Poglavlje 12

AVR-GCC razvojni alati

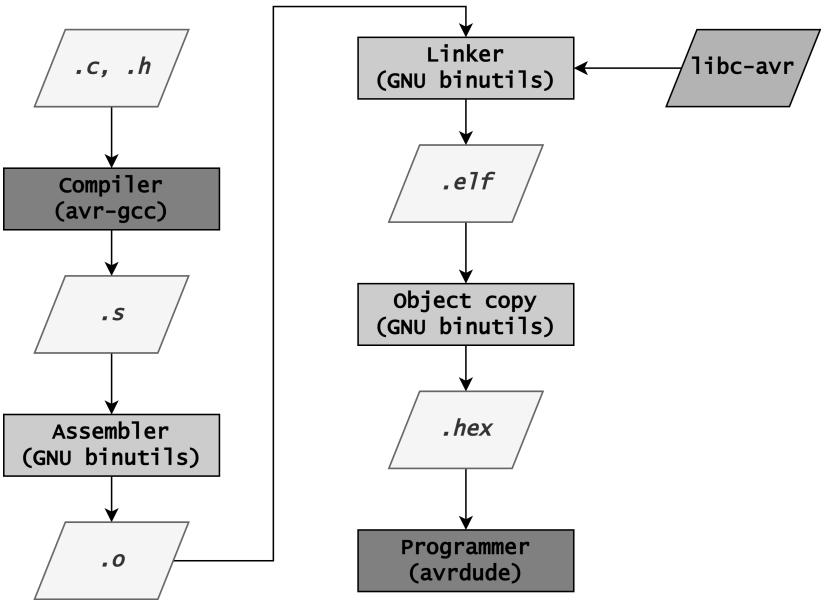
12.1 Uvod

U svim do sada viđenim primerima za mikrokontroler ATmega328P, razvoj aplikacije baziran je na **AVR-GCC** (eng. *AVR-GNU Compiler Collection*) razvojnim alatima. Ovi alati su besplatni za korišćenje, u skladu sa modelom otvorenog koda (eng. *open-source*) i sačinjavaju tzv. *AVR-GCC toolchain*, u koji spadaju:

- *gcc-avr*: C/C++ prevodilac (kompajler, eng. *compiler*) za 8-bitne AVR mikrokontrolere
- *binutils*: kolekcija alata koja uključuje asembler (eng. *assembler*), linker (eng. *linker*) i druge alate za manipulaciju generisanim binarnim kodom
- *libc-avr*: podskup standardnih C biblioteka sa dodatnim funkcionalnostima specifičnim za AVR mikrokontrolere
- *gdb*: alat za otklanjanje grešaka (debager, eng. *debugger*)
- *avrdude*: alat za programiranje sadržaja FLASH/EEPROM memorije AVR mikrokontrolera

GNU Compiler Colection (GCC) familija kompajlera je specifična u odnosu na ostale prevodioce po tome što prevodi kod sa jezika višeg nivoa (C ili C++) u asemblerski kod za datu platformu. Nakon prevođenja, generisani asemblerski kod se asemblira, čime se dobija tzv. objektni kod. Objektni kod predstavlja mašinsku verziju koda koja sadrži i relokacijske informacije, koje koristi linker. Zadatak linkera je da na odgovarajući način poveže fragmente mašinskog koda koji može biti raspoređen u više objektnih datoteka i da kreira izvršnu verziju programa. Detaljno objašnjenje svakog alata ponaosob je dato u nastavku poglavlja.

Ilustracija procesa kompajliranja korišćenjem AVR-GCC alata prikazana je na slici 12.1.



Slika 12.1: Proces kompajliranja

12.2 Kompajler

Kompajliranje predstavlja proces prevođenja programa pisanog u nekom višem programskom jeziku u jezik nižeg nivoa apstrakcije, asemblerski jezik (eng. *assembly*). Rezultat ovog procesa predstavlja asemblerski kod (*.s*), odnosno, kod pisan na nivou instrukcija.

Budući da svaka platforma radi sa jedinstvenim mašinskim jezikom, uvek je potrebno koristiti odgovarajući kompajler za datu platformu. U toku ovog kursa, korišćen je *avr-gcc* kompajler iz *avr-gcc* skupa razvojnih alata.

Sam postupak kompajliranja (ali i dalji postupci, sve do generisanja izvršnih datoteka), prilikom rada sa *embedded*, platformama se skoro uvek izvršava na *host* računaru. Retke su situacije u kojima se kompajliranje izvršava na samoj platformi. Ovakav princip generisanja izvršne datoteke, u okviru kog se kompajliranje izvršava na jednoj, a sam kod koji se produkuje izvršava na drugoj platformi se naziva unakrsno kompajliranje (eng. *cross compile*). Kompajler koji izvršava dati postupak generisanja koda za drugu platformu se naziva unakrsni kompajler (eng. *cross compiler*).

Postupak prevođenja koda pisanog u programskom jeziku C u kod pisan u asembleru se, korišćenjem avr-gcc kompajlera, obavlja pomoću sledeće komande:

```
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -S -o main.s
    main.c
```

Osnovni delovi navedene komande su:

1. `avr-gcc` – predstavlja naziv alata koji se poziva;
2. `-Os -DF_CPU = 16000000UL -mmcu = atmega328p` – predstavljaju dodatne parametre (*napomena*: svi parametri počinju sa znakom "-") na osnovu kojih se specificiraju informacije o platformi za koju je potrebno generisati asemblerski kod. To su nivo optimizacije kompajlera (`-Os` – uključena optimizacija), frekvencija takta na kojoj mikrokontroler radi (`-DF_CPU = 16000000UL` – frekvencija od 16000000Hz) i tip mikrokontrolera za koji je kod pisan (`-mmcu = atmega328p` – mikrokontroler ATmega328P);
3. `-S` – predstavlja vrstu prevođenja koju je potrebno uraditi (prevođenje u asemblerski kod);
4. `-o`(utput) – označava da nakon njega sledi naziv izlazne datoteke (u ovom slučaju to je *main.s*);
5. `main.s` – predstavlja izlaznu datoteku u koju je potrebno smestiti rezultat prevođenja;
6. `main.c` – predstavlja ulaznu datoteku čije se prevođenje vrši.

