

Dušan Bortnik
Milan Lukić

Vladimir Nikić
Ivan Mezei

Razvoj softvera za embeded sisteme

Programiranje Arduino UNO platforme u jeziku C

Agencija Eho
www.infoelektronika.net

- Sva prava zadržana. Nijedan deo ove knjige ne sme biti reprodukovan u bilo kom materijalnom obliku, uključujući fotokopiranje ili slučajno ili nenamerno smeštanje na bilo koji elektronski medijum sa ili uz pomoć bilo kog elektronskog sredstva, bez pismenog odobrenja nosioca autorskih prava osim u skladu sa odredbama zakona o autorskim pravima, dizajnu i patentima. Prijave za pismene dozvole radi štampanja bilo kog dela ove publikacije upućuje se izdavaču ove knjige.
- Izjava: Autori i izdavač su uložili najveće napore da bi se obezbedila tačnost informacija sadržanih u ovoj knjizi. Autor i izdavač ne mogu da pretpostave neprijatnosti i ovom izjavom isključuju bilo kakvu odgovornost za bilo koju stranku koja bi imala gubitke ili štetu uzrokovanu greškama ili propustima u ovoj knjizi, bez obzira da li su greške ili propusti nastali usled nemara, nezgode ili bilo kog drugog razloga.

ISBN 978-86-80134-51-2

Razvoj softvera za embeded sisteme

Autori: Dušan Bortnik, Vladimir Nikić, Milan Lukić, Ivan Mezei

Izdaje i štampa: Agencija Eho, Niš

e-mail: redakcija@infoelektronika.net

Tiraž: 200

Godina izdanja: 2023

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.383/.384
004.432.2C

RAZVOJ softvera za embeded sisteme : programiranje Arduino UNO platforme u jeziku C / Dušan Bortnik ... [et al.]. - Niš : Agencija Eho, 2023 (Niš : Agencija Eho). - 269 str. : ilustr. ; 24 cm

Tiraž 200. - O autorima: str. 3. - Napomene i bibliografske reference uz tekst. - Bibliografija: str. 269-[270].

ISBN 978-86-80134-51-2

1. Бортник, Душан, 1996- [аутор]
а) Микроконтролери б) Програмски језик "C"

COBISS.SR-ID 124464649

O autorima

Dušan Bortnik je rođen u Novom Sadu 1996. godine. Diplomski i master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Embedded sistemi i algoritmi odbranio je 2019. i 2020. godine, respektivno. Trenutno je student doktorskih studija. Kao autor ili koautor objavio je četiri rada na međunarodnim konferencijama.

Vladimir Nikić je rođen u Novom Sadu 1996. godine. Diplomski i master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Embedded sistemi i algoritmi odbranio je 2019. godine, i 2020. godine, respektivno. Trenutno je student doktorskih studija. Kao autor ili koautor objavio je četiri rada na međunarodnim konferencijama.

Dr Milan Lukić je docent na Katedri za elektroniku Fakulteta tehničkih nauka Univerziteta u Novom Sadu. Diplomirao je 2005. godine na Odseku za elektrotehniku i računarstvo Fakulteta tehničkih nauka, a doktorirao 2015. godine na istom fakultetu. Kao autor ili koautor objavio je preko 30 naučnih radova, od toga 7 sa impakt faktorom.

Dr Ivan Mezei je redovni profesor na Katedri za elektroniku Fakulteta tehničkih nauka Univerziteta u Novom Sadu. Diplomski rad, magistarski rad i doktorsku tezu odbranio je na Odseku za elektrotehniku i računarstvo Fakulteta tehničkih nauka, 1999., 2005. i 2012. godine respektivno. Ima preko 20 godina iskustva u projektovanju i programiranju embedded sistema. Kao autor ili koautor objavio je 1 udžbenik, 1 praktikum, 2 poglavlja u međunarodnim monografijama, više od 15 radova u naučnim časopisima, više od 40 publikacija na naučnim konferencijama. Bio je recenzent u više od 20 časopisa sa impakt faktorom kao i član programskog odbora na više od 35 naučnih konferencija. Bio je rukovodilac projekta, stariji istraživač, rukovodilac radnog paketa, supervizor ili vođa tima na više projekata finansiranih od strane EU ili kompanija.

Sadržaj

Predgovor	11
I Osnove razvoja softvera za mikrokontrolere	15
1 Programiranje AVR mikrokontrolera u razvojnom okruženju <i>Eclipse</i>	17
1.1 Uvod	17
1.2 Preuzimanje i instalacija Eclipse IDE	17
1.3 Instalacija AVR-GCC i AVR Dude	18
1.4 Instalacija AVR dodatka za Eclipse	20
1.5 Kreiranje projekta	20
1.6 Podešavanje kompajlera i programatora	22
1.7 Dodavanje izvornog koda	24
1.8 Kompajliranje izvornog koda i programiranje mikrokontrolera .	26
2 Upravljanje portovima mikrokontrolera	27
2.1 Portovi mikrokontrolera ATmega328P	27
2.2 Konfigurisanje pinova AVR mikrokontrolera	30
2.3 Primeri programa koji upravljaju pinovima mikrokontrolera . .	30
3 Prekidi mikrokontrolera	35
3.1 Uvod	35
3.2 Programaska podrška sistemu prekida kod AVR mikrokontrolera	36
3.3 Prekidi usled promene stanja na pinu kod AVR mikrokontrolera	38

4	Tajmer/brojač modul kod mikrokontrolera	43
4.1	Uvod	43
4.2	Struktura tajmer/brojač modula kod AVR mikrokontrolera . .	43
4.2.1	Brojačka jedinica	44
4.2.2	Jedinica za poređenje izlaza	47
4.3	Prekidi tajmer/brojač modula kod AVR mikrokontrolera	53
4.4	Funkcije za upravljanje vremenom	65
4.4.1	Funkcije za kašnjenje	65
4.4.2	Funkcije za upravljanje vremenom bazirane na prekidnoj rutini tajmera	66
4.4.3	Petlje sa vremenskim ograničenjem	68
4.5	Zadaci za vežbu	70
5	Organizacija projekata sa više izvornih datoteka	71
5.1	Uvod	71
5.2	Organizacija projekta i koncept pisanja biblioteka	73
5.3	Zadaci za vežbu	83
6	Analogno-digitalna konverzija AVR mikrokontrolera	85
6.1	Uvod	85
6.2	Registri ADC modula	89
6.3	Interni temperaturni senzor i merenje temperature	92
6.4	Zadaci za vežbu	93
6.4.1	Linear-Feedback pomerački registri	96
7	Statičke biblioteke	101
7.1	Uvod	101
7.2	Kreiranje statičkih biblioteka u razvojnom okruženju <i>Eclipse</i> .	101
7.3	Zadaci za vežbu	106
8	Serijski port i funkcije za serijsku komunikaciju	107
8.1	RS-232 komunikacioni protokol	107
8.2	Naponski nivoi	108

