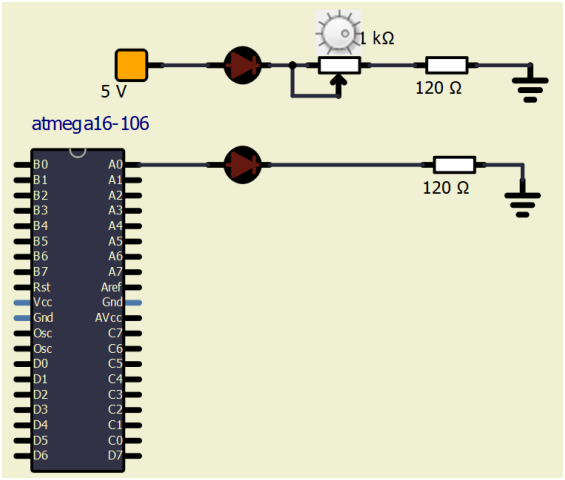


figura 2

Pasul 2: Instalarea Microchip Studio for AVR and SAM Devices

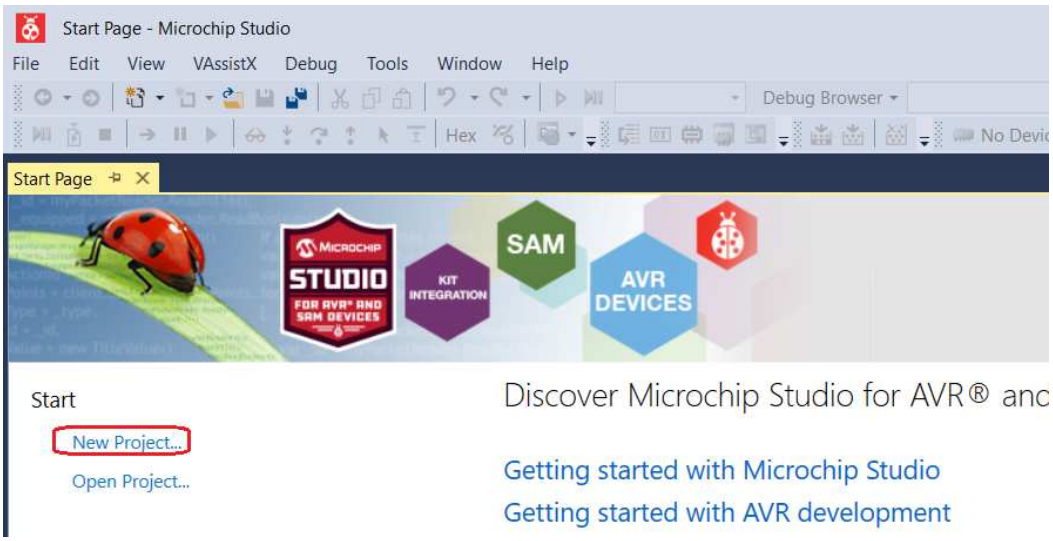
Instalați împreuna cu profesorul Microchip Studio for AVR and SAM Devices

Pasul 3: Crearea proiectului

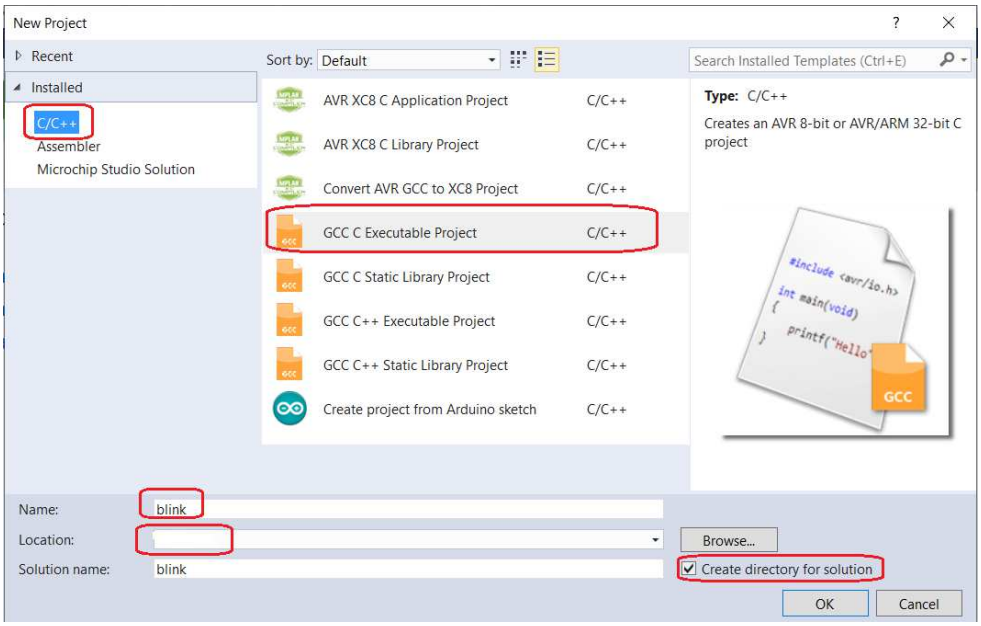
1. Se lansează în execuție **Microchip Studio**:

Va apare pagina de start.

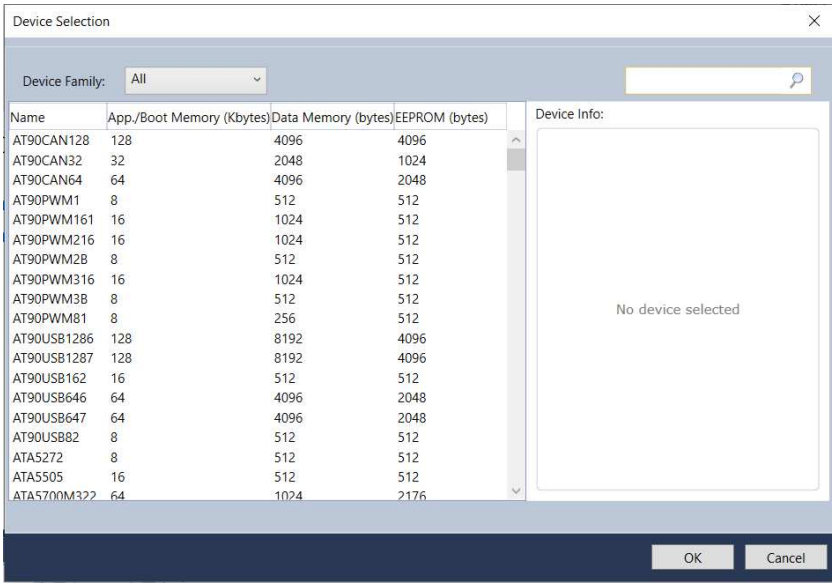
2. Din pagina de start executați **New Project...**



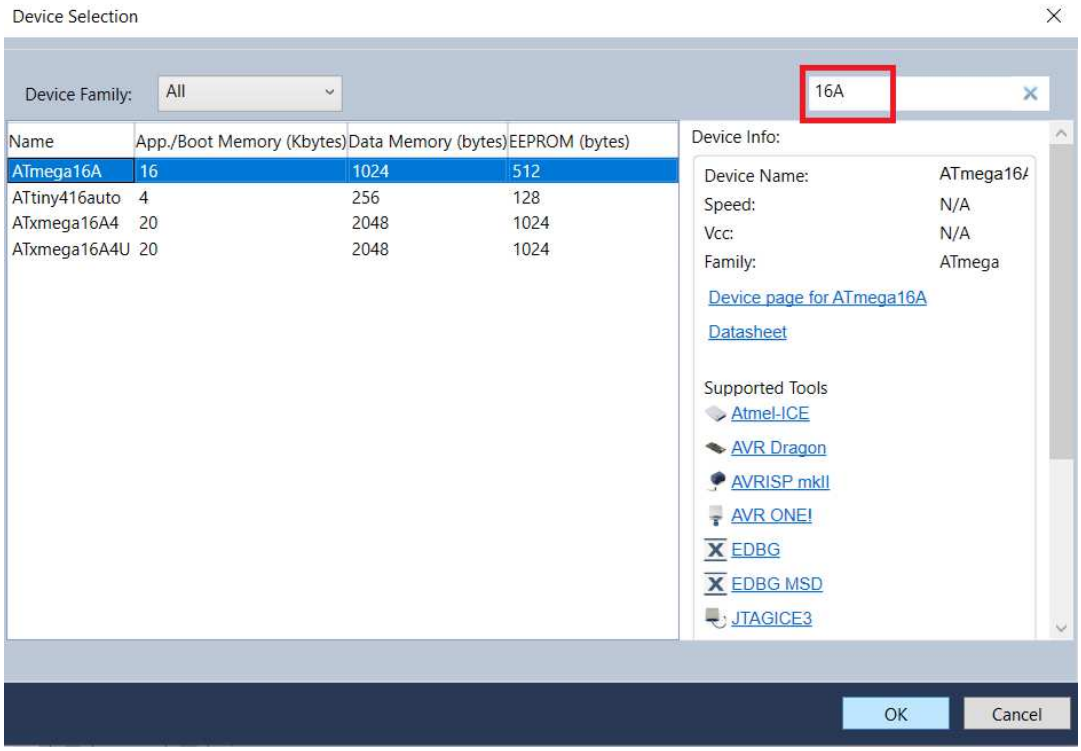
3. În fereastra **New Project** alegeți pentru proiect categoria **C/C++**, tipul „**GCC C Executable Project**”, bifați checkbox-ul **Create directory for solution** ca în figura de mai jos, stabiliți folderul proiectului (unde doriți) și numele proiectului. În această lucrare de laborator numele proiectului va fi **blink**.