12.3 Asembler

Asembler vrši prevođenje asemblerskog jezika (eng. *assembly*) u jedan oblik mašinskog jezika. Prema tome, asembler je u svojoj osnovi takođe kompajler, s tim da su prevođenja koja on vrši, značajno jednostavnija. Rezultat ovog procesa predstavlja objektna datoteka (*.o*). Objektna datoteka zapravo predstavlja specijalno formatiranu binarnu datoteku koja se sastoji od seta instrukcija i drugih podataka koji predstavljaju rezultat prethodnog prevođenja. Sama datoteka nije izvršna, jer ne sadrži sve potrebne informacije za izvršavanje samog koda. Ove informacije biće poznate naknadno, u fazi povezivanja. Takođe, ovako dobijene objektne datoteke sadrže i sve informacije neophodne za eventualnu simulaciju programa.

Postupak prevođenja asemblerske datoteke u objektnu, korišćenjem *avr-gcc* kompajlera obavlja se u sledećoj liniji:

```
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o main.o
    main.s
```

Slično kao i prilikom generisanja asemblerskog koda, komanda se sastoji od poziva odgovarajućeg alata (avr-gcc) i podešavanja odgovarajućih platformskih parametara (nivoa optimizacije, frekvencije takta i tipa mikrokontrolera). Razlika u odnosu na prethodni korak jeste parametar `-c`. Na osnovu ovog parametra, daje se do znanja kompajleru da je potrebno izvršiti prevođenje ulaznog asemblerskog koda (*main.s*) u izlaznu objektnu datoteku (*main.o*).

Važno je napomenuti i to, da se navedeni postupci kompajliranja i asembliranja izvornog koda mogu objediniti, korišćenjem samo jedne komande, date u nastavku.

```
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o main.o
    main.c
```

12.4 Linker

Nakon kompajliranja i asembliranja izvornih datoteka na prethodno opisan način, potrebno je dobijene objektne datoteke povezati u jednu. Kako bi se ovo izvršilo, koristi se poseban alat – linker (eng. *linker*). Zadatak linkera je da spoji objektne datoteke i da razreši sve do tada nerazrešene simbole (promenljive, funkcije i dr. koje nisu definisane unutar objektne datoteke, a koriste se u okviru nje). Kao ulaz linkera prosleđuju se imena objektnih datoteka koje je potrebno povezati, dok se kao izlaz dobija nova objektna datoteka koja u sebi sadrži sav kod i sve promenljive iz pojedinačnih objektnih datoteka.

Prilikom spajanja odgovarajućih sekcija objektnih datoteka, kako za kod, tako i za podatke, vrši se provera razrešenosti simbola. Kada linker naiđe na neku nerazrešenu promenljivu unutar neke objektne datoteke, vrši se provera da li je ta promenljiva definisana unutar neke druge objektne datoteke. U slučaju da jeste, linker vrši povezivanje, tako što umesto nerazrešene promenljive koristi referencu na originalnu promenljivu. U suprotnom, ukoliko promenljiva ne bude razrešena nakon spajanja svih objektnih datoteka, alat će pokušati da je razreši samostalno. Ovaj postupak podrazumeva, u slučaju promenljive, proveru definisanih promenljivih unutar standardnih biblioteka koje su dostupne alatu. Ukoliko je ta promenljiva, ili funkcija, definisana unutar standarde biblioteke, tada se kod i podaci na koje se referencira nerazrešena promenljiva, ili funkcija, takođe uključuju u izlaznu objektnu datoteku. Uključivanje sadržaja u izlaznu objektnu datoteku se vrši selektivno (eng. *Selective linking*), što podrazumeva da se delovi koda i promenljive na koje ne postoji referenca ne uključuju.

Ukoliko je neki simbol definisan u više objektnih datoteka, tada linker neće znati kako da ga razreši, što dovodi do prekida povezivanja. Ovaj problem se najčešće prikazuje kao greška, gde se programeru navodi koji simbol je definisan više puta.

Nakon povezivanja svih sekcija koda i podataka i razrešavanja svih referenci, dobija se "prenosiv" program. Pod prenosivim programom podrazumeva se to, da je on potpun, odnosno da sadrži i kod i podatke. Međutim, unutar ovog programa i dalje nisu dodeljene apsolutne memorijske adrese za odgovarajuće kodne segmente i segmente za podatke. U slučaju da se program koristi na nekoj platformi koja poseduje operativni sistem, ovaj korak bi bio poslednji, budući da su tada dovoljne samo relativne adrese za njegovo izvršavanje. Ipak, kako se u ovom slučaju razvija program koji će se izvršavati na embeded platformi, neophodno je dodeliti apsolutne adrese svakom segmentu.

12.4.1 Početni kod

Jedan proces koji tradicionalni kompajleri rade automatski jeste uključivanje početnog koda (eng. *Startup code*). Početni kod predstavlja mali blok koda, napisan u asemblerskom jeziku, koji vrši pripremu sistema za izvršavanje programa napisanog u programskim jezicima višeg nivoa. Svaki programski jezik višeg nivoa ima svoj skup zahteva za sistem. Na primer, programski jezik C koristi stek (eng. *stack*), pa je zbog toga potrebno alocirati prostor za datu strukturu pre izvršavanja samog programa napisanog u ovom programskom jeziku. Ovo je jedan od zahteva koji je potrebno realizovati u početnom kodu za programe napisane u jeziku C.