8.3	Signalizacija	109
8.4	USART0 mikrokontrolera ATmega328P	111
8.5	Biblioteka za serijsku komunikaciju	116
8.5.1	Opis funkcija	117
8.6	Zadaci za vežbu – I deo	119
8.7	Zadaci za vežbu – II deo	124
9	Watchdog tajmer	137
9.1	Uvod	137
9.2	Rad sa <i>watchdog</i> tajmerom kod AVR mikrokontrolera	140
9.3	Zadaci za vežbu	147
10	Režimi rada AVR mikrokontrolera	149
10.1	Uvod	149
10.2	Zadaci za vežbu	158
11	Memorije AVR mikrokontrolera	159
11.1	Uvod	159
11.2	SRAM – Memorija za podatke	159
11.3	Programska <i>flash</i> memorija	160
11.4	Programska memorija i <i>Look-up</i> tabele	161
11.5	<i>EEPROM</i> memorija kod AVR mikrokontrolera	163
11.6	Zadaci za vežbu	166
12	Softverske mašine stanja	169
12.1	Uvod	169
12.2	Načini predstavljanja mašina stanja	170
12.3	Implementacija mašina stanja	172
12.4	Primena mašina stanja u ostvarivanju pseudo-paralelnog načina izvršavanja	180
12.5	Zadaci za vežbu	183
13	Razno	187

13.1	Bitski operatori	187
13.1.1	Bitsko invertovanje	187
13.1.2	Bitska I operacija	188
13.1.3	Bitska ILI operacija	189
13.1.4	Bitska EKS-ILI operacija	189
13.1.5	Operacija pomeranja u desno	190
13.1.6	Operacija pomeranja u levo	191
13.1.7	Bitske operacije nad pojedinačnim bitima	192
13.1.8	Zadaci za vežbu	193
13.2	Neki napredni koncepti programskog jezika C	194
13.2.1	Kvalifikatori u programskom jeziku C	195
13.2.2	<i>Storage</i> klase	202
13.2.3	Upravljanje memorijom	204
13.2.4	Zadaci za vežbu	209

II Operativni sistemi za rad u realnom vremenu 217

14 Uvod u operativne sisteme za rad u realnom vremenu 219

14.1	Softverske arhitekture za embeded sisteme	219
14.1.1	<i>Round-robin</i> arhitektura	220
14.1.2	<i>Round-robin</i> arhitektura sa prekidima	221
14.1.3	<i>Function-queue-scheduling</i> arhitektura	222
14.1.4	Arhitektura bazirana na operativnom sistemu koji radi u realnom vremenu	224
14.2	Uvod u operativne sisteme koji rade u realnom vremenu	226
14.2.1	Zadaci (Taskovi)	226
14.2.2	Planer	228
14.2.3	Komunikacija između zadataka	229
14.3	RTOS Implementacija na primeru ArdOS	236
14.3.1	Struktura ArdOS	237
14.3.2	Funkcije za pokretanje operativnog sistema	239

14.3.3	Funkcije za kreiranje zadataka	239
14.3.4	Funkcije za promenu izvršavanja zadataka	240
14.3.5	Funkcije za rad sa vremenom	242
14.3.6	Funkcije za rad sa semaforima	242
14.3.7	Funkcije za rad sa muteksima	244
14.3.8	Funkcije za rad sa uslovnim promenljivama	245
14.3.9	Funkcije za rad sa FIFO redovima	246
14.3.10	Funkcije za rad sa prioriternim redom	247
14.4	Primeri aplikacija u ArdOS	249
14.5	Zadaci za vežbu	268

Predgovor

Ova knjiga je nastala tokom školske 2021/22. godine i namenjena je, prevažno, studentima treće godine osnovnih akademskih studija na predmetu *Razvoj softvera za embeded sisteme*, na smeru za Mikroročunarsku elektroniku, na Departmanu za energetiku, elektroniku i telekomunikacije, Fakulteta tehničkih nauka u Novom Sadu. Međutim, treba naglasiti i to, da je knjiga namenjena i svima onima koji žele da dodatno prodube svoje znanje o mikrokontrolerima i pisanju softvera za iste.

Naime, cilj ove knjige jeste osposobljavanje čitalaca za pisanje složenijih aplikacija za AVR familiju mikrokontrolera, specifično za mikrokontroler ATmega328P. Takođe, kao još jedan od ishoda jeste i mogućnost organizacije projekta većeg obima u manje, zasebne, logičke celine. Dodatno, stavljen je akcenat na razumevanje rada određenih periferija mikrokontrolera koje se često koriste u praksi, kao što su različite vrste tajmera, jedinica za serijsku komunikaciju, memorije itd. Konačno, kao poslednji cilj ove knjige, izdvaja se razumevanje koncepta operativnih sistema za rad u realnom vremenu i pisanje jednostavnijih programa u okviru *ArDOS* operativnog sistema. Ovo predstavlja izuzetno korisnu veštinu, budući da se operativni sistemi za rad u realnom vremenu izuzetno eksploatišu u praktičnom radu sa mikrokontrolerima.

U okviru knjige, platforma koja se koristi u svim primerima i zadacima je *Arduino UNO*, koja poseduje mikrokontroler iz AVR familije – ATmega328P, te se podrazumeva neko osnovno predznanje u radu sa njom. Za više informacija o samoj platformi, čitalac se upućuje na sledeći link.

<https://www.arduino.cc/>

Dodatno, tehničku dokumentaciju mikrokontrolera ATmega328P, koji se nalazi na Arduino UNO platformi i na koju se često poziva u okviru ove knjige je moguće pronaći na narednom linku.

<https://www.microchip.com/en-us/product/ATmega328P>

U nastavku je, ukratko, opisan sadržaj ove knjige.

U prvoj glavi objašnjen je rad sa razvojnim okruženjem *Eclipse*, koje predstavlja savremen alat namenjen razvoju aplikacija koje se mogu izvršavati na

različitim platformama. Objašnjen je način preuzimanja i podešavanja samog okruženja u cilju pisanja aplikacija za AVR familiju mikrokontrolera, kao i način na koji se kreiraju projekti unutar njega.

Druga glava posvećena je objašnjavanju rada sa portovima AVR mikrokontrolera. Detaljno su objašnjeni osnovni registri za rad sa pojedinačnim pinovima *ATmega328P* mikrokontrolera.

