

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Miro Bezjak, Davor Delač, Aleksandar Prokopec

VHDLLab
Obrazovni programski sustav za modeliranje i
simuliranje digitalnih sklopova

Zagreb, 4. svibnja 2008.

Ovaj rad izrađen je u sklopu Java projekta na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva pod vodstvom mr. sc. Marka Čupića i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2007./2008.

Sadržaj

1. UVOD	1
2. MODELIRANJE DIGITALNIH SKLOPOVA	4
2.1. DIGITALNI SKLOPOVI	4
2.1.1. Osnovni logički sklopovi	4
2.1.2. Kombinatorijski sklopovi	9
2.1.2.1. Dekoder.....	9
2.1.2.2. Multipleksor	9
2.1.2.3. Permanentna memorija	11
2.1.2.4. Programirajući logički moduli.....	12
2.1.3. Sekvencijski sklopovi	14
2.1.3.1. Bistabil	14
2.1.3.2. Strojevi stanja.....	16
2.2. JEZIK VHDL	19
2.2.1. Povijest jezika.....	19
2.2.2. Struktura jezika.....	19
2.2.2.1. Sučelje sustava	19
2.2.2.2. Implementacija sustava.....	20
2.2.3. Ponašajni opis.....	21
2.2.4. Strukturni opis	22
2.2.5. Implementacija digitalnih sklopova u jeziku VHDL.....	24
2.3. SUSTAV WEBISE.....	27
2.4. SIMULATOR GHDL.....	29
2.5. OSTALE TEHNOLOGIJE MODELIRANJA.....	30
3. SUSTAV VHDLLAB.....	31
3.1. ZNAČAJKE.....	31
3.2. OPIS SUČELJA I SVIH EDITORA	34
3.2.1. Uređivač koda.....	37
3.2.2. Uređivač automata	38
3.2.3. Uređivač sheme sklopa	39
3.2.4. Uređivač ispitnih sklopova.....	41
3.2.5. Preglednik simulacija.....	42
3.3. USPOREDBA S KOMERCIJALNIM RAZVOJNIM OKRUŽENJIMA....	44
4. PRIMJERI KORIŠTENJA SUSTAVA VHDLLAB	46
4.1. OPIS ZADATKA LABORATORIJSKE VJEŽBE.....	46
4.2. OPIS IZVEDBE LABORATORIJSKE VJEŽBE U VHDLLAB-U.....	48
4.2.1. Oblikovanje brojala i dekodera	49
4.2.2. Oblikovanje sklopa <i>timer</i>	53
4.2.3. Oblikovanje automata i upravljača semafora	56
5. TEHNIČKI OPIS SUSTAVA VHDLLAB	61
5.1. IZGRADNJA SUSTAVA VHDLLAB	61
5.2. VHDLLAB POSLUŽITELJ	63

5.2.1. Modeli	65
5.2.2. Podatkovni sloj	66
5.2.3. Logički sloj.....	70
5.2.4. Prezentacijski sloj	72
5.2.5. Sigurnost	73
5.3. VHDLLAB KLIJENT	74
5.3.1. Platforma	76
5.3.2. Uređivači	82
5.3.2.1. Uređivač automata.....	83
5.3.2.2. Uređivač shema digitalnih sklopova.....	85
5.3.3. Pogledi	89
5.3.4. Instalacija	91
6. ZAKLJUČAK.....	93
7. LITERATURA	95
8. DODATAK A: ANKETA.....	96
9. DODATAK B: INDEKS SLIKA, TABLICA I PRIMJERA ...	102
9.1. INDEKS SLIKA	102
9.2. INDEKS TABLICA	103
9.3. INDEKS PRIMJERA	103

1. Uvod

Svaki se student tijekom svog boravka na visokom učilištu suočava s nizom problematičnih situacija. Konstantno učenje i pripremanje za nastavu, zarađivanje za stanarinu ili studentski dom. U takvim stresnim situacijama studenti traže načine da si olakšaju život. Misao vodilju većine pripadnika ljudskog roda, pa tako i nas studenata najbolje je opisao William od Ockhama riječima: „*Entia non sunt multiplicanda praeter necessitatem*.“¹. Ovakvo razmišljanje dovodi do neželjenog ponašanja kao što je varanje i prepisivanje. Ako pogledamo drugu stranu medalje, uvođenje studija u sukladnosti s Bolonjskim procesom na većini fakulteta predstavlja logističku „noćnu moru“ profesorima i asistentima. Kako bismo pomogli u rješavanju ovih problema, prije godinu i pol dana odlučili smo se upustiti u izradu sustava za održavanje laboratorijskih vježbi jednog od najmasovnijih predmeta na našem matičnom fakultetu, Digitalne logike. Taj smo sustav nazvali VHDLLab.

Digitalna logika predmet je sačinjen od osnovnih znanja koja svaki inženjer elektrotehnike ili računarstva mora usvojiti. Predmet je obavezan u prvom semestru svim studentima koji upišu Fakultet elektrotehnike i računarstva. Glavnina predmeta vezana je uz modeliranje digitalnih sustava jezikom VHDL². U svrhu savladavanja jezika VHDL dosada je korišten sustav *Xilinx WebISE* u kombinaciji sa simulatorom *Xilinx SimuLink*. Navedena dva profesionalna alata za modeliranje i simuliranje elektroničkih sustava nude niz kvalitetnih programskih rješenja za izradu kompleksnih sustava. Iako kvalitetu navedenih alata ne želimo i ne možemo osporavati, njihova robusnost i kompleksnost predstavljaju problem velikom broju studenata. Cilj kolegija je shvaćanje osnovnih pojmova digitalne logike, savladavanje znanja o jeziku VHDL, te sticanje iskustva u modeliranju digitalnih sklopova, a ne snalaženje u složenim postupcima instalacije profesionalnih programa. Isto tako, navedeni proizvodi su zaštićeni licencom te je mogućnost njihove distribucije studentima ograničena. Iz ovih razloga odlučili smo napraviti alternativu postojećim sustavima usmjerenu na jednostavnije savladavanje građe predmeta.

VHDLLab je web-orijentirana integrirana okolina za razvoj (*engl. integrated development environment*) izrađena u svrhu učenja jezika za modeliranje sklopovlja VHDL. Sustav krasi jednostavnost korištenja kao i potpuna funkcionalnost pri oblikovanju i simuliranju sklopovlja na različitim razinama apstrakcije. U sklopu sustava implementirana je isključivo funkcionalnost nužna za obrazovne potrebe inženjera elektrotehnike i računarstva. Činjenica da je sustav izveden kao web aplikacija izgrađena u više slojeva osigurala je još jedno jako bitno svojstvo, a to je prenosivost. Klijentska aplikacija izrađena je u programskom jeziku

¹ „Stvari se ne treba umnožavati više nego je nužno“ – Ockhamova britva

² Very high speed integrated circuit Hardware Description Language

Java i za pokretanje zahtjeva samo *JRE*³ 6 instaliran na računalu korisnika, što je već osigurano kod većine korisnika. Ovaj aspekt jednostavnosti čini VHDLLab izvrsnim alatom za rad kod kuće. Nakon kratke instalacije, student može pristupiti izradi laboratorijske vježbe. Na ovaj način student ima više vremena za modeliranje sklopova u jeziku VHDL i ispitivanje njihove funkcionalnosti, kao i ispitivanje rubnih slučajeva i specifičnosti modeliranja u jeziku VHDL. Uz navedeno pojednostavljenje korištenja za studente, VHDLLab nudi i bolju kontrolu izvođenja laboratorijskih vježbi profesorima i asistentima – nakon što je student obavio i predao laboratorijsku vježbu, rezultati njegovog rada spremaju se na poslužitelj na fakultetu, pa ih je moguće pregledati i ocijeniti. Postupak ocjenjivanja moguće je također i automatizirati.

Cilj ovog rada je opis osnovnih pojmova u digitalnoj logici, opis osnovnih problema koje student mora savladati na laboratorijskim vježbama i opis sustava VHDLLab i načina na koji on omogućava savladavanje tih problema. Želja nam je objediniti teorijsku pozadinu predmeta Digitalna logika s izradom aplikacije i njenom primjenom. U tu svrhu rad smo podijelili u četiri osnovna dijela.

Drugo poglavlje opisuje modeliranje digitalnih sklopova. Tematske cjeline su osnove logike, digitalne logike i njena primjena na elektroničke sklopove. Također se pruža uvod u modeliranje i simuliranje digitalnih sklopova jezikom VHDL. Pred kraj poglavlja dajemo opis korištenog simulatora GHDL, kao i opis sustava *WebISE* i *SimuLink* kao uzore po kojima je rađen VHDLLab.

U trećem poglavlju rada opisano je korištenje sustava VHDLLab. Opisuje se izrada projekata i korištenje ugrađenih uređivača za modeliranje kompleksnih digitalnih sklopova. To su uređivač koda, uređivač ispitnih sklopova (*engl. testbench*), te uređivači shema sklopova i dijagrama stanja konačnog automata. Također se opisuje i način simulacije i ispitivanja modeliranih digitalnih sklopova.

Četvrto poglavlje daje primjer izrade jedne složenije laboratorijske vježbe koristeći sustav VHDLLab. Cilj ovog poglavlja je pokazati robusnost i jednostavnost sustava te brzinu i kvalitetu modeliranja složenih digitalnih sklopova pomoću VHDLLab-a.

Peto poglavlje daje opis same implementacije sustava. Ovdje su ukratko predstavljene tehnologije korištene za izradu sustava, kao i sama struktura VHDLLab-a. Dani su opisi sustava po slojevima uz definicije sučelja kojima je ostvarena komunikacija među slojevima.

Rad završava zaključkom i smjernicama za daljnji razvoj ovog i sličnih sustava te mogućom primjenom na neke druge kolegije u sklopu studija elektrotehnike i računarstva. Pokazat ćemo i rezultate ankete koju je ispunila prva generacija korisnika sustava VHDLLab.

³ Java Runtime Environment

Ovaj je rad plod višegodišnjeg rada pod stručnim vodstvom mr. sc. Marka Čupića na području programiranja u Javi, koji je započeo Java tečajem, a nastavio suradnjom na nizu projekata s ciljem jednostavnijeg izvođenja nastave. Niti jedan dio rada nije vezan uz teme i područja diplomskih radova autora.

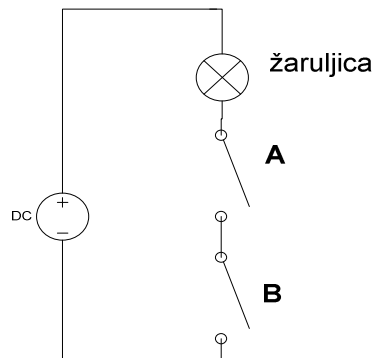
2. Modeliranje digitalnih sklopova

2.1. Digitalni sklopovi

2.1.1. Osnovni logički sklopovi

Mogućnost određenih digitalnih sklopova da obavljaju kompleksne operacije i obrađuju podatke zasniva se na njihovoj sposobnosti obavljanja jednostavnih logičkih operacija. Algebru logičkih operacija prvi je s matematičkog aspekta proveo George Boole godine 1847. te je po njemu dobila ime. Iako je Booleova algebra po definiciji formalni ekvivalent algebre skupova, u digitalnoj logici tim pojmom oslovljavamo algebru sudova pošto je naš jedini interes ne konkretan sadržaj poruke, već je li tvrdnja točna (\top , 1) ili netočna (\perp , 0).

Ponašanje sklopova jednostavno se može modelirati logikom sudova, a vrijedi i obrat, logika sudova sklopovima. Na ovoj se činjenici zasniva ideja jezika za opis sklopova (*HDL*, engl. *Hardware Description Language*). Slika 2.1 prikazuje jednostavan primjer električnog kruga.



Slika 2.1 Jednostavni električni krug (I)

Ovim jednostavnim sklopom modelirali smo tvrdnju:

$$\text{kada } \langle \text{prekidač_A_radi} \rangle \text{ I } \langle \text{prekidač_B_radi} \rangle \text{ tada } \langle \text{žaruljica_svijetli} \rangle \quad (\text{i})$$

to jest, jednostavnu formulu:

$$A \wedge B = f \quad (\text{ii})$$

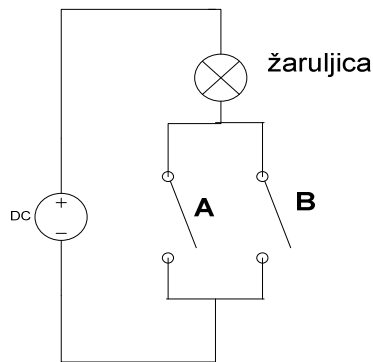
gdje su A , B i f logičke varijable, a izraz (ii) samo skraćena verzija izraza (i). Pri tome varijabla A poprima vrijednost istine ako prekidač A radi, a neistine inače. Isto vrijedi i za varijablu B . Iz kruga je vidljivo da žaruljica svijetli samo kad su oba prekidača uključena, to jest, logička varijabla f poprima vrijednost istine samo kada su varijable A i B istinite. Ovim

primjerom i tablicom 2.1 definiran je prvi od jednostavnih logičkih sklopova, a to je sklop *I* (engl. *and gate*).

Tablica 2.1 Sklop *I*

A	B	$A \wedge B = f$
⊥	⊥	⊥
⊥	⊤	⊥
⊤	⊥	⊥
⊤	⊤	⊤

Algebra sudova definirana samo pomoću sklopa *I* nije kompletna. To znači da postoje sklopovi koje ne možemo modelirati isključivo uporabom sklopa *I*. Potrebni su nam još neki sklopovi. Promotrimo sklop dan slikom 2.2:



Slika 2.2 Jednostavni električni krug (ILI)

Ovim jednostavnim sklopom modelirali smo tvrdnju:

kada $\langle \text{prekidač } A \text{ radi} \rangle$ *ILI* $\langle \text{prekidač } B \text{ radi} \rangle$ *tada* $\langle \text{žaruljica svijetli} \rangle$ (iii)

to jest, jednostavnu formulu:

$$A \vee B = f \quad (\text{iv})$$

Ovaj primjer opisuje logički sklop *ILI* (engl. *or gate*). Ponašanje sklopa *ILI* definirano je tablicom 2.2:

Tablica 2.2 Sklop *ILI*

A	B	$A \vee B = f$
⊥	⊥	⊥
⊥	⊤	⊤
⊤	⊥	⊤
⊤	⊤	⊤

S definiranom binarnom operacijom ILI za kompletnost algebre sudova potrebna nam je još samo jedna relacija. Ta relacija je jednostavna negacija tvrdnje dana tablicom 2.3:

Tablica 2.3 Sklop NE

A	$\neg A = f$
\perp	\top
\top	\perp

Pošto smo iznijeli tri nužne operacije algebre sudova, definirajmo pojam formule. Formula je izraz definiran sljedećim tvrdnjama:

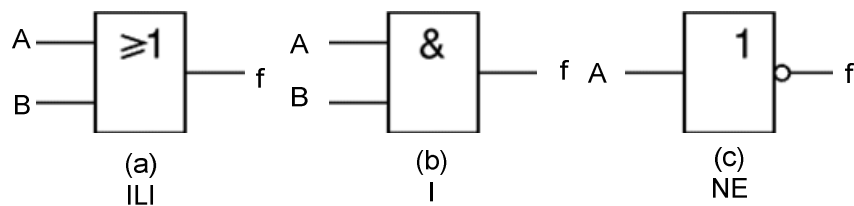
- (i) Logička varijabla je formula.
- (ii) Ako je F formula, tada je i $\neg F$ formula.
- (iii) Ako su F i G formule, tada su formule i:

$$A \vee B = f$$

$$A \wedge B = f$$

- (iv) Ništa drugo nije formula.

Da bi mogli shematski prikazati sklop u digitalnoj logici, uvode se simboli za logičke sklopove. Svaki od logičkih sklopova imaju više različitih simbola koji se pojavljuju u literaturi pa ćemo mi u okviru ove radnje uzeti simbole IEC norme koji se koriste i u Schematic uređivaču VHDLLab-a. Slika 2.3 prikazuje te simbole.



Slika 2.3 Jednostavni logički sklopovi (1)

Iako su ove tri operacije dostatne za prikaz svih logičkih funkcija, u digitalnoj logici javljaju se još četiri osnovna digitalna sklopa:

- (i) Sklop NI : $A NI B = NE (A I B)$
- (ii) Sklop $NILI$: $A NILI B = NE (A ILI B)$
- (iii) Sklop isključivi ILI (XOR): definiran tablicom 2.4

- (iv) Sklop isključivo *NILI* (XNOR): $NE(A \text{ XOR } B)$

Tablica 2.4 Sklop isključivo *ILI*

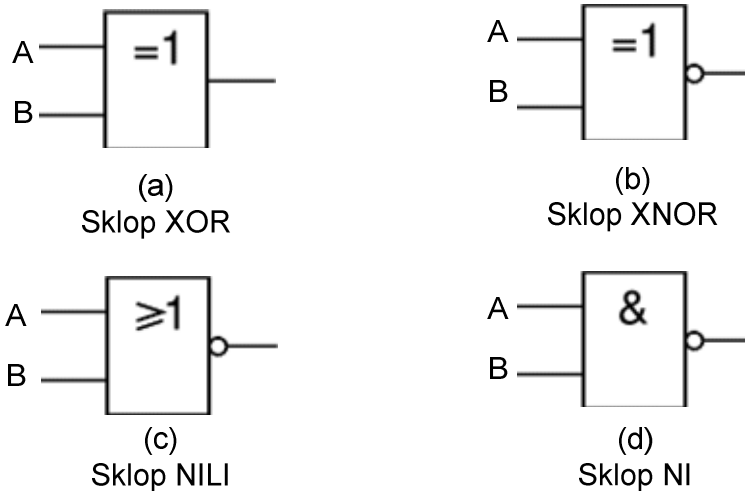
A	B	A XOR B
⊥	⊥	⊥
⊥	⊤	⊤
⊤	⊥	⊤
⊤	⊤	⊥

Među ovim sklopovima najpopularniji su *NAND*, *NOR* i *XOR*. *NAND* kao logička funkcija predstavlja bazu za logičku algebru pošto se uporabom ove funkcije mogu modelirati ranije iznesene funkcije:

- (i) Sklop *NE*: $NOT A = (A \text{ NAND } A)$
 (ii) Sklop *ILI*: $((A \text{ NAND } A) \text{ NAND } (B \text{ NAND } B))$
 (iii) Sklop *I*: $(A \text{ NAND } B) \text{ NAND } (A \text{ NAND } B)$

Ova činjenica omogućava nam izgradnju kompleksnih digitalnih sklopova uporabom samo jednog tipa jednostavnih logičkih sklopova.

Sklop *XOR* je zanimljiv pošto predstavlja logičku operaciju isključivo *ILI* koja je česta u modeliranju digitalnih sklopova zbog svog značenja $A \text{ XOR } B =$ „Ako je A, a nije B ili ako je B, a nije A tada f.“



Slika 2.4 Jednostavni logički sklopovi (2)

Implementacija ovih sklopova u elektronici ima više. Pošto vrijednost \top i \perp treba nekako predstaviti, najčešće se koriste različite naponske razine (naponska logika) i strujne razine (strujna logika). Sklopovi se implementiraju u TTL tehnologiji (Tranzistorsko-Tranzistorska Logika), DTL (Diodno-Tranzistorska Logika), CMOS i sličnima. Ovim dijelom digitalne logike se nećemo baviti pošto se rad fokusira na modeliranje digitalnih sklopova na razini logike. Ipak spominjat će se pojam sinteze sklopa ali u kontekstu preslikavanja opisa sklopa na postojeći FPGA⁴ čip.

⁴ FPGA (Field-Programmable Gate Array) – kombinacijski sklop s matricom logičkih vrata i programirljivom prospojnom mrežom

2.1.2. Kombinacijski sklopovi

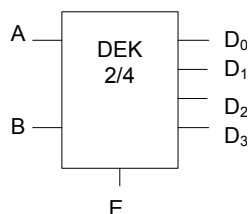
Kombinacijski sklopovi su sklopovi čije izlazne vrijednosti ovise samo o trenutnim ulaznim vrijednostima. Modeliraju se Booleovim funkcijama čiji su parametri ulazni signali, a vrijednost izlazni signal. Ovakvi sklopovi mogu obavljati standardne široko upotrebljavane funkcije, kao što su aritmetičke operacije ili komparacije binarnih nizova. Ovakva vrsta funkcija javlja se primjerice u aritmetičko-logičkoj jedinici mikroprocesora. Druga vrsta kombinacijskih sklopova su univerzalni moduli. Univerzalnim modulima može se ostvariti proizvoljna logička funkcija. Dijelimo ih na module s unaprijed određenom funkcijom (dekoder, multipleksor) i programirljive module (programirljivo logičko polje). U nastavku ćemo razmatrati univerzalne kombinacijske sklopove.

2.1.2.1. Dekoder

Dekoder je kombinacijski sklop koji za svaku kodiranu riječ na ulazu ima posebnu izlaznu liniju. Ta izlazna linija postavlja se u stanje 1 kada je na ulazu kodna riječ koja joj pripada. Ako s n označimo broj ulaza, tada može biti maksimalno 2^n izlaza. Jednostavno se konstruira minimizacijom Booleove funkcije. Može služiti kao BCD-dekadski dekodeer ili u službi demultipleksora. Kao primjer uzet ćemo dekodeer 2/4. Njegova tablica dana je tablicom 2.5, a simbol slikom 2.5.

