Razvojni sistem za rad sa ATMEL mikrokontrolerima

Uputstvo za rukovanje

- napisano za potrebe projekta u okviru takmičenja:

The 2005 International
Future Energy Challenge
Institute of Electrical and Electronics Engineers (IEEE)

Aleksandar Živković
student pete godine na Elektrotehničkom fakultetu
Univerziteta u Beogradu, smer: automatika
aleksandarz@ieee.org

januar 2005

Sadržaj:

Potrebno predznanje

Da bi uspešno mogli obavljati proces pisanja softvera za mikrokontroler i njegovog programiranja potrebno je posedovati sledeća znanja:

1) poznavanje arhitekture mikrokontrolera koji ćemo koristiti
2) poznavanje hardvera u kojem će biti smešten naš mikrokontroler
3) poznavanje programskog jezika u kojem će se pisati kod
4) razvojno okruženje na PC računaru

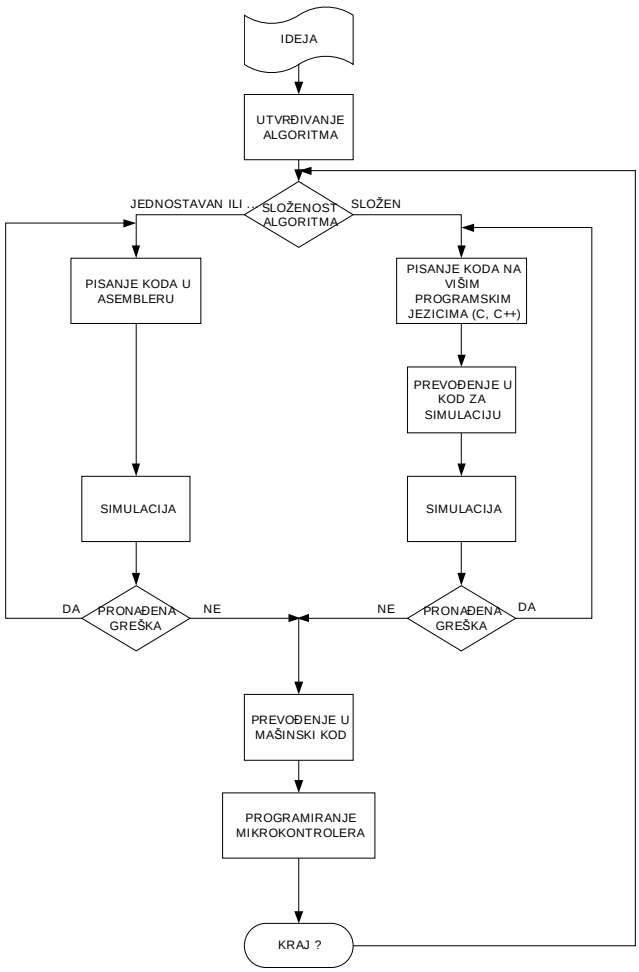
Naredni tekst podrazumeva poznavanje prve tri pomenute tačke i daje kratko uputstvo za korišćenje razvojnog okruženja za programiranje mikrokontrolera.

Uvod

Razvojni sistem koji je pisan ovde, u prvu ruku je namenjen za rad sa mikrokontrolerima proizvođača ATMEL. Glavna osobina ovog razvojnog sistema jeste da je sačinjen od komponenti koji ne potiču iz istog izvora. Iz tog razloga one nisu čvrsto povezane te se kao takve mogu koristiti i pojedinačno, nezavisno jedne od drugih. Sa druge strane, one su dovoljno dobro usklađene sa razlogom da bi rad sa njima u kompletu bio konforan.
Sistem se sastoji od tri celine:

- kompajlera za C kod (*AVR GCC*)
- simulatora napisanog programa za dati mikrokontroler (*AVR Studio 4*)
- programatora i eksperimentalne ploče

Ako posmatramo proces programiranja jednog mikrokontrolera, možemo ići po sledećoj šemi (slika 1):