În final apăsați butonul **OK**. Va apare fereastra **Device Selection**:

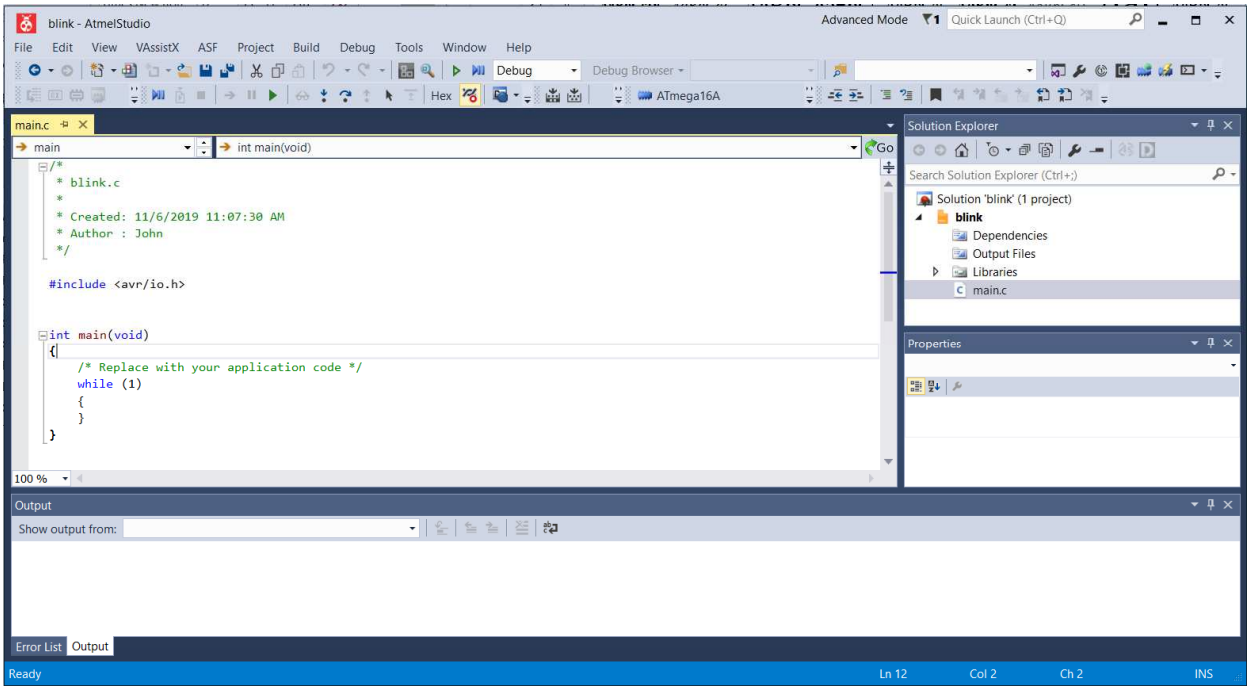


4. În fereastra **Device Selection** în câmpul de căutare introduceți textul 16A. Ca urmare în stânga vor apare toate componentele care conțin în nume textul „**16A**”



Selectați ATmega16A și apoi apăsați OK.