U okviru treće glave, obrađeni su prekidi mikrokontrolera. Nakon generalne priče o konceptu prekida, detaljno je objašnjena programska podrška sistemu prekida AVR mikrokontrolera. Objašnjena je osnovna biblioteka za rad sa prekidima, kao i način pisanja prekidne rutine. Na kraju, opisani su prekidi usled promene stanja na ulaznom pinu i upravljanje njima kroz odgovarajuće primere.

Četvrta glava obrađuje tajmer/brojač modul kod AVR mikrokontrolera, specifično *ATmega328P* kontrolera. Opisana je generalna hardverska struktura modula, uz detaljniji opis pojedinih komponenti – brojačke jedinice, preskalera i jedinice za poređenje izlaza. Nakon toga, objašnjeni su režimi rada samog modula, kao i odgovarajući konfiguracioni registri. Potom, objašnjena je upotreba tajmer/brojač modula u generisanju prekida kod AVR mikrokontrolera, kroz odgovarajući primer. Konačno, objašnjeno je pisanje biblioteke za rad sa vremenom, bazirane na upotrebi tajmer/brojačkog modula.

U petoj glavi je objašnjena organizacija projekata sa više izvornih datoteka. Naime, prilikom pisanja složenijih aplikacija, smisleno je programski kod podeliti u više logičkih celina koje pripadaju različitim datotekama i na taj način povećati preglednost koda, kao i njegovu ponovnu upotrebljivost. Kroz jednostavan primer, objašnjena je organizacija projekta i koncept pisanja biblioteka.

Šesta glava obrađuje analogno-digitalnu konverziju kod AVR mikrokontrolera. U praktičnim aplikacijama, često se javlja potreba za obradom različitih analognih vrednosti. Budući da mikroprocesori ne mogu da obrađuju ovakvu vrstu podataka, neophodno je izvršiti odgovarajuću konverziju. U okviru ove glave, opisana je generalna hardverska struktura ADC modula i način njegovog funkcionisanja. Pored toga, objašnjeni su i osnovni registri neophodni za rad sa njim.

Sedma glava je posvećena pravljenju statičkih biblioteka u okviru *Eclipse* razvojnog okruženja. Naime, ideja statičke biblioteke jeste ta, da se jednom napisani kod može instancirati u proizvoljnom broju projekata, bez potrebe za kopiranjem bilo kakvih izvornih datoteka. Kroz jednostavan primer, detaljno su objašnjeni koraci prilikom pravljenja statičke biblioteke.

U okviru glave osam, opisan je rad sa serijskim portom AVR mikrokontrolera. Detaljno su objašnjene sve karakteristike RS-232 komunikacionog protokola, kao i interfejs za serijsku RS-232 komunikaciju *ATmega328P* mikrokontrolera – *USART0*. Dodatno, objašnjeni su svi registri za rad sa navedenom jedinicom. Konačno, opisana je i biblioteka za rad serijskim portom.

U devetoj glavi je obrađena nova periferija – *watchdog* tajmer. Nakon kratkog opisa načina funkcionisanja, data su uputstva za rad sa *watchdog* tajmerom kod AVR mikrokontrolera. Dodatno, dati su i primeri jednostavnih programa koji koriste pomenuti tajmer.

Glava deset opisuje režime rada AVR mikrokontrolera. Naime, detaljno su opisani različiti režimi rada u kojima se ATmega328P mikrokontroler može naći, kao i standardna biblioteka koja definiše funkcije za promenu režima rada. Dodatno, dati su i primeri jednostavnih programa u okviru kojih se manipuliše režimima rada upotrebom odgovarajuće biblioteke.

U sklopu jedanaeste glave, detaljno je opisana organizacija memorije AVR mikrokontrolera, sa naglaskom na ATmega328P mikrokontroler. Objasnjena je *SRAM* i programska *flash* memorija, kao i EEPROM memorija koju poseduje većina AVR mikrokontrolera (između ostalih i ATmega328P). Konačno, objašnjena je i standardna biblioteka za upravljanje EEPROM memorijom.

U narednoj, dvanaestoj glavi objašnjen je koncept softverskih mašina stanja i način njihovog modelovanja u softveru kroz različite primere. Pored toga, data je i njihova primena u ostvarivanju pseudo-paralelnog načina izvršavanja, kao uvodna priča za drugi deo ove knjige.

Trinaesta glava objašnjava neke generalne koncepte programskog jezika C, a koji se često koriste prilikom pisanja aplikacija za mikrokontrolerske platforme. Naime, objašnjeni su bitski operatori, kvalifikatori u programskom jeziku C i *storage* klase.

Poslednja četrnaesta glava, odnosno drugi deo ove knjige, posvećena je operativnim sistemima za rad u realnom vremenu. Na početku, objašnjene su osnovne softverske arhitekture za embeded sisteme. Nakon toga, akcentat je stavljen na uopštenu priču o operativnim sistemima za rad u realnom vremenu. Potom, opisana je implementacija operativnog sistema za rad u realnom vremenu kroz primer ArdOS-a, gde je detaljno objašnjena biblioteka i njene funkcije za rad sa pomenutim sistemom. Konačno, dati su primeri jednostavnih aplikacija pisanih u ArdOS-u.

Na kraju, autori izražavaju svoju zahvalnost kolegama sa Katedre za elektroniku, na Departmanu za energetiku, elektroniku i telekomunikacije, na Fakultetu tehničkih nauka u Novom Sadu, koji su pomogli u izradi ove knjige. Takođe, autori duguju zahvalnost svima onima koji su svojim korisnim savetima unapredili kvalitet ove knjige.

U Novom Sadu, 2023. godine,

Autori

Deo I

**Osnove razvoja softvera za
mikrokontrolere**

Glava 1

Programiranje AVR mikrokontrolera u razvojnom okruženju *Eclipse*

1.1 Uvod

Integrirano razvojno okruženje *Eclipse IDE* predstavlja moćan i savremen alat koji se koristi za razvoj aplikacija namenjenih izvršavanju na velikom broju različitih platformi. Sve većoj popularnosti ovog okruženja doprinosi veliki broj postojećih dodataka (eng. *plugins*) koji omogućavaju povezivanje sa različitim razvojnim softverskim modulima. Među mnoštvom podržanih opcija, omogućeno je programiranje AVR familije mikrokontrolera u programskom jeziku C. U okviru ovog poglavlja biće ukratko opisan postupak instalacije i podešavanja softverskih modula koji su neophodni za programiranje AVR mikrokontrolera *ATmega328P* koji se nalazi na *Arduino UNO* platformi.