Slika 1

Uvek polazimo od ideje. Prilikom definisanja algoritma moramo biti svesni zahteva koje naš dizajn diktira i sposobnosti hardvera da udovolji tim zahtevima. Ovo je vrlo bitan deo u procesu. Na osnovu kompleksnosti algoritma treba doneti odluku kojim putem ćemo brže stići do cilja ili koji put će omogućiti dalje usavršavanje projekta. Programiranje u asembleru je ponekad nužno i u slučajevima kad je algoritam kompleksan, ali zahteva

tesnu vezu sa hardverom (ukoliko postoje razna ograničenja u brzini izvršavanja, veličini koda... Takve probleme ne mogu da reše ni najbolji kompajleri, već iskusni embedid programeri). Desna grana je najčešći izbor. Tu nas čeka pisanje koda na *C*-u, prevođenje u kod za simulaciju i sama simulacija. Sve do sad pomenuto se u potpunosti može obaviti bez posedovanja samog mikrokontrolera. Petlja *„PRONALAŽENJE GREŠAKA"* - *„PISANJE KODA"* oduzima najveći deo vremena u projektovanju. Od kvaliteta simulatora zavisi to koliko ćemo biti bliže stvarnosti i to da li će mikrokontroler odmah po prebacivanju koda u njega izvršavati željeni algoritam. Ako simulator više ne prijavljuje grešku, možemo preći na kompajliranje i programiranje. Tek sad možemo stvarno da izmerimo željene signale. Petlja koja se vraća sa oblačića *„KRAJ*" treba da sugeriše da u ovom delu takođe sledi testiranje i izmena koda, ali to nije tema ovog teksta.
U narednim poglavljima biće opisani alati i postupci koji prate tok šeme sa slike 1.

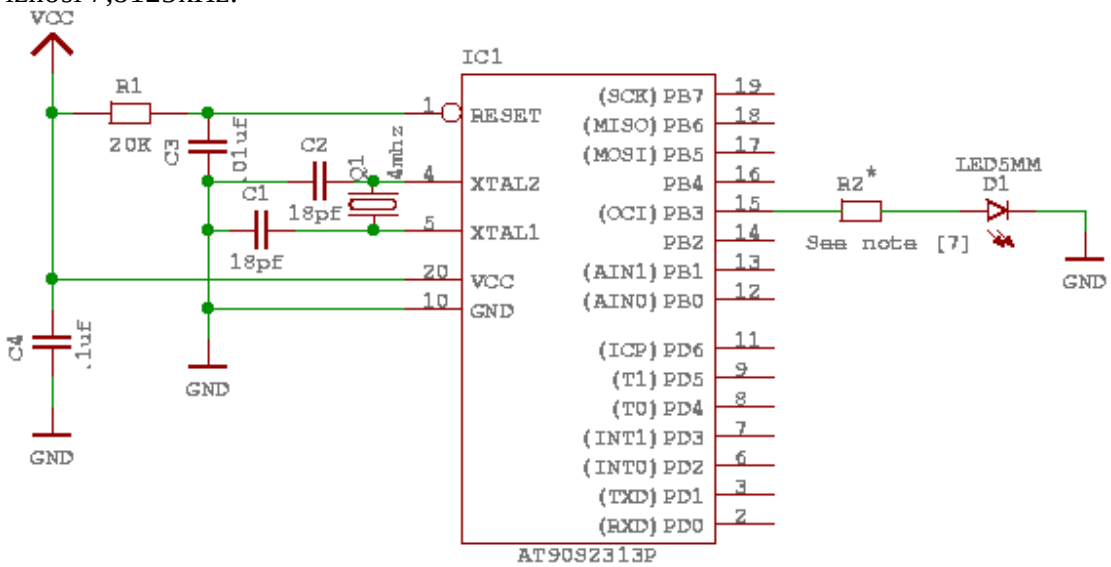
Jednostavan projekat

Ovo poglavlje će pokazati postupak ubacivanja jednostavnog koda u mikrokontroler po sledećem redosledu:

1) hardver
2) otvaranje projekta
3) pisanje *C* koda
4) kompajliranje i provera grešaka
5) priprema za simulaciju
6) simulacija
7) programiranje mikrokontrolera

Hardver (1)

Pre svega treba prikazati gde će biti smešten naš mikrokontroler i šta će raditi. Na eksperimentalnoj ploči potrebno je povezati mikrokontroler sa ostalim komponentama i napajanjem kao na slici 2. Programčić dat u prilogu će kontrolisati paljenje LED indikatora i to PWM modulacijom po zakonu $\sin(\cdot)^2$ sa periodom od jedne sekunde. PWM učestanost zavisi od upotrebljenog kvarcnog oscilatora i za takt na 8MHz ona iznosi 7,8125kHz.

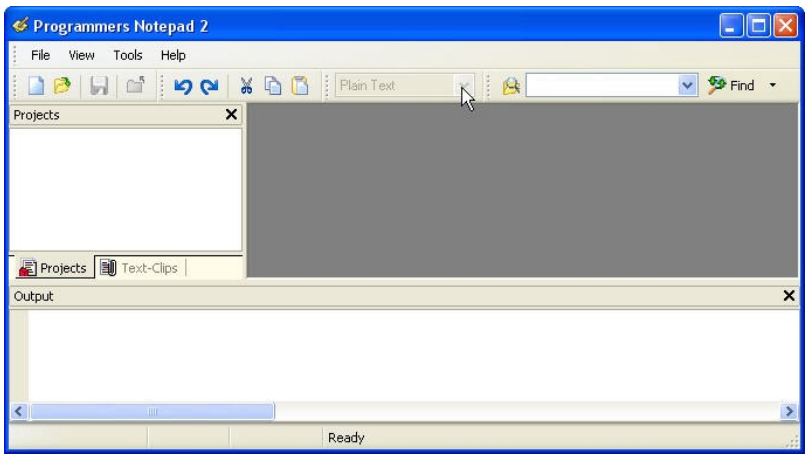


Slika 2

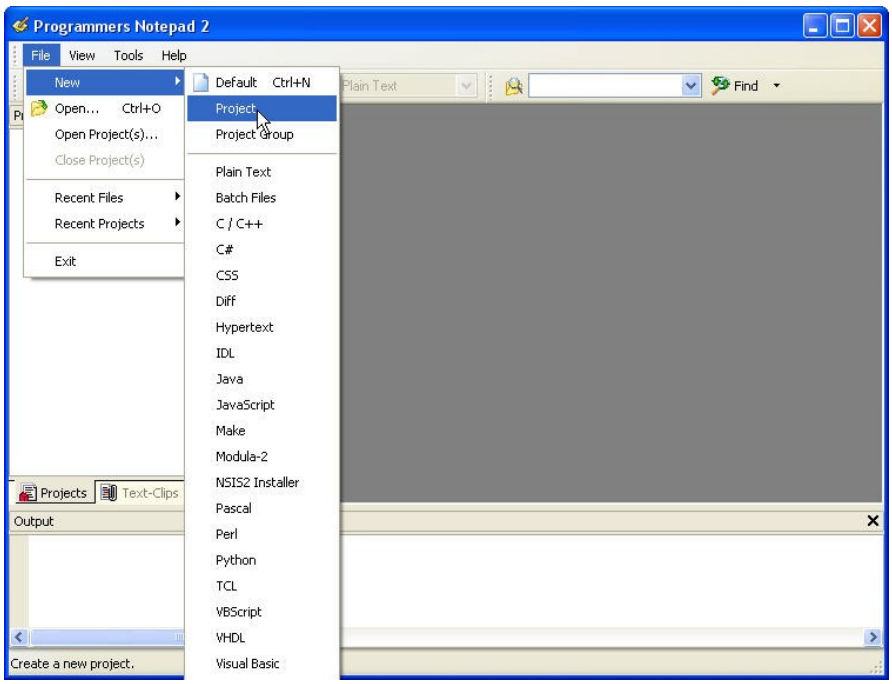
Radno okruženje, pisanje koda i otkrivanje grešaka (2, 3, 4)