Početni kod za programski jezik C se obično sastoji od:

1. isključivanja svih prekida;
2. kopiranja inicijalizovanih podataka iz ROM u RAM memoriju;
3. postavljanja neinicijalizovanih podataka na 0;
4. alociranja prostora za stek i njegove inicijalizacije;
5. inicijalizacije procesorskog pokazivača steka i
6. pozivanja *main()* funkcije.

12.4.2 Lokator

Lokator predstavlja alat koji se uglavnom koristi prilikom upotrebe linkera, gde ga linker aktivira prilikom svog izvršavanja, pri čemu ga je moguće koristiti i nezavisno. Njegova funkcija se ogleda u prevođenju "prenosnog" programa, odnosno programa sa relativnim adresama, u izvršnu binarnu sliku (*.elf*). Pre pokretanja, lokatoru se dostavljaju podaci o memoriji dostupnoj na platformi za koju je potrebno kreirati izvršnu binarnu sliku. Ovi podaci se zatim koriste kako bi se kodnim segmentima i segmentima za podatke dodelile fizičke adrese. Kao rezultat pokretanja ovog alata dobija se datoteka koja sadrži binarnu memorijsku sliku koja se može izvršiti na datoj platformi.

Postupak povezivanja objektnih datoteka u izvršnu binarnu sliku se obavlja u sledećoj liniji:

```
avr-gcc -mmcu=atmega328p -o main.elf main.o
```

12.5 Konvertor binarne slike

Nakon kreiranja izlazne binarne memorijske slike, potrebno je izvršiti konverziju ove slike u format koji je neophodan za programiranje mikrokontrolera. U slučaju Arduino Uno platforme, neophodan format koda je *Intel HEX* (*.hex*).

12.5.1 Intel HEX

Intel hexadecimal object file format je format datoteke koja skladišti binarne informacije u formi ASCII teksta. Često se upotrebljava prilikom programiranja mikrokontrolera, EEPROM-a i drugih uređaja sa programabilnom logikom.

Ovaj tip datoteka sadrži linije ASCII teksta koje se sastoje od heksadecimalnih brojeva koji reprezentuju podatke, adrese i druge vrednosti u zavisnosti od svoje pozicije. Primer sadržaja jedne *.hex* datoteke dat je u nastavku.

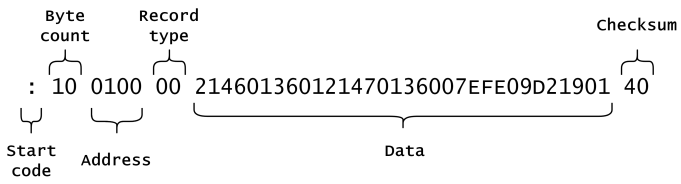
```
: 10 0100 00 214601360121470136007EFE09D21901 40
:100110002146017E17C20001FF5F16002148011928
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

Svaka linija se sastoji od šest polja koja su poređana s leva na desno, pri čemu su u prvoj liniji ova polja razdvojena razmakom radi bolje uočljivosti. To su:

1. *Start code* – polje koje se sastoji od jednog karaktera – ':';
2. *Byte count* – polje koje se sastoji od dve heksadecimalne cifre i reprezentuje broj bajtova u polju za podatke;
3. *Address* – polje koje se sastoji od četiri heksadecimalne cifre i reprezentuje 16-bitni memorijski offset adrese podataka koji se dodaje na prethodno utvđenu početnu adresu;
4. *Record type* – polje koje se sastoji od dve heksadecimalne cifre i označava tip linije, na sledeći način:
   (a) 00 – linija podataka, gde polje *Data* sadrži podatke i 16-bitnu adresu za podatke;
   (b) 01 – linija koja označava kraj datoteke, gde su polja *Byte count* i *Address* uglavnom postavljeni na vrednost 0, a polje *Data* se izostavlja;
   (c) 02 - 05 – su tipovi linija koji podešavaju početne adrese na koje se dodaju adrese iz polja *Address* kako bi se smestili podaci; za različitu vrednost tipa se početna adresa drugačije koristi.
5. *Data* – polje koje se sastoji od onoliko bajtova podataka koliko je određeno poljem *Byte count*;
6. *Checksum* – polje koje se sastoji od dve heksadecimalne cifre i označava vrednost koja se može iskoristiti da se utvrdi da li je prilikom prenosa linije doslo do greške.

Opisana polja su ilustrovana i na slici 12.2.

Postupak prevođenja izvršne binarne slike u *.hex* datoteku se obavlja korišćenjem *avr-objcopy* komande, kao što je prikazano u narednoj liniji, gde `-O` označava format izlazne datoteke:



Slika 12.2: Intel HEX format

```
avr-objcopy -O ihex main.elf main.hex
```

12.6 Programator

Programator je alat koji se upotrebljava za komunikaciju sa *bootloader*-om na ploči i služi za prebacivanje prevedenog programa u programsku memoriju. U slučaju AVR procesora, upotrebljava se programator *AVRDUDE*.

AVRDUDE je alat za preuzimanje (eng. *downloading*) i otpremljivanje (eng. *uploading*) sadržaja memorija na čipu AVR mikrokontrolera. Koristi se za programiranje programske (eng. *flash*) memorije i EEPROM memorije, gde uz pomoć serijskog protokola za programiranje može da konfiguriše registre za upravljanje mikrokontrolerom (eng. *fuses and lock bits*).

Kako se u programskoj memoriji nalazi program koji se izvršava na mikrokontroleru, tada je potrebno ažurirati vrednosti memorijskih lokacija na način opisan *.hex* datotekom.

Konačno, program se upisuje u programsku memoriju mikrokontrolera upotrebom sledeće komande:

```
avrdude -c arduino -p ATMEGA328P -P COM{broj_porta} -B {
    baud_rate} -U flash:w:main.hex
```

U datoj komandi, sledeći parametri se definišu kao:

1. `-c` – označava da nakon njega sledi specifikacija programatora koji je potrebno koristiti;
2. `arduino` – je ime programatora koji je potrebno koristiti;
3. `-p` – označava da nakon njega sledi specifikacija AVR uređaja;
4. `ATmega328P` – je ime AVR uređaja koji se programira;
5. `-P` – označava da nakon njega sledi specifikacija porta na koji je uređaj povezan;
6. `COM{broj_porta}` – je ime porta na koji je uređaj povezan;

7. `-B` – označava da nakon njega sledi specifikacija brzine prenosa podataka na portu;
8. `{baud_rate}` – je brzina prenosa podataka na portu.

12.7 Primer za vežbu

U okviru ovog poglavlja biće objašnjen postupak generisanja izvršne datoteke na osnovu izvorne i njeno upisivanje u programsku memoriju na mikrokontrolerskoj platformi. Izvorni kod aplikacije, korišćen u okviru ovog primera, koji implementira treperenje diode sa periodom od jedne sekunde je dat na listingu 12.1.

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>

int16_t main()
{
    DDRB |= 1 << 5; //PB5 je izlaz

    while (1)
    {
        PORTB |= 1 << 5; // LED ON
        _delay_ms(1000); // Pauza 1s
        PORTB &= ~(1 << 5); // LED OFF
        _delay_ms(1000); // Pauza 1s
    }

    return 0;
}
```

Listing 12.1: Izvorna datoteka hello.c

Napomena: Naredni koraci izvršeni su pod *Windows* operativnim sistemom. Vrlo slično, navedene korake je moguće izvršiti i pod nekom *Linux* distribucijom.

U okviru komandne linije (**cmd.exe** – *Windows Command Processor* ili *Windows PowerShell* u novijim verzijama Windows operativnog sistema) potrebno je promeniti trenutni direktorijum u direktorijum koji sadrži izvorne datoteke (u ovom slučaju *main.c*). Trenutnu poziciju i listu svih direktorijuma i datoteka unutar tekućeg je moguće prikazati korišćenjem komande **dir**[1]. Slično je moguće postići i upotrebom komande **ls**. Ovo je ilustrovano na listingu 12.2.

```
C:\Users\Public>dir
Volume in drive C has no label.
Volume Serial Number is 60FB-8ACE
```

[1]Još jedna korisna komanda je i komanda za brisanje sadržaja konzole – **cls**. U okviru *PowerShell* terminala je moguće koristiti i komandu **clear**, koja ima istu namenu.

```
Directory of C:\Users\Public

10/08/2021  01:22    <DIR>          .
10/08/2021  01:22    <DIR>          ..
10/08/2021  01:22    <DIR>          arduino
05/05/2021  11:32    <DIR>          Documents
07/12/2019  11:14    <DIR>          Downloads
07/12/2019  11:14    <DIR>          Music
07/12/2019  11:14    <DIR>          Pictures
07/12/2019  11:14    <DIR>          Videos
0 File(s)              0 bytes
8 Dir(s)  187,687,469,056 bytes free

C:\Users\Public>
```

Listing 12.2: Komanda dir

Promenu trenutnog direktorijuma je moguće učiniti korišćenjem komande:

```
cd putanja_do_zeljenog_direktorijuma
```

```
C:\Users\Public>cd arduino

C:\Users\Public\arduino>
```

Listing 12.3: Komanda cd

Nakon poziva ove komande, trenutni direktorijum je *arduino* u okviru kog se nalazi izvorna datoteka *main.c*, kao što je i prikazano na listingu 12.4.

```
C:\Users\Public\arduino>dir
Volume in drive C has no label.
Volume Serial Number is 60FB-8ACE

Directory of C:\Users\Public\arduino

10/08/2021  01:22    <DIR>          .
10/08/2021  01:22    <DIR>          ..
10/08/2021  10:09               263 main.c
1 File(s)            263 bytes
2 Dir(s)  187,685,842,944 bytes free

C:\Users\Public\arduino>
```

Listing 12.4: Prikaz sadržaja trenutnog direktorijuma

Kada je to učinjeno, moguće je pristupiti generisanju izvršne datoteke korišćenjem alata iz *avr-gcc toolchain*-a. Prvi korak – kompajliranje i asembliranje se vrši korišćenjem naredne komande.

```
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o main.o
    main.c
```

Nakon poziva ove komande, generiše se objektna datoteka *main.o* u tekućem direktorijumu, što se može i uočiti na listingu 12.5.

```
C:\Users\Public\arduino>avr-gcc -Os -DF_CPU=16000000UL -mmcu=
    atmega328p -c -o main.o main.c

C:\Users\Public\arduino>dir
Volume in drive C has no label.
Volume Serial Number is 60FB-8ACE

Directory of C:\Users\Public\arduino

10/08/2021  10:11    <DIR>          .
10/08/2021  10:11    <DIR>          ..
10/08/2021  10:09               263 main.c
10/08/2021  10:11               792 main.o
2 File(s)          1,055 bytes
2 Dir(s)  187,684,835,328 bytes free

C:\Users\Public\arduino>
```

Listing 12.5: Prikaz sadržaja trenutnog direktorijuma nakon generisanja objektne datoteke

Kada je objektna datoteka generisana, potrebno je pristupiti povezivanju svih objektnih datoteka u jedinstvenu izvršnu *.elf* datoteku. Ovo je moguće učiniti upotrebom sledeće komande:

```
avr-gcc -mmcu=atmega328p -o main.elf main.o
```