2.5 Tablica dekodera 2/4

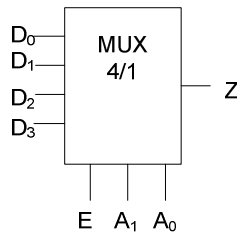
E	A	B	D ₀	D ₁	D ₂	D ₃
⊥	X	X	⊥	⊥	⊥	⊥
⌞	⊥	⊥	⌞	⊥	⊥	⊥
⌞	⊥	⌞	⊥	⌞	⊥	⊥
⌞	⌞	⊥	⊥	⊥	⌞	⊥
⌞	⌞	⌞	⊥	⊥	⊥	⌞



Slika 2.5 Simbol za dekodeer

2.1.2.2. Multipleksor

Multipleksor je sklop koji informaciju prisutnu na jednom od ulaza bira, odnosno selektira između informacija prisutnih na drugim ulazima i šalje na izlaz. Zato se zove i selektor podataka. Ulazi multipleksora su n selekcijskih bitova, jedan bit za stanje aktivnosti i 2^n podatkovnih linija. Kao izlaz multipleksor ima jednu signalnu liniju. Simbol multipleksora 4/1 dan je slikom 2.6:



Slika 2.6 Simbol za multipleksor

Ponašanje multipleksora je takvo da kad je signal E u nuli tada je izlaz Z u nuli. Kada je signal E u stanju 1 (τ) tada vrijedi $Z = D_A$ gdje je A dekadski ekvivalent vrijednosti selekcijskog vektora. Primjerice, tablica 2.6 predstavlja tablicu multipleksora 4/1 prikazanog slikom 2.6.

2.6 Tablica dekodera 2/4

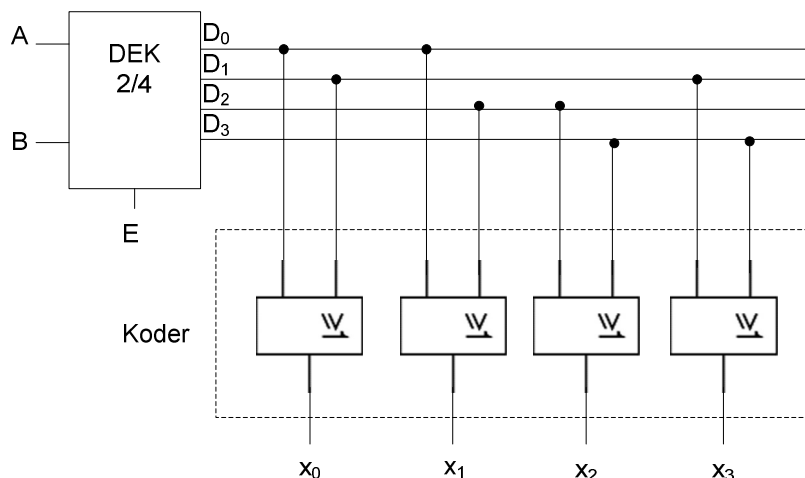
E	A	B	Z
\perp	X	X	\perp
τ	\perp	\perp	D_0
τ	\perp	τ	D_1
τ	τ	\perp	D_2
τ	τ	τ	D_3

Multipleksori i dekoderi većeg stupnja (MUX 16/1) mogu se dobiti spajanjem manjih multipleksora u kaskade. Tako se pravilnim spajanjem u kaskadu 5 MUX 4/1 dobiva MUX 16/1, spajanjem 2 MUX 16/1 dobiva MUX 32/1 itd. Za dekoder, spajanjem 5 DEK 2/4 dobiva se DEK 4/16.

U programu VHDLlab, upravo zbog ovakvih slučajeva, podržano je crtanje shema sklopova što omogućava jednostavno modeliranje ovakvih kompleksnih spojeva u preglednom i jednostavnom sučelju.

2.1.2.3. Permanentna memorija

Permanentna memorija sastoji se od dekodera, koda i prospojnog polja na kojem su ukodirane riječi. Slika 2.7 prikazuje primjer permanentne memorije ROM 4×4. Općenito, na ulazu u memoriju je dekodeer koji na osnovi n ulaznih bitova postavlja jednu od 2^n memorijskih linija na vrijednost 1. Na izlazu iz memorije je koder koji se sastoji od m izlaznih linija, a jednostavno se realizira sklopovima *ILI*. Oznaka za memoriju je ROM $2^n \times m$ gdje je n broj ulaznih linija (adresa), a m veličina pohranjenih podataka u bitovima.



Slika 2.7 ROM 4×4

Primjer memorije dan slikom 2.7. predstavlja memoriju ROM 4×4. To znači da se adresira s 2 bita (A i B), a na izlazu se dobiva 4-bitni podatak. U našem slučaju, ako na ulaz dovedemo $AB = 00_2$ tada će se na izlazu pojaviti podatak na adresi 0 koji iznosi:

$$X_0 X_1 X_2 X_3 = 1100_2$$

Ovaj sklop služi za trajno pamćenje informacija pošto su podatci određeni fizičkim vezama u prospojnom polju. Zato se ova vrsta memorije zove **permanentna memorija** ili ROM⁵. Koristi se još i naziv **ispisna memorija** pošto se informacije iz nje samo čitaju to jest ispisuju. Ima puno načina na koje je realizirana memorija, pogotovo koder na izlazu. Primjer dan slikom 2.7 daje jednu od jednostavnih mogućnost koja je dovoljna za naša razmatranja u sklopu ovog rada.

⁵ eng. ROM = Read Only Memory

2.1.2.4. Programirljivi logički moduli

Veliki broj logičkih funkcija može se u praksi realizirati mnogo manjim brojem sklopova I nego što ih ima dekode permanentne memorije. Stoga se na osnovi slične logičke strukture kao što je struktura permanentne memorije razvijaju programirljiva logička polja (*PLA*, *engl. Programmable Logic Array*). Ovi jednostavni sklopovi realiziraju logičku funkciju zapisanu kao suma produkata ili produkt suma. Na ulaz prve prospojne mreže sklopa dovode se ulazni signali i njihove negirane vrijednosti. Pomoću ove mreže realiziraju se mintermi spajanjem ulaznih signala na sklopove I na izlazu. Druga prospojna mreža na ulazima ima izlaze iz sloja sklopova I , a na izlazima sklopove II koji onda daju sume produkata. Programiranje se radi pomoću maske slične onoj kod permanentnih memorija. Proizvođači ovakvih sklopova daju korisniku mogućnost da ih na licu mjesta sam programira. U tu se svrhu koriste rastalni osigurači na vezama.

Drugi standardni programirljivi modul je PAL (*engl. Programmable Array Logic*). PAL je dosta sličan PLA sklopu. Sastoji se od jednog prospojnog polja te sloja sklopova I i sloja sklopova II . Na ulaze prospojnog polja dovode se ulazi sklopa, njihove negirane vrijednosti te izlazi sklopa i njihove negirane vrijednosti. Ova struktura nam omogućava da realiziramo logičke funkcije koje nisu nužno u obliku sume produkata. Funkcije koje nisu u obliku sume produkata često se daju oblikovati korištenjem manje logičkih sklopova. Uzmimo za primjer logičku funkciju:

$$f = \overline{(A \wedge B)} \vee (A \wedge C) \vee (B \wedge C) \vee (A \wedge C)$$

Ovu logičku funkciju možemo direktno modelirati na PAL-u, a za PLA ju moramo pretvarati u sumu produkata. Mala napomena vezana uz primjer je da se minimizacijom funkcije dobiva:

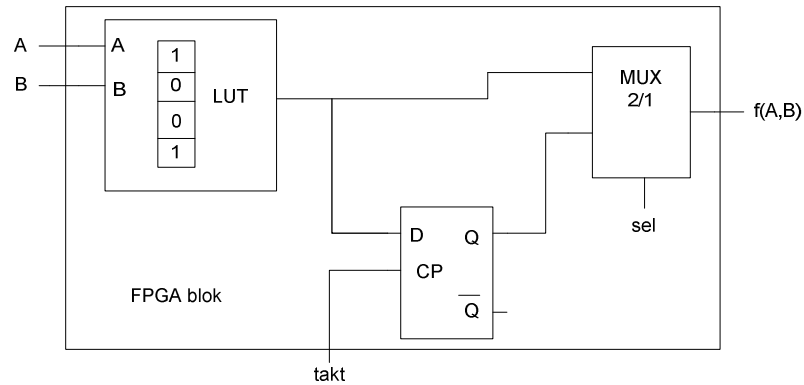
$$f = \overline{A} \vee B \vee \overline{C}$$

što je bolji oblik pošto zahtjeva manje logičkih sklopova za implementaciju.

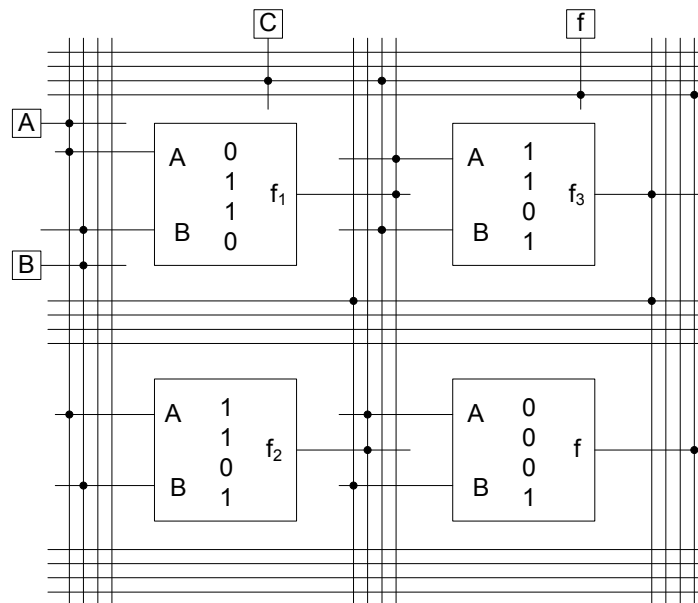
Iako smo ovdje promatrali PAL i PLA kao dvoslojne II od I sklopove sama implementacija ne mora biti točno takva. PAL i PLA mogu biti napravljeni kao I - II sklopovi pa su tako primjereni za implementaciju produkta suma. Isto tako za implementaciju slojeva često se koriste i ostali osnovni logički sklopovi, tako da sama implementacija funkcija u okviru PLA-a i PAL-a uvelike ovisi o njihovoj arhitekturi.

Treći programirljivi sklop je FPGA (Field-Programmable Gate Array). U najjednostavnijem primjeru, FPGA se sastoji od matrice logičkih blokova i prospojne mreže između njih. Logički blok FPGA sklopa sastoji se od preglednih tablica (*LUT*, *engl. look-up table*), bistabila i multipleksora na izlazu. Pregledne tablice su tablice kojima implementiramo logičke funkcije n varijabli. Imaju n ulaza i za svaku od 2^n vrijednosti postavljaju jedan od zadanih izlaza. U najjednostavnijem primjeru imamo LUT $2n+1$. Bistabili služe za modeliranje sekvencijskih sklopova, a multipleksor na izlazu zadužen je za proslijeđivanje izlaza bistabila ili izlaza tablice na izlaz bloka. Shema bloka dana je slikom 2.8. Prospojna mreža je programirljiva i

omogućava spajanje blokova međusobno te na ulaze i izlaze. Slika 2.9. pokazuje jednostavni FPGA sklop. FPGA se često koristi u sintezi računalno ostvarenih modela digitalnih sklopova. Pravi FPGA sklopovi kao što su oni proizvođača *Xilinx* (*MicroBlaze*, *PicoBlaze*) imaju puno kompleksniju strukturu koja im omogućava efikasnije i optimalnije implementiranje digitalnih sklopova.



Slika 2.8 Blok FPGA sklopa



Slika 2.9 Primjer FPGA sklopa

2.1.3. Sekvencijski sklopovi

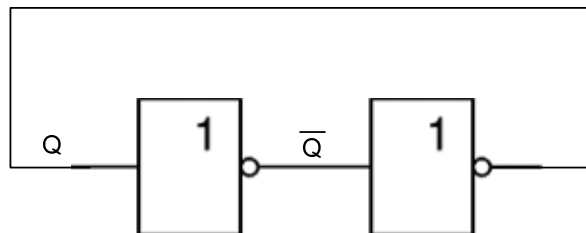
Logički sklopovi koji su dosad razmatrani (kombinacijski sklopovi) nemaju svojstvo pamćenja. Izlaz takvih sklopova ovisi samo o ulazu. Ovo svojstvo vrijedi i za permanentne memorije. Memorije, iako se tako zovu, samo predstavljaju funkcionalnu vezu između ulaza i izlaza. Memorijska svojstva sklopovima daju povratne veze, stoga ovakvi sklopovi moraju biti građeni logičkom povratnom vezom.

Digitalni sklopovi koji sadrže elemente za pamćenje nazivaju se sekvencijski sklopovi. Izlaz takvih sklopova je u najopćenitijem slučaju funkcija ulaza i trenutnog stanja. Stanje je u ovom slučaju binarna informacija pohranjena u elementima za pamćenje sekvencijskog sklopa. Nad ovakvim sklopovima definira se dodatno još jedna funkcija, a to je funkcija sljedeće stanje. Funkcija sljedeće stanje kao rezultat daje binarnu informaciju koja će biti pohranjena u elementima za pamćenje nakon što se za trenutno stanje na ulazima pojavi određena kombinacija. Sinkrone sekvencijske sklopove nazivamo još i strojevi stanja.

Prema načinu vremenskog odziva, sekvencijske sklopove dijelimo na asinkrone i sinkrone. Asinkroni sklopovi reagiraju na promjenu ulaznih varijabli. Sinkroni sklopovi reagiraju na promjenu sinkronizacijskog impulsa. Sinkronizacijski impuls nazivamo još i impuls takta ili samo takt. U nastavku ćemo se držati samo sinkronih izvedbi sekvencijskih sklopova.

2.1.3.1. Bistabil

Bistabil (*engl. flip-flop*) je najjednostavniji elektronički sklop s dva stabilna stanja. U načelu se sastoji od dva invertora (sklopa *NE*) povezana u povratnu vezu kao što je prikazano na slici 2.9. U tom je slučaju uvijek na ulazu prvog negirana vrijednost ulaza drugog invertora, tj $Q = \overline{Q}$ gdje je Q ulaz prvog invertora, a \overline{Q} ulaz drugog invertora. Ova razmatranja vrijede za sve izvedbe logičkih invertora. Neke od sklopovskih izvedbi invertora, kao što je korištenjem sklopa *NI*, već su opisane. Izvedbe invertora u elektronici neće se razmatrati u sklopu ovog rada.



Slika 2.10 Bistabil izveden invertorima

Opisani sklop u stanju je pamtiiti informaciju veličine jednog bita. Sa stajališta digitalne elektronike razlikujemo nekoliko tipova bistabila. Tipovi bistabila razlikuju se po ulazima kojima reguliramo njihovo interno stanje.

SR-bistabil ima dva ulaza, postavi (S , engl. *set*) i poništi (R , engl. *reset*). Kad je postavljen ulaz S i dođe do okidanja na izlaz se postavlja vrijednost 1 i pamti se to stanje. Kada je postavljen ulaz R na izlaz se postavlja vrijednost 0 i pamti se to stanje. Ako su oba ulaza u logičkoj nuli u trenutku okidanja, neće doći do promjene stanja ni izlaza. Tablica stanja dana je tablicom 2.7, a grafički simbol dan je slikom 2.11.a. Ova izvedba ne definira što se sa sklopom događa kada su postavljeni i ulaz S i ulaz R , štoviše, za ovaj tip bistabila ovo je zabranjena pobuda. Ako se ograničimo na način da kažemo da se na ulazu ne smije i ne može pojaviti ta kombinacija, funkciju novo stanje (Q_{n+1}) i izlaz za SR-bistabil možemo dati formulom (Q_n je trenutno stanje):

$$Q_{n+1} = S \vee (Q_n \wedge \neg R)$$

2.7 Tablica SR-bistabila

S	R	Q_{n+1}
\perp	\perp	Q_n
\perp	\top	\perp
\top	\perp	\top
\top	\top	?

JK-bistabil je drugi u nizu bistabila s dva ulaza. Ime i oznake za ulaze ovaj je tip bistabila dobio u čast Jack Killbya. JK-bistabil rješava problem SR-bistabila s postavljenim signalima oba ulaza. Samo ponašanje JK-bistabila ekvivalentno je ponašanju SR-bistabila uz $J = S$ i $K = R$. Jedina promjena je stanje $J = 1$ i $K = 1$ za koje JK-bistabil prelazi u stanje komplementarno trenutnom. Funkciju novo stanje i izlaz za JK-bistabil bez ograničenja dajemo formulom:

$$Q_{n+1} = (J \wedge \neg Q_n) \vee (\neg K \wedge Q_n)$$

2.8 Tablica JK-bistabila

S	R	Q_{n+1}
\perp	\perp	Q_n
\perp	\top	\perp
\top	\perp	\top
\top	\top	$\neg Q_n$

Ako se ulazi JK-bistabila spoje međusobno dobiva se T-bistabil. Na osnovi tablice stanja JK-bistabila lako je odrediti tablicu T-bistabila. T-bistabil ima jedan ulaz. Ime je dobio po svom ponašanju pošto u trenutku okidanja mijenja stanje ako je ulaz postavljen. Oznaka T dolazi od engleske riječi *toggle* što znači prebacivati.

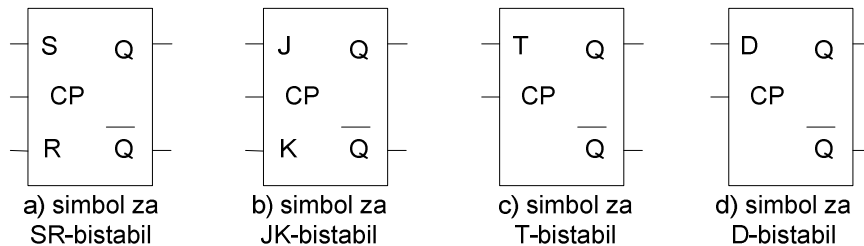
2.9 Tablica T-bistabila

T	Q_{n+1}
1	Q_n
0	$\neg Q_n$

Modifikacija SR-bistabila kod kojeg je jedan ulaz spojen i na S i njegova negirana vrijednost na R ulaz daje D-bistabil. Oznaka D dolazi od engleskih riječi *data* i *delay*. D-bistabil pamti trenutni ulaz i preslikava ga na izlaz.

2.10 Tablica D-bistabila

D	Q_{n+1}
1	1
0	0



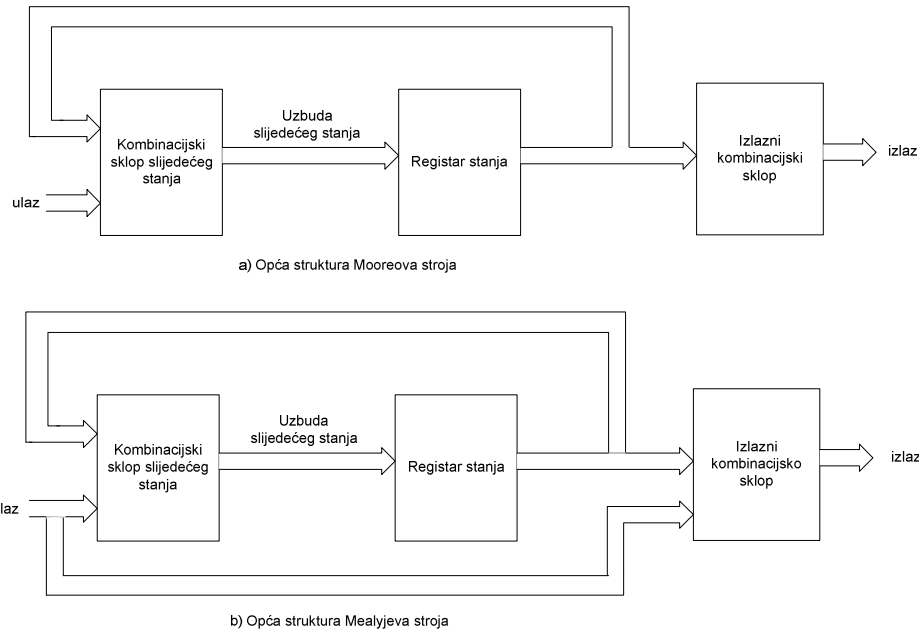
Slika 2.11 Simboli bistabila

2.1.3.2. Strojevi stanja

Sinkroni sekvencijski sklopovi sastoje se od bistabila i kombinacijskih logičkih sklopova. Sklopovi koji u sebi sadrže n bistabila sposobni su pamti n -bitnu informaciju. Takve informacije imaju 2^n mogućih različitih vrijednosti što znači da je sklop u stanju pamti da se nalazi u jednom od 2^n različitih stanja. Zbog ove činjenice sinkrone sekvencijski sklopove nazivamo i strojevi stanja⁶. Budući da je broj stanja konačan nekad se rabi i naziv strojevi s konačnim brojem stanja ili konačni automati.

