Radionica digitalnog dizajna II
Dan I

dr. sc. Jurica Kundrata

Centar za svemirsku i inovativnu tehnologiju

17. - 18. lipnja 2023.

Raspored - Dan I

17. lipnja 2023., subota	
9.00 - 12.00	● Kratko ponavljanje sadržaja prošle radionice ● Instalacija lokalne podrške za Verilog jezik ● Brainstorm ideja za zajednički integrirani krug MyChip
12.00 - 12.45	Pauza za ručak
12.45 - 15.00	● Uvod u dizajn procesorske jezgre - Arhitektura i instrukcijski set ● Dizajn modula procesorske jezgre: ○ Registarska mapa ○ Memorijski sustav

Raspored - Dan II

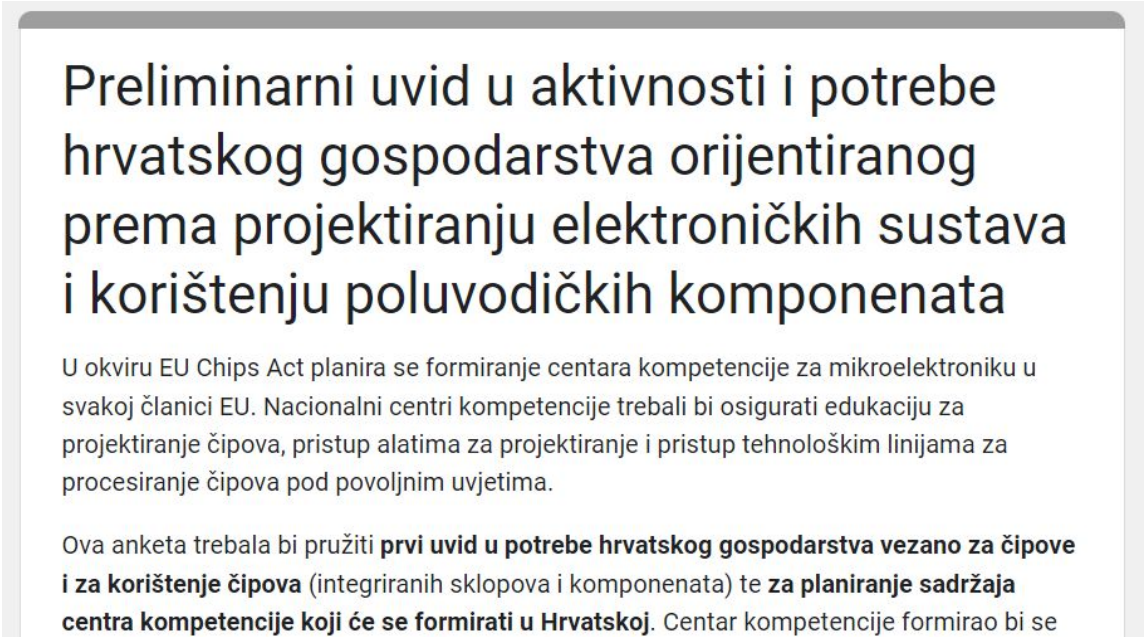
18. lipnja 2023., nedjelja	
9.00 - 12.00	● Dizajn modula procesorske jezgre: ○ Upravljačka jedinica i Aritmetičko-logička jedinica ○ Brojač programa i Dekoder naredbi
12.00 - 12.45	Pauza za ručak
12.45 - 15.00	● Integracija modula procesorske jezgre ● Sinteza i implementacija projekta na FPGA razvojnoj ploči ● Verifikacija projekta

Dan I

Vijesti iz zemlje i svijeta

Vijesti iz RH

- EU Chips Act i RH → Centar kompetencija za integrirane krugove u RH



Preliminarni uvid u aktivnosti i potrebe hrvatskog gospodarstva orijentiranog prema projektiranju elektroničkih sustava i korištenju poluvodičkih komponenata

U okviru EU Chips Act planira se formiranje centara kompetencije za mikroelektroniku u svakoj članici EU. Nacionalni centri kompetencije trebali bi osigurati edukaciju za projektiranje čipova, pristup alatima za projektiranje i pristup tehnološkim linijama za procesiranje čipova pod povoljnim uvjetima.

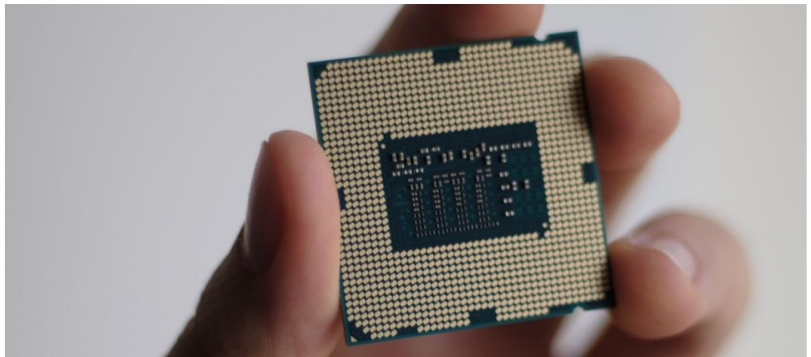
Ova anketa trebala bi pružiti **prvi uvid u potrebe hrvatskog gospodarstva vezano za čipove i za korištenje čipova** (integriranih sklopova i komponenata) te **za planiranje sadržaja centra kompetencije koji će se formirati u Hrvatskoj**. Centar kompetencije formirao bi se

Vijesti iz EU

- EU Chips Act i ‘onshoaring’ u EU:
  - https://www.poslovni.hr/strane/intel-ulaze-46-milijardi-dolara-u-novi-pogon-u-europi-i-zaposljava-2-000-radnika-4394643

Intel ulaže 4,6 milijardi dolara u novi pogon u Europi i zapošljava 2.000 radnika

Autor: Poslovni.hr/Hina , 16. lipanj 2023. u 13:09 0 komentara



Sadržaj

- Kratko ponavljanje sadržaja prošle radionice
- Instalacija lokalne podrške za Verilog jezik
- Brainstorm ideja za zajednički integrirani krug MyChip

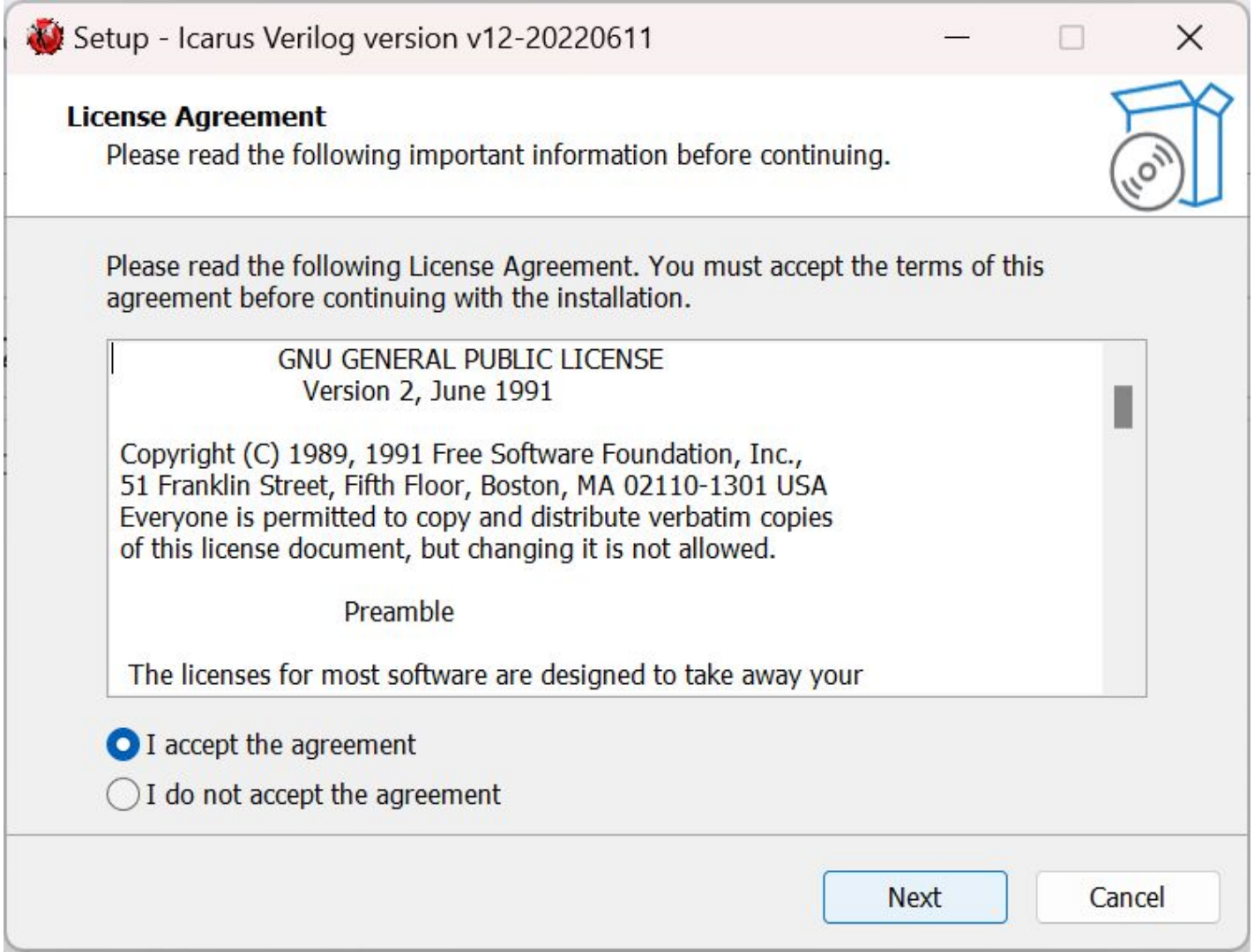
- Uvod u dizajn procesorske jezgre:
  - Arhitektura procesorske jezgre
  - Primjer minimalnog instrukcijskog seta
  - Strojni kod za LED blinky program
- Dizajn modula procesorske jezgre:
  - Registarska mapa
  - Memorijski sustav