Svi alati koji će biti korišćeni u nastavku su alati otvorenog koda (eng. *open-source*) i omogućeno je njihovo besplatno preuzimanje od strane korisnika. Budući da je *Eclipse* zapravo *cross-platform* okruženje koje je razvijano u programskom jeziku Java, za njegovo funkcionisanje je neophodna postojeća instalacija Jave. U okviru ovog poglavlja će biti opisan postupak instalacije i korišćenja *Eclipse* razvojnog okruženja pod operativnim sistemom *Windows*. Vrlo sličnu proceduru je potrebno sprovesti i u slučaju nekog drugog operativnog sistema.

1.2 Preuzimanje i instalacija Eclipse IDE

Preuzimanje Eclipse razvojnog okruženja je moguće izvršiti putem sledećeg linka:

<https://www.eclipse.org/downloads/packages/>

Među ponuđenim verzijama, potrebno je odabrati "*Eclipse IDE for C/C++ developers*". U zavisnosti od platforme, bira se 32-bitna ili 64-bitna verzija za odgovarajući operativni sistem. Nakon otpakivanja preuzete arhive, pokreće se izvršna verzija programa *Eclipse* koja se nalazi u otpakovanom direktorijumu.

Pokretanjem razvojnog okruženja, prvo se vrši izbor direktorijuma u kojem će biti smešteni svi izvorni kodovi i ostale datoteke koje sačinjavaju strukturu projekata koje korisnik bude razvijao. Takav direktorijum se naziva *Workspace*, odnosno radni prostor. Ukoliko više korisnika koristi istu mašinu, preporučljivo je da svaki korisnik definiše zaseban *Workspace* direktorijum, kako bi se izbegle neželjene situacije koje bi mogle nastati usled korišćenja zajedničkog radnog prostora.

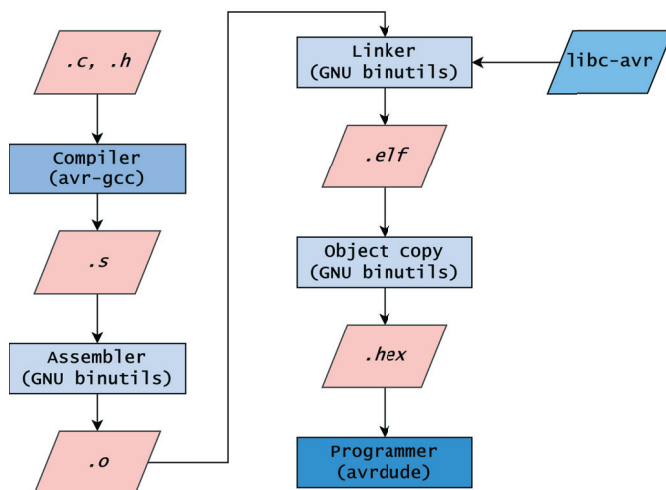
1.3 Instalacija AVR-GCC i AVR Dude

AVR-GCC predstavlja skup razvojnih alata (eng. *toolchain*) koji se koriste za prevođenje programa za AVR mikrokontrolere napisanih u programskom jeziku C. *GNU Compiler Collection* (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.

Ilustracija procesa kompajliranja korišćenjem *avr-gcc* alata prikazana je na slici 1.1.

Sa druge strane, *AVR Dude* je pomoćni alat koji komunicira sa hardverskim programatorom i posredstvom kog se prevedeni program upisuje u programsku (eng. *flash*) memoriju mikrokontrolera. Naime, alat šalje pakete podataka u odgovarajućem formatu putem serijskog interfejsa ka mikrokontroleru. Sa druge strane, mikrokontroler u svojoj programskoj memoriji poseduje specijalan program – *bootloader*. Reč je o aplikaciji malih dimenzija, veličine par stotina bajtova, koja je smeštena na kraju adresnog prostora programske memorije. Nakon reseta mikrokontrolera, *bootloader* se pokreće i proverava pakete podataka koji mu pristižu putem serijskog porta, u skladu sa odgovarajućim protokolom. Ukoliko aplikacija detektuje zahtev za upisom nove korisničke aplikacije, ona će izvršiti reprogramiranje korisničkog dela programske memorije sa novom korisničkom aplikacijom. Ako ovaj zahtev, nakon reseta mikrokontrolera izostane, *bootloader* će inicirati programsku instrukciju skoka i započeti izvršavanje korisničke aplikacije. Način organizacije programske memorije je prikazan na slici 1.2.

Treba napomenuti i to, da upotreba *bootloader*-a nije jedini način pomoću kojeg je moguće izvršiti programiranje kontrolera. Naime, postoje i tzv. eksterni



Slika 1.1: Proces kompajliranja

programatorski uređaji – *programatori* (eng. *programmer*) koji se povezuju sa odgovarajućim pinovima mikrokontrolera i putem njih vrše direktan upis nove korisničke aplikacije u programsku memoriju. Prednost ovog pristupa jeste ostvarivanje većeg korisničkog prostora u programskoj memoriji, budući da deo memorije nije zauzet od strane *bootloader*-a, koji se u ovoj situaciji ne koristi. Pored toga, početno kašnjenje koje se javlja prilikom izvršavanja *bootloader*-a ne postoji.



Slika 1.2: Organizacija programske memorije

Pod operativnim sistemom *Windows*, oba alata se instaliraju u okviru programskog paketa *WinAVR*, koji je moguće besplatno preuzeti putem narednog linka.

`winavr.sourceforge.net/download.html`

Napomena: Ukoliko se koristi *Windows 10* operativni sistem potrebno je, pored navedene instalacije, izvršiti dodatne korake koji podrazumevaju sledeće:

1. u instalacionom direktorijumu na adresi `\WinAVR-20100110\utils\bin` je potrebno locirati datoteku `msys-1.0.dll`;
2. preuzeti novu, zamensku datoteku na sledećem linku.

`http://www.madwizard.org/download/electronics/msys-1.0-vista64.zip`

3. zameniti postojeću datoteku sa preuzetom datotekom.

1.4 Instalacija AVR dodatka za Eclipse

Dodatak za AVR mikrokontrolere (*AVR plugin*) omogućava i u znatnoj meri olakšava rad sa ovom familijom mikrokontrolera u okviru *Eclipse* razvojnog okruženja. Njegovom instalacijom omogućeno je kreiranje, kompajliranje i učitavanje C projekta koji će se izvršavati na željenom AVR mikrokontroleru.

Instalacija ovog dodatka vrši se izborom opcije *Help* → *Install New Software* u okviru *Eclipse* okruženja. Po otvaranju prozora čiji izgled je prikazan na slici, u polju *Work with* potrebno je upisati naredni link.