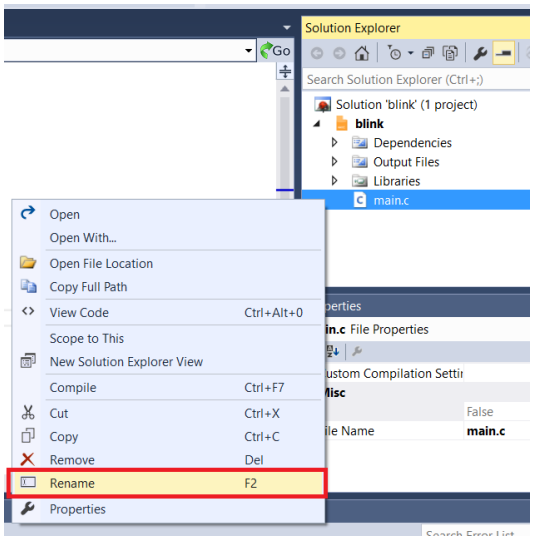
6. Ca urmare se creează proiectul și deschide fereastra principală a IDE-ului Microchip Studio:



7. Structura este similara cu structura multor IDE-uri. În secțiunea „Solution Explorer" apare structura proiectului **blink**. Se observă că în proiect a fost adăugată automat sursa **main.c** și aceasta a fost deschisă în editare în partea stângă.

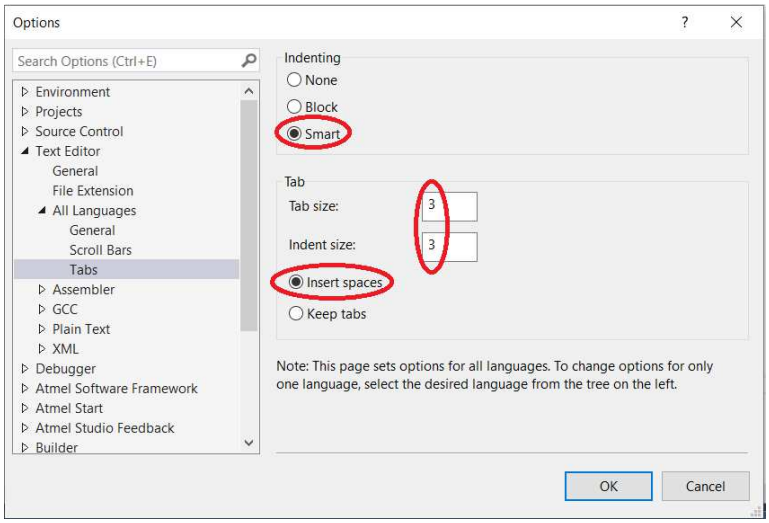
   Structura unei aplicații micro este prezentată în **prelegerea 2 Porturi, capitolul 3. Citiți acest capitol!** Se observă că structura lui main.c are structura tipică a unei aplicații micro, adică are în componență o buclă infinită.

8. Fișierul creat și adăugat automat la proiect se numește main.c, care este numele implicit pentru toate proiectele. În momentul în care avem mai multe proiecte, toate vor avea în componența fișierul main.c, ceea ce poate crea confuzii. Din acest motiv vom redenumi main.c ca blink.c. Redenumirea se face în Atmel Studio cu F2 sau cu click stânga pe fișierul selectat, ca în figura următoare:



Pasul 3: Modificarea sursei blink.c

Obligatoriu: Pentru o editare mai uşoară a sursei, în fereastra **Tools → Options** faceți următoarele setări:



Atenție: codul sursă se scrie cu indentare. Profesorul verifică indentarea și în caz că acesta este incorectă, va trebui refăcută. Refacerea consumă timp!

Atenție: în C numărul de biţi pe care se reprezintă o variabilă de tip **int** depinde de compilator. În cazul compilatorului GCC din AVR Studio **variabilele de tip int se reprezintă pe 16 biţi** iar cele de tip **long int** pe 32 de biţi.