Razlikujemo dvije vrste strojeva stanja ovisno i njihovom dizajnu. To su Mooreovi i Mealyjevi strojevi stanja. Njihova se struktura razlikuje u logičkoj funkciji izlaza. Kod Mooreova stroja izlaz ovisi samo o trenutnom stanju, dok u slučaju Mealyjeva izlaz stroja ovisi o trenutnom stanju i pobudi. Funkcija sljedeće stanje u oba slučaja ovisi o ulazima i trenutnom stanju. Shematski prikazi za oba stroja dani su slikom 2.12.

⁶ engl. state machine



Slika 2.12 Opća struktura strojeva stanja

Konačni automat kao matematički model digitalnog sklopa stroja stanja definiramo kao uređenu šestorku:

$$DKA = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

gdje je:

- Q konačan skup stanja (binarna informacija stanja pohranjena u sklopu)
- Σ konačan skup ulaznih znakova (sve kombinacije koje se mogu pojaviti na ulazu sklopa)
- Δ konačan skup izlaznih znakova (sve kombinacije koje se mogu pojaviti na izlazu sklopa)
- δ funkcija prijelaza – funkcija novo_stanje
- $q_0 \in Q$ početno stanje

Razlika Mooreovog i Mealyjevog automata je definicija funkcije λ . Ona je kod Mooreovog automata definirana kao $\lambda : Q \rightarrow \Delta$, a kod Mealyjevog automata kao $\lambda : Q \times \Sigma \rightarrow \Delta$. Nad ovako definiranim matematičkim modelom strojeve stanja možemo prikazivati dijagramom toka ili strukturom usmjerenog grafa. Struktura usmjerenog grafa koristi se tako da se stanja prikazuju kao vrhovi grafa, a funkcija prijelaza δ kao bridovi usmjerenog grafa. Izlaz se kod Mooreova automata upisuje u vrhove, a kod Mealyjeva na bridove. Ovu strukturu nazivamo i dijagram stanja sekvencijskog sklopa. Sastavni dio sustava VHDLLab je i uređivač koji

omogućava modeliranje automata dijagramom stanja sekvencijskog sklopa te generiranje VHDL opisa sklopa i njegovu simulaciju. O ovoj temi će se raspravljati u jednom od kasnijih poglavlja ovog rada.

2.2. Jezik VHDL

2.2.1. Povijest jezika

U dizajnu elektroničkih sklopova postalo je gotovo nemoguće napraviti kompleksni sklop koristeći se klasičnim metodama (olovkom i papirom). U tu svrhu, kao pripomoć inženjerima, nastao je niz jezika za opis sklopovlja (*HDL*, engl. *Hardware Description Language*). Ovakvi jezici najčešće služe za oblikovanje sustava temeljenih na programirljivim sklopovima (*PLD*, *FPGA*). Najpopularniji su VHDL, Verilog i Abel. Jezik VHDL (engl. *Very High Speed Integrated Circuit HDL*) je razvijen pod sponzorstvom ministarstva obrane SAD-a. Prvi simulatori pojavili su se početkom 90-ih godina, a u široku upotrebu VHDL dolazi godine 1994. Iako VHDL pokriva područje automatske sinteze, glavni razlozi kreiranja jezika su modeliranje i simulacija i dokumentacija složenih sustava. Sinteza je dodana u kasnijoj fazi razvoja, a ovisi o proizvođačima programske potpore i programirljivih sklopova na koje se sintetizira željeni model. Jezik je relativno brzo normiran od strane udruženja IEEE. Prva norma pojavila se već 1987., dok je 1993. objavljena njena revizija, te skoro svi alati u potpunosti podržavaju tu normu. Pored temeljne norme (1076, VHDL) definirane su još i norme formata za razmjenu podataka za testiranje (1029, WAVES) i standardni tip podataka za viševrijednosnu logiku (1164, STD_LOGIC). Jezik je brojne koncepte i sintaksu posudio iz jezika Ada i Pascal.

Prednosti VHDL-a u odnosu na ostale formalne metode projektiranja su opis na različitim razinama, pogodnost za velike sustave (hierarhijski pristup projektiranju i ispitivanju složenih sustava), simulacija, sinteza i široko područje primjene (VHDL nije ograničen na modeliranje elektronike).

2.2.2. Struktura jezika

Svaki tehnički model, bez obzira na svoju funkciju, mora imati vezu za komunikaciju s okolinom. Taj komunikacijski dio sklopa nazivamo sučelje. Kvaliteta sučelja uvelike ovisi o kvaliteti samog sustava. U jeziku VHDL sučelje sustava deklariramo ključnom riječi *entity*. Deklaracija sučelja je temeljna jedinica oblikovanja sustava. Svaki VHDL model mora imati opis sučelja, odnosno konstrukt *entity*.

Da bi se postigao željeni cilj modela, podatci primljeni preko sučelja moraju proći nekakvu transformaciju. Ta se transformacija zbiva u unutrašnjosti (implementaciji) sustava. U VHDL-u ovaj se dio opisuje arhitekturom sklopa. Arhitektura se deklarira ključnom riječi *architecture*. Bez obzira dali je sustav jednostavan ili kompleksan, svaki se sustav u VHDL-u može opisati pomoću sučelja i implementacije.

2.2.2.1. Sučelje sustava

Bilo koje projektiranje mora započeti analizom okoline u kojoj bi sustav trebao raditi. Definicija sučelja sustava temeljni je element njegova opisa. Najčešći način dokumentiranja

sučelja je korištenje shematskih simbola kao što su simboli dani slikama 2.3, 2.4, 2.5, 2.6 i 2.10. Elementi sučelja u grubo se mogu podijeliti u dvije glavne grupe:

- **podatkovne veze** – prenose podatke iz sustava i u sustav (deklarirane ključnom riječi *port*)
- **parametri sustava** – podatci kao širina sabirnice ili frekvencija rada ili kašnjenje sklopa (deklarirani ključnom riječi *generic*)

Primjer jednog sučelja koje sadrži oba tipa elemenata dan je sljedećim VHDL-kodom:

```
entity mux2nal is
    generic(
        N:integer := 2;
        Td: time := 20ns;
    );
    port(
        sin0: in std_logic_vector(N-1 downto 0);
        sin1: in std_logic_vector(N-1 downto 0);
        sel: in std_logic;
        e: in std_logic;
        dout: out std_logic_vector(N-1 downto 0);
    );
end entity mux2nal;
```

Primjer 2.1 Sučelje multipleksora 2 na 1

2.2.2.2. Implementacija sustava

Implementacija sustava u VHDL-u opisuje se zasebnim odjeljkom (arhitekturom). Arhitektura se označava ključnom riječi *architecture*. Uz zaglavlje arhitekture mora biti navedeno sučelje koje ta arhitektura implementira. Za jedno sučelje moguće je definirati više arhitektura. Unutar *begin-end architecture* bloka naredbe se ne odvijaju jedna iza druge nego istodobno. Skica arhitekture za sučelje iz primjera bila bi:

```
architecture A of mux2nal is
    -- deklaracija internih signala sklopa
begin
    -- implementacija sklopa
end architecture A1;
```

Primjer 2.2 Skica architecture bloka

Implementacija sustava može biti dana opisom njegova ponašanja ili strukture. Obično se tijekom razvoja koriste oba načina i to tako da se prvo sustav modelira ponašajno, a zatim se alatima za transformiranje pretvara u strukturni model – to se naziva sinteza. Velik dio sinteze može se obaviti automatski, ali potpuno funkcionalna sinteza još uvijek ne postoji.

2.2.3. Ponašajni opis

Ponašajni opis sustava opis je koji odgovara na pitanje: „Što sustav radi?“. Ovaj pristup modeliranju ne daje vidljivo sklopovsko rješenje problema nego prisiljava korisnika da konstruktima jezika VHDL opiše željeni sklop. Iz takvog se opisa može alatima za sintezu izgraditi strukturni model.

Pri razvoju modela često je javlja potreba za opisom slijednih operacija koje valja izvesti kako bi se postigao željeni rezultat. Prema analogiji s operacijskim sustavima, jezični konstrukt u VHDL-u koji omogućava takav redoslijed izvođenja naziva se *process* blok. Sintaksa opisa *process* bloka je:

```
stavak_procesa <=
naziv:process(lista_osjetljivosti)
    deklaracije
begin
    slijedne naredbe
end process naziv;
```

Primjer 2.3 Sintaksa process bloka

Uporaba procesa može se ilustrirati na već definiranom primjeru multipleksora 2 na 1. Sučelje je opisano ranije pa ćemo samo opisati implementaciju:

```
architecture behavioral of mux2na1 is
begin
    p:process(sin0,sin1,sel)
    begin
        if (sel = '1') then
            dout <= sin1;
        else
            dout <= sin0
        end if;
    end process p;
end architecture behavioral;
```

Primjer 2.4 Implementacija multipleksora 2 na 1

Tijek izvođenja naredbi u procesu je slijedan. Proces se pokreće (okida) kada se promijeni vrijednost jednom od signala navedenih u listi osjetljivosti (*sin0*, *sin1* i *sel*). Unutar procesa možemo izazvati kašnjenje korištenjem *wait* naredbe, a u protivnom, vrijeme izvođenja naredbi između dvije *wait* naredbe sa stajališta simulatora je trenutno.

Konstrukt procesa u VHDL-u omogućava i deklariranje varijabli. Varijable se od signala razlikuju prema mjestu deklaracije, trenutku kada im se pridružuje vrijednost i kašnjenju. Što se deklaracije tiče, varijable se deklariraju unutar bloka procesa, a signali se deklariraju samo u *port* dijelu opisa sučelja i deklaracijskom dijelu izvedbe. Kada razmatramo pravila

pridruživanja, razlika između varijable i signala je u tome da se vrijednost varijabli pridružuje odmah, a signalu tek na kraju bloka ili pri nailasku na naredbu *wait*. Kašnjenje kod varijabli nije podržano pošto ne bi imalo smisla. Konstrukt procesa nudi i naredbe za kontrolu tijeka. U primjeru se vidi korištenje naredbe *if*. Podržano je još i *case* grananje. Od programskih petlji podržani su *while* i *for* konstrukti te dodatno uz njih naredbe za upravljanje petljama *next* i *exit*. Ako se pojavi više procesa u implementaciji, njihovo je izvršavanje usporedno, kao što je opisano kod naredbi pridruživanja.

2.2.4. Strukturni opis

Za razliku od opisanog ponašajnog modela, koji na sustav gleda kao cjelinu, strukturni model raščlanjuje sustav na jednostavnije podsustave čijim se spajanjem dobiva željena implementacija. Svaki od tih podsustava mora biti opisan zasebnim modelom koji može biti strukturni ili ponašajni.

Jedan od najbitnijih konstrukata strukturnog opisa sustava je izravno stvaranje primjeraka komponenti. Svaki modelirani podsustav može se upotrijebiti kao građevni element složenijeg ili hijerarhijski višeg sustava uporabom naredbe za stvaranje primjeraka. Prilikom stvaranja primjerka povezuju se signali na ulaze i izlaze komponente koja se stvara, što je u stvari analogno spajanju žica na neki sklop.

Formalna sintaksa naredbe stvaranja primjeraka predviđa mogućnost da se u složenom sustavu ne iskoriste svi formalni argumenti komponente. Na takvim mjestima potrebno je navesti ključnu riječ *open*. Ime arhitekture koja se koristi nije potrebno navesti ako sučelje koje koristimo ima samo jednu arhitekturu.

```
instanciranje_komponente <=
  labela:entity ime [(ime_arhitekture)]
    [generic map (lista_parametara)]
    port map(lista_veza);
lista_veza <=
  ( [formalna_veza => ] (signal | izraz | open))
```

Primjer 2.5 Sintaksa naredbe za stvaranje komponenti

Pokažimo stvaranje primjeraka komponenti na primjeru. Pokušat ćemo na strukturnoj razini opisati D-bistabil sa ulazom za omogućavanje uz pomoć D-bistabila i sklopa *I*. Sučelja ova dva sklopa dana su sljedećim kodom:

```
entity sklop_i is
  port(
    a,b: in std_logic;
    z: out std_logic);
end entity sklop_i;
```

Primjer 2.6 Opis sučelja za sklop I

```
entity bist_d is
    port(
        d,clk: in std_logic;
        q: out std_logic);
end entity bb_d;
```

Primjer 2.7 Opis sučelja za D-bistabil

```
entity bist_d_en is
    port(
        d,clk,e: in std_logic;
        q: out std_logic);
end entity bist_d_en;

architecture structural of dist_d_en is
    signal clke: std_logic;
begin
    skl: entity work.sklop_i
        port map (clk, e, clke);
    bb: entity work.dist_d
        port map (
            d => d,
            clk => clke,
            q => q
        );
end architecture;
```

Primjer 2.8 D-bistabil s ulazom za omogućavanje

Strukturno modeliranje u okviru VHDLLab sustava obavlja se uređivačem shema. Uređivač shema sustava VHDLLab nudi mogućnost crtanja sheme sustava i generiranje VHDL opisa za nacrtani model.

2.2.5. Implementacija digitalnih sklopova u jeziku VHDL

Kao završetak ovog kratkog uvoda u jezik VHDL dati ćemo par primjera VHDL kodova za sklopove opisane u poglavlju 2.1. Kod za multipleksor smo već imali kao primjer sučelja i implementacije u poglavlju 2.2.

Kao drugi primjer kombinacijskog sklopa uzet ćemo dekodekter opisan u poglavlju 2.1.2.1. Dekoder 2/4 s ulazima A0 i A1 te izlazima Y0, Y1, Y2, Y3 dan je nizom logičkih funkcija. Te logičke funkcije mogu se u okviru ponašajnog modela direktno modelirati.

$$Y0(A0, A1) = \neg A1 \wedge \neg A0$$

$$Y1(A0, A1) = \neg A1 \wedge A0$$

$$Y2(A0, A1) = A1 \wedge \neg A0$$

$$Y3(A0, A1) = A1 \wedge A0$$

```
entity dek24 is port(
    A0,A1: in std_logic;
    Y0,Y1,Y2,Y3: out std_logic);
end entity dek24;
architecture ponasajna of dek24 is
begin
    Y0 <= not A1 and not A0;
    Y1 <= not A1 and A0;
    Y2 <= A1 and not A0;
    Y3 <= A1 and A0;
end ponasajna;
```

Primjer 2.9 VHDL model za dekodekter 2/4

Ovaj jednostavan model opisan je na razini između ponašajne i strukturne. U nekim literaturama za ovaj način opisivanja modela koristi se termin „Data flow“ opis dok ostale i ovu razinu stavljaju u grupu ponašajnih opisa. Zgodno je još napomenuti da je simbol „<=“ pažljivo odabran da ne bi došlo do zabune sa simbolom „=“ ili „:=“. Simbol „<=“ ne označava prijenos podataka u programskom jeziku nego odnos između signala u VHDL modelu.

Za slijedeći primjer uzet ćemo predložak za VHDL model digitalnih automata. Mooreov i Mealyjev automat lako se po predlošku iz dijagrama stanja pretvaraju u VHDL kod. Automati se opisuju na ponašajnoj razini.

```
ENTITY Automat1 IS PORT(  
    --deklaracija ulaza i izlaza  
    clock: IN std_logic;  
    reset: IN std_logic;  
);  
END Automat1;  
  
ARCHITECTURE Behavioral OF Automat1 IS  
    --definiranje konstanti kao kodova stanja  
    CONSTANT ST_A:  std_logic := "0...000";  
  
    SIGNAL state_present, state_next:  std_logic;  
  
BEGIN  
  
    PROCESS(state_present, a)  
    BEGIN  
        CASE state_present IS  
            --opis funkcije novo_stanje  
            WHEN ST_A=>  
                state_next<=ST_A;  
            WHEN OTHERS => state_next <= state_present;  
        END CASE;  
    END PROCESS;  
  
    PROCESS(state_present)  
    BEGIN  
        CASE state_present IS  
            --opis funkcije izlaz  
            WHEN ST_A =>  
                b <= '0';  
            WHEN OTHERS =>  
                b <= '0';  
        END CASE;  
    END PROCESS;  
  
    PROCESS(clock, reset)  
    BEGIN  
        --okidanje automata  
        IF reset='0' THEN  
            state_present <= ST_A;
```

```
ELSIF falling_edge(clock) THEN
    state_present <= state_next;
END IF;
END PROCESS;
END Behavioral;
```

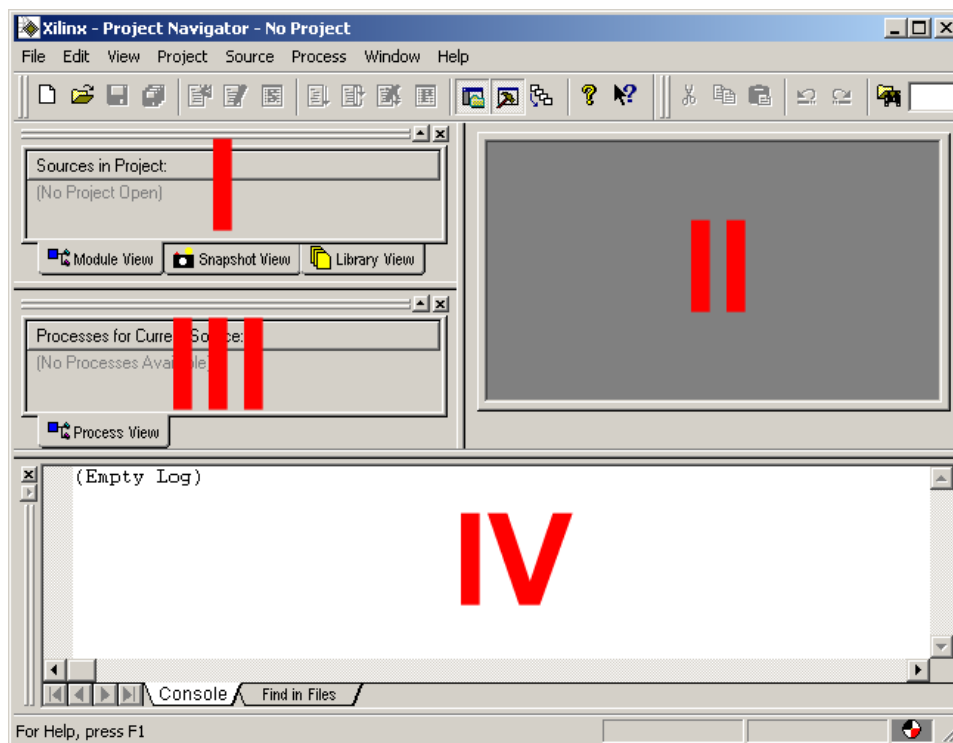
Primjer 2.10 Opis Mooreova digitalnog automata

Primjerom 2.10 dan je predložak za Mooreov automata. Opis je ponašajni i sastoji se od tri bloka procesa. Prvi blok predstavlja funkciju novog stanja i u listi osjetljivosti sadrži ulaze i signal trenutnog stanja pošto funkcija novog stanja ovisi o tim parametrima. Drugi proces blok predstavlja funkciju izlaza i ima samo trenutno stanje u listi pošto kod Mooreovog automata izlaz ovisi samo o trenutnom stanju. Zadnji se blok pojavljuje kao kontrola okidanja to jest sinkronizacije automata.

Mealyjevi automati se modeliraju na sličan način. Jedina veća razlika je u tome što i funkcija novog stanja i funkcija izlaza ovise o trenutnom stanju i ulazima pa se stoga mogu zajedno modelirati jednim proces blokom. U sklopu VHDLLabu postoji uređivač automata u sklopu kojeg se dijagramom stanja definira automat te se navedenom šablonom generira kod modela.

2.3. Sustav WebISE

U sklopu predmeta Digitalna logika na Fakultetu elektrotehnike i računarstva na Sveučilištu u Zagrebu koristi se program *WebISE* u kombinaciji s programom *ModelSim* proizvođača *Xilinx*. *WebISE* je jedan od rijetkih besplatnih „front – to – end“ programa za oblikovanje digitalnih sustava s verzijama za operacijske sustave *Linux* i *Windows*. Nudi implementaciju, simulaciju i sintezu sustava za sklopove FPGA i CPLD, kao i još niz funkcionalnosti koje se ne koriste u okviru predmeta Digitalna logika. Sklopovi na koje se sintetiziraju sustavi su Xilinx-ovi FPGA sklopovi iz serija *Spartan* i *Virtex* te CPLD sklopovi *CoolRunner* i *XC9500*. Iako proizvođač navodi kako je ovaj program jednostavno koristiti, u svrhe učenja pokazao se je dosta kompliciranim.