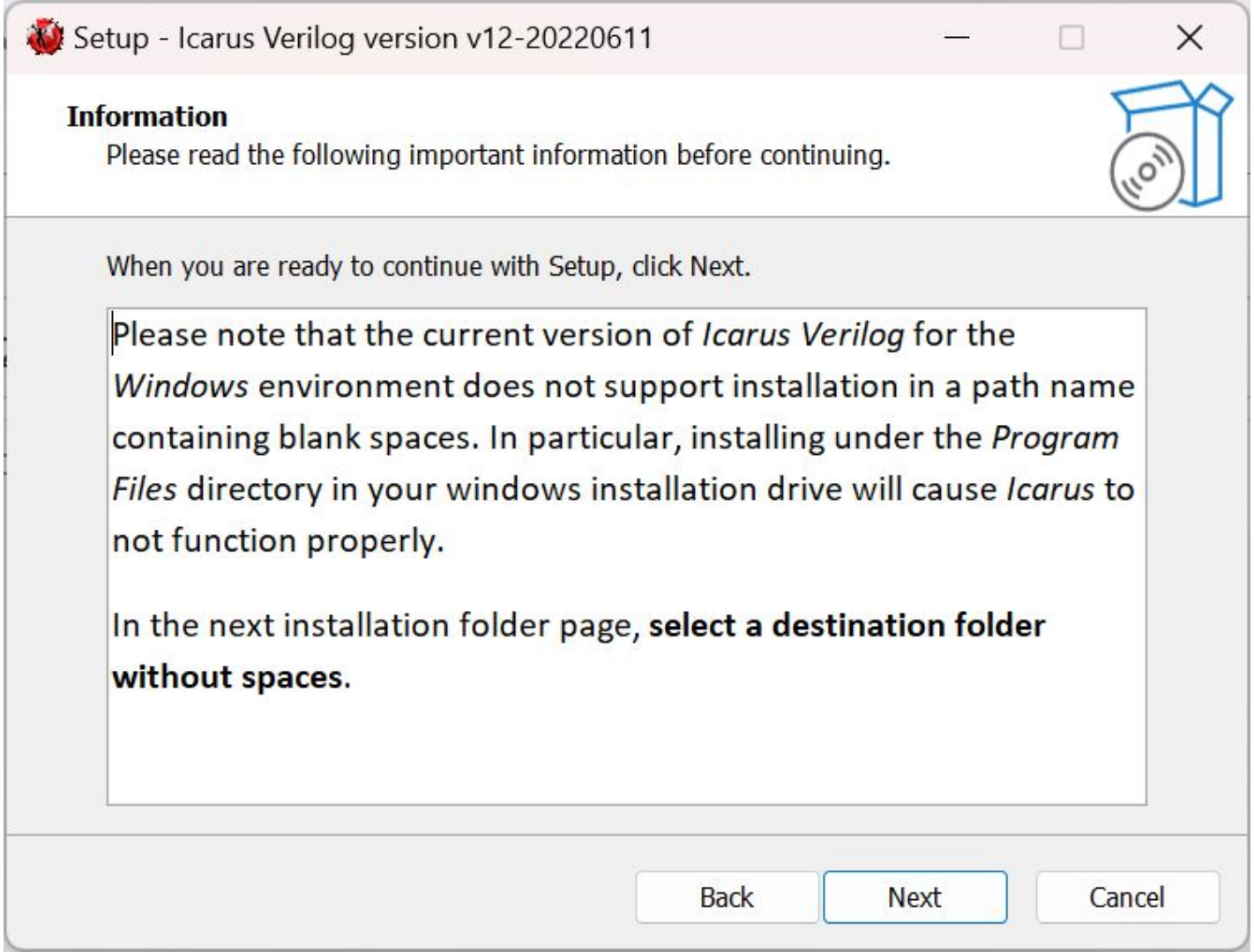
Ponavljanje sadržaja prošle radionice

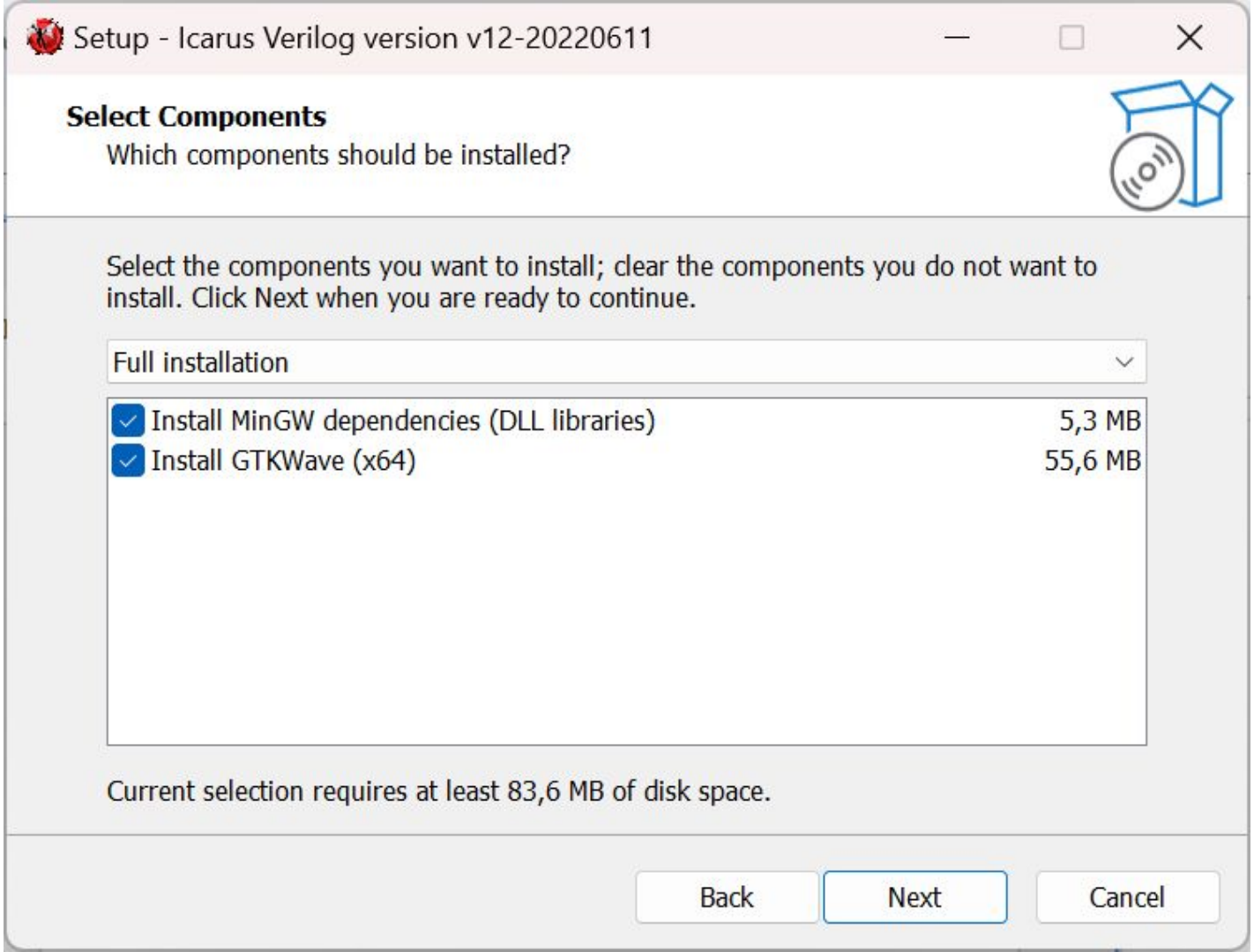
Instalacija lokalne podrške za Verilog

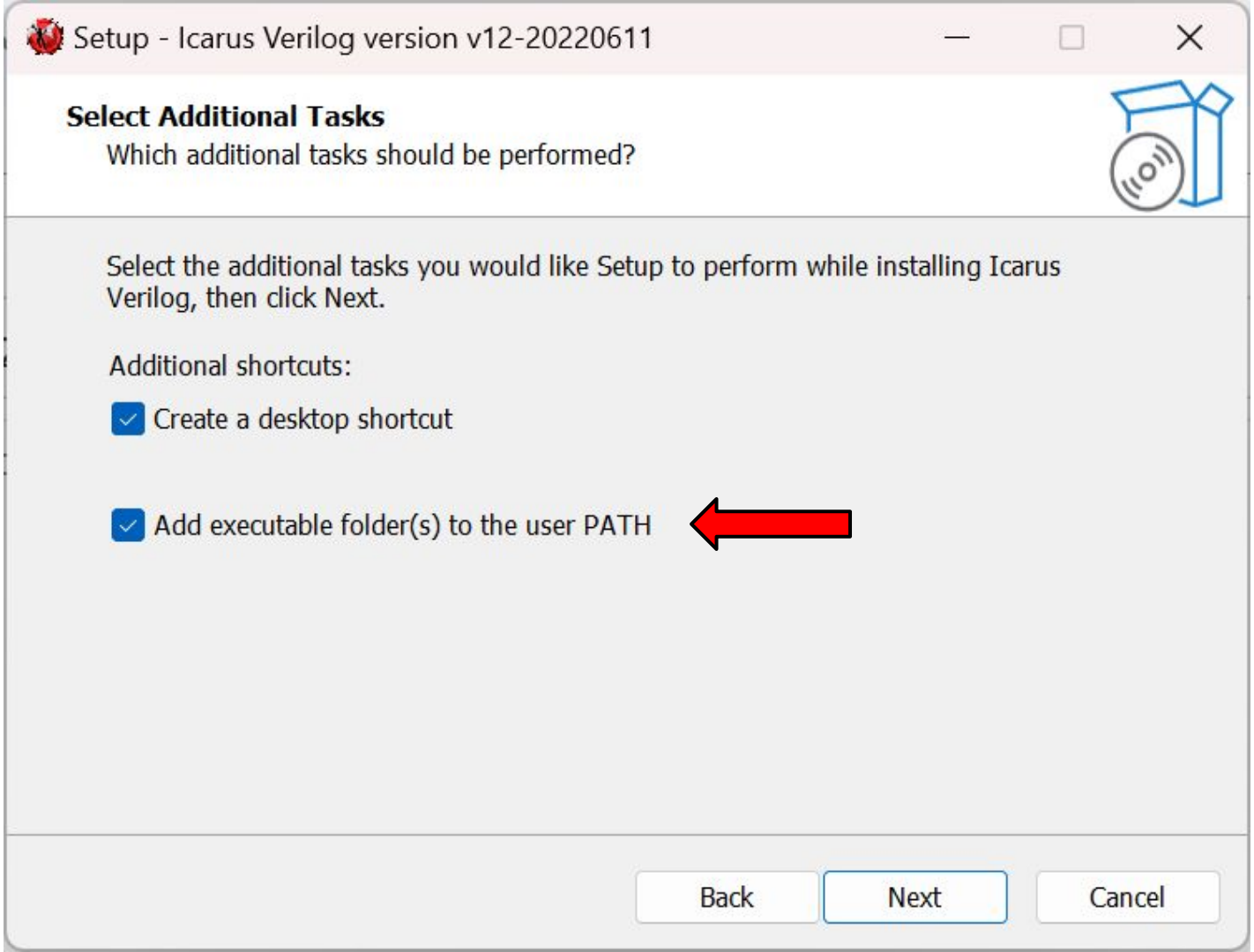
Icarus Verilog

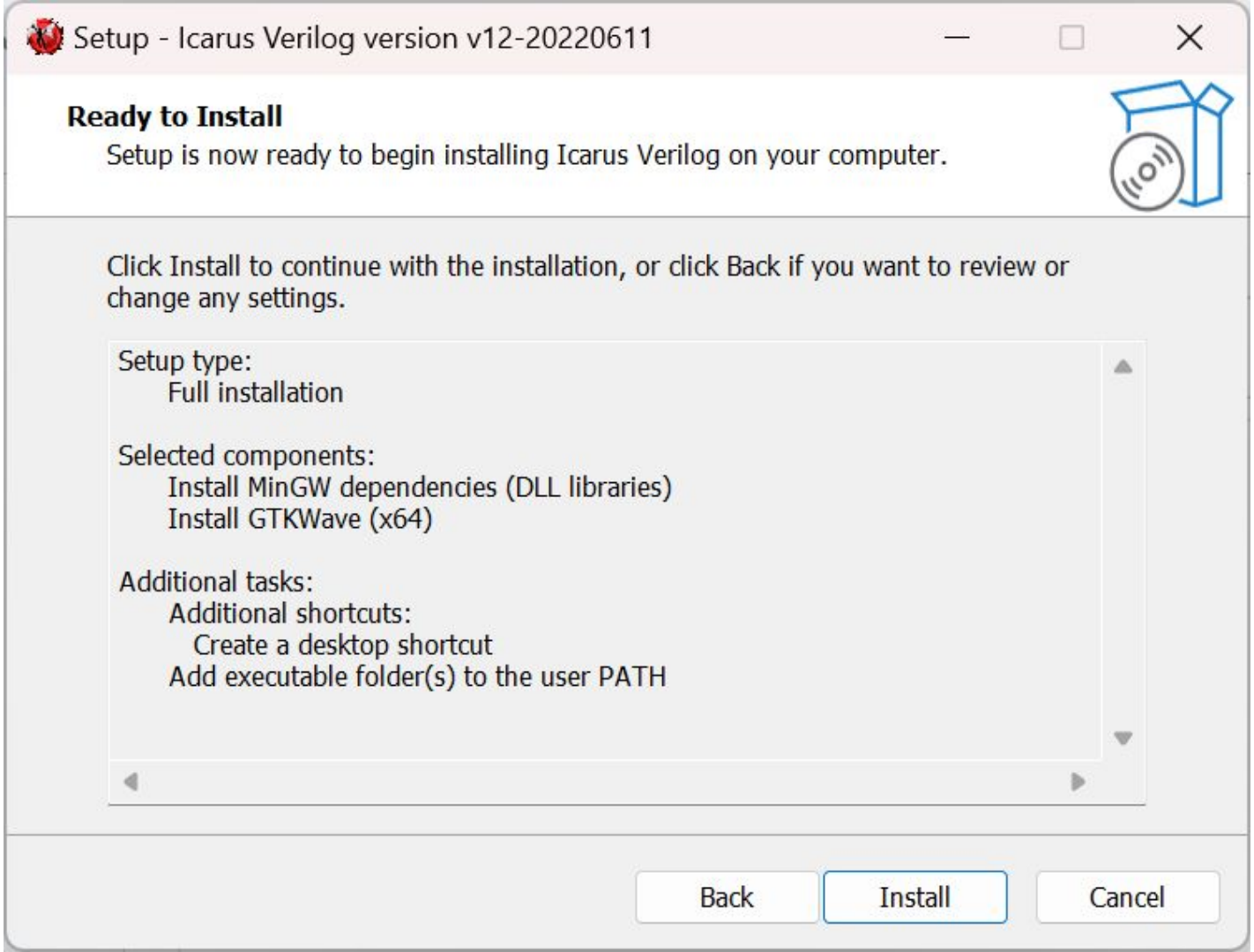
- https://bleyer.org/icarus/
- https://bleyer.org/icarus/iverilog-v12-20220611-x64_setup.exe


Setup - Icarus Verilog version v12-20220611
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
Preamble
The licenses for most software are designed to take away your
I accept the agreement
I do not accept the agreement
Next
Cancel


Setup - Icarus Verilog version v12-20220611
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
Please note that the current version of *Icarus Verilog* for the *Windows* environment does not support installation in a path name containing blank spaces. In particular, installing under the *Program Files* directory in your windows installation drive will cause *Icarus* to not function properly.
In the next installation folder page, **select a destination folder without spaces**.
Back
Next
Cancel


Setup - Icarus Verilog version v12-20220611
Select Components
Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Full installation