Codul care face să clipească LED-ul conectat pe bitul 0 al portului A este:

```
#include  <avr/io.h>
#define F_CPU 8000000UL
#define __DELAY_BACKWARD_COMPATIBLE__
#include <util/delay.h>

//period
#define T 1000UL

//duty factor
#define DF 50UL

#define THIGH (T*DF/100)
#define TLOW  (T*(100-DF)/100)

int main(){
   DDRA=0xff;

   while(1){
      PORTA=1;  //turn on LED
      _delay_ms(THIGH);

      PORTA=0;  //turn off LED
      _delay_ms(TLOW);
   }
}
```

Ștergeți tot conținutul fișierului blink.c și apoi copiați în acesta codul de mai sus.

PORTA şi DDRA sunt porturi ale procesorului din microcontroler.

```
DDRA=0xFF;
```

face ca portul A să fie un port de tip buffer (vezi prelegere 2, figura 1). Adresele porturilor PORTA și DDRA se găsesc în documentația ATmega16 la pagina 334. Porturile vor fi tratate pe larg la curs.

Pentru a crește claritatea programelor, adresele porturilor au primit nume simbolice. Astfel, în loc de 0x1B putem folosi PORTA. Atribuirea de nume simbolice este făcută în fișierul **io.h**, motiv pentru care acest fişier a fost inclus în codul sursă.

Clipirea LED-ului se bazează pe funcția `_delay_ms(double ms_val)`. Durata de execuție a acestei funcții este de `ms_val` **milisecunde**.

Aceasta funcție se implementează prin intermediul unei bucle `for` cu corp vid:

```
for(long int i=initial_val; i>0; i--){}
```

Timpul de execuție al for-ului depinde de `initial_val` și de viteaza procesorului. Viteaza procesorului este dată de frecvența semnalului de ceas al procesorului. Pentru ca funcția să poată calcula `initial_val` se specifică frecvența semnalului de ceas al procesorului:

```
#define F_CPU 8000000UL
```

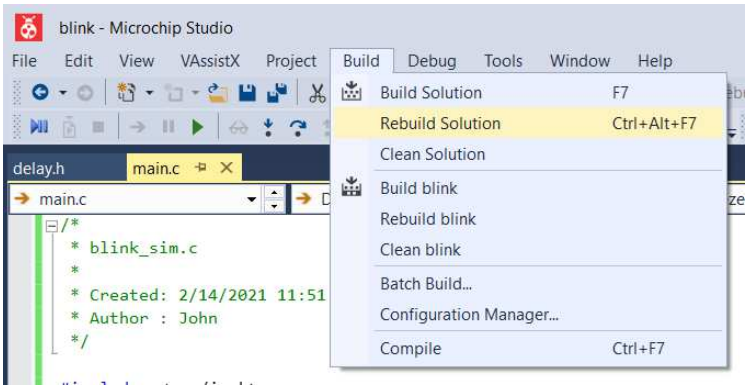
Apoi se include headerul unde este definit prototipul funcției `_delay_ms(double ms_val)`:

```
#include <util/delay.h>
```

Este obligatoriu să scriem întâi `#define`-ul și apoi `#include`-ul.

În rest ar trebui să fie clar ce face programul blink. Dacă există ceva ce nu aţi înţeles sau ceva neclar, **chemaţi profesorul!**

În continuare compilați și link-editaţi selectând Rebuild Solution, ca în figura următoare:



Pasul 4: Execuția programului blink

În simulIDE setați frecvența ceasului microcontrolerului la 8MHz. Această setare se face modificând proprietatea „MHz" a modulului atmeg16, ca în figura 3 :