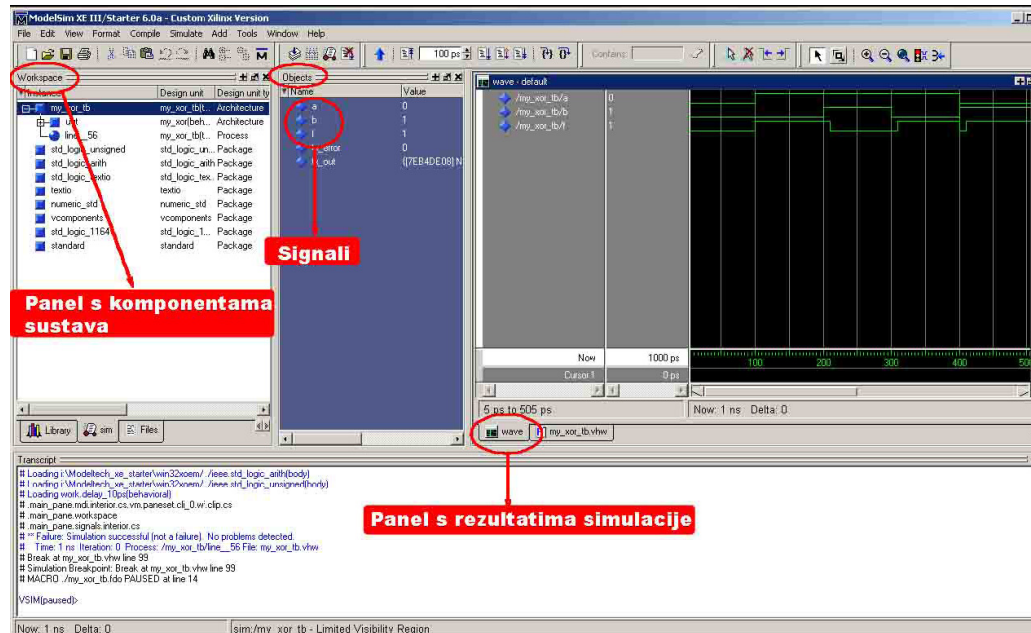


Slika 2.13 Sučelje programa WebISE

Prikaz sučelja *WebISE* programa dan je slikom 2.13. Na njoj se vidi da je sučelje podijeljeno u četiri osnovna djela:

- I. Sources in project – prikazuje sve datoteke koje čine dizajn
- II. Prozor u kojem se otvaraju uređivači VHDL-koda ili nekih drugih dodatnih alata u sklopu programa
- III. Prozor koji prikazuje trenutno raspoložive operacije. Ovisi o odabranoj datoteci u prozoru I.
- IV. Statusni prozor u kojemu se ispisuju naredbe koje se pokreću tijekom rada programa te prikazuju njihovi izlazi.

Pored samog *Web/SE* programa koji služi za modeliranje, za simulaciju se koristi program *ModelSim*. *ModelSim* je PC HDL simulacijska okolina koja omogućava verifikaciju HDL kodova funkcijskih i vremenskih modela u dizajnu sustava. Ovaj program nije ograničen samo na VHDL, nego pruža mogućnost verifikacije modela dizajniranih u Verilogu ili nekom drugom HDL-u. Što se samog VHDL-a tiče, ovaj program je u skladu sa normama IEEE 1076 iz 1987. i 1993. i kao takav nudi brzo i kvalitetno sučelje za verifikaciju VHDL modela.



Slika 2.14 Sučelje programa ModelSim

Mana kombinacije ova dva programa je u tome da je riječ o kompleksnom programskom paketu koji nije namijenjen u obrazovne svrhe. Sama instalacija navedenog programskog paketa na računalu predstavlja zamoran i kompliciran postupak. Programi nude velik broj opcija koje razumiju samo iskusni dizajneri sklopova, a koje redovito zbunjuju prosječnog korisnika.

2.4. Simulator GHDL

GHDL⁷ je besplatni VHDL simulator baziran na GCC tehnologiji. Napisan je u programskom jeziku Ada95. U sklopu njega podržane su VHDL norme IEEE 1076-1987 i IEEE 1076-1993, kao i norme viševrijednosne logike IEEE 1164 – Std_Logic. GHDL ne pruža opciju sinteze modela definiranog jezikom VHDL na programirajući logički sklop. GHDL je u stvari prevodioc koji VHDL datoteku prevodi direktno u strojni kod, a ne u prijelazni kod. Ta ideja smišljena je da bi se dobilo na brzini prilikom simulacije sustava. Ovaj simulator pruža opciju generiranja VCD datoteke koja sadrži valne oblike dobivene simulacijom sustava u nekom testnom okruženju. Prikažimo rad GHDL-a na jednostavnom primjeru „Hello world“. Prvo napravimo VHDL datoteku koju nazovemo hello.vhdl. U nju unesemo kod dan primjerom 2.11:

```
use std.textio.all;
entity hello_world is
end hello_world;
architecture behaviour of hello_world is
begin
    process
        variable l : line;
    begin
        write (l, String'("Hello world!"));
        writeline (output, l);
        wait;
    end process;
end behaviour;
```

Primjer 2.11 VHDL model „Hello world!“

Prvo model trebamo prevesti (kompajlirati). To radimo, pozivom ghdl-a iz komandne linije ili terminala, na sljedeći način:

```
$ ghdl -a hello.vhdl
```

Sada trebamo napraviti izvršnu datoteku. Ponovno pozivamo GHDL na sljedeći način:

```
$ ghdl -e hello_world
```

Dobivenu izvršnu datoteku pokrećemo sa:

```
$ ghdl -r hello_world
```

```
Hello world!
```

Više riječi o upotrebi GHDL simulatora u okviru VHDLlab-a bit će u kasnijim poglavljima.

⁷ kratica za G Hardware Description Language, G trenutno nema značenje

2.5. Ostale tehnologije modeliranja

Osim jezika VHDL u svijetu modeliranja sustava postoji još niz tehnologija. Jedna od njih je već spominjani jezik Verilog. Jezik Verilog spada kao i VHDL u skup jezika koje nazivamo HDL. Verilog omogućava dizajn, verifikaciju i implementaciju analognih i digitalnih sklopova na različitim razinama apstrakcije. Jezik Verilog osmislili su Phil Moorby i Prabhu Goel krajem 1983. godine. Godine 1995 Verilog postaje IEEE norma (IEEE 1364-1995) pod nazivom Verilog-95. Trenutna norma je Verilog-2005 (IEEE 1364-2005). Svoje jezične konstrukte Verilog je posudio iz jezika C i C++. Zajedno s VHDL-om, Verilog je danas najrašireniji jezik za modeliranje sustava.

Sintaksa Veriloga ponekad je jednostavnija i čitljivija od VHDL-a. Da ilustriramo tu tvrdnju dat ćemo primjer za ponašajni opis bistabila napisan u jeziku Verilog.

```
module toplevel(clock,reset);
    input clock;
    input reset;
    reg flop1;
    reg flop2;
    always @ (posedge reset or posedge clock)
        if (reset)
            begin
                flop1 <= 0;
                flop2 <= 1;
            end
        else
            begin
                flop1 <= flop2;
                flop2 <= flop1;
            end
    end
endmodule
```

Primjer 2.12 Ponašajni opis bistabila jezikom Verilog

Prva velika razlika je da Verilog nema odvojeno sučelje od implementacije. Iako izgleda da je to jednostavnije rješenje, odvojena implementacija VHDL-u omogućava definiranje više različitih implementacija za jedno sučelje. Konstrukti signala i varijabli u oba jezika su slični sintaksno i semantički. Oba jezika nude ponašajnu i strukturnu razinu apstrakcije te sama odluka koji jezik koristiti leži isključivo na korisniku ili željenoj ponuđenoj tehnologiji sinteze.

VHDLLab ne omogućava modeliranje sustava u jeziku Verilog makar svojom strukturom i modularnom izvedbom nudi jednostavnu opciju proširenja i na taj slučaj.

Jezik Abel je danas toliko podređen VHDL-u i Verilogu da nema smisla opisivati ga u sklopu ovog rada.

3. Sustav VHDLLab

VHDLLab je Web orijentirana integrirana okolina za razvoj (*IDE, engl. integrated development environment*) koja služi za izgradnju i testiranje digitalnih sklopova putem jezika VHDL. Pojam integrirane okoline za razvoj, označava skup alata objedinjen u jednom programu koji služi za ubrzani razvoj programske i druge opreme, najčešće putem nekog jezika, u ovom slučaju VHDL-a. VHDLLab je Web orijentiran zato što se, kao što ćemo vidjeti u nastavku, svi podaci drže na poslužitelju. Prevođenje i simulacija se također obavljaju na poslužitelju, dok je na korisničkoj strani isključivo sučelje za uređivanje VHDL koda, shema digitalnih sklopova, automata, ispitnih sklopova i preglednik simulacija.

Skup alata se kod VHDLLab razvojne okoline sastoji od već spomenutih sučelja na korisničkoj strani i od VHDL prevodioca i simulatora na poslužiteljskoj strani. Svrha ovog poglavlja je upoznavanje sa značajkama korisničke strane VHDLLaba, opisom sučelja i komunikacijom korisničkog sučelja s poslužiteljem, dok je tehnički opis dan u sljedećem poglavlju.

3.1. Značajke

VHDLLab je integrirana okolina za razvoj temeljena na Java tehnologiji, što znači da je prenosiv (*cross-compatible*) između svih platformi za koje postoji Java podrška. Drugim riječima, VHDLLab je moguće pokrenuti na operacijskim sustavima *Windows, Linux, MacOS, Solaris*, itd. Ovo je ujedno i njegova prva bitna značajka, jer nije potrebno izdavati niz različitih inačica programa za različita računala i operacijske sustave – dovoljna je jedna inačica, i ona radi na svim sustavima.

Konkretno, VHDLLab se pokreće uporabom tehnologije *Java Web Start*. Unošenjem odgovarajuće adrese u internetski preglednik program se instalira, te ga je potom moguće pokretati izravno iz operacijskog sustava. *Java Web Start* dodatno omogućuje automatsko nadograđivanje na najnoviju verziju programa – kad god se izradi poboljšana verzija sustava VHDLLab, svi će korisnici već pri sljedećem pokretanju imati tu verziju na svome računalu. Ovo korisno svojstvo omogućuje lakšu ugradnju novih mogućnosti u program i djelotvorno ispravljanje eventualnih problema.

VHDLLab zahtijeva da svaki korisnik ima svoje korisničko ime i zaporku, koju mu dodjeljuje administrator sustava (osoba koja instalira i podešava poslužitelj za VHDLLab). Jednom kad se VHDLLab pokrene, potrebno je stoga unijeti odgovarajuće korisničko ime i zaporku, nakon čega je moguće započeti s radom. Smisao zaporke je da se omogući pristup projektima isključivo korisniku kome isti pripadaju. Primjerice, u slučaju korištenja VHDLLaba u okviru laboratorijskih vježbi, poželjno je da isključivo student ima pristup svojim projektima

(uz iznimku asistenta koji njegove projekte smije nadgledati). VHDLLab u svom radu koristi zaštitu putem SSL⁸ protokola, što ga čini dobro zaštićenim od potencijalnih napada.

Jednom kad se korisnik autentificira korisničkim imenom i zaporkom, dobiva pristup skupu svojih projekata, kao što je ranije spomenuto. U VHDLLabu su sve datoteke logički organizirane u projekte, pa je, dakle, projekt skup datoteka koje mu pripadaju. Svaki projekt ima svoj naziv koji korisnik odabire prilikom stvaranja projekta. Tipičan ciklus rada u VHDLLab sustavu sastoji se od stvaranja novog ili otvaranja postojećeg projekta, stvaranja novih datoteka različitih sadržaja ili rada na postojećim, izrade ispitnih sklopova, slanja datoteka na poslužitelj gdje se obavlja prevođenje i simuliranje, te proučavanja rezultata simulacija. Kad se VHDLLab koristi u okviru laboratorijskih vježbi, tad najčešće jedan projekt predstavlja jednu laboratorijsku vježbu čiji je cilj modelirati neki digitalni sklop.

Datoteke koje se nalaze unutar projekta mogu biti različitih sadržaja. Trenutno podržan skup datoteka su datoteka s vhdl kodom, datoteka apstraktnog automata sklopa, datoteka sheme digitalnog sklopa, datoteka s ispitnim sklopom i datoteka s rezultatima simulacije. Za svaku od tih datoteka postoji zasebni uređivač koji služi za pregledavanje ili uređivanje te vrste datoteke, kao što će to biti opisano u daljnjim potpoglavljima.

Za pisanje VHDL koda VHDLLab ima uređivač koda unutar kojeg je moguće VHDL kodom strukturno, ponašajno ili hibridno opisati digitalni sklop. VHDL je detaljno opisan u prethodnom poglavlju. Nakon slanja VHDL koda na poslužitelj i prevođenja istog, unutar uređivača koda su naznačene eventualne korisničke pogreške. Ako je VHDL kod kojim je digitalni sklop opisan ispravan, moguće je pristupiti izradi ispitnog sklopa, koji služi za provjeru ispravnosti rada sklopa. Za izradu ispitnog sklopa predviđen je poseban uređivač u kojem je moguće podesiti vrijednosti svih ulaznih signala sklopa koji se ispituje, i to u različitim vremenskim trenucima. Sam uređivač na temelju vrijednosti signala u određenim trenucima iz predloška generira VHDL kod ispitnog sklopa, koji se potom šalje na poslužitelj, tamo prevodi i simulira. Rezultati simulacije se potom šalju nazad do korisnika u obliku datoteke koja sadrži simulaciju. Za pregled simulacije opet postoji poseban uređivač u kojem je moguće promatrati vrijednosti svih ulaznih, izlaznih i unutarnjih signala sklopa u svim vremenskim trenucima, te na temelju toga odrediti funkcionira li sklop ispravno. Uređivač za pregledavanje simulacije omogućuje prikaz dijagrama vrijednosti svih signala u vremenu, pomicanje duž vremenski osi i povećanje mjerila vremenske osi, te ispis svih vrijednosti signala u odabranom trenutku.

Kako se opis sklopova VHDL kodom često svodi na ponašajni i strukturni opis, napravljena su i dva dodatna alata kojima je moguće ubrzati razvoj digitalnih sklopova. Prvi

⁸ protokol prijenosnog sloja koji osigurava sigurnu komunikaciju na Internetu (*engl. Secure Socket Layer*)

od njih je uređivač apstraktnih automata koji predstavljaju digitalne sklopove. Kao što je spomenuto u prethodnom poglavlju, svaki se digitalni sklop može predstaviti apstraktnim automatom koji ima konačni broj stanja, pa je određivanjem svih stanja sklopa i prijelaza između pojedinih stanja na temelju vanjskih pobuda (ulaznih signala) potpuno opisan digitalni sklop. Uređivač apstraktnih automata omogućuje korisniku stvaranje novih stanja, njihovo razmještanje po platnu za crtanje automata, određivanje početnog stanja i određivanje prijelaza između stanja. Jednom kad korisnik odredi sva stanja, prijelaze između njih i početno stanje, informacije o automatu spremaju se u datoteku pripadnog tipa koja se šalje na poslužitelj. Na poslužitelju se iz datoteke s informacijama o automatu generira VHDL ponašajni opis digitalnog sklopa koji predstavlja taj automat, nakon čega je taj VHDL opis moguće prevesti, te za njega napraviti ispitni sklop i obaviti simulaciju. Ponašajni VHDL opis izgeneriran na temelju automata moguće je pregledati u korisničkom sučelju, što između ostalog ima obrazovnu vrijednost, jer studentu koji se tek upoznaje s VHDL-om daje priliku da bolje shvati povezanost modela pomoću automata i VHDL modela istog digitalnog sklopa.

Drugi alat za ubrzano generiranje VHDL koda stvara strukturne opise digitalnih sklopova na temelju sheme digitalnog sklopa. Naime, većinu je digitalnih sklopova moguće modelirati na temelju više međusobno povezanih digitalnih sklopova, koji služe kao građevne komponente. Građevne komponente mogu biti osnovni digitalni sklopovi i bistabila, koji predstavljaju građevne primitive u izgradnji digitalnih sklopova, ili složeni digitalni sklopovi poput registara, multipleksora ili dekodera. U modeliranju digitalnog sklopa shemom, moguće je koristiti velik broj unaprijed definiranih sklopova, kao i sve sklopove koje je korisnik napravio unutar istog projekta. Primjerice, ako korisnik VHDL kodom opiše multipleksor, tad ga može koristiti unutar shematskog opisa nekog drugog digitalnog sklopa. Jednom razmještene sklopove moguće je međusobno povezati žicama, te žicama njihove ulaze i izlaze povezati na ulaze i izlaze modeliranog sklopa. Na taj način dobiva se složeniji sklop nove funkcionalnosti, koji uređivač shema sklopova pohranjuje u datoteku za opis sheme, i koja se šalje na poslužitelj. Baš kao i kod datoteke za opis putem apstraktnog automata, iz datoteke za opis sheme se na poslužitelju generira opis digitalnog sklopa VHDL kodom, u ovom slučaju strukturni opis, koji je moguće prevesti, izgenerirati mu ispitni sklop i simulirati ga. U slučaju da je korisnik pojedine sklopove nepravilno ili nepotpuno povezao, strukturni VHDL opis neće biti izgeneriran, a korisniku će se dojaviti opis pogreške, kako bi je lakše ispravio. VHDL opis dobiven iz sheme digitalnog sklopa moguće je pregledati, što korisniku koji se tek upoznaje s VHDL-om i strukturnim opisom omogućuje da bolje shvati povezanost između sheme digitalnog sklopa i VHDL opisa, te da si lakše predoči što su ulazni, izlazni i unutarnji signali sklopa.

Sve korisničke pogreške u napisanom VHDL kodu, modelu apstraktnog automata ili shematskom opisu digitalnog sklopa pronalaze se jednom kad se pripadne datoteke prebace

od korisnika na poslužitelj. Nakon toga poslužitelj određuje gdje se nalazi pogreška i koja je vrsta pogreške, te na temelju toga stvara poruku o pogrešci i šalje je korisniku. Unutar korisničkog sučelja se tad prikazuje pripadna poruka pogreške, kao i njen kratak opis.

Sve projekte koje korisnik stvori, kao i sve datoteke unutar njih, na uvid ima administrator sustava. Smisao ovoga je da u slučaju laboratorijskih vježbi student radi projekt i unutar njega smješta opise digitalnih sklopova koje je potrebno ostvariti u danoj laboratorijskoj vježbi. Nakon što je završio vježbu, njegov projekt ostaje pohranjen na poslužitelju, gdje asistent pregledava ispravnost simulacija njegovih sklopova, te na temelju toga odlučuje je li laboratorijska vježba uspješno izvršena. Sustav je moguće proširiti na način da se automatizira pregled rezultata laboratorijskih vježbi na način da sam poslužitelj odluči o ispravnosti rezultata simulacija, bez uplitanja asistenta, budući da za velik broj studenata ovo može biti dug i mukotrpan posao.

Naposljetku, treba napomenuti da VHDLLab nudi i mehanizam za jednostavno prevođenje programa na različite jezike, što omogućuje korištenje VHDLLab-a i u drugim zemljama.

3.2. Opis sučelja i svih editora

Nakon unošenja korisničkog imena i zaporka, i uspješne autentifikacije, prikazuje se korisničko sučelje VHDLLab-a. Na samom vrhu je glavni izbornik. S lijeve strane nalazi se preglednik projekata (*engl. project explorer*). Na dnu se nalazi lista prethodnih radnji (*engl. status history*). S desne se strane nalazi glavni prozor u kojem se može nalaziti bilo koji od ranije navedenih uređivača.



The image shows a horizontal menu bar with five items: 'File', 'Edit', 'View', 'Tools', and 'Help'. Each item is underlined, suggesting a standard menu interface.

Slika 3.1 Glavni izbornik

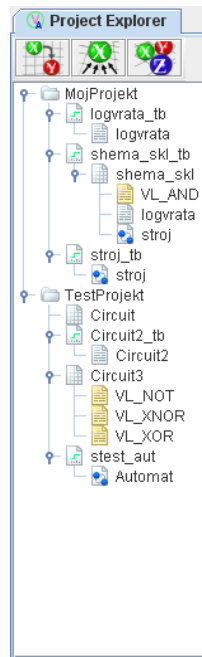
Glavni se izbornik sastoji od izbornika *File*, *Edit*, *View*, *Tools* i *Help*. U izborniku *File* moguće je odabrati neku od osnovnih operacija vezanih uz rad s projektima i datotekama. Moguće je stvoriti novi projekt odabirom opcije *New* i potom *New project*, a moguće je u postojećem projektu stvoriti novu datoteku za VHDL kod odabirom *VHDL source*, novi ispitni sklop odabirom *Testbench*, novi shematski opis sklopa odabirom *Schema* ili novi automat odabirom na opciju *Automat*. Ovdje je također moguće pohraniti datoteku koja je trenutno odabrana u glavnom prozoru opcijom *Save*, ili pohraniti sve otvorene datoteke opcijom *Save all*, te zatvoriti trenutno odabranu datoteku odabirom *Close* ili zatvoriti sve otvorene datoteke opcijom *Close all*. Naposljetku, moguće je zatvoriti program opcijom *Exit*.

U izborniku *Edit* moguće je poništiti zadnje učinjenu operaciju odabirom *Undo*. Primjerice, ako smo u uređivaču VHDL koda zabunom obrisali neki dio koda koji ne želimo pisati nanovo, tad ova opcija poništava brisanje tog koda, te vraća VHDL kod u prethodno stanje. Ako, nakon toga, ipak želimo obrisati navedeni kod, tu je još opcija i za to odabirom *Redo*.

Izbornik *View* nudi funkcionalnost vezanu uz razmještanje pojedinih uređivača i elemenata sučelja po ekranu, te prikazivanje raznih elemenata sučelja. Moguće je povećati trenutno odabrani uređivač na prostor čitavog ekrana kako bi rad bio ugodniji (*Maximize active window*). U podizborniku *Show view* moguće je uključiti podprozore za popis pogrešaka nastalih prilikom prevođenja ili simulacije, podprozor za listu prethodnih radnji, te preglednik projekata.