Install MinGW dependencies (DLL libraries) 5,3 MB
Install GTKWave (x64) 55,6 MB
Current selection requires at least 83,6 MB of disk space.
Back
Next
Cancel


Setup - Icarus Verilog version v12-20220611
Select Additional Tasks
Which additional tasks should be performed?
Select the additional tasks you would like Setup to perform while installing Icarus Verilog, then click Next.
Additional shortcuts:
Create a desktop shortcut
Add executable folder(s) to the user PATH
Back
Next
Cancel


Setup - Icarus Verilog version v12-20220611
Ready to Install
Setup is now ready to begin installing Icarus Verilog on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Setup type:
Full installation
Selected components:
Install MinGW dependencies (DLL libraries)
Install GTKWave (x64)
Additional tasks:
Additional shortcuts:
Create a desktop shortcut
Add executable folder(s) to the user PATH
Back
Install
Cancel

Setup - Icarus Verilog version v12-20220611

Completing the Icarus Verilog Setup Wizard

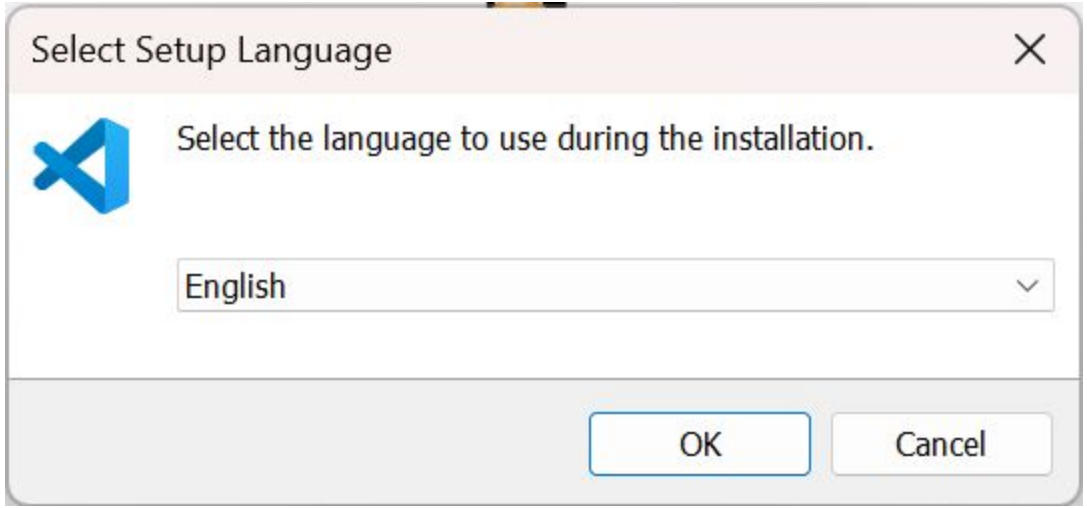
Setup has finished installing Icarus Verilog on your computer. The application may be launched by selecting the installed shortcuts.

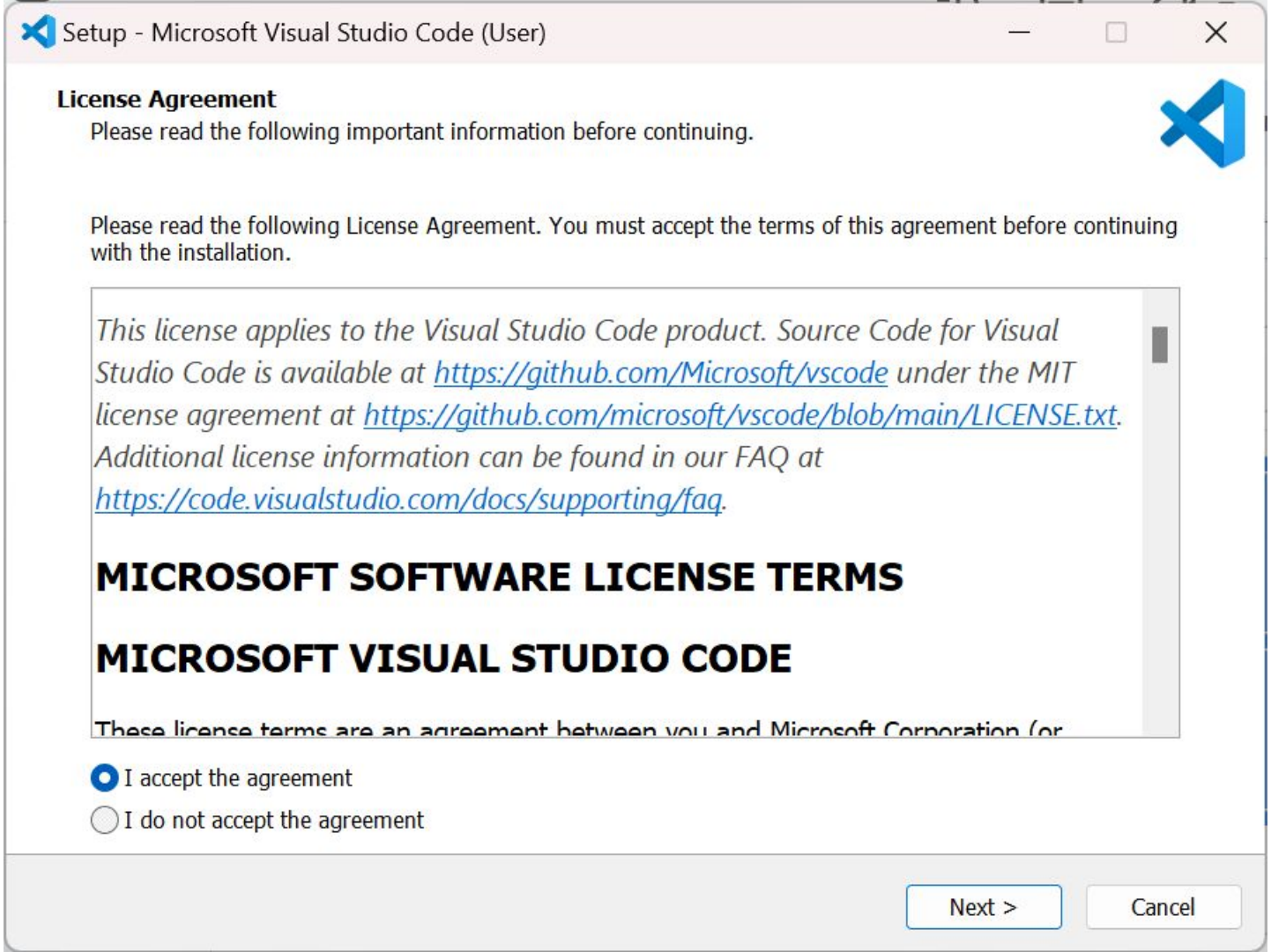
Click Finish to exit Setup.