Ceo postupak pisanja, kompajliranja i debagovanja *C* koda se obavlja u programu pod nazivom „ *Programmers Notepad [WinAVR]"*. U ovom primeru se podrazumeva da je C kod već napisan i da se nalazi u dokumentu pod nazivom: *uputstvo.c*

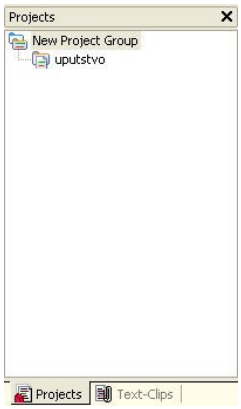
Naredne slike u najkraćem prikazuju kako doći do pozicije kad možemo da kompajliramo i debagujemo *C* kod:



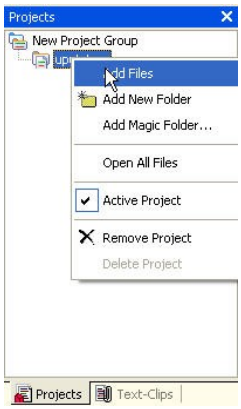
Slika 3. Izgled po otvaranju „*Programmers Notepad [WinAVR]*''



Slika 4. Otvoriti: *File/New/Project* i snimiti projekat



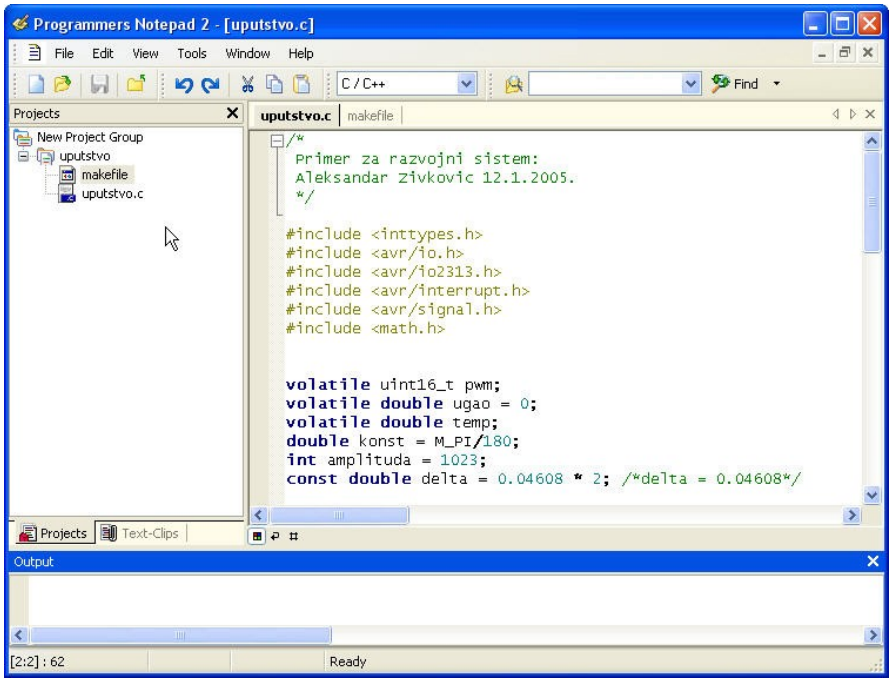
Slika 5 U prozoru *Projects* se pojavljuje snimljeni projekat