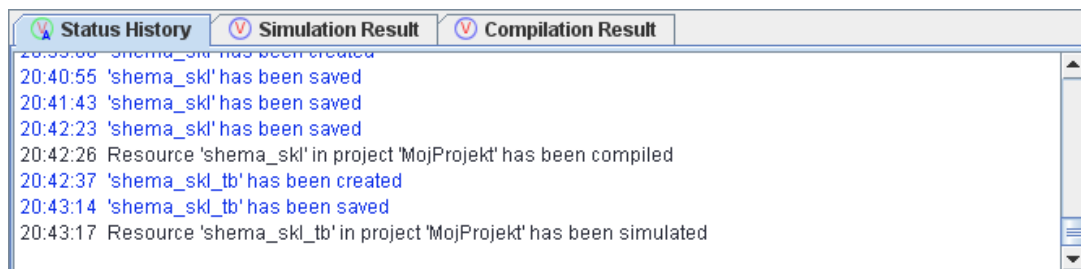
Izbornik *Tools* omogućuje prevođenje i simulaciju trenutno odabranog VHDL koda, modela sklopa pomoću automata ili sheme digitalnog sklopa. Opcije *Compile...* i *Compile active* pokreću prevođenje sklopa opisanog VHDL kodom, modelom automata ili shemo. Opcije *Simulate...* i *Simulate active* pokreću simulaciju ispitnog sklopa i rezultate uspješne simulacije prikazuju u novom uređivaču. Rezultate simulacije moguće je potom spremiti putem izbornika *File*. Opcija *View VHDL code* ima različit učinak ovisno o vrsti trenutno odabrane datoteke. Ako je trenutno odabran VHDL kod, tad ništa nije potrebno učiniti, s obzirom da je VHDL kod već vidljiv. Međutim, ako je odabran shematski opis digitalnog sklopa, tad se datoteka sa shematskim opisom prevodi u strukturni opis VHDL kodom, te se taj VHDL kod prikazuje u novom prozoru. Učinak u slučaju odabrane datoteke s modelom automata je sličan. Naposljetku opcija *View preferences* otvara dijalog s popisom svih postavki korisnika, u kojem korisnik može podesiti niz opcija svojim potrebama.

Izbornik *Help* nudi opcije vezane uz pomoć pri radu s VHDLLab-om, kao i popis programera koji su radili na razvoju VHDLLab-a.



Slika 3.2 Preglednik projekata

Kao što je već spomenuto, s lijeve se strane nalazi preglednik projekata. Unutar preglednika projekata popisani su nazivi svih projekata. Klikom na pojedini projekt prikazuju se sve datoteke koje se nalaze unutar tog projekta. Dvostrukim klikom na pojedinu datoteku u glavnom se prozoru otvara uređivač za pripadnu vrstu datoteke. Iznad liste projekata i datoteka unutar njih nalaze se tri gumba kojima je moguće određivati način prikaza datoteka unutar projekata. Najjednostavniji prikaz je tzv. plosnati prikaz (*engl. flat view*) kod kojeg su datoteke unutar pojedinog projekta naprosto izlistane. Ako se odabere X-koristi-Y (*engl. X-uses-Y*) prikaz, tad su datoteke prikazane hijerarhijski u stablu na način da je datoteka VHDL sklopa koji je opisan pomoću drugih sklopova na vrhu stabla, a da se datoteke sklopova koji su korišteni u opisu drugih sklopova nalaze na dnu stabla. Treća vrsta prikaza je tzv. X-je-korišten-od-Y (*engl. X-is-used-by-Y*) prikaz kod kojeg je hijerarhija obrnuta slučaj one kod X-koristi-Y prikaza.



Slika 3.3 Lista prethodnih radnji

Lista prethodnih radnji nalazi se na dnu ekrana. U njoj su opisane sve radnje i događaji vezani uz rad u VHDLLab-u, komunikaciju s poslužiteljem, prevođenje i simulaciju, kao i njihova pripadna vremenska oznaka. U slučaju eventualnih grešaka u radu sustava, korisnik će o tome biti obaviješten putem ovog prozora. Ovisno o prirodi sadržaja, poruke su označene različitim bojama, pa je tako u slučaju pogreške poruka ispisana crvenom bojom, a u slučaju uspješno izvršene radnje plavom bojom.

3.2.1. Uređivač koda

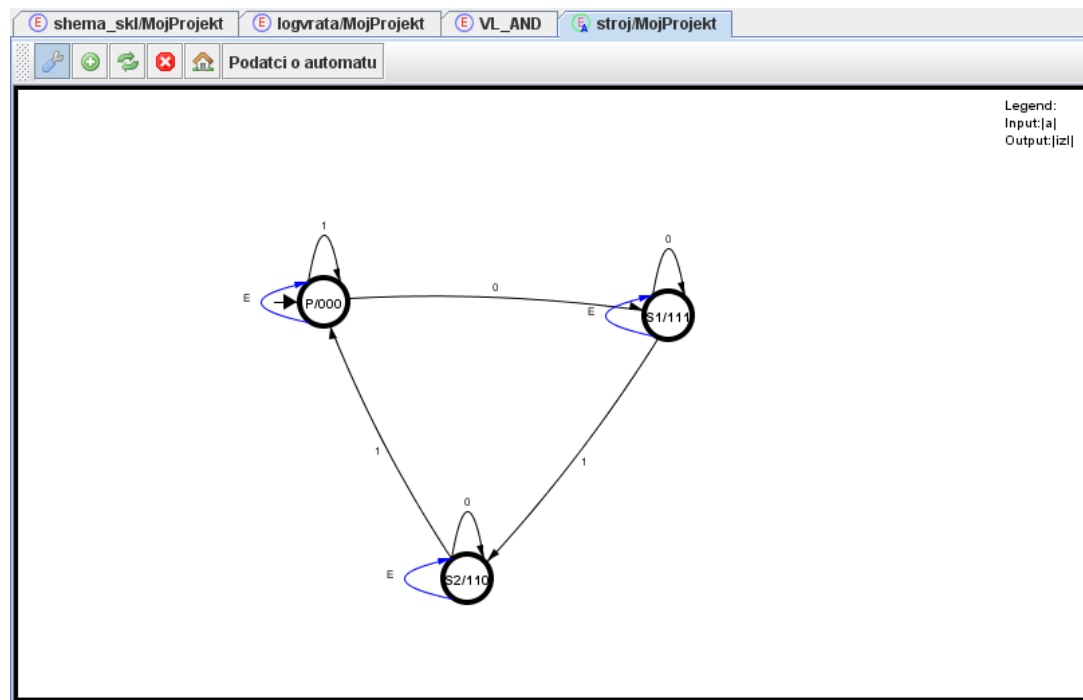


Slika 3.4 Uređivač koda

Za pisanje datoteka s VHDL kodom koristi se uređivač koda. U slučaju stvaranja datoteke s VHDL kodom, otvara se dijalog u kojem je moguće odrediti ulazne i izlazne signale modeliranog digitalnog sklopa, pa stvorena datoteka neće biti prazna, nego će u njoj biti izgeneriran *entity* blok VHDL opisa sa svim pripadnim ulaznim i izlaznim signalima. Uređivač koda otvorit će se u glavnom prozoru, nakon čega je moguće mijenjati sadržaj datoteke. Nakon slanja datoteke na poslužitelj i prevođenja, u sve će korisničke pogreške u datoteci biti prijavljene na dnu korisničkog sučelja. Klikom na poruku pojedine pogreške u uređivaču koda će se pokazivač trenutnog znaka smjestiti u liniju u kojoj se nalazi pogreška.

3.2.2. Uređivač automata

Stvaranjem datoteke s modelom automata, ili odabirom iste u pregledniku projekata, otvara se uređivač automata. U slučaju novog automata, baš kao i kod uređivača koda, uređivač automata će ponuditi dijalog, unutar kojeg je moguće odrediti je li automat Mooreov ili Mealyev, da li će biti resetiran na visoku ili nisku razinu signala za reset, i da li će reagirati na vanjsku pobudu uz rastući ili padajući brid, nisku ili visoku razinu signala vremenskog takta. Potom se otvara još jedan dijalog u kojem je moguće odrediti ime sklopa, te ulazne i izlazne signale. Uređivač automata sastoji se od platna na kojem su prikazana stanja automata i prijelazi između njih, koje zauzima većinu uređivača i koje je moguće klizati u svim smjerovima, za slučaj da sva stanja automata ne stanu na ekran, te alatne trake na vrhu.



Slika 3.5 Uređivač automata

Na početku je platno za crtanje automata prazno, jer još nisu dodana stanja ni prijelazi između njih. Odabirom opcije na alatnoj traci predstavljene zelenim plusom na platno se pritiskom miša dodaju nova stanja automata. Pri dodavanju svakog stanja, otvara se dijalog u kojem je moguće unijeti ime stanja i izlaz digitalnog sklopa dok je u tom stanju. Nakon što su sva stanja postavljena na platno, moguće ih je razmještati odabirom opcije s odvijačem na alatnoj traci, te pritiskom na stanje i razvlačenje po platnu. Na taj je način moguće kasnije tijekom rada stvoriti jasniji i intuitivniji raspored stanja. Odabirom opcije predstavljene crvenim križićem, i pritiskom na određeno stanje automata, ono se briše. Ovo je pogodno

ako korisnik tijekom rada zaključi da mu neko stanje koje je ranije stvorio više ne treba, ili je ono suvišno za automat. Početno stanje bira se pritiskom na gumb na alatnoj traci na kojem se nalazi slika kuće.

Nakon što su sva stanja postavljena, odabrano je početno stanje i određeni su izlazi pojedinih stanja, nužno je definirati prijelaze između stanja, odnosno odrediti koje vrijednosti ulaznih signala digitalnog sklopa uzrokuju prijelaze i u koja stanja ti prijelazi vode. To se obavlja odabirom opcije predstavljene zelenim strelicama na alatnoj traci, te povlačenjem miša od jednog stanja do drugog uz pritisak. Pritom je nužno u dijalogu definirati za koju se vrijednost ulaznog signala događa prijelaz iz jednog stanja u drugo. Ako se vrijednost signala definira samo djelomično, ili uopće ne, smatra se da se prijelaz događa uz svaku vrijednost nedefiniranog dijela signala (*engl. don't care*). Sve postavke automata moguće je naknadno pregledati u dijalogu koji se otvara pritiskom na opciju za podatke o automatu koja se nalazi na alatnoj traci, pri čemu je moguće mijenjati isključivo imena signala.

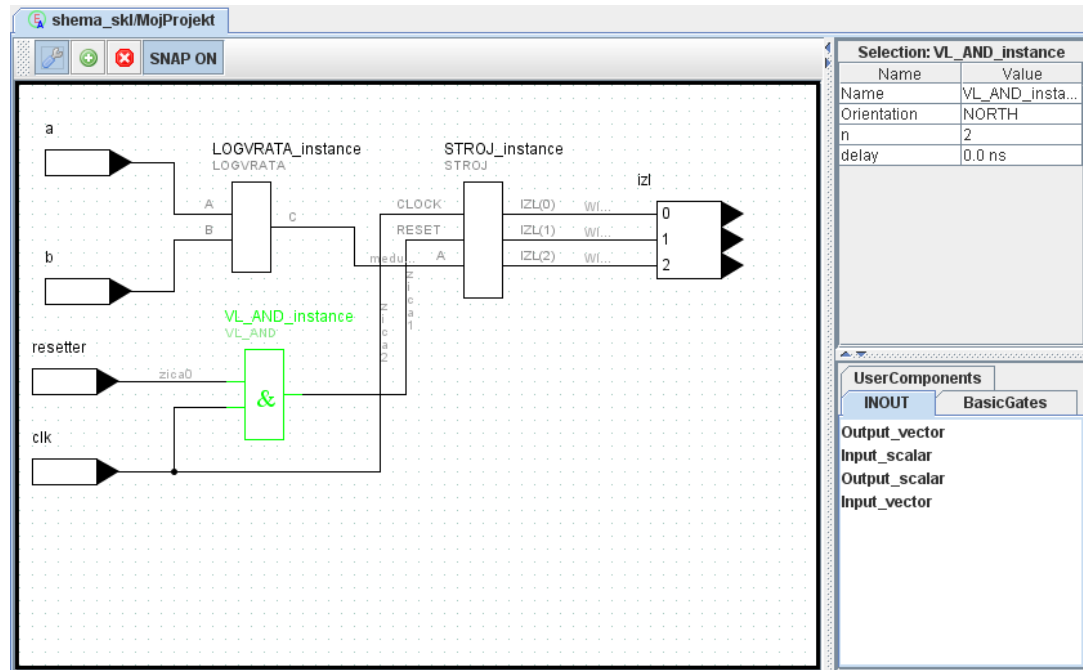
Pošto je automat izmodeliran, stanja i njihov raspored spremaju se u datoteku, te se šalju na poslužitelj, gdje se iz opisa datoteke automatom generira VHDL kod. U slučaju da automat nema nijedno stanje, početno stanje nije definirano ili model automata na bilo koji način nije dobar i VHDL kod se ne može izgenerirati, u korisničkom će se sučelju prikazati poruka o pogrešci s njenim kratkim opisom.

3.2.3. Uređivač sheme sklopa

Uređivač za crtanje sheme sklopa otvara se stvaranjem nove sheme ili odabirom postojeće datoteke sa shemom sklopa iz preglednika projekata. U slučaju stvaranja nove sheme digitalnog sklopa, otvara se dijalog unutar kojeg je moguće podesiti broj ulaznih i izlaznih signala u digitalni sklop. Nakon što se odrede ulazni i izlazni signali, otvara se uređivač shema digitalnih sklopova na kojem su na početku prikazani ulazi i izlazi u sklop. Uređivač za crtanje sheme sklopova sastoji se od više dijelova, od kojih je središnje platno na kojem se nalazi shema, dok se na vrhu nalazi alatna traka, a zdesna popis svojstava trenutno odabranog podsklopa ili žice i izbornik s popisom sklopova koje je moguće dodavati u shemu sklopa.

U shemi digitalnog sklopa, sklop se nastoji modelirati nizom podsklopova od kojih se sastoji, koji su međusobno povezani žicama, kako bi se signali prenosili s izlaza jednih na ulaze drugih. Izbornik s popisom dostupnih sklopova sadrži veći broj osnovnih digitalnih sklopova kao što su to sklop / ili bistabili, ali i ostale sklopove koje korisnik prethodno stvorio unutar istog projekta. Primjerice, ako je korisnik VHDL kodom opisao potpuno zbrajalo, tad će mu to isto potpuno zbrajalo biti dostupno u popisu sklopova, te će pomoću njega shemom digitalnog sklopa moći graditi složenije sklopove. U izborniku se također nalaze i izlazi i ulazi u digitalni sklop, pa je sučelje digitalnog sklopa moguće mijenjati i nakon što je on već

jednom stvoren – ako tijekom rada korisnik odluči da mu je potreban još jedan ulaz u sklop, tada to može učiniti jednostavnim dodavanjem jednog ulaza koji je dostupan u popisu sklopova.



Slika 3.6 Uređivač sheme digitalnog sklopa

Crtaње sheme sklopa obavlja se tako da se na platno iz izbornika s popisom sklopova pritiskom miša odaberu podsklopovi od kojih će se sastojati sklop koji se modelira, te se potom isti smjeste na shemu. Jednom smješteni digitalni sklop nekog tipa naziva se *instanca*. Na shemi odjednom može biti više instanci istog tipa sklopa. Svaki digitalni sklop ima neko ime, pa se tako u slučaju sklopa / pripadni sklop naziva *VL_AND*, a u slučaju nekog korisnikovog sklopa unutar istog projekta sklop se naziva onako kako ga je nazvao korisnik. Neovisno o tome, jednom kad se sklop nekog tipa (*VL_AND*, *VL_OR* ili *MojMultipleksor*) smjesti na shemu, tj. stvori se instanca sklopa, automatski mu se pridijeli tzv. *ime instance*. Budući da na shemi može biti više sklopova istog tipa (npr. više sklopova *IL1*), svaki od smještenih sklopova (svaka instanca sklopa) mora imati jedinstveni identifikator. Pri smještanju sklopa na shemu početno ime instance dodjeljuje sam VHDLLab – ono je ispisano iznad samog sklopa, i u popisu svojstava sklopa zdesna. Međutim, ako korisnik nije zadovoljan dodijeljenim imenom, može ga promijeniti pritiskom miša na svojstvo *Name* u popisu svojstava, upisom novog imena i pritiskom tipke *Enter*. Ako novoodabrano ime ne pripada nekom drugom sklopu, ono će uspješno biti promijenjeno. To je ujedno i ilustracija prvog svojstva, objekata u shemi, koje se ujedno i uvijek pojavljuje u popisu svojstava sklopova na shemi. Za neke je sklopove moguće mijenjati i neka druga svojstva.

Primjerice, postoje dvoulazni, troulazni i općenito N-ulazni osnovni logički sklopovi. Mijenjanjem parametra N za instance sklopova tipa *VL_AND*, *VL_OR*, *VL_XOR*, *VL_NAND*, *VL_NOR* i *VL_XNOR*, mijenja se broj ulaznih signala pripadne instance.

Jednom postavljene i imenovane sklopove na shemi mogu se odabirom odvijča na alatnoj traci pomicati po platnu onako kako to korisniku najviše odgovara. Jednom kad je postignut odgovarajući razmještaj instanci sklopova na shemi, moguće je pristupiti povezivanju sklopova žicama. Žice se stvaraju tako da se odabere zeleni plus na alatnoj traci i da se uz pritisak miša povuče linija koja povezuje dva mjesta na shemi. Na taj je način moguće povezati i izlaz jednog sklopa s ulazom u drugi sklop, ili povezati dvije žice skupa. Nakon što je žica stvorena, žicu je moguće odabrati, čime se prikazuju njena svojstva u popisu svojstava. Najbitnije je uočiti svojstvo *Name* kojim je moguće, baš kao i kod sklopova, dodijeliti žici neko drugo ime, ovisno o potrebi korisnika. Kao i ranije, dvije žice ne mogu imati isto ime. Svojstvo *Delay* određuje kašnjenje signala koji se širi žicom.

Ako korisnik tijekom rada zaključi da mu je neki sklop ili žica nepotreban, može ga obrisati odabirom crvenog križića na alatnoj traci i pritiskom na sklop ili žicu. Od opcija na alatnoj traci, treba spomenuti još i *Snap* koji određuje finoću razmještanja sklopova na shemi tijekom rada.

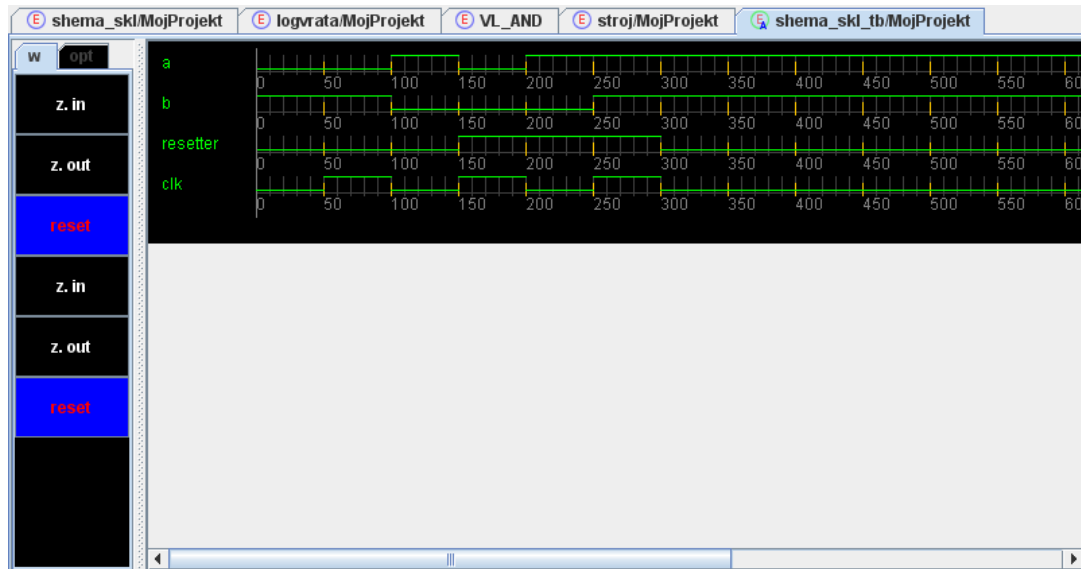
Jednom dovršena shema digitalnog sklopa pohranjuje se u datoteku i šalje na poslužitelj, gdje se na temelju nje generira VHDL opis digitalnog sklopa. U slučaju da na ulaze sklopova na shemi nisu dovedeni signali, nikakav signal nije doveden na izlaz sklopa koji se modelira u shemi ili je neka žica na neki drugi način neispravno spojena, korisniku će se prijaviti pogreška i njen kratak opis, nakon čega korisnik mora potražiti pogrešku u shemi i napraviti nužne preinake.

VHDL opise dobivene iz modela automata ili sheme digitalnog sklopa moguće je proučiti u korisničkom sučelju, i povući usporednice između VHDL koda i automata ili sheme. Tako alat za modeliranje sklopa automatom jasno pokazuje način kako iz modela apstraktnog automata izgraditi ponašajni VHDL opis digitalnog sklopa, a alat za izradu shema digitalnih sklopova jasno ukazuje na povezanost strukturnog VHDL opisa i međusobno povezanih podsklopova od kojih se sklop sastoji.