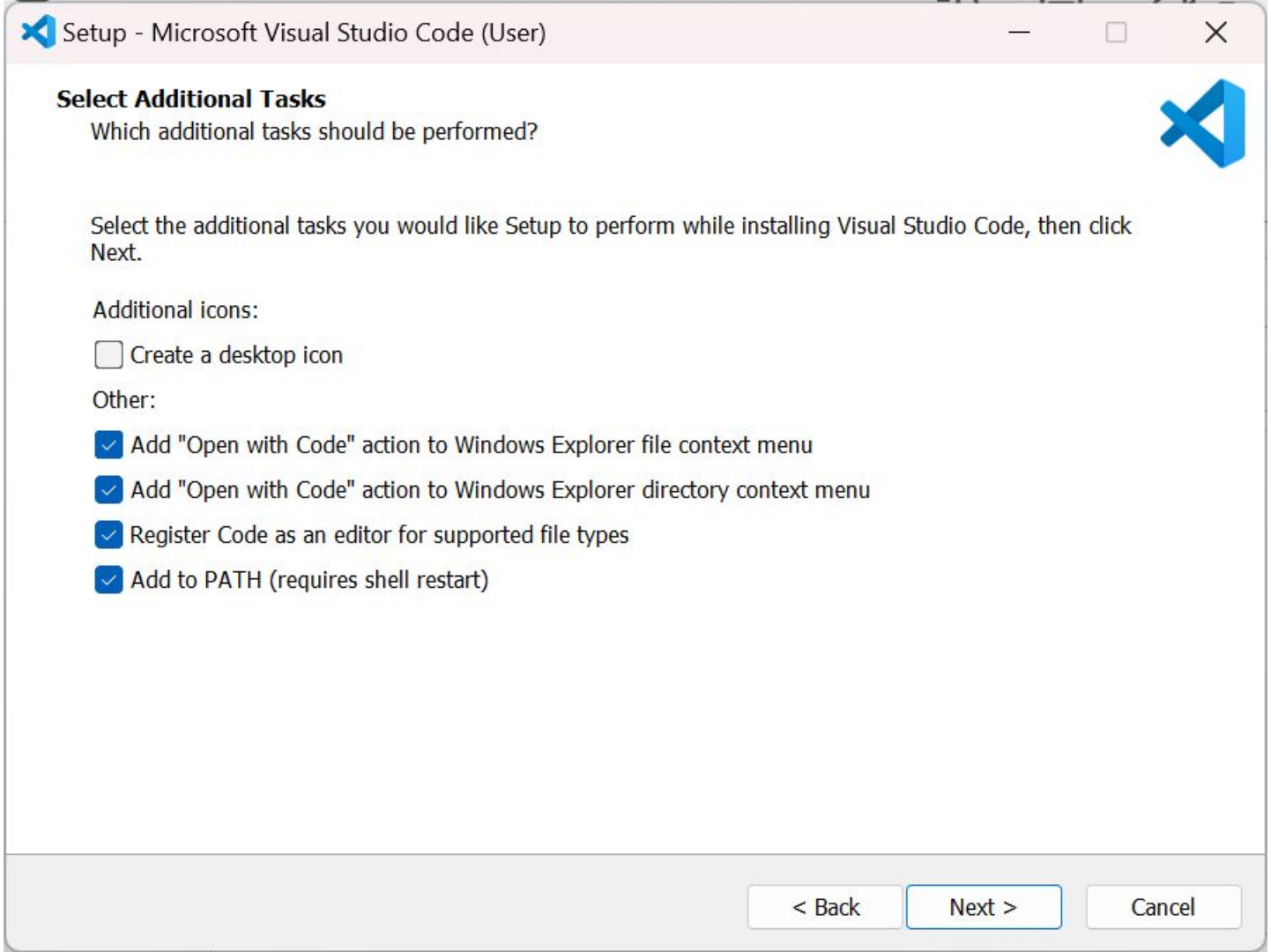
Finish

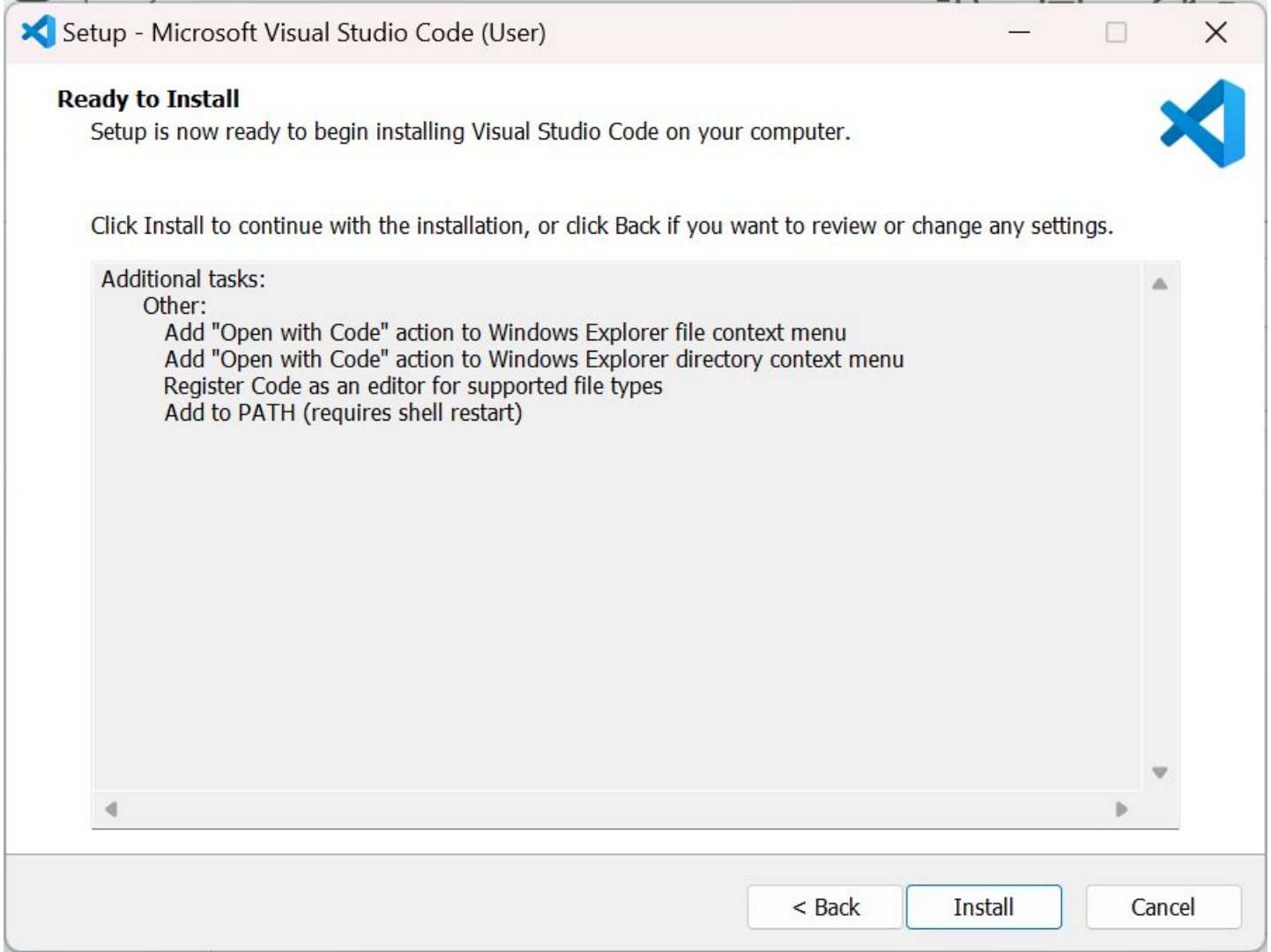
Visual Studio Code

- https://code.visualstudio.com/download


Select Setup Language
Select the language to use during the installation.
English
OK
Cancel


Setup - Microsoft Visual Studio Code (User)
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
This license applies to the Visual Studio Code product. Source Code for Visual Studio Code is available at https://github.com/Microsoft/vscode under the MIT license agreement at https://github.com/microsoft/vscode/blob/main/LICENSE.txt. Additional license information can be found in our FAQ at https://code.visualstudio.com/docs/supporting/faq.
MICROSOFT SOFTWARE LICENSE TERMS
MICROSOFT VISUAL STUDIO CODE
I accept the agreement
I do not accept the agreement
Next >
Cancel


Setup - Microsoft Visual Studio Code (User)
Select Additional Tasks
Which additional tasks should be performed?
Select the additional tasks you would like Setup to perform while installing Visual Studio Code, then click Next.
Additional icons:
Create a desktop icon
Other:
Add "Open with Code" action to Windows Explorer file context menu
Add "Open with Code" action to Windows Explorer directory context menu
Register Code as an editor for supported file types
Add to PATH (requires shell restart)
< Back
Next >
Cancel


Setup - Microsoft Visual Studio Code (User)
Ready to Install
Setup is now ready to begin installing Visual Studio Code on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Additional tasks:
Other:
Add "Open with Code" action to Windows Explorer file context menu
Add "Open with Code" action to Windows Explorer directory context menu
Register Code as an editor for supported file types
Add to PATH (requires shell restart)
< Back
Install
Cancel

Setup - Microsoft Visual Studio Code (User)

Completing the Visual Studio Code Setup Wizard

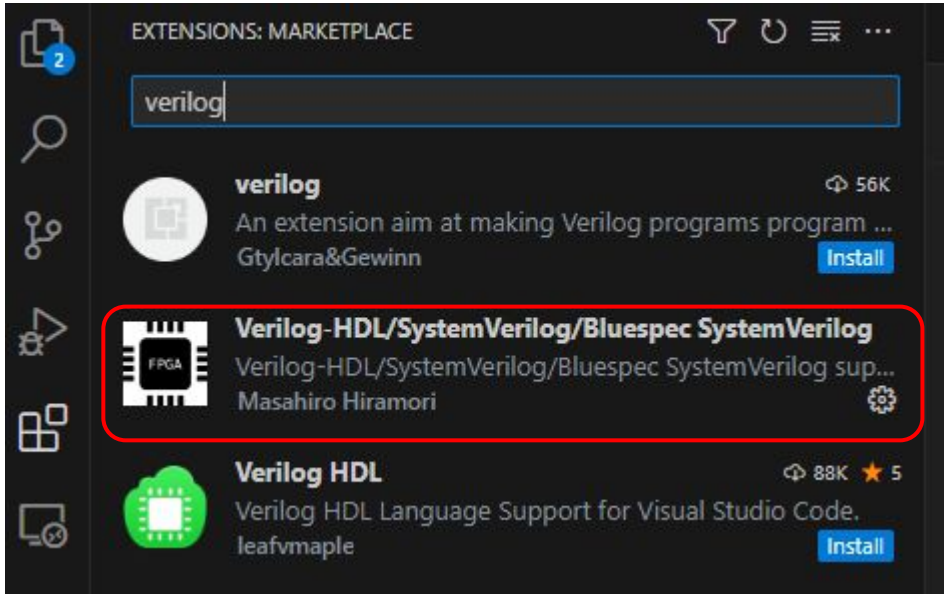
Setup has finished installing Visual Studio Code on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