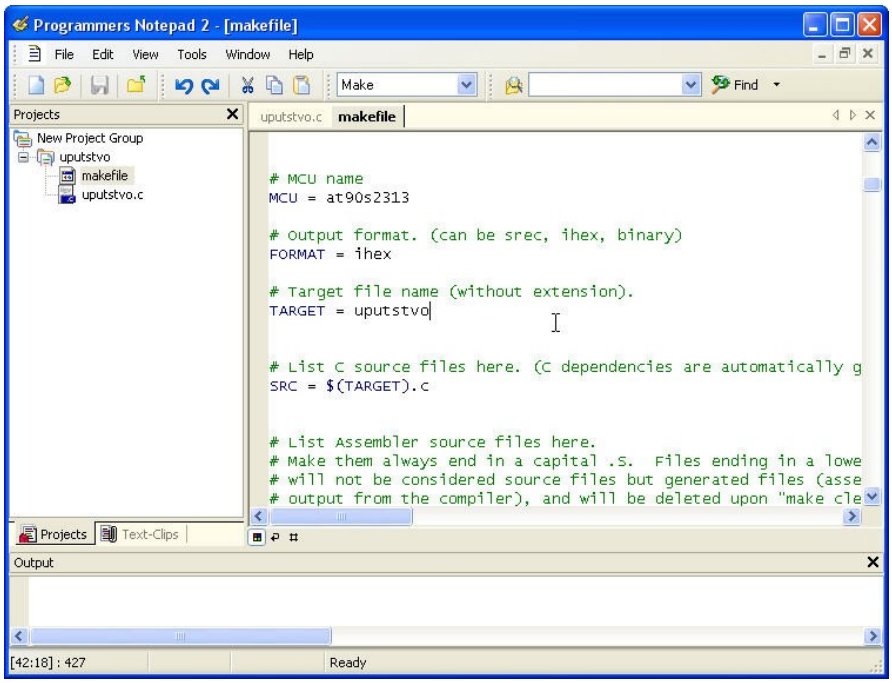
Slika 6 Praznom projektu dodeliti fajl sa C kodom i tzv. *makefile*-om

Slika 7 Dodate fajlove treba otvoriti

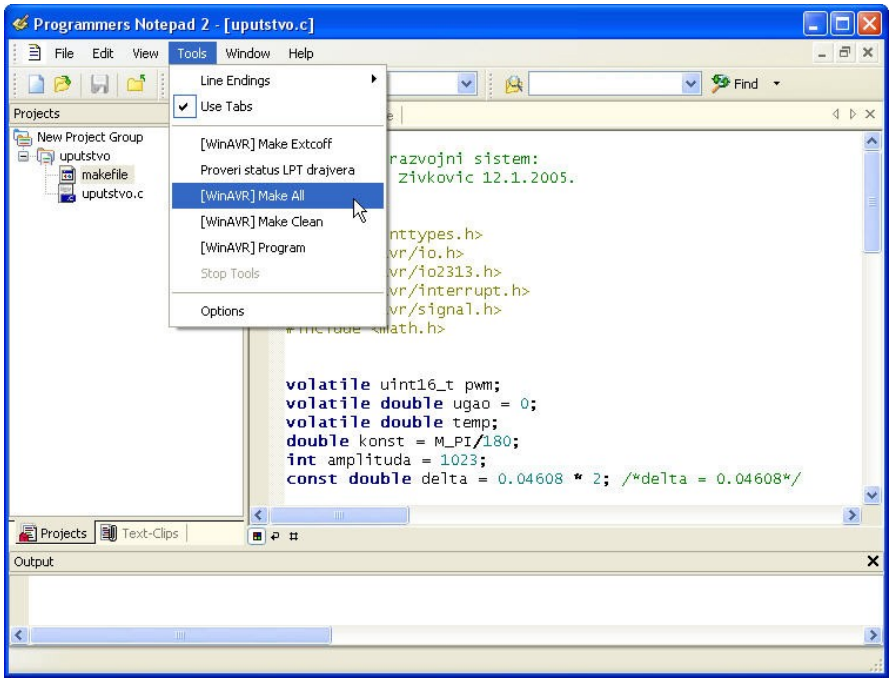
Makefile je skript fajl u kome se definiše šta računar sve treba da uradi. Ime potiče iz *Unix* operativnog sistema. Sva podešavanja vezana za: tip mikrokontrolera, stepen optimizacije kompajlera, vrste programatora itd. će moći ovde da se izmene.