Nakon poziva ove komande, generiše se datoteka *main.elf* u tekućem direktorijumu, kao što je prikazano na listingu 12.6.

```
C:\Users\Public\arduino>avr-gcc -mmcu=atmega328p -o main.elf main.o

C:\Users\Public\arduino>dir
Volume in drive C has no label.
Volume Serial Number is 60FB-8ACE

Directory of C:\Users\Public\arduino

10/08/2021  10:12    <DIR>          .
10/08/2021  10:12    <DIR>          ..
10/08/2021  10:09               263 main.c
10/08/2021  10:12             2,003 main.elf
10/08/2021  10:11               792 main.o
3 File(s)          3,058 bytes
2 Dir(s)  187,684,331,520 bytes free

C:\Users\Public\arduino>
```

Listing 12.6: Prikaz sadržaja trenutnog direktorijuma nakon generisanja .elf datoteke

Poslednji korak predstavlja generisanje izvršne *.hex* datoteke na osnovu *.elf* datoteke koja poseduje format koji je neophodan za programiranje mikrokontrolera. Naredna komanda omogućava ovu proceduru.

```
avr-objcopy -O ihex main.elf main.hex
```

Nakon što je ova komanda pozvana, generiše se datoteka *main.hex* u tekućem direktorijumu, čiji je sadržaj prikazan na listingu 12.7.

```
C:\Users\Public\arduino>avr-objcopy -O ihex main.elf main.hex

C:\Users\Public\arduino>dir
Volume in drive C has no label.
Volume Serial Number is 60FB-8ACE

Directory of C:\Users\Public\arduino

10/08/2021  10:13    <DIR>          .
10/08/2021  10:13    <DIR>          ..
10/08/2021  10:09               263 main.c
10/08/2021  10:12             2,003 main.elf
10/08/2021  10:13               500 main.hex
10/08/2021  10:11               792 main.o
4 File(s)          3,558 bytes
2 Dir(s)  187,683,897,344 bytes free

C:\Users\Public\arduino>
```

Listing 12.7: Prikaz sadržaja trenutnog direktorijuma nakon generisanja .hex datoteke

Konačno, upis programa u programsku memoriju mikrokontrolera se izvršava korišćenjem naredne komande.

```
avrdude -c arduino -p ATMEGA328P -P COM3 -b 115200 -U flash:w:
    main.hex
```

U okviru same komande potrebno je manuelno konfigurisati port i brzinu komunikacije (eng. *baudrate*). U ovom slučaju, port na koji je povezana Arduino UNO platforma je *COM3*, a brzina komunikacije iznosi *115200*.

Nakon njenog poziva, u komandnoj liniji se ispisuju informacije o uspešnosti izvršene komande. Sadržaj konzole nakon navedenog koraka je dat u nastavku.

```
C:\Users\Public\arduino>avrdude -c arduino -p ATMEGA328P -P COM3 -b
    115200 -U flash:w:main.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100%
    0.02s

avrdude: Device signature = 0x1e950f
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 "main.hex"
avrdude: input file main.hex auto detected as Intel Hex
```

```
avrdude: writing flash (172 bytes):

Writing | ################################################## | 100%
    0.03s

avrdude: 172 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex auto detected as Intel Hex
avrdude: input file main.hex contains 172 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100%
    0.02s

avrdude: verifying ...
avrdude: 172 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.


C:\Users\Public\arduino>
```

Listing 12.8: Prikaz sadržaja konzole nakon upisa programa u programsku memoriju

12.8 Optimizacija na kompajlerskom nivou

Prilikom izvršavanja procesa prevođenja kako programa pisanog u nekom višem programskom jeziku u jezik nižeg nivoa, npr. asembler, tako i programa napisanog u asemblerskom jeziku u mašinski kod, izvršavaju se određene vrste optimizacije. Ove optimizacije se nazivaju optimizacije na kompajlerskom nivou i imaju zadatak da modifikuju program tako da neki njegovi aspekti budu efikasniji ili zahtevaju manje resursa. U najvećem broju slučaja, ovo podrazumeva optimizaciju u cilju boljih performansi sa stanovišta brzine rada i mogućnosti rada u uslovima u kojima nije dostupna velika količina memorijskih resursa.

Iako kompajlerske optimizacije menjaju program, one moraju ispoštovati određene zahteve. Najvažniji zahtev svakako predstavlja potreba da optimizacija bude ispravna, odnosno da ni na koji način ne promeni značenje programa. Kako je ideja optimizacije da, na neki način, unapredi performanse ili smanji upotrebu resursa, optimizacija se smatra neuspešnom ukoliko ni jedan od ova dva aspekta nije unapređen. Konačno, proces optimizovanja bi trebalo da traje razumno dugo, odnosno ne bi trebalo da značajno usporava prevođenje.

AVR-GCC alati omogućavaju nekoliko nivoa optimizacije. To su:

- -O0 – nivo na kojem su isključene sve vrste optimizacije. Ovom konfiguracijom se postiže da zadatak kompajlera bude samo smanjenje vremena

prevođenja i očuvanje ispravnosti programa.

- -O1 – prvi nivo optimizacije koji zahteva nešto više vremena i značajno veću količinu memorije za velike funkcije. Ova konfiguracija pokušava da smanji veličinu i brzinu izvršavanja programa bez upotrebe optimizacija koje zahtevaju značajnu količinu vremena.
- -O2 – nivo koji podržava sve optimizacije sadržane u -O1 nivou i uključuje nove tehnike koje ne zahtevaju kompromis između brzine izvšravanja i veličine programa. Ovaj nivo povećava vreme neophodno za prevođenje, ali se njime postižu bolje performanse generisanog koda.
- -O3 – nivo koji podržava sve optimizacije sadržane u -O2 nivou i uključuje dodatne tehnike u cilju poboljšavanja performansi. Kao rezultat može biti generisan program čije su performanse bolje, ali mu veličina može biti veća od veličine dobijene upotrebnom nivoa -O2.
- -Os – nivo koji podržava većinu optimizacije sadržane u -O2 nivou i uključuje tehnike koje smanjuju veličinu programa.