☑ Launch Visual Studio Code

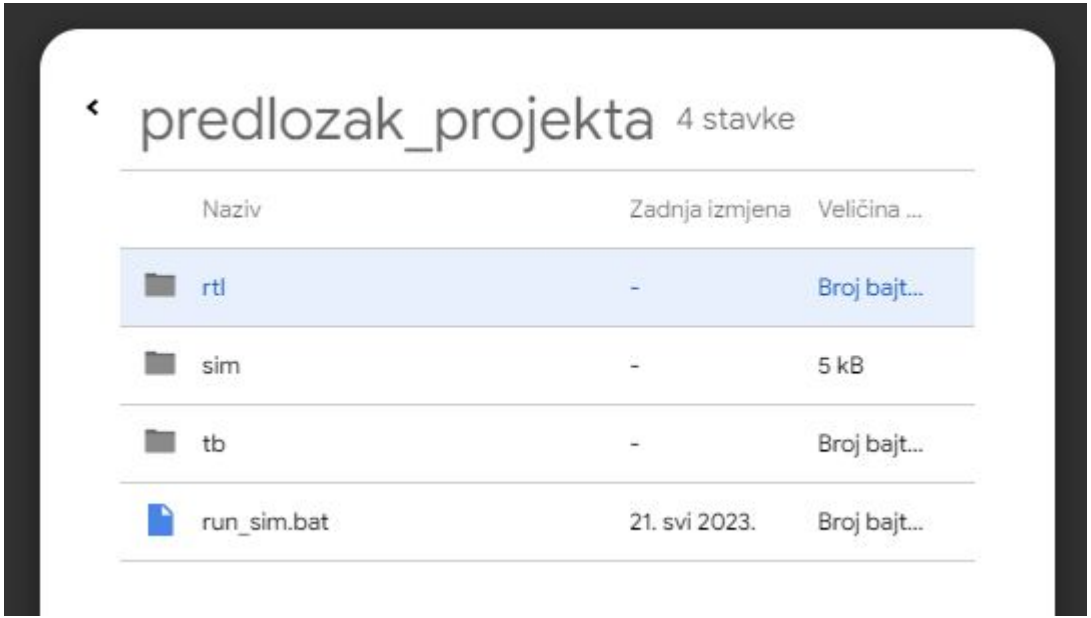
Finish

Ekstra: Ekstenzija za Verilog



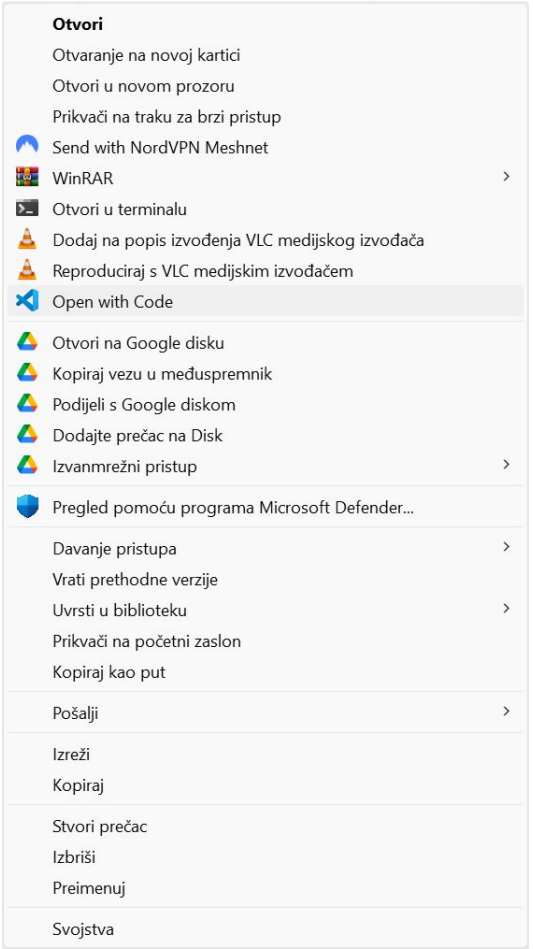
Predložak projekta

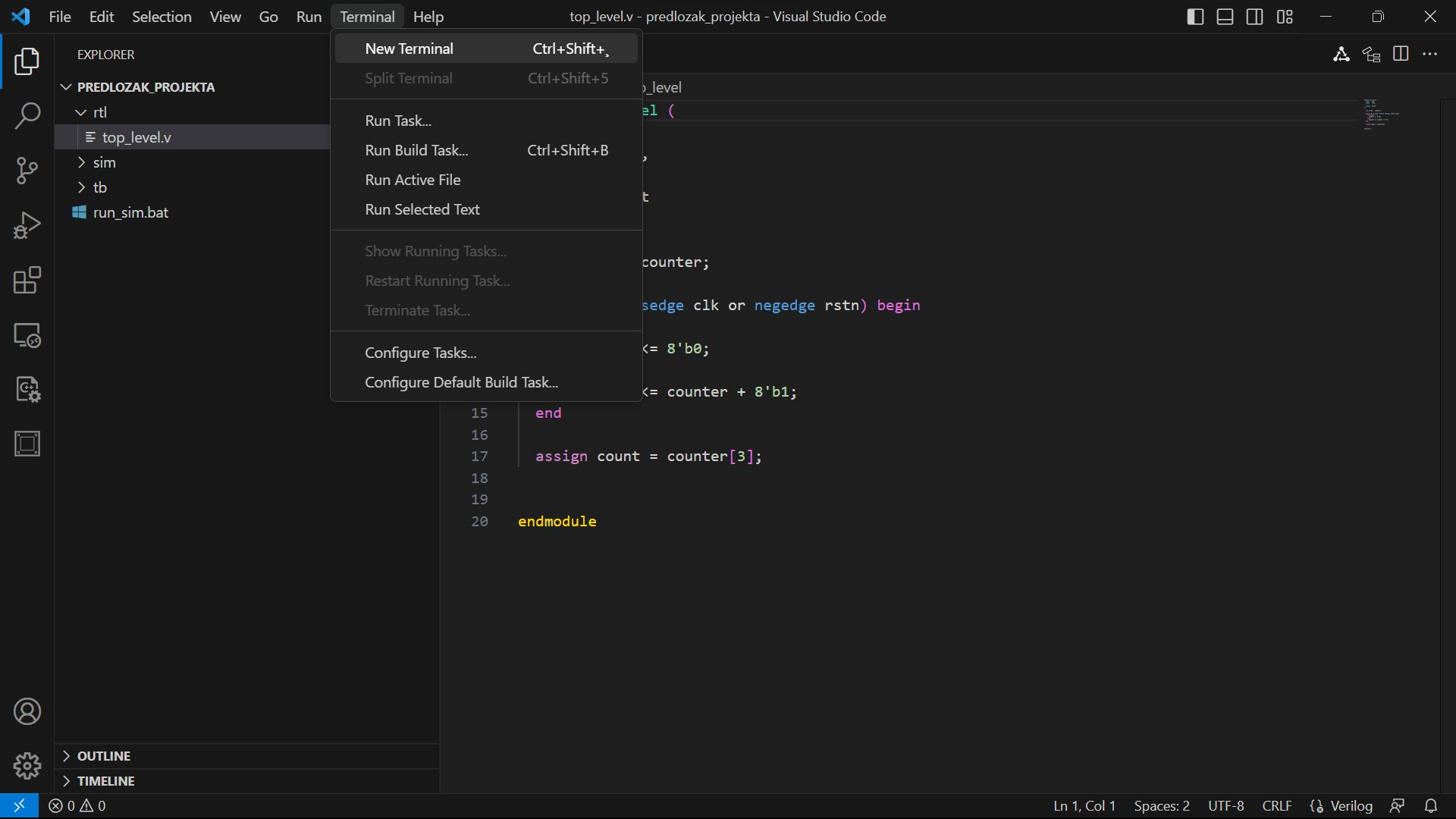
- https://tinyurl.com/f87n8sda

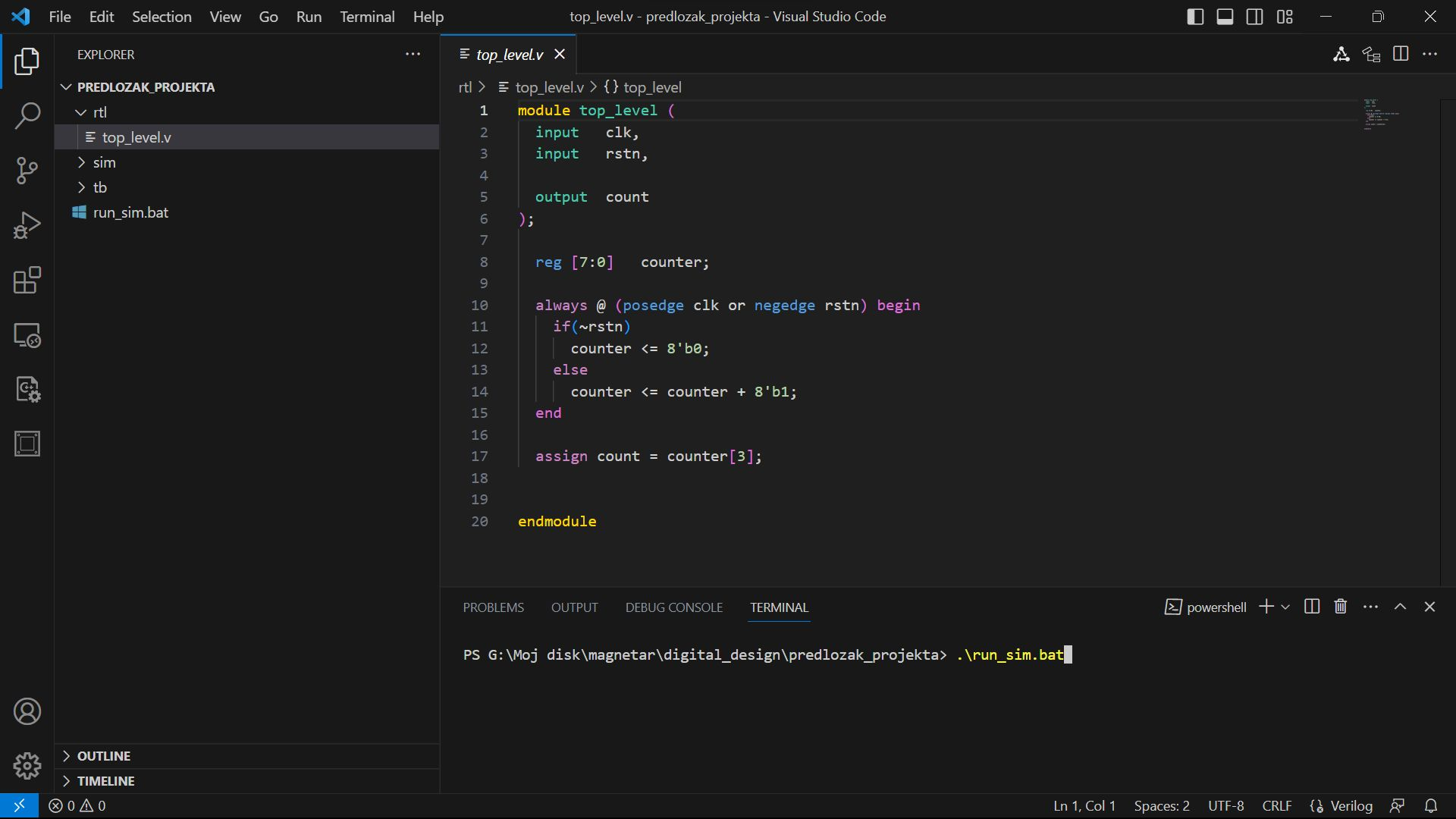


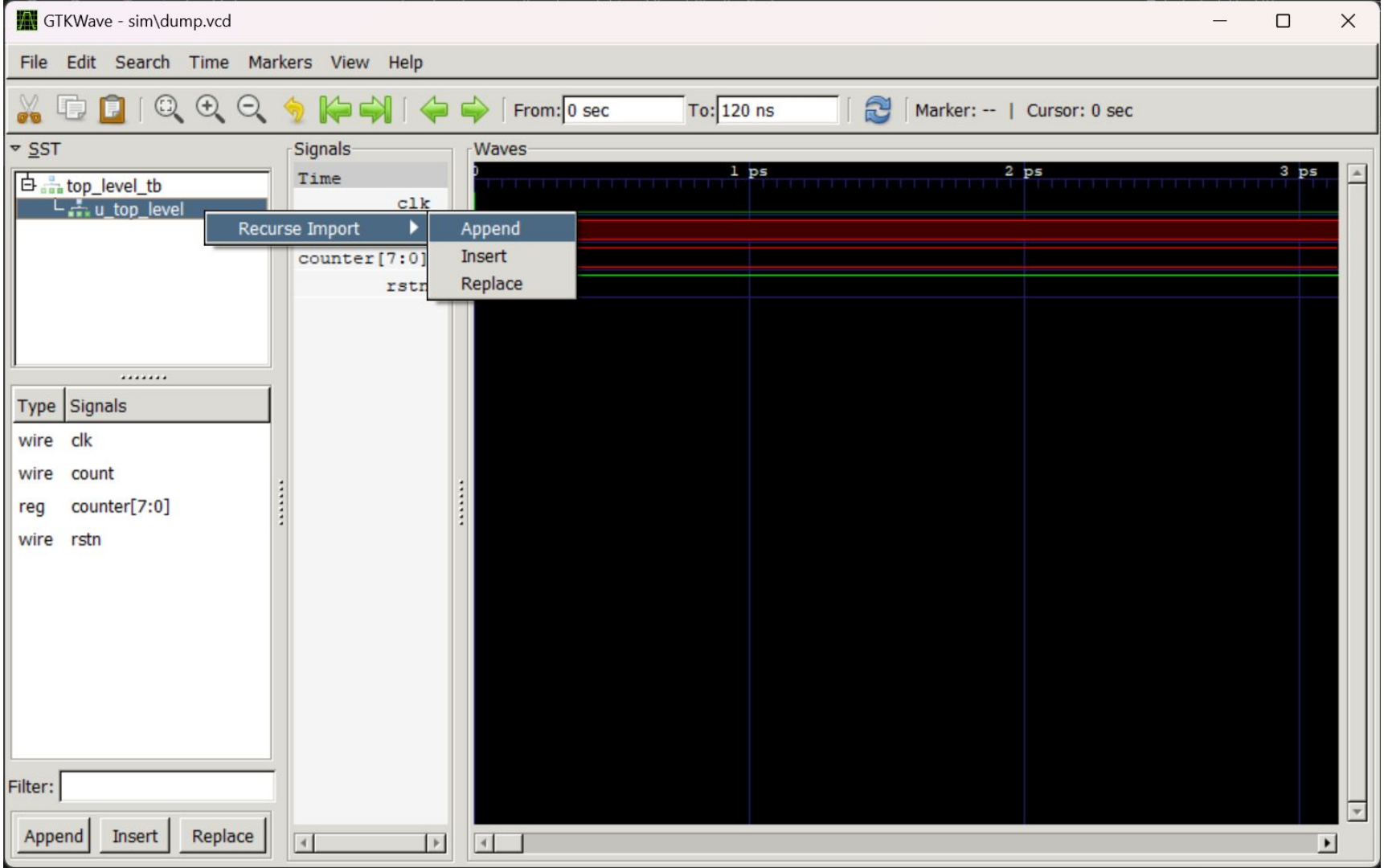
Radni tijek

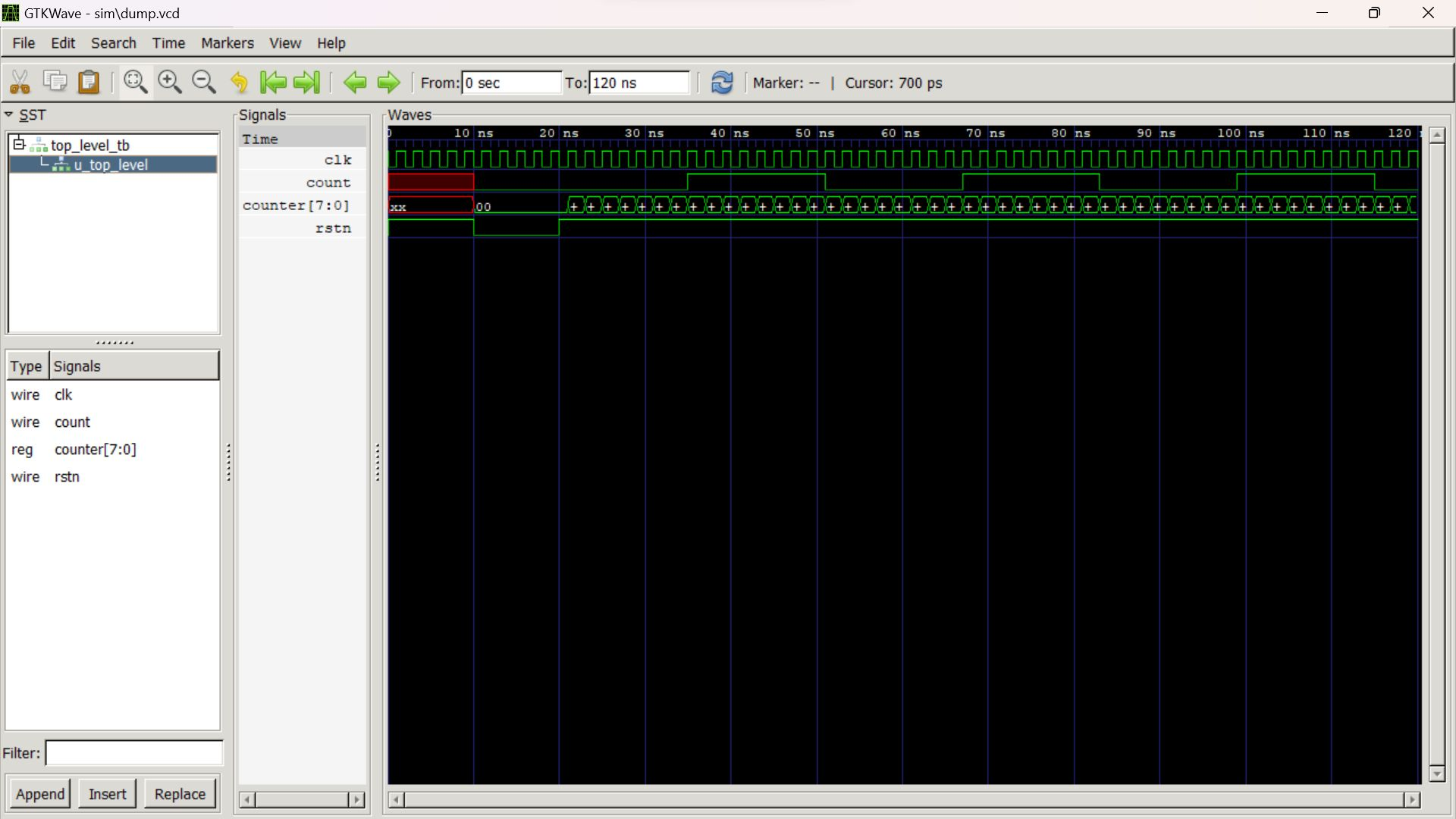
- Otvaranje predloška projekta
- Kodiranje u Visual Studio Code
- Pokretanje simulacije iz terminala batch skriptom
- Pregled rezultata simulacije u GTKwave-u




File Edit Selection View Go Run Terminal Help
top_level.v - predlozak_projekta - Visual Studio Code
New Terminal Ctrl+Shift+`
Split Terminal Ctrl+Shift+5
Run Task...
Run Build Task... Ctrl+Shift+B
Run Active File
Run Selected Text
Show Running Tasks...
Restart Running Task...
Terminate Task...
Configure Tasks...
Configure Default Build Task...
EXPLORER
PREDLOZAK_PROJEKTA
rtl
top_level.v
sim
tb
run_sim.bat
OUTLINE
TIMELINE
sedge clk or negedge rstn) begin
= 8'b0;
= counter + 8'b1;
end
assign count = counter[3];
endmodule
Ln 1, Col 1
Spaces: 2