3.2.4. Uređivač ispitnih sklopova

Kad je dobiven VHDL opis sklopa, moguće je pristupiti izgradnji ispitnog sklopa. Odabirom novog ispitnog sklopa iz izbornika otvara se dijalog u kojem se određuje za koji će postojeći sklop u projektu biti stvoren ispitni sklop, te koje će biti ime ispitnog sklopa. U uređivaču ispitnih sklopova otvorit će se datoteka s novostvorenim ispitnim sklopom, u kojoj će prikazani vremenski dijagram svih ulaznih signala. Korisnik potom može mijenjati

vrijednosti pojedinih signala u različitim vremenskim trenucima, s ciljem da isproba što je moguće veći broj situacija u kojima se digitalnim sklop može naći, te na taj način utvrdi da li digitalni sklop radi ispravno. Vrijednosti pojedinih signala mijenjaju se pritiskom miša na dijagram signala, pri čemu se razina signala mijenja od tog vremenskog trenutka nadalje.



Slika 3.7 Uređivač ispitnih sklopova

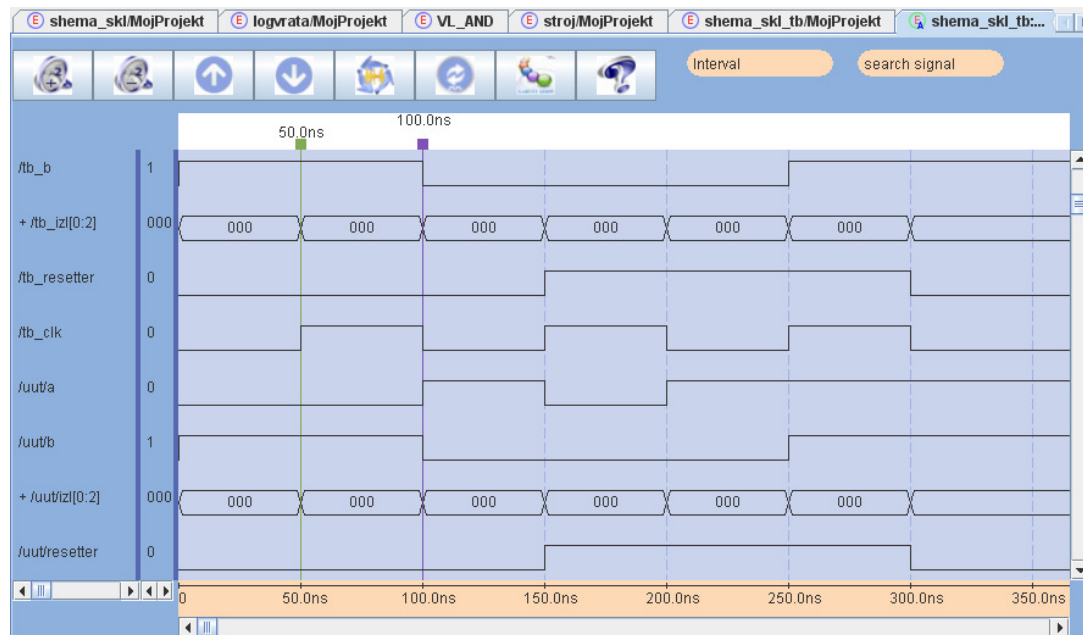
U ispitnom se sklopu vrijednosti signala mogu postavljati do bilo kojeg trenutka. Ako vremenski dijagram svojom veličinom prelazi granice ekrana moguće se duž njega pomicati trakom na dnu uređivača, ili koristiti tipke za uvećanje i smanjivanje mjerila vremenskog dijagrama koje se nalaze slijeva.

Odabirom opcije *View VHDL code*, moguće je pregledati VHDL kod koji se izgenerira na temelju ispitnog sklopa. Ovo je također korisno za obrazovne svrhe, jer korisniku pojašnjava što je to ispitni sklop u stvarnosti, koja je njegova uloga i kako se on može modelirati VHDL kodom. VHDL kod kojim se modelira ispitni sklop je tipično kombinacija strukturnog i ponašajnog modela, i sastoji se od stvaranja primjerka ispitivanog sklopa i procesa unutar kojeg se postavljaju vrijednosti ulaznih signala, pa je tako ostvaren i ovdje.

3.2.5. Preglednik simulacija

Kad je ispitni sklop napravljen, pristupa se simulaciji odabirom odgovarajuće opcije u izborniku *Tools*. Nakon što se na poslužitelju obavi simulacija, njeni se rezultati prikazuju u korisničkom sučelju. Preglednik simulacija otvara se u sredini korisničkog sučelja, a u njemu su prikazani vremenski dijagrami svih signala, ne samo ulaznih kao što je to slučaj kod uređivača ispitnog sklopa. Na alatnoj traci na vrhu preglednika simulacije nalaze se opcije za uvećanje i smanjivanje mjerila da bi se odjednom mogao vidjeti čitav raspon vrijednosti ili

samo jedan detalj na vremenskom dijagramu. Tu je i opcija za pomicanje signala gore i dolje po listi kako bi se postiglo da je skup signala koji korisnik želi promatrati na jednom mjestu.



Slika 3.8 Preglednik simulacija

Ispod alatne trake nalaze se vremenski dijagrami signala, na kojima se nalaze dva pokazivača koja služe za mjerenje intervala. Vremenski interval između dva pokazivača prikazan je na vrhu pokraj alatne trake. Korisnik može pomicati pokazivače po vremenskim dijagramima, te ih koristeći opciju *Move to next/previous right/left edge* postavljati na uzlazne i silazne bridove signala. Ovo je naročito korisno ako korisnik želi izmjeriti interval između dva brida signala ili bridova različitih signala, radi mjerenja kašnjenja signala, ili dužine visoke ili niske razine signala.

U slučaju složenih signala na vremenskom se dijagramu prikazuje binarna vrijednost signala. S lijeve strane svakog složenog signala nalazi se plus, te se pritiskom miša na njega vremenski dijagram složenog signala širi više dijagrama, prikazujući tako vrijednost svakog pojedinog bita. Ponovnim pritiskom na minus s lijeve strane imena signala vraća se originalni prikaz složenog signala.

U slučaju da se promatra simulacija nekog složenog sklopa u kojem postoji velik broj ulaznih, izlaznih i unutarnjih signala, svi signali ne stanu na ekran, pa je željeni signal potrebno tražiti povlačenjem trake desno od vremenskih dijagrama gore ili dolje. Tad je moguće signal potražiti pomoću polja za pretraživanja koje se nalazi na vrhu, desno od alatne trake. Dovoljno je upisati puno signala, i isti će biti nađen u listi signala, te će se lista pomaknuti niže ili više tako da pripadni dijagram bude prikazan.

Na vrhu preglednika rezultata simulacija postoji opcija za prikaz pomoći pri radu u pregledniku, gdje se nalazi dosta korisnih informacija. Od posebne su važnosti kratice na tipkovnici (*engl. keyboard shortcuts*), koje mogu ubrzati rad u simulatoru.

U uređivaču ispitnih sklopova i pregledniku rezultata simulacija moguće je podesiti boje signala, dijagrama i vremenskih oznaka na način na koji to korisniku odgovara. U pregledniku simulacija također postoje i unaprijed ponuđeni predlošci kombinacija boja. Radi ugodnosti rada, ovo je dosta korisna opcija.

Rezultate simulacije moguće je pohraniti, nakon čega se oni otvaraju odabirom odgovarajuće datoteke u pregledniku projekata. Ovo je posebno korisno, ako se primjerice rezultati simulacije trebaju pokazati asistentu pri predaji laboratorijske vježbe.

3.3. Usporedba s komercijalnim razvojnim okruženjima

U ovom ćemo odjeljku usporediti neke značajke postojećih komercijalnih razvojnih okruženja s onima koje nudi VHDLLab. Od često korištenih komercijalnih razvojnih okruženja za VHDL svakako treba napomenuti *Xilinx WebISE*, koji koristi nezavisno okruženje *ModelSim* koje služi za simulaciju. Ova dva okruženja se najčešće koriste u kombinaciji za razvoj digitalnih sklopova kako u velikim tvrtkama, tako i u sveučilištima u sklopu laboratorijskih vježbi za upoznavanje s digitalnom logikom.

Međutim, već je prvi nedostatak ovakvog rješenja činjenica da razvojno okruženje za VHDL i okruženje za simulaciju i testiranje digitalnih sklopova nisu integrirani u jedan cjelovit sustav, što usporava i otežava rad. VHDLLab ima simulator uključen u razvojno okruženje, te rezultate simulacije može prikazivati u okviru korisničkog sučelja, što je njegova prednost u odnosu na komercijalne proizvode, koji često simulator odvajaju od razvojnog okruženja.

Nadalje, sama činjenica da su *Xilinx WebISE* i *ModelSim* komercijalni programi podrazumijeva da za njih treba plaćati, nabavljati i obnavljati licencu. Čak i ako se, primjerice, *ModelSim* koristi isključivo u obrazovne svrhe, za njega treba nabavljati vremenski ograničenu licencu. Ovo predstavlja problem za velik broj studenata koji svake godine treba nabavljati licencu i instalirati te programe na svojim računalima.

ModelSim i *Xilinx WebISE* su osim toga i programi koji zauzimaju mnogo resursa na računalu. Pri instalaciji navedenih okruženja, na disku je potrebno odvojiti više od pola gigabajta prostora, što je razumno za računala koja pripadaju tvrtkama koje se bave razvojem digitalnih sklopova. S druge strane, studenti koriste mali dio funkcionalnosti koje nude ti dugo razvijani sustavi, s obzirom da se tek upoznaju s VHDL-om i modeliranjem digitalnih sklopova.

Još jedna prednost VHDLLab-a u odnosu na postojeće komercijalne programe je i činjenica da je on Web orijentiran, što omogućuje da se svi podaci drže na jednom mjestu na poslužitelju. Ovo je također idealno za laboratorijske vježbe s obzirom na to da studenti mogu predavati vježbe putem računala, a asistent ih može pregledavati u bilo kojem trenutku s administratorskim ovlastima. Dodatno, s obzirom na proširivu arhitekturu, VHDLLab je moguće povezati s nekim drugim automatiziranim sustavom za stvaranje laboratorijskih vježbi i njihovu verifikaciju, od kojih je jedan u razvoju na Fakultetu elektrotehnike i računarstva.

Naposljetku, treba napomenuti da je VHDLLab razvijan kao zamjena za skupa i neprikladna komercijalna rješenja za modeliranje digitalnih sklopova koja su se koristila u okviru predmeta Digitalna logika na Fakultetu elektrotehnike i računarstva. Kako se radi o predmetu u kojem se studenti upoznaju s okosnicom VHDL koda i njegovih konstrukata, neki manje korišteni, konstrukti nisu predviđeni u sklopu sustava VHDLLab, jer se uostalom ne predviđa ni njihovo proučavanje u sklopu tog kolegija. Tako su neke specifičnosti jezika VHDL izbačene s obzirom da nisu suštinski važne za njegovo shvaćanje, pa tako VHDLLab npr. ne podržava *generic* dio *entity* bloka u opisu sklopa, te druge vrste signala osim *std_logic* i *std_logic_vector* u sučelju sklopa. Bez obzira na to, svi dijelovi VHDL-a bitni za taj kolegij su podržani, a ovisno o potrebama, u budućnosti je moguće proširenje sustava VHDLLab tako da trenutno nepodržane značajke budu implementirane.

4. Primjeri korištenja sustava VHDLLab

U ovom je odjeljku prikazan primjer korištenja VHDLLab-a. Konkretno, opisana je izvedba jedne od laboratorijskih vježbi s predmeta Digitalna logika, koji se predaje kao obavezan predmet na prvoj godini Fakulteta elektrotehnike i računarstva. Primjer je pogodan, jer su njime pokazane sve mogućnosti i prednosti sustava VHDLLab.

4.1. Opis zadatka laboratorijske vježbe

Cilj je navedene laboratorijske vježbe izmodelirati digitalni sklop koji obavlja ulogu jednostavnog upravljača semaforom pješačkom prijelazu. Upravljač semaforom je pritom digitalni sklop čiji su ulazi signal takta i signal za resetiranje upravljača, a izlazi su mu signali za paljenje crvenog, žutog i zelenog svjetla semafora na cesti, te signal za upravljanje semaforom na pješačkom prijelazu. Pritom, upravljač semafora prolazi kroz više stanja koja traju različit broj perioda takta, a u svakom od njih daje različite vrijednosti izlaznih signala. Stanja su prikazana i opisana u tablici, pri čemu je trajanje pojedinog stanja izraženo u broju taktova dužine T .

Tablica 4.1 Stanja upravljača semafora

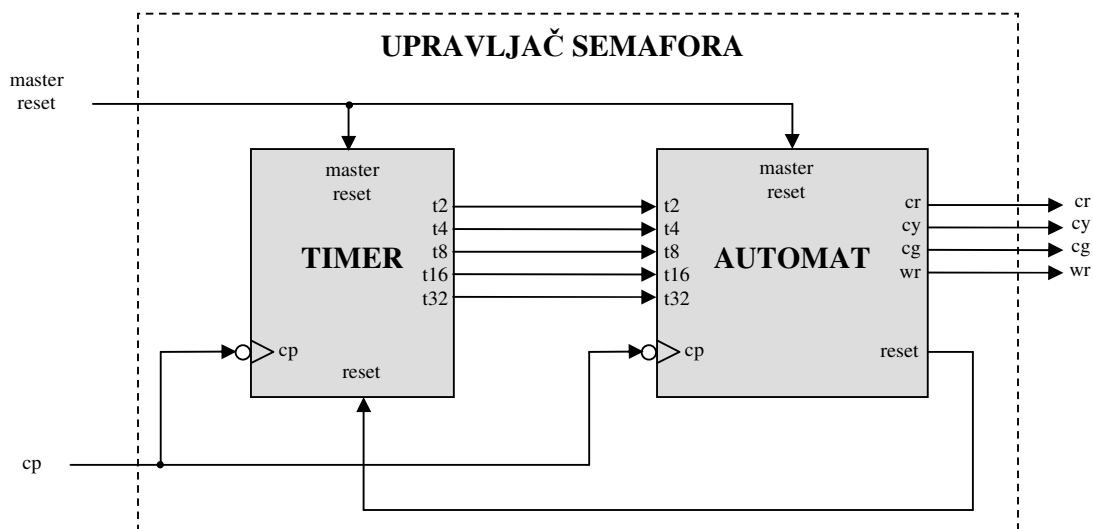
Stanja	Trajanje
Svi semafori pokazuju crveno (početno stanje)	8T
Crveno i žuto svjetlo na semaforu za automobile	2T
Zeleno svjetlo na semaforu za automobile	32T
Žuto svjetlo na semaforu za automobile	4T
Crveno svjetlo na semaforu za automobile	2T
Zeleno svjetlo na semaforu za pješake	16T
Crveno svjetlo na semaforu za pješake	4T

U prvom (početnom) stanju upravljač se nalazi samo jednom i to na početku rada, odnosno nakon resetiranja. Nakon osam perioda takta upravljač prelazi u stanje u kojem je automobilima upaljeno crveno i žuto svjetlo, te nakon toga prolazi sva stanja kružno, redom kojim su napisana u tablici.

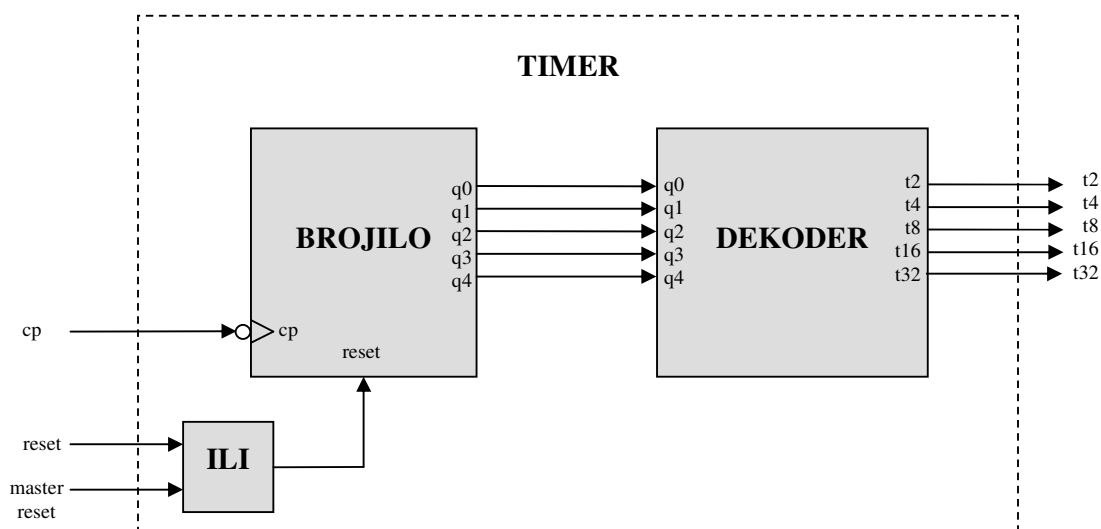
Da bi se ostvarila ova funkcionalnost, zahtijeva se da se upravljač modelira strukturno, na način da se podijeli na podsklopove *automat* i *timer*⁹. Sklop *timer* dalje se dijeli na

⁹ Prikladan prijevod mogao bi biti *štoperica*.

podsklopove brojilo i dekodeer. Ideja je da automat na ulazu prima informaciju o tome koliko je prošlo perioda vremenskog takta, te na temelju toga mijenja svoja stanja, i sukladno s time svoje izlaze. Informaciju o proteklom vremenu daje mu *timer*, koji ima izlazne signale kojima signalizira da li su prošle 2, 4, 8, 16 ili 32 periode vremenskog takta. *Timer* se sastoji od brojila koje na izlazu daje binarni kodiran broj proteklih perioda vremenskog takta i dekodeera koji na temelju tog broja postavlja izlazne signale sklopa *timer*. Navedene dekompozicije prikazane su na slikama 4.1 i 4.2.

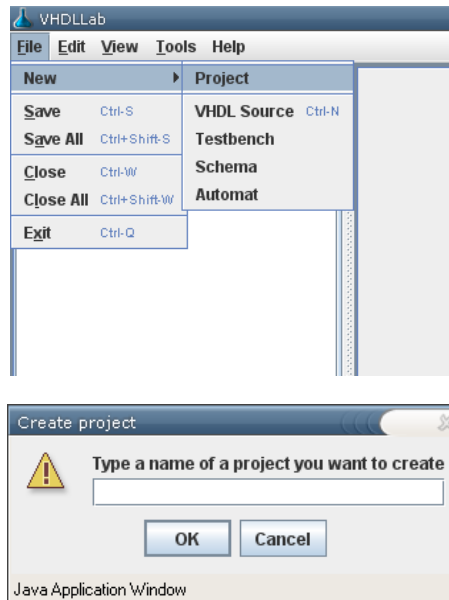


Slika 4.1 Shema upravljača semafora



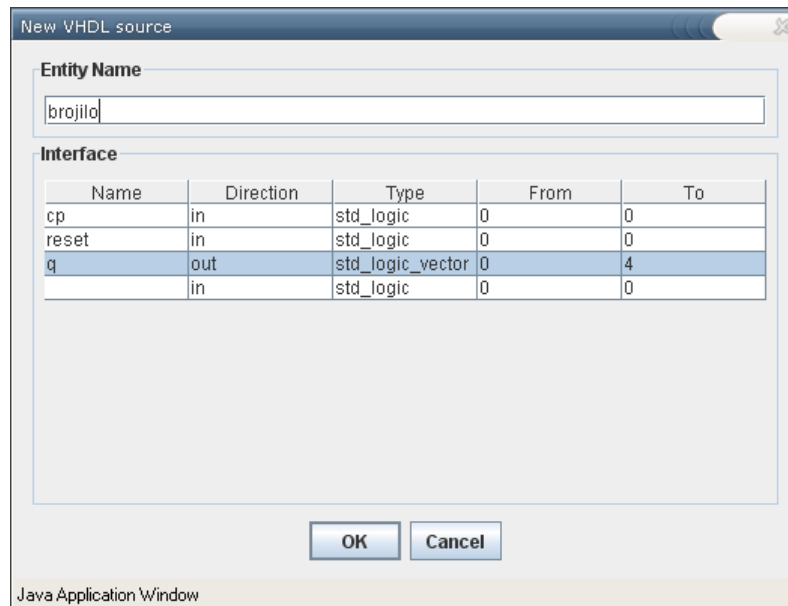
Slika 4.2 Shema sklopa *timer*

4.2. Opis izvedbe laboratorijske vježbe u VHDLLab-u



Slika 4.1 Stvaranje novog projekta

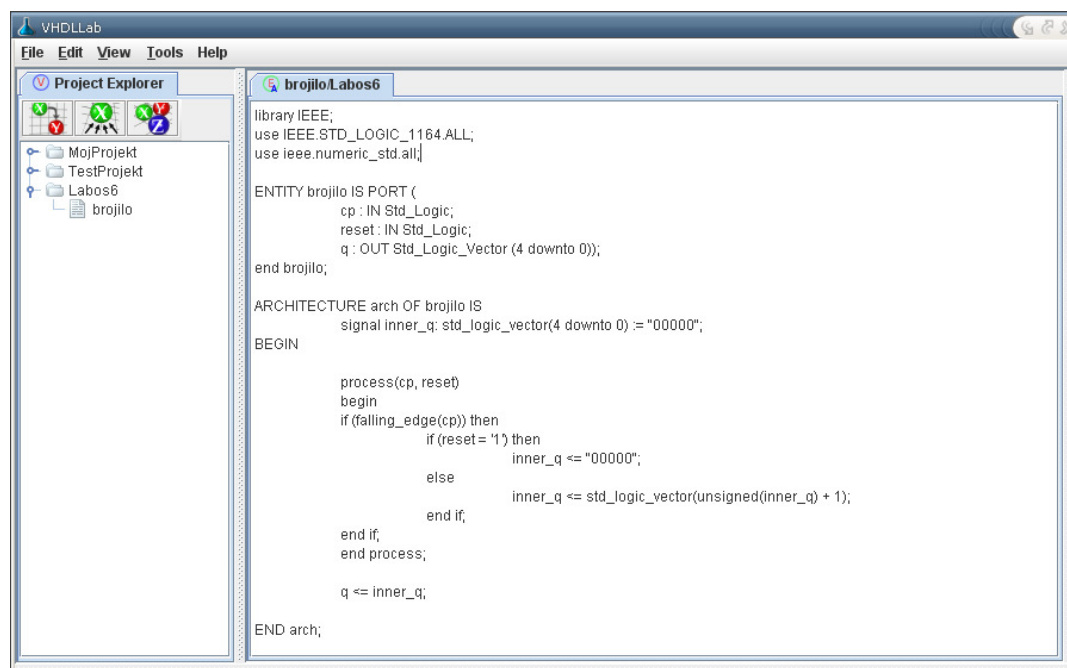
Na početku rada u VHDLLab-u nužno je stvoriti novi projekt. Ovo se obavlja odabirom stavke *File* na izborniku, i potom stavcima *New* i *Project*, čime se otvara dijalog za unos imena novog projekta. Nakon toga se stvara novi projekt, koji je prikazan s lijeve strane u pregledniku projekata.