U okviru listinga 12.8 prikazan je ispis dobijen prilikom programiranja mikrokontrolera u kojem je navedena veličina mašinskog koda koji je potrebno upisati u programsku memoriju. Za konkretan primer, ona iznosi 172 bajta, gde je prilikom prevođenja upotrebljena optimizacija -Os. Prolaskom kroz čitav primer za različite vrste optimizacije može se primetiti da se veličina programa smanjuje kako se povećava nivo optimizacije. Nasuprot veličini programa, performanse, odnosno brzinu izvršavanja programa nije jednostavno izmeriti. Postoje različiti načini kako se ona može odrediti, međutim, oni će biti izostavljeni na ovom mestu.

Rezultati upotrebe različitih nivoa optimizacije sa stanovišta veličine programa su prikazani u tabeli 12.1.

Tabela 12.1: Veličina programa za različite nivoe optimizacije

Optimizacija	**Veličina**
-O0	3532 B
-O1	196 B
-O2	176 B
-O3	188 B
-Os	172 B

Poglavlje 13

Makefile

13.1 Uvod

U okviru Poglavlja 12 je obrađena tema o *AVR-GCC* razvojnim alatima, gde su objašnjene neophodne komande za prevođenje izvornog koda u njegovu izvršnu verziju. Kao što je prikazano, prolazeći kroz niz komandi, izvršavaju se različiti koraci u procesu prevođenja. Kako su se za prevođenje i programiranje projekta, koji se sastoji od jedne datoteke, iskoristile četiri komande, gde bi za svaku dodatnu datoteku ovaj broj rastao, iz razloga što je za prevođenje svake izvorne datoteke u odgovarajuću objektnu neophodna po jedna komanda, možemo pretpostaviti da bi za relativno kompleksniji projekat broj komandi bio ogroman. Ovo bi kao posledicu imalo značajan utrošak vremena za ručno pokretanje svih komandi, kao i mogućnost unošenja greške prilikom pokretanja tolikog broja komandi. Pored ovoga, ručno unošenje velikog broja komandi posle kratkog vremena postaje vrlo zamorno. Kako bi se ovi problemi prevazišli, upotrebljavaju se *Makefile* datoteke.

Makefile predstavlja skriptu koja daje instrukcije alatu *make*[1] kako da generiše određeni program. Ovaj alat prati pravila koja se definišu u *Makefile* datoteci u cilju automatizovanog generisanja izlaznih datoteka na osnovu skupa ulaznih izvornih datoteka.

Iako su *Makefile* datoteke nešto kompleksnije za kreiranje, one predstavljaju poprilično moćne alate prilikom repetitivnog generisanja projekta zbog toga što, u velikoj meri, smanjuju vreme koje će biti utrošeno.

Svako pravilo u okviru *Makefile* datoteke ima sledeći oblik:

```
target: prerequisite
    command
```

Napomena: Potrebno je obratiti pažnju na to, da nakon karaktera ':' sledi tabulator, kao i pre poziva same komande. U slučaju da se tu nalazi niz razmaka,

[1] *Make* alat se tipično instalira uz ostale GNU alate.

potrebno ga je zameniti, iz razloga što se u tom slučaju *Makefile* datoteka neće izvršiti. Ova greška se često javlja prilikom kopiranja koda.

Target predstavlja cilj koji je potrebno generisati, preduslov (eng. *prerequisite*) predstavlja datoteku koja mora da postoji pre generisanja samog *target*-a, dok komanda (eng. *command*) predstavlja način na koji se *target* generiše. Pri tome, moguće je postojanje više preduslova (odvojenih razmacima (eng. *whitespace*)), kao i postojanje više komandi u okviru jednog pravila.

13.2 Osnovna *Makefile* datoteka

Osnovni primer primene *Makefile* datoteke biće prikazan kroz program koji implementira treperenje diode, korišćen i u okviru Poglavlja 12. Prema tome, izvorna datoteka ovog programa će imati sadržaj prikazan na listingu 13.1.

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>

int16_t main()
{
    DDRB |= 1 << 5;

    while (1)
    {
        PORTB |= 1 << 5;
        _delay_ms(1000);
        PORTB &= ~(1 << 5);
        _delay_ms(1000);
    }

    return 0;
}
```

Listing 13.1: Primer sa treperenjem diode u jednoj datoteci

Primer jednostavne *Makefile* datoteke koja generiše izvršnu *.hex* datoteku, za ovaj primer, je data u nastavku, na listingu 13.2.

```
main.hex:   main.elf
    avr-objcopy -O ihex main.elf main.hex

main.elf:   main.o
    avr-gcc -mmcu=atmega328p main.o -o main.elf

main.o: main.c
    avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o main
        .o main.c
```

```
program: main.hex
    avrdude -c arduino -p ATMEGA328P -P COM3 -b 115200 -U
        flash:w:main.hex