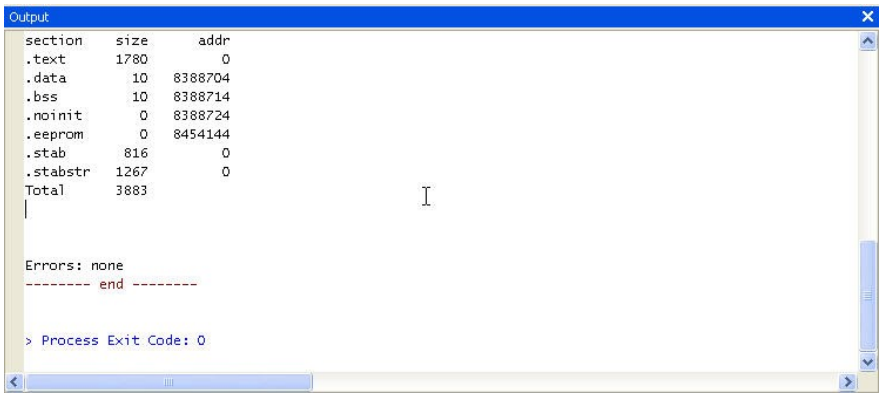
Slika 8 Otvoreni fajlovi: *uputstvo.c* i *makefile*



Slika 9 Jedan deo *makefile*-a (definisanje tipa mikrokontrolera i imena fajla u kome se nalazi *C* kod)

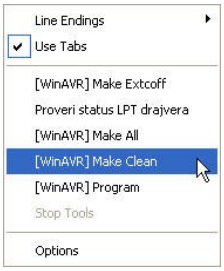


Slika 10 Kada je učitan odgovarajući *C* kod i *makefile* može se preći na kompajliranje. To se čini komandom „*Make All*"



Slika 11 Sve informacije o toku kompajliranja se ispisuju u donjem prozoru „*Output*"

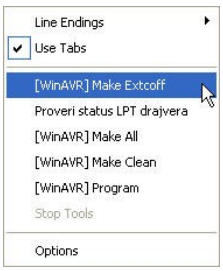
Ovo je bitan deo u celom procesu jer nam kompajler javlja sintaksne greške u programu i eventualne nepravilnosti u opisu *makefile*-a. Za vreme ispravljanja ovih grešaka stalno ćemo se vrteti u krug: ispravi grešku ↔ proveri da li ih ima još (kompajliranje). Za vreme kompajliranja u direktorijumu u kome je snimljen projekat će se generisati mnogo dokumenata koji mogu da se pre novog kompajliranja izbrišu naredbom „*Make Clean*" (slika 12).



Slika 12

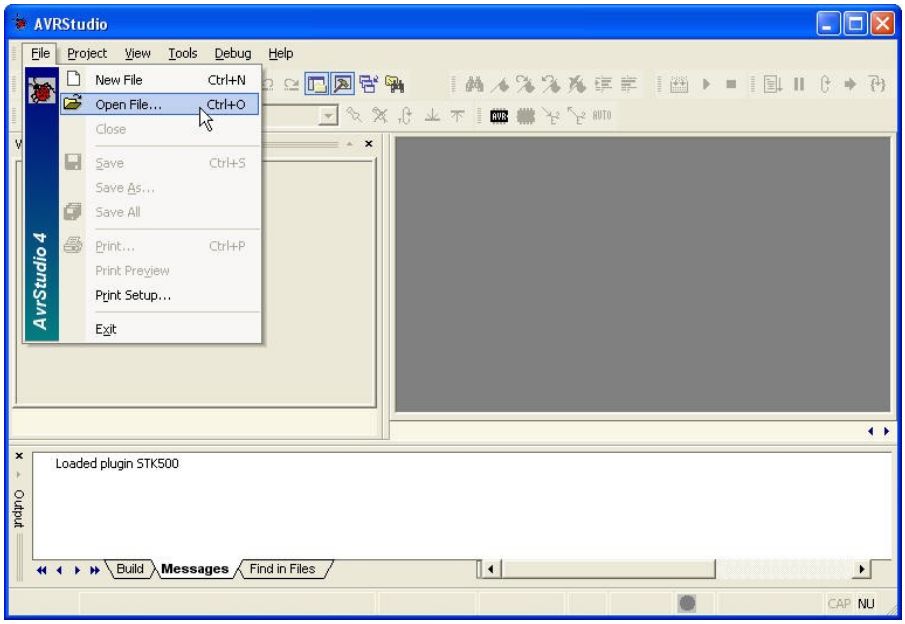
Simulacija: priprema i izvršavanje (5, 6)