Slika 4.2 Dijalog za opis sučelja sklopa

4.2.1. Oblikovanje brojila i dekodera

Upravljač semafora modelirat ćemo od dna prema vrhu, stoga krećemo s opisom brojila. Radi jednostavnosti, brojilo ćemo opisati ponašajnim VHDL opisom, stoga iz izbornika odabiremo stavku *File*, i potom *New* i *VHDL source*. Otvara se dijalog u koji treba unijeti ime sklopa, te opisati njegovo sučelje, tj. ulazne i izlazne signale. Za svaki je signal moguće unijeti ime, tip (*std_logic* ili *std_logic_vector*), i smjer (*in* ili *out*). Za brojilo ćemo dodati signal vremenskog vođenja *cp*, signal za resetiranje *reset* i izlazni signal *q* tipa *std_logic_vector*, preko kojeg je moguće očitati stanje brojila. Nakon potvrde, u pregledniku projekata stvara se nova datoteka tipa izvornog koda, a njen se sadržaj prikazuje zdesna. Može se primijetiti da je u datoteci odmah stvoreno sučelje sklopa (blok *entity*) i svi signali navedeni u prethodnom dijalogu. Također je stvoren arhitekturni blok unutar kojeg je nužno unijeti sam opis sklopa.

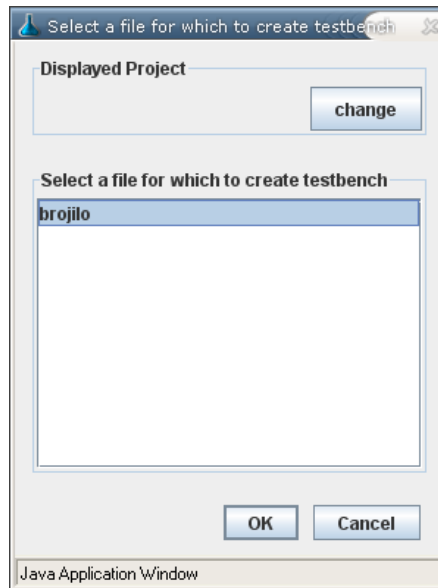


Slika 4.3 Opis brojila u uređivaču koda

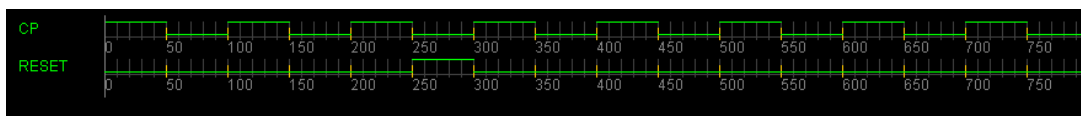
Unutar arhitekturnog bloka definiran je potom jedan *process* blok unutar kojeg je opisano ponašanje brojila – sinkroni reset okidan na visoku razinu signala *reset*, i povećanje brojila pri svakom padajućem bridu signala vremenskog vođenja.

Nakon što je brojilo opisano opisano, nužno je ispitati njegov rad izradom ispitnog sklopa. Općenito se ispitni sklop može napisati u VHDL-u, kao i svaki drugi sklop, tipično uporabom jednog *process* bloka i nekoliko instanciranih sklopova. U VHDLLab-u postoji uređivač ispitnih sklopova, u kojem je nužno definirati jedino razine ulaznih signala sklopa kroz

vrijeme, nakon čega se VHDL opis ispitnog sklopa stvara automatski. Odabirom izbornika *File*, pa *New* i stavke *Testbench* na izborniku otvara se dijalog za stvaranje novog ispitnog sklopa u kojem treba odabrati sklop za koji će biti izgeneriran ispitni sklop.

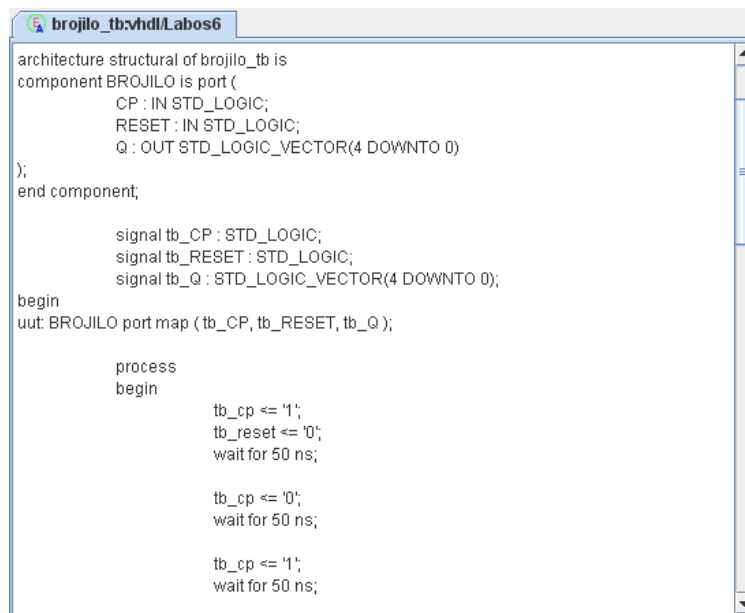


Slika 4.4 Odabir sklopa za izradu ispitnog sklopa



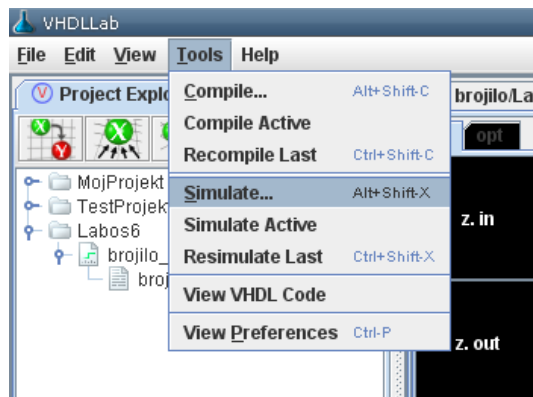
Slika 4.5 Ispitni signali za brojilo

Nakon što je stvoren ispitni sklop, moguće je podesiti razine ulaznih signala. U ovom slučaju je dovoljno podesiti razine signala vremenskog vođenja, te na jednom mjestu definirati visoku razinu signala *reset*. Očekujemo da će brojilo na padajuće bridove signala vremenskog vođenja povećavati vrijednost na svom izlazu, a da će kod onog padajućeg brida kod kojeg je vrijednost signala *reset* na visokoj razini postaviti vrijednost na izlazu na nulu.



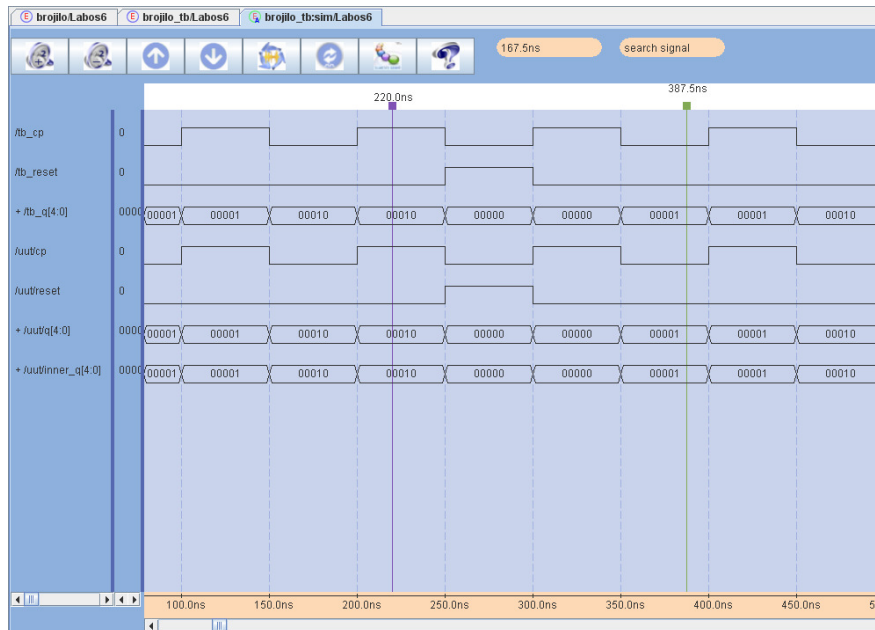
Slika 4.6 VHDL kod ispitnog sklopa

Odabirom stavke *View VHDL Code* iz izbornika *Tools* moguće je pregledati VHDL kod koji se stvara na temelju željenih valnih oblika na ulazima. Ovo naročitu obrazovnu vrijednost za studenta, s obzirom da jasno ilustrira na koji se način u VHDL-u pišu ispitni sklopovi – za studenta koji se prvi put susreće s digitalnom logikom i VHDL-om, pojam ispitnog sklopa može biti prilično nejasan, pa mu je ova funkcionalnost značajna pomoć u savladavanju gradiva.



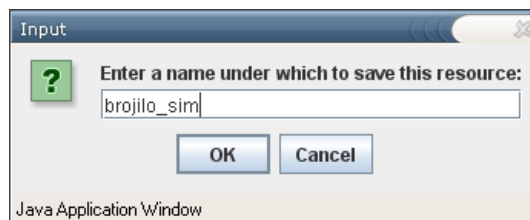
Slika 4.7 Pokretanje simulacija sklopa

Nakon što su postavljeni valni oblici na ulazima, može se obaviti simulacija sklopa odabirom izbornika *Tools* i potom stavke *Simulate...* u izborniku, kao što je prikazano na slici. Pritom će se otvoriti dijalog za odabir ispitnog sklopa kojeg se želi simulirati.



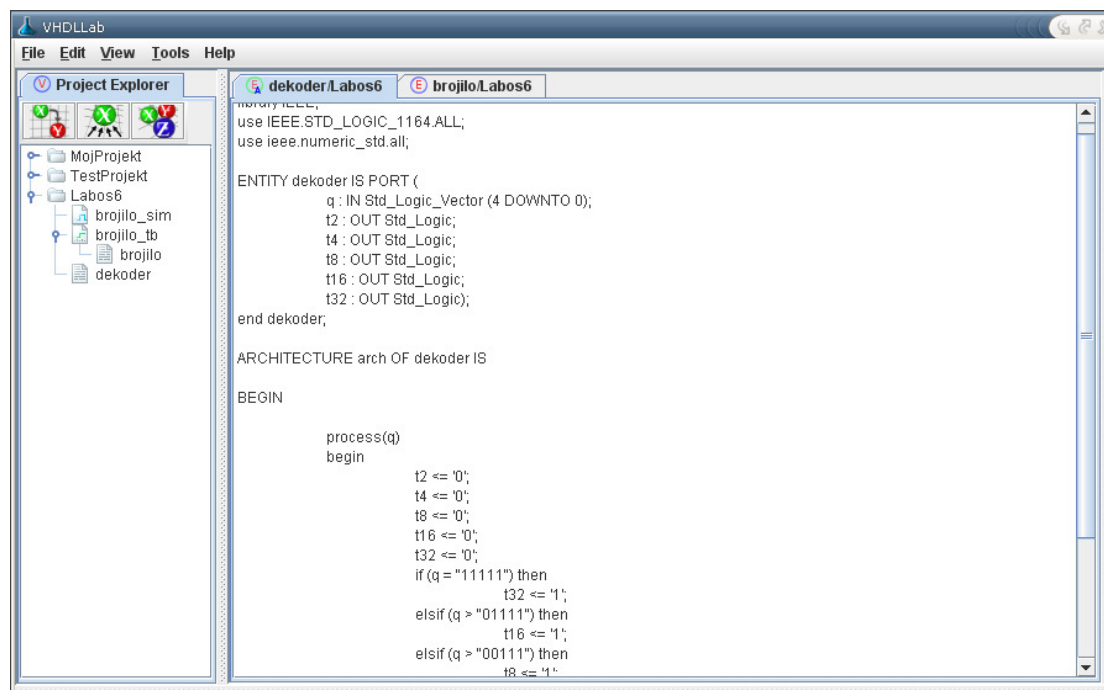
Slika 4.8 Rezultati simulacije brojala

Nakon potvrde, prikazuje se prozor s rezultatima simulacije na kojem je moguće pratiti vrijednosti pojedinih signala kroz vrijeme. Primijetimo na vrhu slike signal vremenskog vođenja *cp*. Sljedeći je signal *reset*, a treći po redu je izlazni signal *q*, koji je tipa *std_logic_vector*, pa je prikazana njegova binarna vrijednost. Brojilo se ponaša kao što je i zamišljeno – povećava vrijednost na izlazu sve do binarne vrijednosti 10, kad se signal *reset* nađe u visokom stanju, te nakon toga nastavlja brojanje.



Slika 4.9 Pohrana rezultata simulacije

Rezultate simulacije je moguće pohraniti u datoteku kako bi se kasnije mogle pogledati. Ovo je korisno sa stajališta laboratorijskih vježbi, jer omogućuje studentu obavljanje laboratorijske vježbe kod kuće i prikaz rezultata simulacije asistentu na predaji vježbe.

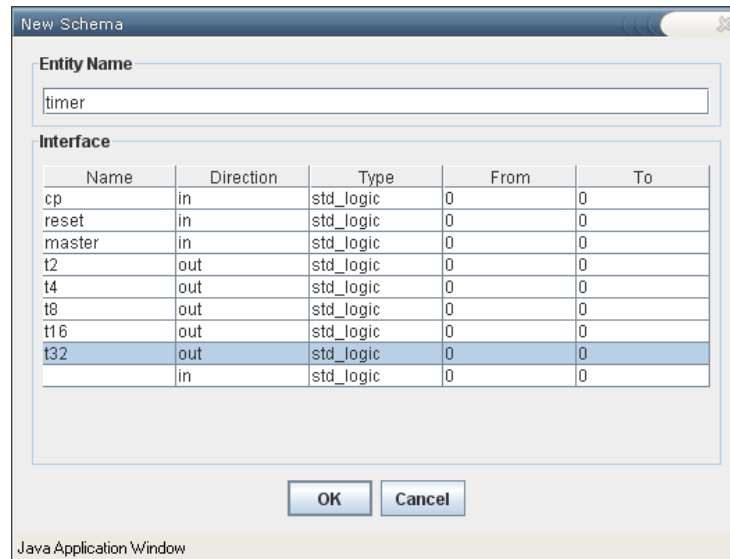
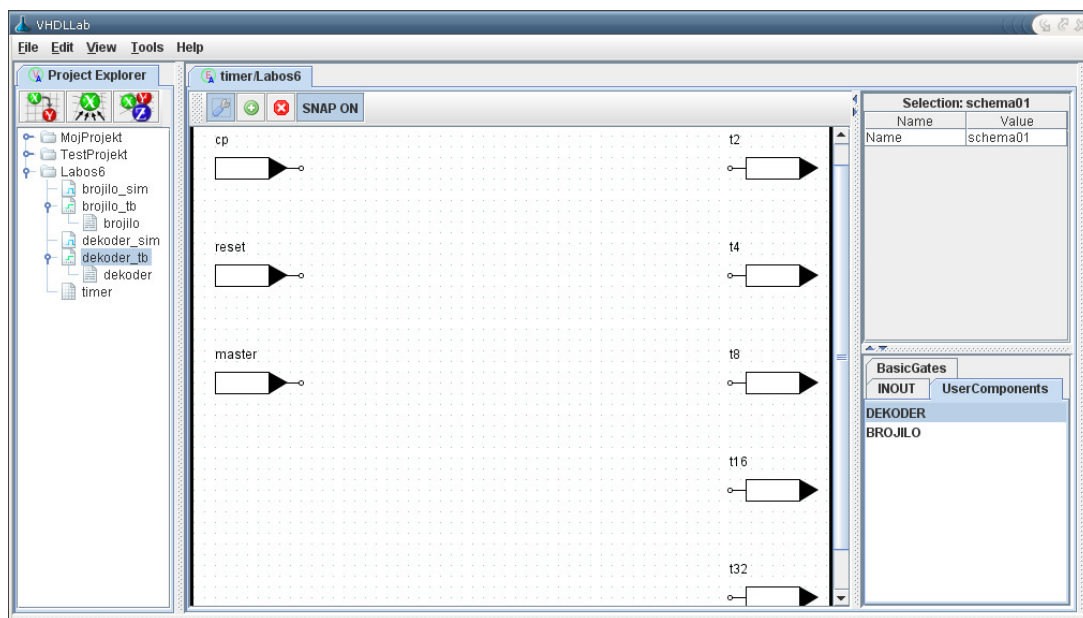


Slika 4.10 Ponašajni opis dekode

Nakon što je izmodelirano brojilo, moguće je pristupiti modeliranju dekode. Dekoder ćemo također modelirati ponašajno, pa se čitav postupak malo razlikuje od postupka primijenjenog na modeliranje brojila. Dovoljno je napomenuti da je dekode moguće opisati jednim *process* blokom unutar kojeg se provjerava je li binarni broj na izlazu iz brojila veći od 2, 4, 8, 16 ili 32, te se na temelju toga odgovarajući izlazni signal postavlja na visoku razinu. Tome slijedi stvaranje ispitnog sklopa i provjera ispravnosti rada simulacijom. Navedeni koraci vrlo su slični onima za modeliranje brojila, pa ih nećemo detaljno razmatrati.

4.2.2. Oblikovanje sklopa *timer*

Budući da je sad izmodelirano i brojilo i dekode, moguće je stvoriti sklop *timer*, čije su brojilo i dekode sastavnice. Ovo je moguće izvesti uporabom strukturnog VHDL opisa, dakle pisanje VHDL izvornog koda, no mi ćemo se poslužiti uređivačem shema digitalnih sklopova čiji je ionako smisao strukturno modeliranje digitalnih sklopova, te ćemo time ubrzati čitav postupak. Odabirom stavki *File*, *New* i *Schema*, otvara se dijalog za stvaranje nove sheme. Pritom je nužno navesti ime digitalnog sklopa, a moguće je navesti i ulazne i izlazne signale kao što je prikazano na slici. Ulazne i izlazne signale moguće je dodavati i brisati i kasnije tijekom rada u uređivaču shema, te na taj način mijenjati sučelje sklopa naknadno.