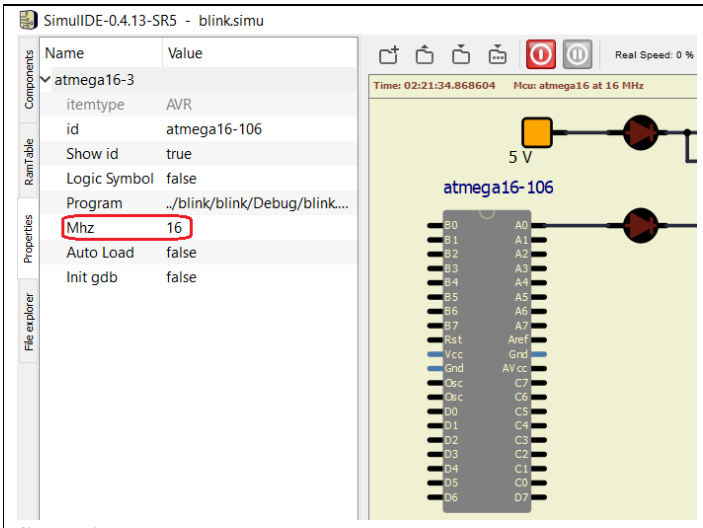


figura 3

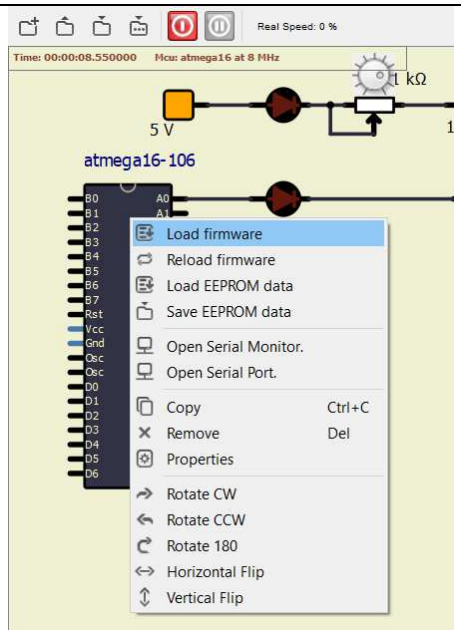


figura 4

Codul mașină obținut după Build va fi programat în atmega16 din simulIDE. Accesați proprietățile circuitului atmega16 cu clic dreapta și selectați „**Load firmware**", ca în figura 4.

Selectați fișierul **blink.hex** din folderul **calea_către foderul proiectului\blink\blink\debug**

După ce ați specificat fișierul .hex prima dată, ulterior puteți reîncărca acest fișier cu „**Reload firmware**". Reîncărcarea se face ori de câte ori regenerați fișierul .hex în urma unei modificări a sursei .c urmată de rebuild.

Executați 4 experimente:

1. Modificaţi codul pentru ca LED-ul să clipească cu un factor de umplere **DF=25** (DF=Duty Factor). Observaţi efectul.
2. Modificaţi codul pentru ca LED-ul să clipească cu un factor de umplere **DF=65**. Observaţi efectul.

3. Modificaţi codul pentru a implementa clipirea cu frecvenţă mare. În programul blink.c micșorați perioada **T la 10**. Acum un ciclu aprins-stins va dura 10 ms. Rulați programul pentru **DF=25**.
4. La fel ca la pasul 3, perioada trebuie să fie 10. Rulaţi programul pentru **DF=65** (DF=Duty Factor).

Care este efectul P=10 şi DF=25? Dar P=10 şi DF=65? Mai clipește LED-ul? Când simularea funcționează și puteți răspunde la întrebări, atenționați profesorul pentru a prezenta.

Pasul 5: Clipire semi aprins - semi stins (simulare pentru bonus)

Pentru T=1000 şi DF=50 programul anterior face ca LED-ul să clipească la nesfârșit, o dată pe secundă. Din perioada de o secundă jumătate din timp LED-ul este aprins și jumătate de secundă stins.

În continuare se dorește ca LED-ul să clipească la fel ca mai înainte, o data pe secundă, dar în loc să fie stins va fi aprins cu intensitate mică şi în loc să fie aprins la maxim va fi aprins cu intensitate mare. Pentru a obţine intensitate mică şi intensitate mare vom seta T=10. Valoarea lui T nu se va modifica. Pentru a obţine cele două intensităţi se va modifica DF. LED-ul va lumina astfel:

- 0.5 secunde cu DF=25, adică va lumina slab.
- 0.5 secunde cu DF=65, adică va lumina puternic.

Modificaţi codul pentru ca LED-ul să lumineze la nesfârșit după cum s-a precizat mai sus.

Indicaţie de implementare: Definiți o pereche THIGH -TLOW pentru DF=25 și o pereche THIGH -TLOW pentru DF=65:

În `main` folosiți o pereche de variabile thigh – tlow. În secundele pare această pereche va lua valoarea perechii THIGH -TLOW pentru DF=25 iar în secundele impare va lua valoarea THIGH -TLOW pentru DF=65.

Dacă nu vă place aceasta idee, orice altă idee este acceptată.

Când funcționează, atenționați profesorul pentru a prezenta!