Pošto je utvrđeno da ne postoje sintaksne greške i da *C* kod može da se kompajlira bez grešaka može se preći na proces simulacije. Simulacijom dobijate uvid u to šta će se dešavati u mikrokontroleru i kad će se to dešavati (u kom taktu). Simulacija se obavlja pomoću programa pod nazivom *AVR Studio 4*. Ovaj softver je prvobitno bio namenjen za simulaciju koda pisanog u asembleru. Ova opcija i dalje posoji i u tom slučaju nam nije potreban AVR GCC kompajler. Da bi se simulirao kod pisan u *C*-u potrebno ga je prilagoditi. Fajl koji može da generiše AVR GCC kompajler, a koji AVR Studio 4 prepoznaje jeste tzv. *Coff* fajl. Generisanje ovog fajla se u *„Programmers Notepad"* radi izborom opcije *„[WinAVR] Make Extcoff"*. Posle ove operacije u direktorijumu vašeg projekta će se generisati fajl pod nazivom `ime_C_fajla`.*coff*

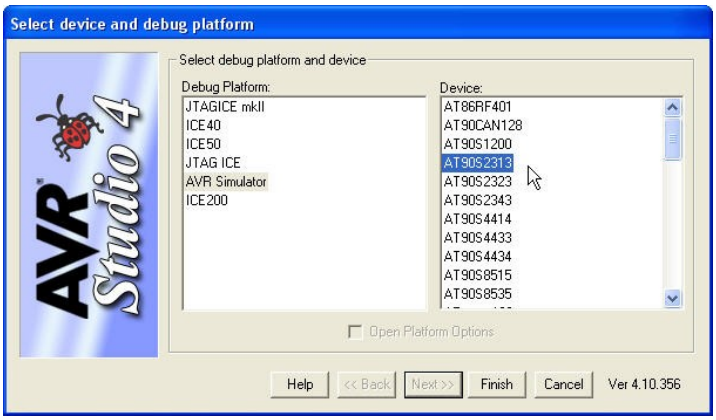


Slika 12 Generisanje *coff* fajla

U našem slučaju potrebno je iz programa AVR Studio 4 otvoriti fajl pod nazivom *uputstvo.cof*. Po učitanom fajlu, potrebno je izabrati opciju *AVR Simulator* i tip mikrokontrolera kao na slici 15.



Slika 13 Početak rada sa AVR Studiom



Slika 14 Izbor simulacije i tipa mikrokontrolera

Tada se učitava radno okruženje i moguće je započeti simulaciju programa. Ukoliko nismo zadovoljni programom potrebno je:

- vratiti se u *„Programmers Notepad"* i izmeniti kod
- kompajlirati kod i proveriti da li ima sintaksnih grešaka
- generisati *coff* fajl
- vratiti se u AVR Studio 4 (za ovo ne treba ponovo učitavati fajl, AVR Studio 4 prepoznaje da je fajl izmenjen i pita vas da li da ga ponovo učita, što malo ubrzava rad u ovoj petlji)

Prethodne četiri stavke predstavljaju drugu, veću petlju u kojoj se treba vrteti kako bi se program što više oslobodio od grešaka.

Prebacivanje koda: PC → μC (7)

Poslednji korak u radu sa opisivanim razvojnim sistemom je prebacivanje napisanog koda sa PC računara u mikrokontroler. To se radi posebnim hardverom koji se naziva programator. U našem razvojnom sistemu koristi se programator povezan na PC preko paralelnog porta (LPT1). Programiranje mikrokontrolera se obavlja u *„Programmers Notepad"*-u tako što se otvori dokument *uputstvo.c* i aktivira programator izborom opcije: *„[WinAVR] Program"*. Ovime se, u stvari poziva program *AVRDude.exe* koji kontroliše programator i preko njega upis i čitanje u mikrokontroler.