`http://avr-eclipse.sourceforge.net/updatesite/`

i kliknuti *Add*. Po iskakanju prozora *Add Repository*, u polju *Name* imenuje se AVR dodatak, na primer *AVR Plugin*, kao što je i prikazano na slici 1.3.

Nakon pojavljivanja opcije *AVR Eclipse Plugin*, potrebno je potvrditi odabir opcije klikom na kvadrat koji se nalazi sa leve strane, nakon čega sledi pritisak na taster *Next*. Praćenjem i potvrdom opcija koje će se pojavljivati u narednim prozorima, stiže se do opcije *Finish*, čime se okončava instalacija AVR dodatka. Kako bi dodatak bio aktiviran, potrebno je restartovati *Eclipse* okruženje.

Instalacijom razvojnog okruženja *Eclipse* sa dodatkom AVR plugin i WinAVR programskog paketa, korisnik ima na raspolaganju sve softverske alate koji su mu potrebni za programiranje AVR mikrokontrolera u programskom jeziku C.

1.5 Kreiranje projekta

Nakon što je razvojno okruženje instalirano i podešeno, naredni korak podrazumeva kreiranje projekta. U okviru projekta, nalaziće se sve potrebne datoteke

Glava 3

Prekidi mikrokontrolera

3.1 Uvod

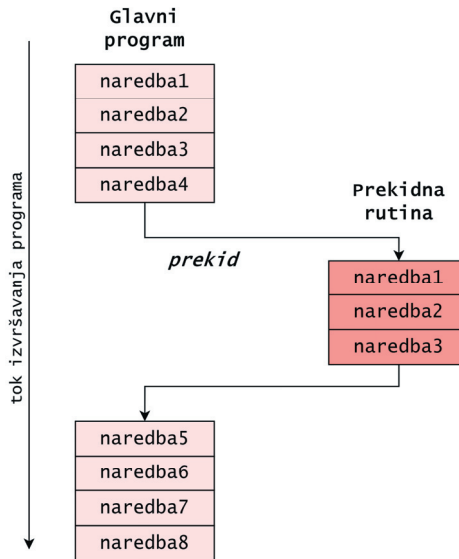
Prekid (eng. *Interrupt*) predstavlja događaj koji zaustavlja trenutni tok izvršavanja programa, kako bi se izvršilo nešto drugo. Kada se prekid dogodi, procesor će kompletirati trenutnu naredbu (instrukciju) i pristupiti tzv. *obradi prekida*. Tom prilikom, procesor zabranjuje nove prekide, čuva povratnu adresu i izvršava prekidnu rutinu. Posmatrano sa implementacione strane, prekidna rutina (eng. *interrupt service routine*) predstavlja funkciju u okviru koje se izvršavaju željene akcije definisane od strane korisnika. Na primer, kada se dogodi prekid usled pristizanja karaktera putem serijskog porta, prekidna rutina će izvršiti njegovo čitanje i smeštanje u prijemni bafer podataka. Koncept prekida je ilustrovan na slici 3.1.

Postoji dve osnovne podele prekida kod mikrokontrolera. Na osnovu toga da li se prekid može zabraniti ili ne, svi prekidi se mogu podeliti na:

- **maskabilni** (eng. *maskable*) – moguće ih je zabraniti, pomoću nekog jednostavnog mehanizma koji sprovodi programer (najčešće je to postavljanje odgovarajućih bita u statusnim registrima);
- **nemaskabilni** (eng. *nonmaskable*) – nije ih moguće zabraniti (uglavnom su to neki kritični događaji u sistemu).

Druga podela se vrši na osnovu toga da li su prekidi inicirani spoljašnjim ili unutrašnjim događajima u odnosu na procesor. Prema tome, razlikuju se:

- unutrašnji prekidi (eng. *internal interrupts*) (primer: izuzeci prilikom izvršavanja aritmetičkih operacija)
- spoljašnji prekidi (eng. *external interrupts*) (primer: izuzeci usled isteka tajmer/brojač modula)



Slika 3.1: Ilustracija koncepta prekida

Prilikom pisanja prekidnih rutina osnovna preporuka je da one budu *što je moguće kraće*. Razlog za to jeste, što se na taj način sprečava predugačko čekanje na izvršavanje drugih delova programa (na primer, druge prekidne rutine višeg prioriteta). Prema tome, prekidna rutina treba da izvršava samo kritične delove koda. Ostatak je poželjno izvršiti u sklopu glavne petlje. Ipak, isključivo poštovanje ovog pravila ponekad nije dovoljno. Prilikom realizacije složenijih sistema, često je potrebno izvršiti i znatno kompleksniju vremensku analizu trajanja prekidnih rutina.

3.2 Programaska podrška sistemu prekida kod AVR mikrokontrolera

AVR-GCC razvojni alati pružaju mogućnost upravljanja prekidima AVR familije mikrokontrolera na jednostavan način. Ovo će biti ilustrovano primerima koji koriste prekide tajmer/brojač modula 0, kao i prekide izazvane promenom stanja ulaznih pinova (eng. *Pin Change Interrupt*).

Za početak, potrebno je uključiti biblioteku koja podržava rad sa prekidima:

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

Da bi prekid bio omogućen, potrebno je obezbediti sledeće:

1. Mora biti setovan bit za dozvolu prekida u odgovarajućem konfiguracionom registru.

2. Mora biti setovan bit I u statusnom registru. U slučaju kada je I=0 podrazumeva se globalna zabrana prekida. Bit I se setuje pomoći makro funkcije `sei()`, a resetuje pomoću makro funkcije `cli()`.
3. Mora postojati prekidna rutina. Prekidne rutine su funkcije koje se automatski pozivaju prilikom pojave zahteva za prekidom, pod uslovom da je prekid dozvoljen. Definišu se na sledeći način:

```
ISR(vektor_prekida)
{
    //telo prekidne rutine...
}
```

Kao parametar prekidne rutine navodi se vektor prekida, u zavisnosti od toga šta je izvor prekida. Mikrokontroler ATmega328P poseduje 25 vektora (izvora) prekida + reset vektor. Ovi vektori su smešteni na početku programske memorije u tzv. *tabeli vektora prekida*, počevši od reset vektora prekida koji se nalazi na adresi 0. U okviru tabele vektora prekida, nalaze se i početne adrese prekidnih rutina namenjenih za obradu prekida. Kod AVR mikrokontrolera, ove adrese su fiksne, odnosno ne mogu se menjati od strane korisnika. Redosled ovih vektora u tabeli ujedno definiše i njihov prioritet, tako da je vektor koji se nalazi na nižoj adresi, većeg prioriteta (prema tome, reset vektor je vektor sa najvišim prioritetom).