UTF-8
CRLF
Verilog


File Edit Selection View Go Run Terminal Help
top_level.v - predlozak_projekta - Visual Studio Code
EXPLORER
PREDLOZAK_PROJEKTA
rtl
top_level.v
sim
tb
run_sim.bat
top_level.v
rtl > top_level.v > {} top_level
1 module top_level (
2 input clk,
3 input rstn,
4
5 output count
6);
7
8 reg [7:0] counter;
9
10 always @ (posedge clk or negedge rstn) begin
11 if(~rstn)
12 counter <= 8'b0;
13 else
14 counter <= counter + 8'b1;
15 end
16
17 assign count = counter[3];
18
19
20 endmodule
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
powershell
PS G:\Moj disk\magnetar\digital_design\predlozak_projekta> .\run_sim.bat
OUTLINE
TIMELINE
0 0
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF Verilog

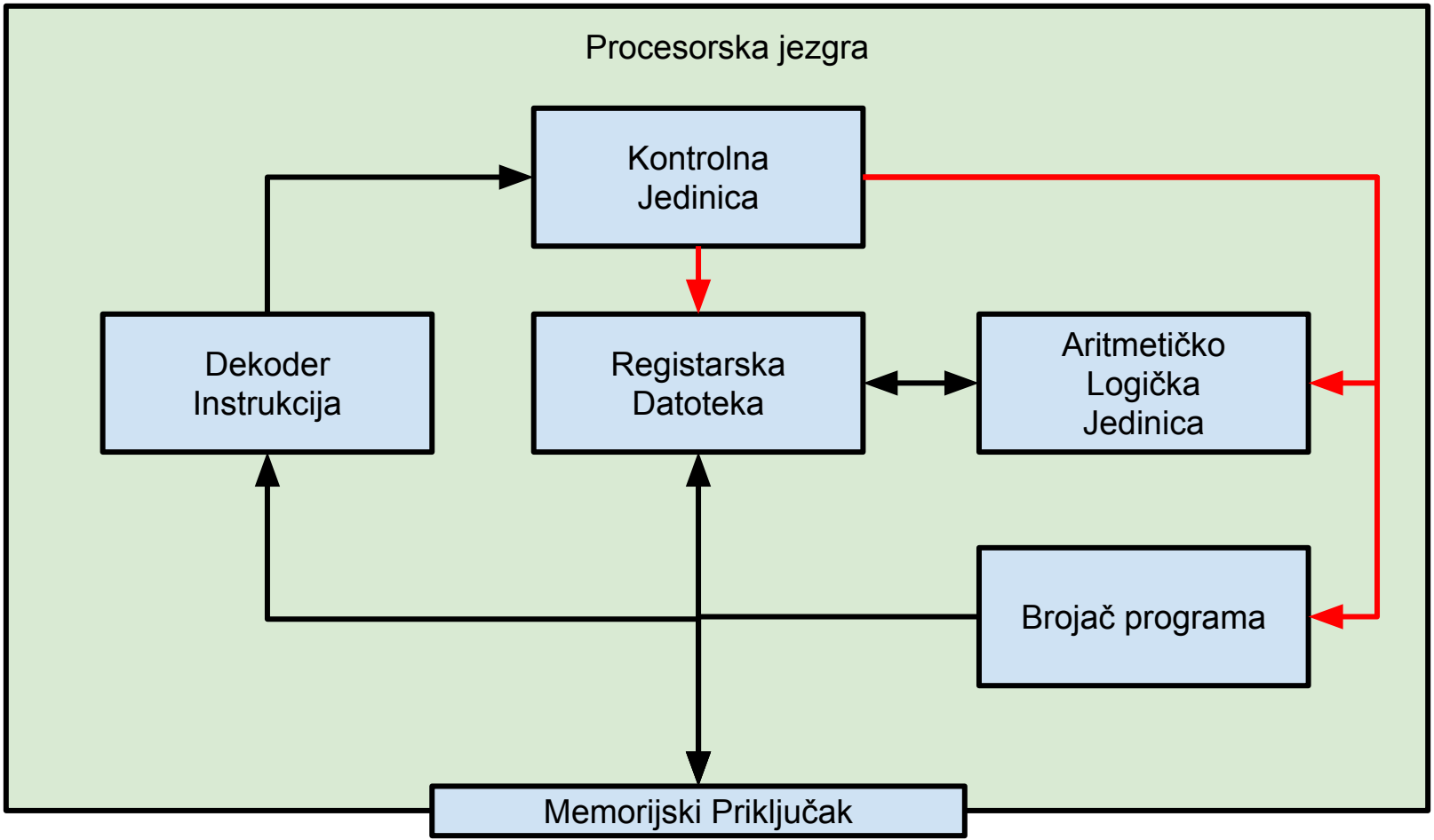

GTKWave - sim\dump.vcd
File
Edit
Search
Time
Markers
View
Help
From: 0 sec
To: 120 ns
Marker: -- | Cursor: 0 sec
SST
top_level_tb
u_top_level
Recurse Import
Append
Insert
Replace
Signals
Time
clk
counter[7:0]
rstn
Waves
1 ps
2 ps
3 ps
Type
Signals
wire clk
wire count
reg counter[7:0]
wire rstn
Filter:
Append
Insert
Replace