Slika 15

Za vreme programiranja se u prozoru *Output* ispisuje detaljan izveštaj o tome šta se dešava. Za naš slučaj izveštaj ima sledeći oblik:

```
> "make.exe" program
avrdude -p at90s2313 -P lpt1     -c sp12     -E novcc -U flash:w:uputstvo.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e9101
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "uputstvo.hex"
avrdude: input file uputstvo.hex auto detected as Intel Hex
avrdude: writing flash (1790 bytes):

Writing | ################################################## | 100% 7.02s

avrdude: 1790 bytes of flash written
avrdude: verifying flash memory against uputstvo.hex:
avrdude: load data flash data from input file uputstvo.hex:
avrdude: input file uputstvo.hex auto detected as Intel Hex
avrdude: input file uputstvo.hex contains 1790 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.72s

avrdude: verifying ...
avrdude: 1790 bytes of flash verified

avrdude done.  Thank you.


> Process Exit Code: 0
```

Sve radnje koje su opisane u ovom izveštaju mogu da se definišu u *makefile*-u. Ovde se na primer posle, učitavanja koda u mikrokontroler vrši čitanje upravo ubačenog koda i poredi sa kodom na PC-ju. Tako se vrši verifikacija upisa u mikrokontroler.
Ovime je završen proces programiranja jednog ATMEL mikrokontrolera ☺.

Literatura

Prethodni tekst obuhvata veoma veliku oblast i zbog njegovog malog obima nemoguće je ukazati na detalje koji mogu da se pojave u toku rada sa mikrokontrolerom. Zbog toga se čitalac upućuje na bogatu literaturu i podršku u vidu *Windows Help*-a koju poseduje na računaru gde je instaliran ovakav razvojni sistem:

- AVR GCC dokumentacija: c:/WinAVR/doc/avr-libc/
- AVR Dude – softwer za programiranje: c:/WinAVR/doc/avrdude-4.4.0/
- AVR Studio 4 - simulacija: Glavni meni >> Help >> AVR Studio User Guide
- Korisna adresa: http://www.avrfreaks.net - odlično organizovan sajt gde možete naći mnoštvo informacija o ovoj tematici i gde ćete na forumu vrlo brzo dobiti odgovor na vaše pitanje

Dodatak – primer program

Uputstvo.c

```
/*
 Primer za razvojni sistem:
 Aleksandar Zivkovic 1.2.2005.
 */

#include <inttypes.h>
#include <avr/io.h>
#include <avr/io2313.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <math.h>


volatile uint16_t pwm;
volatile double ugao = 0;
volatile double temp;
double konst = M_PI/180;
int amplituda = 1023;
const double delta = 0.04608 ; /*delta = 0.04608*/

SIGNAL (SIG_OVERFLOW1)
{
    ugao = ugao + delta;
      if (ugao>180) ugao = ugao - 180;
}

void
ioinit (void)
{
    /* tmr1 je 10-bit PWM */
    TCCR1A = _BV (PWM10) | _BV (PWM11) | _BV (COM1A1);

    /* tmr1 radi na punom taktu mikrokontrolera */
```

```
    TCCR1B = _BV (CS10);

    /* postavi vrednost PWM-a na 0 */
    OCR1 = 0;

    /* omoguci OC1 i PB2 kao izlaz */
    DDRB = _BV (PB3);

    timer_enable_int (_BV (TOIE1));

    /* omoguci interapte */
    sei ();
}

int
main (void)
{
    ioinit ();

    /* for-petlja se vrti zauvek, a interapti obavljaju ostalo*/

    for (;;){
                temp = ugao*konst;
                pwm = (int) (amplituda*sin(temp)*sin(temp));
                OCR1 = pwm;
                }
    return (0);
}
```