Svi izvori prekida kod AVR mikrokontrolera se mogu podeliti u sledeće grupe:

1. spoljašnji (eksterni) prekidi;
2. prekidi na osnovu promene stanja na pinu;
3. prekidi *watchdog* tajmera;
4. prekidi tajmer/brojač modula 0, 1 i 2;
5. prekidi serijskih interfejsa (USART, SPI i TWI);
6. prekidi analognih modula (AD konvertor i analogni komparator);
7. sistemski prekidi (reset, EEPROM i SPM ready).

Definicije vektora prekida navedene su u nastavku (listing 3.1) i dostupne su korisniku ukoliko je u okviru programa uključena biblioteka *avr/interrupt.h*.

```
/* Interrupt Vectors */
/* Interrupt Vector 0 is the reset vector. */

#define INTO_vect    _VECTOR(1) /* External Interrupt Request 0 */
#define INT1_vect    _VECTOR(2) /* External Interrupt Request 1 */

#define PCINT0_vect  _VECTOR(3) /* Pin Change Interrupt Request 0 */
#define PCINT1_vect  _VECTOR(4) /* Pin Change Interrupt Request 1 */
```

Glava 4

Tajmer/brojač modul kod mikrokontrolera

4.1 Uvod

Mikrokontroler ATmega328P, iz familije AVR mikrokontrolera, poseduje tri tajmer/brojačka (eng. *Timer/Counter*) modula. To su 8-bitni tajmer/brojač modul 0, 16-bitni tajmer/brojač modul 1 i 8-bitni tajmer/brojač modul 2. Iako se tri tajmera međusobno razlikuju, koncept njihovog dizajna je vrlo sličan, zbog čega će u okviru ovog poglavlja biti objašnjen samo tajmer/brojač 0.

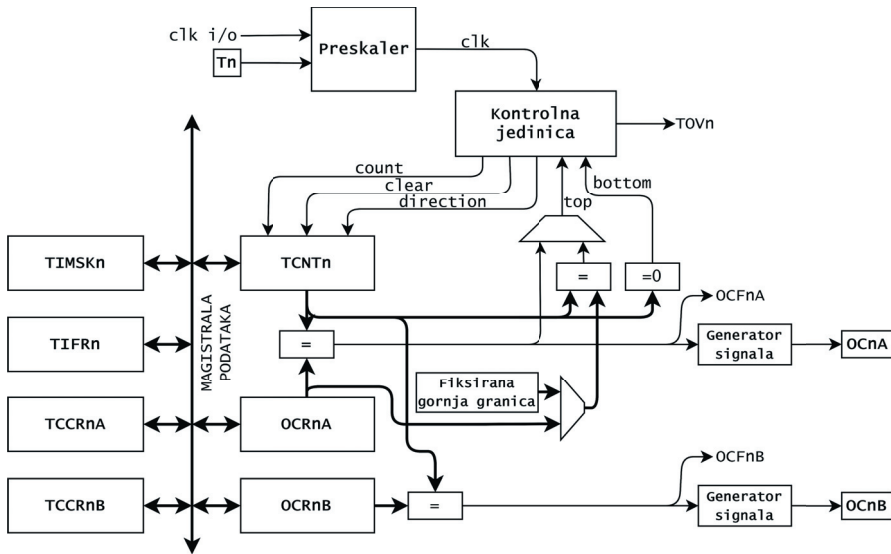
Osobine koje odlikuju tajmer/brojač 0 su:

- dve nezavisne jedinice za poređenje izlaza;
- baferovani izlazni registri;
- automatska reinicijalizacija brojača na vrednost 0 nakon uspešnog poređenja;
- generisanje impulsno širinske modulacije bez gličeva;
- promenljiva perioda impulsno širinske modulacije;
- tri nezavisna izvora prekida (COMP_A, COMP_B i OVF).

U nastavku je objašnjena struktura tajmer/brojač modula, osnovni konfiguracioni registri i način njegove upotrebe. Detaljno objašnjenje ovog modula je moguće pronaći i u tehničkoj dokumentaciji ATmega328P mikrokontrolera [2].

4.2 Struktura tajmer/brojač modula kod AVR mikrokontrolera

Blok šema prikazana na slici 4.1, predstavlja strukturu tajmer/brojač modula 0 mikrokontrolera ATmega328P. Ova shema se u velikoj meri podudara sa



Slika 4.1: Struktura tajmer/brojač modula mikrokontrolera ATmega328P

shemama preostala dva tajmer/brojačka modula, gde postoje neznatne razlike u funkcionalnosti. Simbol n se može zemeniti sa vrednostima 0, 1 i 2, a x se može zameniti sa A i B, čime se dobijaju imena registara za odgovarajuće tajmer/brojač module. Ovaj modul se može podeliti na nekoliko funkcionalnih celina:

- *Brojačka jedinica* (eng. *Counter unit*) – je zadužena za kontrolu rada bidirekcionog brojačkog registra. Ovu jedinicu sačinjavaju preskaler, kontrolna jedinica, bidirekciono brojački registar TCNT n i logika za generisanje *top* i *bottom* signala.
- *Jedinica za poređenje izlaza* (eng. *Output compare unit*) – su zadužene za generisanje izlaznih signala i odgovarajućih prekida. Sačinjavaju je bidirekciono brojački registar TCNT n , registari za poređenje izlaza OCR n x, komparator, generator signala i izlazni pin OC n x.
- *Konfiguracioni registri* – su namenjeni za podešavanje načina funkcionisanja tajmer/brojača. U ove registre spadaju TIMSK n , TIFR n , TCCR n A, TCCR n B, OCR n A i OCR n B.

4.2.1 Brojačka jedinica

Glavni zadatak brojačke jedinice je upravljanje tokom brojanja, što uglavnom predstavlja kontrolisanje rada bidirekcionog brojačkog registra TCNT n . Za

kontrolu ovog dela sistema zadužena je kontrolna jedinica, koja ima 3 ulazna i 4 izlazna jednobitna signala. To su:

- *clk* – ulazni taktni signal na osnovu čije aktivne ivice se inicira inkrementovanje ili dekrementovanje bidirekcionog brojačkog registra TCNTn. Ovaj signal se generiše u okviru preskalera.
- *TOVn* – izlazni signal koji postavlja istoimeni bit koji označava pojavu prekida usled prekoračenja opsega brojanja.
- *count* – izlazni signal koji inicira inkrementovanje ili dekrementovanje bidirekcionog brojačkog registra TCNTn.
- *clear* – izlazni signal kojim se postavlja stanje brojačkog registra TCNTn na vrednost 0.
- *direction* – izlazni signal koji označava smer u kojem je potrebno da broji bidirekciono brojački registar TCNTn. Na osnovu ovog signala se određuje da li će pojavom aktivne ivice signala count doći do inkrementovanja ili dekrementovanja bidirekcionog brojačkog registra TCNTn.
- *top* – ulazni signal koji označava da je brojački registar TCNTn dostigao gornju granicu brojanja.
- *bottom* – ulazni signal koji označava da je brojački registar TCNTn dostigao donju granicu brojanja.