GTKWave - sim\dump.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 120 ns
Marker: --
Cursor: 700 ps
SST
top_level_tb
u_top_level
Type Signals
wire clk
wire count
reg counter[7:0]
wire rstn
Filter:
Append Insert Replace
Signals
Time
clk
count
counter[7:0]
rstn
Waves
10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns 100 ns 110 ns 120
xx
00

Brainstorm ideja za zajednički integrirani krug MyChip

Uvod u dizajn procesorske jezgre

Arhitektura procesorske jezgre


Procesorska jezgra
Kontrolna Jedinica
Dekoder Instrukcija
Registarska Datoteka
Aritmetičko Logička Jedinica
Brojač programa
Memorijski Priključak

Kontrolna jedinica (engl. Control Unit)

- Upravljanje Instrukcijama: Kontrolna jedinica je zadužena za preuzimanje, dekodiranje i izvršavanje instrukcija koje su pohranjene u memoriji računala. Ona tumači instrukcije i pretvara ih u niz signala koji upravljaju ostalim dijelovima računala.
- Koordinacija Komponenti: Kontrolna jedinica upravlja radom ostalih komponenti procesorske jezgre. Osigurava da se operacije izvode u ispravnom redoslijedu i da se podaci pravilno prenose između registara, ALU (jedinice za aritmetičko-logičke operacije), predmemorije i glavne memorije.
- Kontrola Tempa Izvršenja: Kontrolna jedinica kontrolira radni takt procesora, odnosno određuje tempo kojim se instrukcije izvršavaju. To čini sinkroniziranjem svih operacija s internim radnim taktom procesora.

Brojač programa (engl. Program Counter)

- Vođenje Reda Instrukcija: Brojač programa (Program Counter ili PC) je specijalizirani registar procesora koji sadrži adresu sljedeće instrukcije koju treba izvršiti. On omogućava procesoru da zna koje instrukcije treba izvršiti i u kojem redoslijedu.
- Automatsko Povećanje: Nakon što je instrukcija preuzeta za izvršavanje, brojač programa se automatski povećava kako bi pokazao na sljedeću instrukciju. Kod većine instrukcija, to je jednostavno povećanje za veličinu instrukcije u memoriji.
- Skokovi i Grananja: Tijekom izvršenja skokova ili grana (jumps or branches), brojač programa se mijenja tako da pokazuje na novu adresu iz koje će se preuzeti sljedeća instrukcija, omogućujući tako kontrolu toka programa.

Dekoder instrukcija (engl. Instruction Decoder)

- Tumačenje Instrukcija: Dekoder instrukcija je komponenta procesorske jezgre koja tumači (dekodira) instrukcije. Svaka instrukcija, predstavljena u binarnom obliku, prenosi se u dekoder instrukcija gdje se analizira i tumači.
- Raščlamba Instrukcija: Dekoder instrukcija također raščlanjuje instrukciju na operacijski, strojni kod i prateće argumente (adrese registara, memorijske adrese, konstante i sl.).
- Povezanost s Ostalim Komponentama: Dekoder instrukcija ima direktnu vezu sa brojačem programa (koji određuje koja se instrukcija izvršava) i kontrolnom jedinicom (koja koordinira proces izvršavanja instrukcija).

Registarska datoteka (engl. Register File)

- Skladištenje Podataka: Registarska datoteka je set registara unutar procesorske jezgre koji pružaju brzo lokalno skladištenje za privremeno čuvanje podataka tijekom izvršenja instrukcija.
- Rad s Instrukcijama: Registri često služe kao operandi za aritmetičke i logičke instrukcije. Podaci u registrima mogu biti izravno pristupljivi i procesirani, što procesiranje čini bržim nego kada bi se podaci dohvatili iz memorije.
- Različite Vrste Registara: U registarskoj datoteci mogu postojati različite vrste registara, uključujući opće namjene registre (koji se mogu koristiti za bilo koju operaciju), specijalizirane registre (kao što su brojač programa i registar statusa) i registre za brzo pohranjivanje privremenih rezultata.

Aritmetičko logička jedinica - ALU

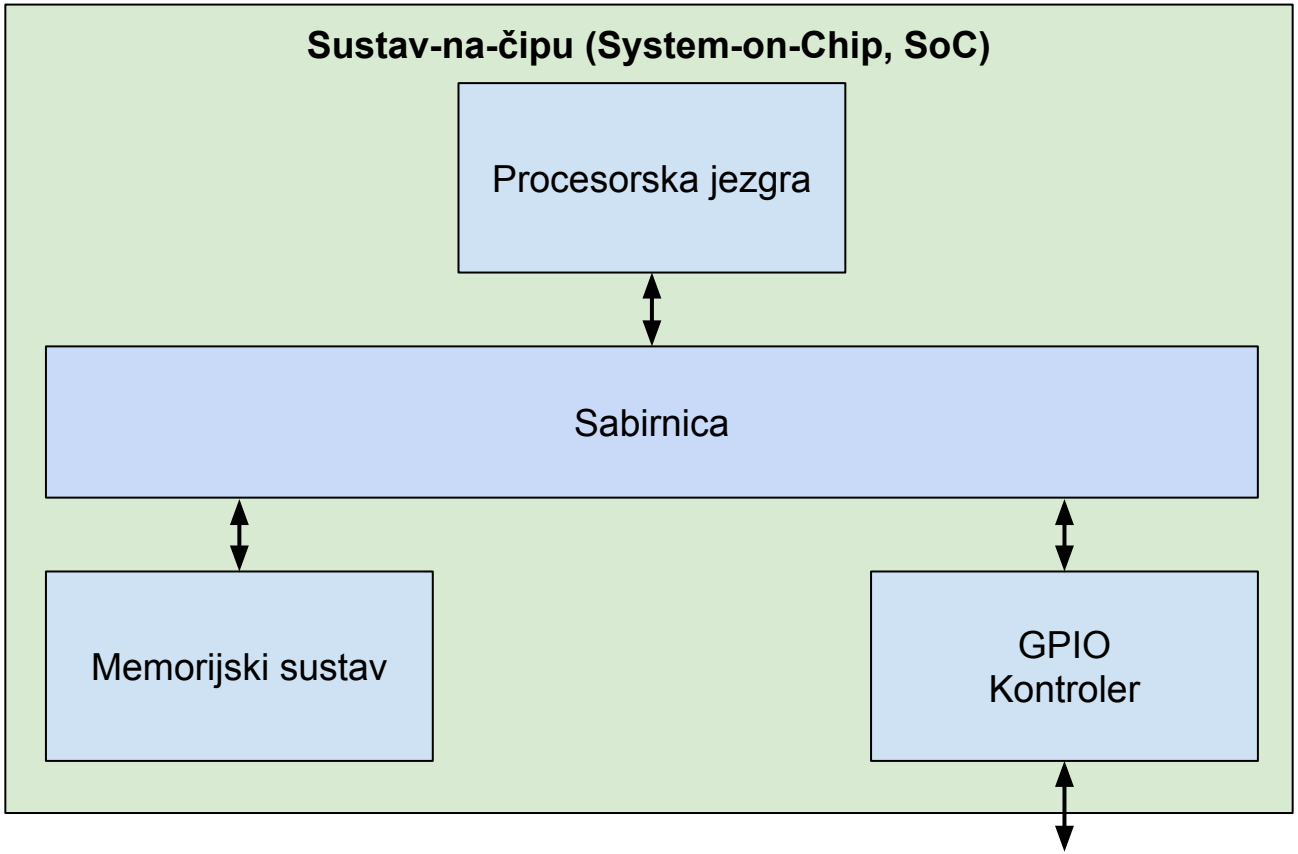
- Izvođenje Operacija: ALU (Arithmetic Logic Unit ili Jedinica za aritmetičko-logičke operacije) je ključna komponenta procesorske jezgre koja izvodi osnovne aritmetičke i logičke operacije, uključujući zbrajanje, oduzimanje, množenje, dijeljenje i operacije uspoređivanja.
- Rad s Registarskom Datotekom: ALU često koristi podatke pohranjene u registarskoj datoteci. Na primjer, može preuzeti dva broja iz registara, izvršiti operaciju nad njima (npr. zbrajanje), a zatim rezultat pohraniti natrag u registar.
- Generiranje Statusa: ALU također generira informacije o statusu, poput zastavica prekoračenja (overflow) ili nule (zero), koje mogu utjecati na daljnje operacije i odluke procesora. Te informacije o statusu obično se pohranjuju u specijaliziranom registru statusa.

Memorijski priključak (engl. Memory Port)

- Komunikacija s Memorijom: Memorijski priključak (ili memory interface) omogućava procesorskoj jezgri komunikaciju s glavnom memorijom i drugim vanjskim memorijskim jedinicama. On prenosi podatke između procesora i memorije.
- Izvršavanje Memorijskih Operacija: Putem memorijskog priključka, procesorska jezgra izvršava operacije učitavanja (read) i pohrane (write). Učitavanje prenosi podatke iz memorije u procesor, dok pohrana prenosi podatke iz procesora u memoriju.
- Upotreba Adresne i Podatkovne Sabirnice: Memorijski priključak koristi adresnu sabirnicu za specificiranje lokacije u memoriji s koje se čitaju ili na koju se zapisuju podaci, i podatkovnu sabirnicu za prijenos samih podataka. Ove sabirnice su ključne za pravilno funkcioniranje memorijskog priključka.

Sustav na Čipu (System-on-Chip)

- Integracija Komponenti: Sustav na čipu (SoC - System on a Chip) je integrirani sklop koji uključuje većinu ili sve komponente računalnog sustava na jednom čipu. Ovo uključuje procesorsku jezgru, memoriju, ulazno-izlazne portove, i potencijalno druge funkcionalne blokove poput grafičkih procesora (GPU), modula za mrežnu komunikaciju, itd.
- Optimizacija Performansi i Potrošnje: SoC pruža optimizaciju performansi i smanjenje potrošnje energije kroz integraciju komponenti na jednom čipu. To smanjuje vremena pristupa memoriji i olakšava koordinaciju između različitih komponenti sustava.
- Primjena u uređajima: SoC-ovi su široko korišteni u modernim uređajima kao što su pametni telefoni, tableti, pametni televizori, IoT uređaji (Internet of Things) i slično, gdje su prostor i energetska učinkovitost ključni.


Sustav-na-čipu (System-on-Chip, SoC)
Procesorska jezgra
Sabirnica
Memorijski sustav
GPIO
Kontroler

Primjer minimalnog instrukcijskog seta

Arhitektura instrukcijskog seta (ISA)

- Definicija Arhitekture: ISA (Instruction Set Architecture) je apstraktan model računalne arhitekture koji definira set instrukcija koje procesor može izvršiti. To uključuje opise operacija, modova adresiranja, registara, format instrukcija i drugih aspekata relevantnih za programere.
- Most Između Hardvera i Softvera: ISA djeluje kao sučelje između softvera i hardvera, omogućujući programerima da pišu programe bez potrebe za dubokim razumijevanjem specifičnih detalja hardvera.
- Primjeri ISA: Postoje različite vrste ISAs koje koriste različiti procesori. Neki poznati primjeri uključuju x86 i x86-64 (koristi se u većini osobnih računala), ARM (koristi se u većini mobilnih uređaja), MIPS (često se koristi u akademskim i obrazovnim kontekstima) i RISC-V.