clean:
    rm *.o *.elf *.hex
```

Listing 13.2: Primer jednostavne Makefile datoteke

U okviru ove *Makefile* datoteke definisana su tri standardna pravila za generisanje i dva posebna pravila `program` i `clean`.

Prvo pravilo za generisanje kao cilj ima kreiranje *main.hex* datoteke na osnovu izvršne binarne slike *main.elf*. Komanda koja se poziva je `avr-objcopy`, na način opisan u Poglavlju 12. Kako bi se kreirala neophodna izvršna binarna slika, definiše se sledeće pravilo gde se kao preduslov zahteva objektna datoteka *main.o*. Parametrizovanje komande *avr-gcc* u cilju povezivanja objektnih datoteka je opisano u Poglavlju 12. Na kraju, u cilju generisanja zahtevane objektne datoteke, dato je poslednje pravilo gde se kao preduslov koristi izvorna datoteka *main.c*. Parametrizovanje komande *avr-gcc* u cilju kreiranja objektnih datoteka je opisano na Poglavlju 12.

Pravilo `program` definiše način na koji će se izvršiti programiranje mikrokontrolera ukoliko postoji generisana *.hex* datoteka. Komanda koja se upotrebljava je *avrdude*, na način opisan u Poglavlju 12. Konačno, pravilo `clean` predstavlja jedno od podrazumevanih, odnosno vrlo čestih pravila koja se pojavljuju unutar *Makefile* datoteka, ne samo za projekte usmerene ka mikrokontrolerima, nego i za softver uopšte. Ovo pravilo obično za cilj ima brisanje datoteka koje se generišu prilikom upotrebe *Makefile* datoteke. Kako bi se ovo postiglo, upotrebljava se komanda *rm* koja ima funkcionalnost brisanja datoteka koje se proslede kao parametri. Datoteke se mogu navesti na različite načine, svaka pojedinačno, kao datoteke koje pripadaju određenom direktorijumu ili kao datoteke određenog tipa. Za adresiranje se mogu koristiti i apsolutne i relativne adrese, gde se kao dobra praksa uvek koriste relativne u odnosu na direktorijum u kojem se nalazi *Makefile* datoteka. U okviru ove *Makefile* datoteke, za brisanje generisanih datoteka koristi se selektovanje na osnovu tipa datoteke koje se postiže upotrebom konstrukcije **.extension_name*. Na primer, to su tipovi *.o*, *.elf* i *.hex*, koji se mogu generisati neograničen broj puta na osnovu izvorne datoteke, pa ih iz tog razloga nije potrebno čuvati nakon programiranja, nego se mogu uvek generisati ukoliko za to postoji potreba.

Kako bi se izvršila *Makefile* datoteka potrebno je pozicionirati se u direktorijum u okviru kog se nalazi *Makefile* datoteka, a nakon toga pozivanjem komande *make* unutar komandne linije vrši se pokretanje skripte i kreiranje izvršne datoteke (*.hex*). Nakon toga, pozivanjem komande *make program*, takođe unutar komandne linije, vrši se njen upis u programsku memoriju mikrokontrolera. Dodatno, pozivom komande *make clean*, omogućava se brisanje svih prethodno generisanih datoteka.

13.3 Upotreba promenljivih u *Makefile* datotekama

Prilikom rada sa ovako pisanim *Makefile* datotekama uvek je potrebno voditi računa o napravljenim izmenama u tekućem projektu. Prilikom izmena datoteka za koje je potrebno generisati izvršnu datoteku ili promenu porta na kom je vezan hardver, potrebno je u skladu sa tim konfigurisati i dati *Makefile*. Budući da ovo podrazumeva detaljan prolazak kroz samu *Makefile* datoteku i izmenu odgovarajućih parametara, čitava procedura vrlo brzo postaje zamorna. Kako bi se ovaj postupak automatizovao, u okviru *Makefile* datoteka je moguća upotreba promenljivih.

Promenljiva u *Makefile* datoteci se definiše na sledeći način.

```
VARIABLE_NAME = VALUE
```

Sa druge strane, poziv promenljive se vrši na sledeći način.

```
$(VARIABLE_NAME)
```

Uzimajući ovo u obzir, prethodni primer *Makefile* datoteke je moguće napisati i na sledeći način.

```
CC = avr-gcc
OBJCOPY = avr-objcopy

SOURCE = main
TARGET = app
PORT = COM3

$(TARGET).hex:  $(TARGET).elf
    ${OBJCOPY} -O ihex $(TARGET).elf $(TARGET).hex

$(TARGET).elf:  $(SOURCE).o
    ${CC} -mmcu=atmega328p $(SOURCE).o -o $(TARGET).elf

$(SOURCE).o:    $(SOURCE).c
    ${CC} -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o $(
        SOURCE).o $(SOURCE).c

program: $(TARGET).hex
    avrdude -c arduino -p ATMEGA328P -P ${PORT} -b 115200 -U
        flash:w:$(TARGET).hex

clean:
    rm *.o *.elf *.hex
```

Listing 13.3: Primer Makefile datoteke u okviru koje se koriste promenljive

U okviru ove *Makefile* datoteke definisano je pet promenljivih. Prve dve promenljive `CC` i `OBJCOPY` definišu alate koji se upotrebljavaju u sklopu korišćenih

komandi. U ovom slučaju to su alati iz skupa AVR-GCC razvojnih alata: *avr-gcc* i *avr-objcopy*. Pored toga, definisane su i promenljive `SOURCE`, `TARGET` i `PORT`. Promenljiva `SOURCE` služi za definisanje imena izvorne datoteke i njenih derivata. `TARGET` je promenljiva koja definiše ime datoteka nakon povezivanja u izvršnu binarnu sliku. `PORT` promenljiva predstavlja broj porta preko kojeg se vrši serijska komunikacija sa mikrokontrolerom, respektivno. Prilikom izvršavanja *Makefile* datoteke, svi pozivi promenljivih (korišćenjem `$(VARIABLE_NAME)`) biće zamenjeni odgovarajućim vrednostima definisanim u okviru te datoteke.

13.4 *Makefile* datoteka u projektima sa više izvornih datoteka

U narednom primeru biće ilustrovano pisanje *Makefile* datoteka u slučaju da je potrebno generisati izvršnu datoteku na osnovu projekta koji sadrži više izvornih datoteka. Navedeni primer koristi biblioteku **timer0**, čija je realizacija objašnjena u okviru Poglavlja 5. Podsećanja radi, njegova realizacija je data na listingu 13.4.