Logika koja određuje da li je dostignuta donja ili gornja granica brojanja se sastoji od dva komparatora i dva multipleksera. Donja granica se dobija poređenjem sadržaja brojačkog registra TCNTn sa minimalnom vrednošću koju može da ima ovaj registar, što je 0. Logika za generisanje gornje granice je nešto složenija. Na shemi 4.1 su izostavljeni selekcionni signali za dva multipleksera. Konfiguracijom bita u kontrolnim registrima se određuje koja vrednost će predstavljati gornju granicu brojanja. Ona može biti:

- rezultat komparatora koji poredi vrednosti bidirekcionog registra TCNTn i izlaznog registra za poređenje OCRnA.
- rezultat komparatora koji poredi vrednosti bidirekcionog registra TCNTn i fiksirane gornje granice, u kojoj svi biti imaju vrednost 1, što u slučaju 8-bitnog tajmer/brojačkog modula iznosi *0xFF*.

Preskaler

Preskaler predstavlja komponentu sa ulogom da obezbedi taktni signal na osnovu kog se inicira brojanje tajmer/brojača. Ova komponenta se može konfigurisati na način da, taktni signal koji generiše bude proizvod različitih događaja. Izvor ovog taktnog signala može biti interni taktni signal mikrokontrolera

Glava 5

Organizacija projekata sa više izvornih datoteka

5.1 Uvod

U svim dosadašnjim primerima, razvijane su jednostavnije aplikacije koje su realizovane pisanjem izvornog koda koji je smešten u jednu izvornu (.c) datoteku. U slučaju složenijih aplikacija, programski kod se deli u više logičkih celina koje pripadaju različitim izvornim datotekama. Tipičan primer programskih logičkih celina su biblioteke sa drajverskim rutinama za različite tipove perifernih uređaja. Na ovaj način postiže se bolja preglednost koda, kao i mogućnost ponovnog korišćenja jednom napisanog koda u više različitih projekata (eng. *code reuse*). Ovakav način pisanja koda je izuzetno cenjen u industriji i predstavlja nešto čemu treba težiti prilikom realizacije praktičnih projekata [4].

Izvorni kod napisan u programskom jeziku C smešta se u dva tipa datoteka:

- **.h datoteke** sadrže deklaracije (prototipove) funkcija, makro konstante, makro funkcije, konstante, definicije tipova podataka korišćenih prilikom implementacije funkcija i sl.
- **.c datoteke** sadrže definicije (implementacije) funkcija deklariranih u okviru .h datoteke, kao i varijabli korišćenih u okviru implementacije.

U nastavku je prikazan primer jednostavnog, dobro organizovanog, projekta koji sadrži ukupno pet izvornih datoteka:

- *main.c* (sadrži glavni program, odnosno funkciju *main*)
- *addition.h*
- *addition.c*

- *subtraction.h*
- *subtraction.c*

Sadržaj ovih datoteka je prikazan na listinzima koji slede u nastavku.

```
#include <stdint.h>
#include "addition.h"
#include "subtraction.h"

void main()
{
    int16_t a;
    a = add(1, subtract(4, 2));

    while(1)
        ;
}
```

Listing 5.1: Datoteka main.c

```
int16_t add(int16_t a, int16_t b);
```

Listing 5.2: Datoteka addition.h

```
#include "addition.h"

int16_t add(int16_t a, int16_t b)
{
    return (a + b);
}
```

Listing 5.3: Datoteka addition.c

```
int16_t subtract(int16_t a, int16_t b);
```

Listing 5.4: Datoteka subtraction.h

```
#include "subtraction.h"

int16_t subtract(int16_t a, int16_t b)
{
    return (a - b);
}
```

Listing 5.5: Datoteka subtraction.c

Kao što se može uočiti sa prethodnih listinga, čitav projekat je dekomponovan na tri logičke celine: deo koji implementira sabiranje dve vrednosti, deo koji

Glava 6

Analogno-digitalna konverzija AVR mikrokontrolera

6.1 Uvod

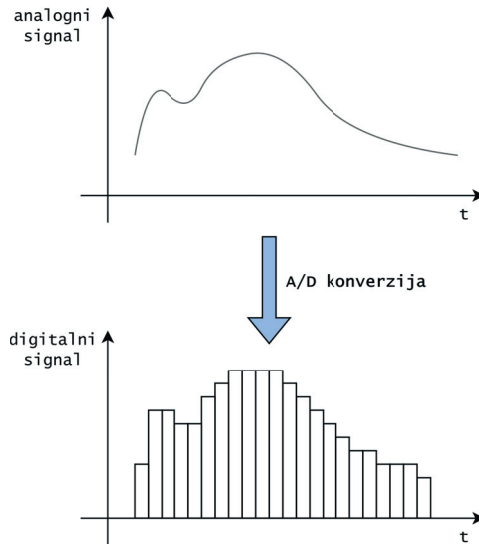
U praktičnim aplikacijama često se javlja potreba za očitavanjem različitih vrednosti (temperatura, pritisak, vlažnost, napon itd.) u analognom formatu. Međutim, obrada takvih podataka prilično je neefikasna u pogledu brzine. Takođe, mikroprocesorski sistemi, ne mogu da obrađuju ovakvu vrstu podataka. Kako bi se ovo prevazišlo, koristi se postupak, poznat pod nazivom *analogno-digitalna konverzija*.

Proces konverzije analognog signala u digitalni signal, koji jedan mikroprocesorski sistem može da obradi, se sastoji iz dva osnovna koraka:

- diskretizacija u vremenu (*odabiranje*);
- diskretizacija u amplitudi (*kvantizacija*).

U procesu odabiranja, vrši se konverzija signala kontinualnog u vremenu u diskretni signal, tako što se vrednosti polaznog signala uzimaju u unapred definisanim vremenskim intervalima. Sa druge strane, u procesu kvantizacije vrši se konverzija polazne kontinualne amplitude u unapred definisan (konačan) skup mogućih vrednosti. Kako se broj bita za reprezentaciju datih vrednosti povećava, tačnost samog procesa kvantizacije se, takođe, povećava, odnosno, greška kvantizacije se smanjuje. Idealno, ukoliko bi broj bita težio beskonačnosti, govorili bismo o idealnom procesu konverzije.