RISC-V ISA

- RISC Arhitektura: RISC-V je otvorena standardna instrukcijska skupina (ISA) temeljena na RISC (Reduced Instruction Set Computing) principima. RISC arhitektura koristi jednostavne instrukcije koje se mogu izvršiti u jednom taktu procesora, čime se omogućava visoka performansa i energetska učinkovitost.
- Otvoreni Standard: Za razliku od većine drugih ISA, RISC-V je otvoren standard, što znači da ga bilo koji proizvođač čipova može koristiti bez plaćanja licenci. To otvara put za inovacije i personalizaciju u dizajnu procesora.
- Modularnost: RISC-V je modularan, što znači da proizvođači mogu odabrati samo one dijelove instrukcijskog skupa koji su im potrebni za određeni proizvod. To ga čini pogodnim za širok spektar uređaja, od mikrokontrolera do superkompjutora.

RISC-V format instrukcija

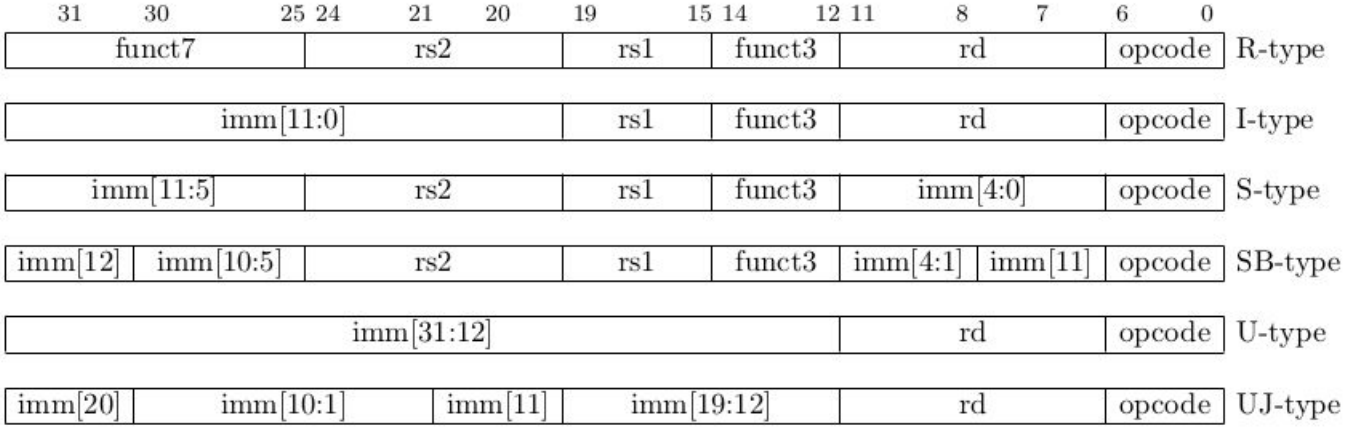


Figure 2.3: RISC-V base instruction formats showing immediate variants.

Format instrukcija

31	24	23	16	15	8	7	0
Izvor 2		Izvor 1		Odredište		Kod instrukcije	

- Odredište → Registar, memorijska adresa (za pohranu ili skok)
- Izvor 1 → Registar ili memorijska adresa (za čitanje)
- Izvor 2 → Uvijek registar

Strojni kod (opcode)	Asemblerski mnemonik	Opis naredbe
0x00	LOAD RX, #ADRESA	Učitaj sadržaj s memorijske lokacije na #ADRESI u registar procesora RX
0x01	STORE RX, #ADRESA	Zapiši sadržaj registra procesora RX u memorijsku lokaciju na #ADRESI
0x02	ADD RX, RY, RZ	Zbroji sadržaje registara procesora RX i RY te ih spremi u registar RZ
0x03	SUB RX, RY, RZ	Oduzmi sadržaje registara procesora RX i RY te ih spremi u registar RZ

Strojni kod (opcode)	Asemblerski mnemonik	Opis naredbe
0x04	JMP #ADRESA	Skoči na naredbu na #ADRESI
0x05	JZ RX, #ADRESA	Skoči na naredbu na #ADRESI ako je sadržaj registra procesor RX jednak nuli
0x06	JNZ RX, #ADRESA	Skoči na naredbu na #ADRESI ako sadržaj registra procesor RX nije jednak nuli

Strojni kod (opcode)	Asemblerski mnemonik
0x00	LOAD RX, #ADRESA
0x01	STORE RX, #ADRESA
0x02	ADD RX, RY, RZ
0x03	SUB RX, RY, RZ

Strojni kod (opcode)	Asemblerski mnemonik
0x04	JMP #ADRESA
0x05	JZ RX, #ADRESA
0x06	JNZ RX, #ADRESA

Strojni kod za LED blinky program

Strojni kod

- Strojni Kod: Strojni kod je najniža razina programskog koda koju procesor izravno može izvršiti. Sastoji se od binarnih instrukcija koje odgovaraju specifičnom setu instrukcija procesora (ISA). Strojni kod je specifičan za određeni tip procesora i nije lako čitljiv ljudima.
- Asembler: Asembler je vrsta programskog jezika koji koristi lako čitljive simboličke reprezentacije strojnog koda. Svaka instrukcija u asembleru odgovara točno jednoj instrukciji strojnog koda.
- Asembliranje: Asembliranje je proces pretvaranja asemblerskog koda u strojni kod. Ovo radi asembler - poseban program koji čita asemblerski kod i generira ekvivalentni strojni kod koji procesor može izvršiti.

Adresa	Asemblerski mnemonik	Komentar
0	LOAD R1, #LED_ADRESA	U registar R1 učitaj lokaciju priključka LEDice
1	LOAD R2, #UKLJUCENO	U registar R2 učitaj vrijednost uključene LEDice
2	STORE R2, (R1)	Na lokaciju priključka LEDice zapiši vrijednost uključene LEDice
3	LOAD R2, #IZNOS_BROJILA	U registar R2 učitaj iznos brojila za pauzu
4	LOAD R3, #KORAK_BROJILA	U registar R3 učitaj iznos koraka brojila za pauzu
5	SUB R2, R3, R2	Oduzmi korak od iznosa brojila i zapiši u registar R2

Adresa	Asemblerski mnemonik	Komentar
6	JNZ R2, #5	Ako sadržaj registra R2 nije jednak nuli, skoči na adresu #5
7	LOAD R2, #ISKLJUCENO	U registar R2 učitaj vrijednost isključene LEDice
8	STORE R2, (R1)	Na lokaciju priključka LEDice zapiši vrijednost islkjučene LEDice
9	LOAD R2, #IZNOS_BROJILA	U registar R2 učitaj iznos brojila za pauzu
10	SUB R2, R3, R2	Oduzmi korak od iznosa brojila i zapiši u registar R2
11	JNZ R2, #10	Ako sadržaj registra R2 nije jednak nuli, skoči na adresu #10

Adresa	Asemblerski mnemonik	Komentar
12	JMP #0	Bezuvjetno skoči na početak programa
13	#LED_ADRESA	Vrijednost adrese priključka LEDice
14	#UKLJUCENO	Vrijednost uključene LEDice
15	#ISKLJUCENO	Vrijednost isključene LEDice
16	#IZNOS_BROJILA	Vrijednost iznosa brojila za pauzu
17	#KORAK_BROJILA	Vrijednost koraka brojila za pauzu

LOAD R1, #LED_ADRESA	JNZ R2, #5	JMP #0
LOAD R2, #UKLJUCENO	LOAD R2, #ISKLJUCENO	#LED_ADRESA
STORE R2, (R1)	STORE R2, (R1)	#UKLJUCENO
LOAD R2, #IZNOS_BROJILA	LOAD R2, #IZNOS_BROJILA	#ISKLJUCENO
LOAD R3, #KORAK_BROJILA	SUB R2, R3, R2	#IZNOS_BROJILA
SUB R2, R3, R2	JNZ R2, #10	#KORAK_BROJILA

Dizajn modula procesorske jezgre

Organizacija RTL koda

- rtl
  - cpu
    - alu.v
    - central_processing_unit.v
    - control_unit.v
    - instruction_decoder.v
    - program_counter.v
    - register_file.v
  - memory_system.v
  - system_on_chip.v

Memorijsko sučelje

Memorijsko sučelje (I)

- Adresni Signali: Adresni signali se koriste za određivanje lokacije u memoriji na kojoj se izvršava operacija. Procesorska jezgra generira adresni signal kako bi odredila određenu adresu u memoriji za čitanje ili pisanje podataka.
- Kontrolni Signali: Kontrolni signali su signali koji upravljaju radom memorije. To uključuje signale koji određuju je li operacija čitanja ili pisanja, te signale koji sinkroniziraju prijenos podataka između procesora i memorije.
- Podatkovni Signali: Podatkovni signali predstavljaju stvarne podatke koji se prenose između procesora i memorije. Kada procesor piše u memoriju, podaci se prenose kroz podatkovne signale; kada procesor čita iz memorije, podaci se prenose iz memorije u procesor putem podatkovne šine.

Memorijsko sučelje (II)

- Širina memorijskog sučelja → WIDTH = 32 bita

```
// Memory port
input wire [WIDTH-1:0]      ADDRESS,
input wire [WIDTH-1:0]      WRITE_DATA,
input wire                  WRITE_ENABLE,
input wire                  READ_ENABLE,
output  [WIDTH-1:0]         READ_DATA
```

Registarska mapa

Specifikacija registarske mape

- Širina registara → WIDTH = 32 bita
- Broj registara → 16
- Resetiranje sadržaja svih registara na RSTn signal
- Dva priključka za čitanje (argumenti za ALU)
- Jedan priključak za pisanje

Sučelje registarske mape

```
// Global
input wire CLK,
input wire RSTn,                        // Active-low reset signal
// Module specific
input wire [3:0] READ_REG1,             // Read address for the first register
input wire [3:0] READ_REG2,             // Read address for the second register
output wire [WIDTH-1:0] READ_DATA1,     // Read data from the first register
output wire [WIDTH-1:0] READ_DATA2,     // Read data from the second register
input wire [3:0] WRITE_REG,             // Write address for the register
input wire [WIDTH-1:0] WRITE_REG_DATA,  // Data to write to the register
input wire WRITE_REG_ENABLE             // Write Enable signal
```

Memorijski sustav

Specifikacija memorijskog sučelja

- Širina riječi → WIDTH = 32 bita
- Dubina memorije → Depth = 256
- Initial naredba za učitavanje sadržaja memorije

Sučelje memorijskog sustava

```
// Global
input wire                      CLK,
input wire                      RSTn,

// Memory port
input wire [WIDTH-1:0]          ADDRESS,
input wire [WIDTH-1:0]          WRITE_DATA,
input wire                      WRITE_ENABLE,
input wire                      READ_ENABLE,
output  [WIDTH-1:0]             READ_DATA
```

Initial naredba za učitavanje sadržaja memorije

```
// Initial block to define memory contents
initial begin
    memory[0]   = 32'h00_00_00_00; // instruction
    memory[1]   = 32'h11_22_33_44; // data
    memory[2]   = 32'hAB_CD_EF_01; //

end
```

Definiranje sadržaja memorije

- Koristite instrukcijski set za implementaciju asemblerskog koda LED blinky koda