```
#include <avr/io.h>
#include <stdint.h>
#include "timer0.h"

int16_t main()
{
    DDRB |= (1 << 5);
    timer0InteruptInit();

    while (1)
    {
        PORTB |= (1 << 5);
        timer0DelayMs(500);
        PORTB &= ~(1 << 5);
        timer0DelayMs(500);
    }

    return 0;
}
```

Listing 13.4: Primer sa treperenjem diode uz korišćenje biblioteke timer0

Kako bi se omogućilo generisanje izvršne datoteke na osnovu više izvornih datoteka, potrebno je izvršiti određene izmene u sklopu ranije napisane *Makefile* datoteke. Sadržaj izmenjene *Makefile* datoteke je dat u nastavku.

```
CC = avr-gcc
OBJCOPY = avr-objcopy

TARGET = main
OBJECTS = main.o timer0.o
```

```
PORT = COM3

$(TARGET).hex:  $(TARGET).elf
    ${OBJCOPY} -O ihex $(TARGET).elf $(TARGET).hex

$(TARGET).elf:  $(OBJECTS)
    ${CC} -mmcu=atmega328p -o $(TARGET).elf $(OBJECTS)

%.o : %.c
    ${CC} -Os -DF_CPU=16000000UL -mmcu=atmega328p -c $< -o $@

program: $(TARGET).hex
    avrdude -c arduino -p ATMEGA328P -P $(PORT) -b 115200 -U
        flash:w:$(TARGET).hex

clean:
    rm *.o *.elf *.hex
```

Listing 13.5: Primer Makefile datoteke koja omogućava rad sa više izvornih datoteka

Prva izmena predstavlja definiciju nove promenljive `OBJECTS` u okviru koje se navode sve objektne datoteke koje je potrebno generisati. U ovom slučaju, potrebno je navesti dve datoteke: *main.o* i *timer0.o*, budući da je potrebno izvršiti kompajliranje dve izvorne datoteke: *main.c* datoteke koja sadrži glavni program i *timer0.c* datoteke koja sadrži implementacije funkcija koje se koriste u okviru glavnog programa.

Pored toga, definiše se i novo pravilo za generisanje objektne datoteke na osnovu izvorne datoteke za svaku izvornu datoteku ponaosob. Ovo pravilo dato je u nastavku.

```
%.o : %.c
    ${CC} -Os -DF_CPU=16000000UL -mmcu=atmega328p -c $< -o $@
```

Kako bi ovo bilo moguće potrebno je koristiti takozvane *predefinisane* (automatske) promenljive na osnovu kojih je moguće izvršiti generalizaciju datoteka koje je potrebno koristiti. Poziv ovih promenljivih se vrši na sličan način kao i u slučaju klasičnih promenljivih (započinje karakterom $), gde je, umesto naziva promenljive, potrebno pisati neki od specijalnih karaktera (@, *, +, ? itd.).

U skladu sa tim, neke od najčešće korišćenih automatskih promenljivih su:

- `$@` – promenljiva koja biva zamenjena sa imenom datoteke cilja (eng. *target*);
- `$*` – promenljiva koja biva zamenjena sa imenom datoteke cilja, bez ekstenzije;
- `$<` – promenljiva koja biva zamenjena sa imenom datoteke prvog preduslova;
- `$^` – promenljiva koja biva zamenjena svim imenima datoteka preduslova.

U okviru komande navedenog pravila koriste se dve automatske promenljive $< i $@ u sledećem obliku:

```
-c $< -o $@
```

Na ovaj način specificira se sledeća tvrdnja: "*Izvrši kompajliranje datoteke čije je ime zadato u prvom preduslovu (%.c), i generiši izlaznu datoteku čije je ime zadato u okviru cilja (%.o)*".

Parametri %.c i %.o predstavljaju generalizaciju imena izvornih i objektni datoteka koje je potrebno generisati. U pitanju su, takozvana, šablonska pravila. Ona podrazumevaju korišćenje specijalnog karaktera % i koriste se za specificiranje cilja, odnosno preduslova, kako bi se izvršila generalizacija samog pravila i na taj način izbeglo korišćenje eksplicitnih imena. Iz navedenog razloga, ovako definisana pravila se nazivaju implicitna pravila.

Cilj (eng. *target*) i preduslov (eng. *prerequisite*) u navedenom pravilu su zadati u ovom obliku.

```
%.o : %.c
```

Na ovaj način specificira se sledeća tvrdnja: "*Prilikom pravljenja objektne datoteke (.o) na osnovu izvorne datoteke (.c) upotrebom komande u nastavku, napravi objektnu datoteku sa istim imenom kao izvorna datoteka*". Prema tome, ukoliko je izvorna datoteka *main.c*, na osnovu nje biće generisana objektna datoteka *main.o*.

Alternativni način za generisanje objektnih datoteka bi podrazumevao pisanje po jednog pravila za svaku objektnu datoteku koju je potrebno kreirati. Jasan zaključak koji se izvodi jeste taj da, za veći broj izvornih datoteki ova procedura postaje prilično zamorna, pa se stoga i retko koristi.

13.5 *Makefile* datoteka u projektima koji koriste statičke biblioteke

U prethodnom odeljku je prikazan način realizacije *Makefile* datoteke u slučaju da se projekat sastoji iz više izvornih datoteka. Sledeći primer prikazuje kako se kreira *Makefile* u slučaju projekta koji koristi statičku biblioteku. U okviru ovog primera, statička biblioteka će implementirati funkcionalnost manipulacije pinovima na način opisan u Poglavlju 5.

Prvo će biti prikazan način na koji se prevodi statička biblioteka. Statička biblioteka se kreira kao samostalan projekat u direktorijumu odvojenom od aplikacije. Ovaj direktorijum treba da sadrži sve neophodne datoteke za implementaciju date biblioteke. Takođe, direktorijum će posedovati *Makefile* datoteku čija će namena biti generisanje datoteka koje će biti upotrebljene od strane drugih projekata. Sadržaj *Makefile* datoteke za generisanje ovih datoteka je prikazan na listingu 13.6.