Prema tome, analogno-digitalna konverzija predstavlja proces pretvaranja analognog signala u neku digitalnu vrednost (ceo broj). Ilustracija procesa konverzije je prikazana na slici 6.1.



Slika 6.1: Analogno-digitalna konverzija

Za realizaciju opisanog procesa koristi se modul unutar AVR mikrokontrolera koji se naziva AD (analogno-digitalni) konvertor. Vrednost analognog signala je moguće čitati sa jednog od 6 analognih ulaza koji se nalaze na Arduino UNO razvojnom sistemu. Što se samog ADC modula tiče, on poseduje 10-bitnu tačnost, odnosno rezoluciju. Ovo znači da, na izlazu, modul može da prikaže vrednosti iz intervala $0-(2^{10}-1)$ sa preciznošću od $\pm 2LSB$. Napon na ulazu ADC modula može da uzme vrednost iz intervala $0-V_{cc}$. Tipično trajanje konverzije iznosi 13 perioda taktnog signala na frekvenciji rada ADC modula (koja se podešava pomoću preskalera). Izuzetak je inicijalna konverzija (nakon pokretanja modula), koja traje 25 perioda taktnog signala usled potrebe za inicijalizacijom pojedinih komponenti u okviru modula. Generalna struktura ADC modula je ilustrovana na slici 6.2.

Osnovni registri neophodni za rad sa ADC modulom su:

- **ADMUX** – *ADC Multiplexer Selection Register*
- **ADCSRA** – *ADC Control and Status Register A*
- **ADC** – *ADC Data Register*

Izbor ulaza sa kog je potrebno očitati analognu vrednost se vrši postavljanjem MUX bita unutar ADMUX registra. Dozvolu rada ADC modula je moguće postaviti pomoću ADEN bita u okviru kontrolnog, ADCSRA registra. Kada je ovaj bit postavljen na vrednost 0, ADC modul ne troši energiju, pa je stoga preporučljivo izvršiti isključivanje modula prilikom prelaska u neki od režima

Glava 7

Statičke biblioteke

7.1 Uvod

Problem ponovnog korišćenja jednom napisanog koda u više projekata na elegantan način se rešava kreiranjem statičkih biblioteka. Naime, ideja je ta, da se napisani kod može instancirati u proizvoljnom broju projekata, bez potrebe za kopiranjem bilo kakvih izvornih datoteka. Takođe, eventualne izmene i dopune koje se izvrše u samoj implementaciji biblioteke se prilikom linkovanja automatski odražavaju na sve projekte koji tu biblioteku koriste. Kreiranje i korišćenje statičke biblioteke biće ilustrovano na primeru biblioteke **pin** za manipulaciju pinovima mikrokontrolera, koja je realizovana u Poglavlju 5.

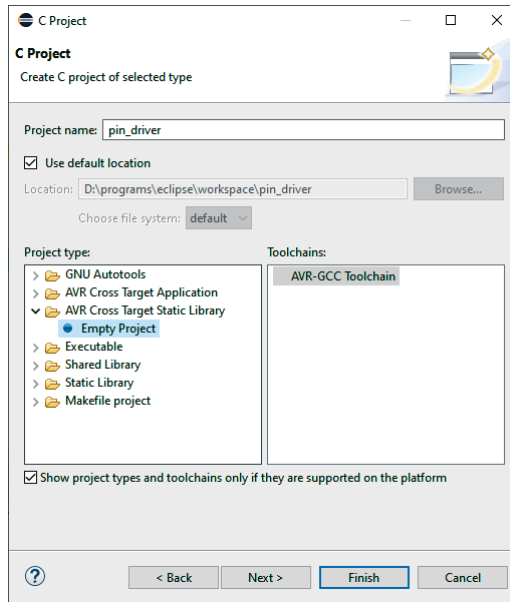
U okviru ovog poglavlja biće ilustrovan način kreiranja statičkih biblioteka pomoću *Eclipse* razvojnog okruženja.

7.2 Kreiranje statičkih biblioteka u razvojnom okruženju *Eclipse*

Za početak, potrebno je kreirati novi projekat, koji u ovom slučaju predstavlja statičku biblioteku, izborom naredne opcije:

New → Project → C project → AVR Cross Target Static Library

U ovom slučaju, biblioteka je nazvana **pin_driver**, a ostale opcije su identične kao i u prethodnim primerima. Ovo je ilustrovano na slici 7.1.



Slika 7.1: Kreiranje statičke biblioteke

Nakon kreiranja projekta, potrebno je dodati datoteke *pin.h* i *pin.c* koje sadrže izvorni kod, naveden na listinzima 7.1 i 7.2.

```

/*
 * pin.h - Datoteka koja deklarise funkcije za upravljanje
 *         pinovima
 */
#ifndef PIN_H_
#define PIN_H_

#include <avr/io.h>
#include <stdint.h>

// Makro za podesavanje visoke vrednosti signala na pinu
#define HIGH 1
// Makro za podesavanje niske vrednosti signala na pinu
#define LOW 0

// Makro za podesavanje izlaznog smera pina
#define OUTPUT 1
// Makro za podesavanje ulaznog smera pina
#define INPUT 0

// Makro za selektovanje porta B
#define PORT_B 0
// Makro za selektovanje porta C

```

```
#define PORT_C 1
// Makro za selektovanje porta D
#define PORT_D 2

/*
 * pinSetValue - Funkcija koja postavlja vrednost na pinu
 */
void pinSetValue(uint8_t port, uint8_t pin, uint8_t value);

/*
 * pinInit - Funkcija koja implementiran inicijalizaciju pina
 */
void pinInit(uint8_t port, uint8_t pin, uint8_t direction);

#endif /* PIN_H_ */
```

Listing 7.1: pin.h datoteka

```
/*
 * pin.c - Datoteka koja definise funkcije za upravljanje
 * pinovima
 */
#include "pin.h"

void pinSetValue(uint8_t port, uint8_t pin, uint8_t value)
{
    // Postavljanje vrednosti pina
    switch(port)
    {
        case PORT_B:
            if (value == HIGH)
                PORTB |= 1 << pin;
            else
                PORTB &= ~(1 << pin);
            break;

        case PORT_C:
            if (value == HIGH)
                PORTC |= 1 << pin;
            else
                PORTC &= ~(1 << pin);
            break;

        case PORT_D:
            if (value == HIGH)
                PORTD |= 1 << pin;
            else
                PORTD &= ~(1 << pin);
            break;
    }
}
```