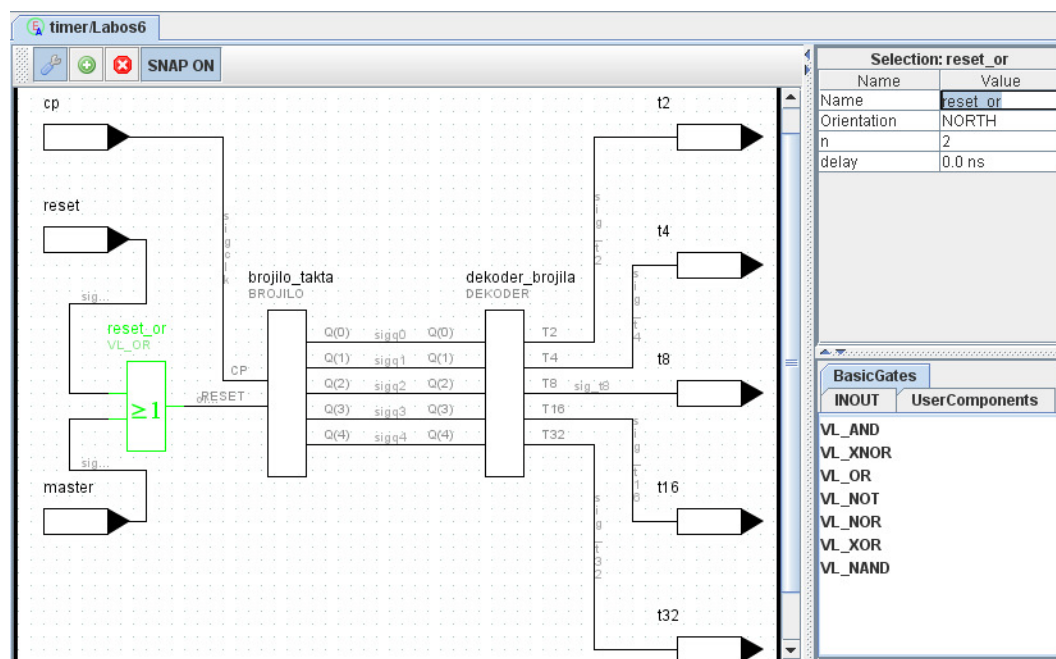
Slika 4.11 Unos opisa sučelja *timer-a*Slika 4.12 Inicijalno stvorena shema *timer-a*

Potvrđivanjem dijaloga otvara se prozor u kojem je prikazana shema. Na početku se na shemi nalaze isključivo ulazi i izlazi iz sklopa koji su odabrani u dijalogu za stvaranje sheme. Kako bismo dodali nove sklopove, nužno je odabrati neku od kartica s desne strane ekrana, npr. *UserComponents* u kojem su izlistani ostali sklopovi iz projekta, te potom odabrati neki od sklopova. Kao što se vidi na slici, trenutno su na raspolaganju dekodier i brojilo koje smo modelirali ranije. Nakon odabira, klikom na platno s lijeve strane moguće je primjerak sklopa

smjestiti na shemu. Položaj jednom smještenog sklopa moguće je i promijeniti odabirom prvog gumba na alatnoj traci, te pritiskom i povlačenjem sklopa po platnu.

Uređivač shema također nudi niz unaprijed definiranih logičkih sklopova kao što su to sklop *I*, sklop *ILI*, sklop isključivo *ILI*, sklop *NI*, sklop *NILI* i drugi. Odabirom kartice *BasicGates* moguće je na shemu smještati te sklopove. U našem primjeru trebat će nam isključivo sklop *ILI* na čiji ćemo ulaz spojiti signale *reset* i *master reset*, a čiji ćemo izlaz spojiti na ulaz brojila – sa stajališta sklopa *timer* je svejedno da li se brojanje ponavlja uslijed resetiranja čitavog sustava, ili zbog prijelaza automata u sljedeće stanje.

Jednom kad su svi sklopovi smješteni na platno, nužno ih je povezati žicama. Odabirom gumba sa slikom plusa na alatnoj traci i pritiskom na platno započinje crtanje žice. Jednom kad se otpusti pritisak miša, crtanje žice završava.

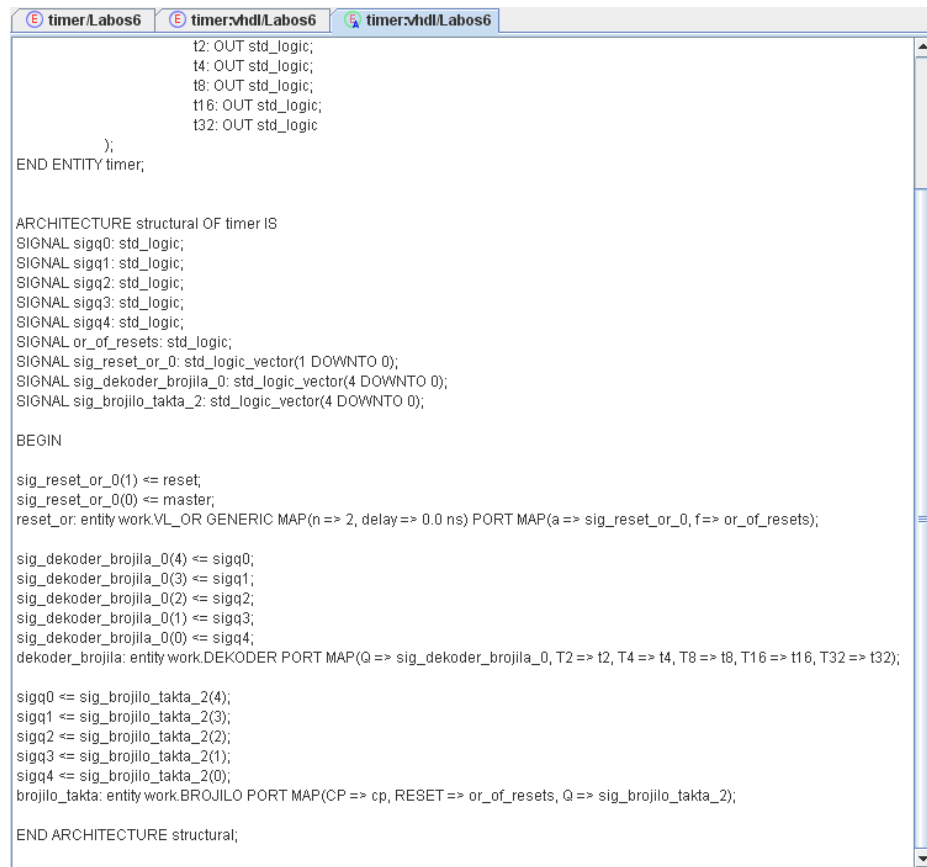


Slika 4.13 Shema *timer-a*

Primjercima smještenih sklopova, kao i žicama koje ih povezuju, moguće je davati imena. Smještanjem sklopa ili žice na platno dodijeljuju im se podrazumijevana imena, no moguće ih je i promijeniti odabirom sklopa ili žice i odabirom svojstva *Name* u popisu svojstava sklopa, te promjenom te vrijednosti. Neki sklopovi imaju više različitih svojstava, pa je tako logičkim vratima moguće promijeniti i kašnjenje, ili broj ulaza.

Slika 4.13 prikazuje shemu kakvu bismo trebali dobiti na kraju modeliranja. Sada je moguće pristupiti izradi ispitnog sklopa i simulaciji, no prije toga, sjetimo se da se naš

shematski prikaz mora pretvoriti u VHDL opis, što VHDLLab obavlja automatski. Odabirom stavke *View VHDL Source...* možemo vidjeti VHDL kod stvoren na temelju sheme sklopa.



```

t2: OUT std_logic;
t4: OUT std_logic;
t8: OUT std_logic;
t16: OUT std_logic;
t32: OUT std_logic

);
END ENTITY timer;

ARCHITECTURE structural OF timer IS
SIGNAL sigq0: std_logic;
SIGNAL sigq1: std_logic;
SIGNAL sigq2: std_logic;
SIGNAL sigq3: std_logic;
SIGNAL sigq4: std_logic;
SIGNAL or_of_resets: std_logic;
SIGNAL sig_reset_or_0: std_logic_vector(1 DOWNTO 0);
SIGNAL sig_dekoder_brojila_0: std_logic_vector(4 DOWNTO 0);
SIGNAL sig_brojilo_takta_2: std_logic_vector(4 DOWNTO 0);

BEGIN

sig_reset_or_0(1) <= reset;
sig_reset_or_0(0) <= master;
reset_or: entity work.VL_OR GENERIC MAP(n => 2, delay => 0.0 ns) PORT MAP(a => sig_reset_or_0, f => or_of_resets);

sig_dekoder_brojila_0(4) <= sigq0;
sig_dekoder_brojila_0(3) <= sigq1;
sig_dekoder_brojila_0(2) <= sigq2;
sig_dekoder_brojila_0(1) <= sigq3;
sig_dekoder_brojila_0(0) <= sigq4;
dekoder_brojila: entity work.DEKODER PORT MAP(Q => sig_dekoder_brojila_0, T2 => t2, T4 => t4, T8 => t8, T16 => t16, T32 => t32);

sigq0 <= sig_brojilo_takta_2(4);
sigq1 <= sig_brojilo_takta_2(3);
sigq2 <= sig_brojilo_takta_2(2);
sigq3 <= sig_brojilo_takta_2(1);
sigq4 <= sig_brojilo_takta_2(0);
brojilo_takta: entity work.BROJILO PORT MAP(CP => cp, RESET => or_of_resets, Q => sig_brojilo_takta_2);

END ARCHITECTURE structural;

```

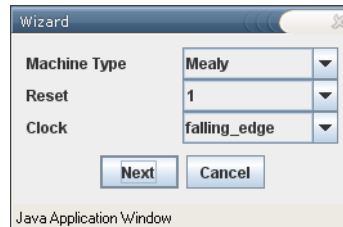
Slika 4.14 VHDL kod stvoren na temelju sheme sklopa

Ovo ponovno ima iznimnu obrazovnu vrijednost za studenta, jer mu omogućuje da za bilo koju shemu digitalnog sklopa, dobije odgovarajući VHDL strukturni opis, te na taj način shvati smisao i značenje strukturnog opisa. Student na ovaj način shvaća što su to primjerci sklopova i kako stvoriti primjerak nekog sklopa unutar drugoga, kakva je veza između signala i žica, te kako se koriste VHDL konstrukti poput *port map* i *generic map*.

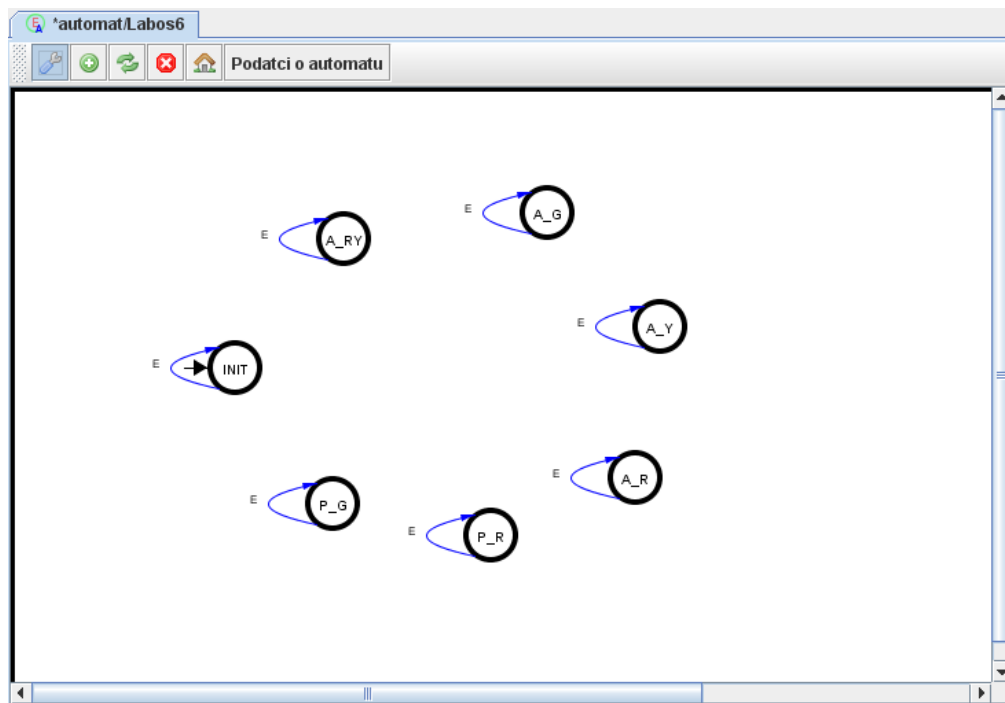
4.2.3. Oblikovanje automata i upravljača semafora

Nakon što smo izmodelirali brojilo, slijedi modeliranje automata. Automati se najčešće u VHDL-u modeliraju ponašajnim opisom, no VHDLLab skraćuje vrijeme izrade automata pomoću ugrađenog uređivača automata, u kojem je moguće naprosto odrediti stanja i prijelaze između njih, te izlaze iz automata, na temelju čega se stvara odgovarajući VHDL kod. Odabirom stavki *File*, *New* i *Automat* otvara se dijalog unutar kojeg je moguće odrediti da li je automat Mealyev ili Mooreov, da li ga u početno stanje vraća visoka ili niska razina

signala *reset*, te da li se stanja mijenjaju na padajući ili rastući brid signala vremenskog vođenja. Odabiremo Mealyev automat, visoku razinu signala *reset* i padajući brid, pa se povrdom otvara novi, otprije poznati, dijalog u kojem je moguće odrediti ulaze i izlaze u automat. Nakon postavljanja ulaza i izlaza, prikazuje se prozor unutar kojeg je moguće uređivati automat.



Slika 4.15 Dijalog za odabir vrste automata



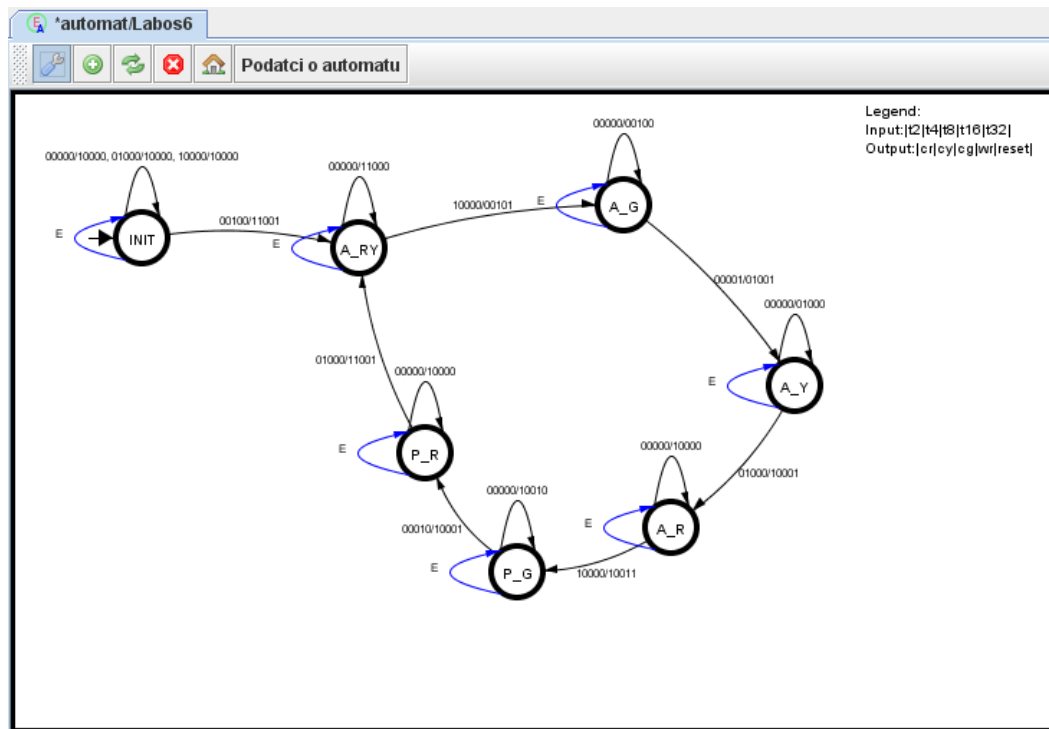
Slika 4.16 Stanja automata

Pritiskom na gumb sa slikom plusa na alatnoj traci i potom pritiskom na platno dodaje se novo stanje. Svakom od stanja moguće je odrediti neko ime prilikom dodavanja na platno. Nakon što su stanja razmještena, početno je stanje moguće odrediti pritiskom na gumb alatne trake sa slikom kućice, te potom pritiskom na željeno stanje. Prijelazi između stanja dodaju se pritiskom na gumb alatne trake sa slikom strelica, te pritiskom na izvorišno stanje i potom na odredišno stanje. Kod dodavanja prijelaza otvara se dijalog u kojem je nužno

upisati uz koju se pobudu prijelaz dešava, a u slučaju Mealyevog automata, i koji je odziv na izlazi.

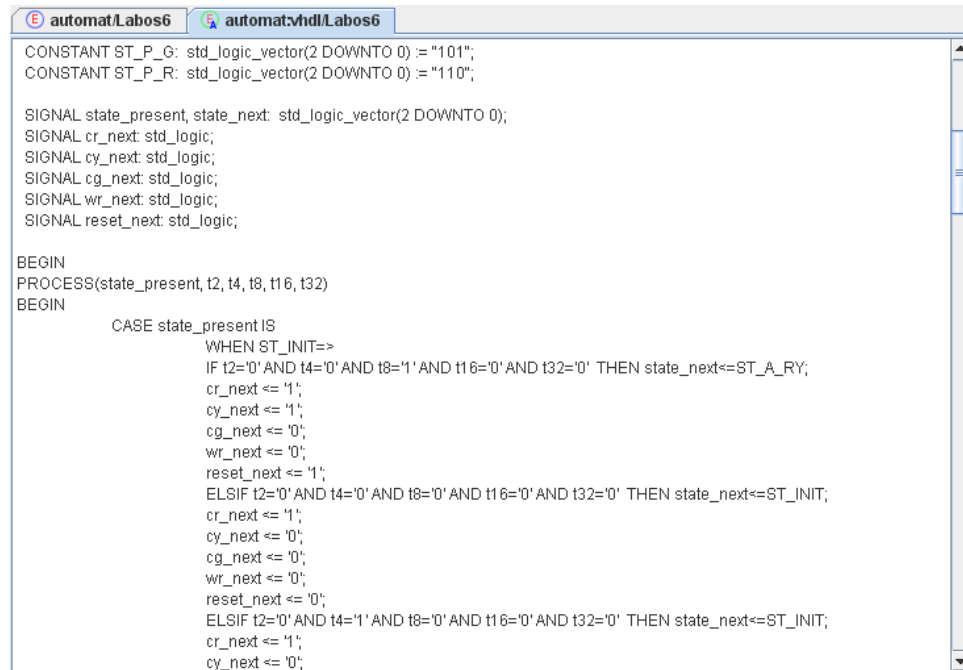
U našem slučaju, moramo modelirati automat prema tablici na početku odjeljka. Pritom se prijelazi događaju za pobudu određenu brojem taktova koje automat mora provesti u danom stanju. Primjerice, iz početnog se stanja prelazi u stanje u kojem je autima upaljeno žuto i crveno svjetlo nakon 8 taktova, što znači da se prijelaz mora dogoditi kad se ulazni signal $t8$ nađe na visokoj razini. Poredak signala na ulazu i izlazu prikazan je zdesna-gore.

Naposlijetku dobivamo automat prikazan na slici.



Slika 4.17 Automat sa stanjima i prijelazima

Sljedeći korak je izrada ispitnog sklopa i simulacija, no VHDLLab nudi prikaz VHDL koda izgeneriranog na temelju sheme automata. Za studenta koji prvi put u VHDL-u modelira automat je ovo izuzetno korisno, s obzirom da mu za bilo koji automat pokazuje ekvivalentan opis u VHDL kodu pomoću *case* konstrukata.



```

E automat.Labos6  automat.vhdl.Labos6
CONSTANT ST_P_G: std_logic_vector(2 DOWNTO 0) := "101";
CONSTANT ST_P_R: std_logic_vector(2 DOWNTO 0) := "110";

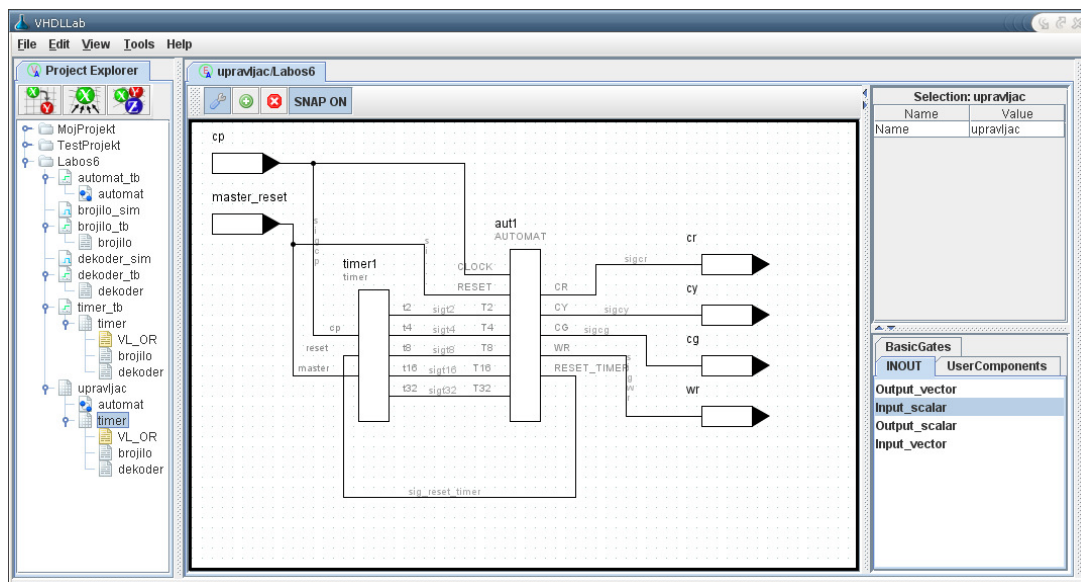
SIGNAL state_present, state_next: std_logic_vector(2 DOWNTO 0);
SIGNAL cr_next: std_logic;
SIGNAL cy_next: std_logic;
SIGNAL cg_next: std_logic;
SIGNAL wr_next: std_logic;
SIGNAL reset_next: std_logic;

BEGIN
PROCESS(state_present, t2, t4, t8, t16, t32)
BEGIN
    CASE state_present IS
        WHEN ST_INIT =>
            IF t2='0' AND t4='0' AND t8='1' AND t16='0' AND t32='0' THEN state_next<=ST_A_RY;
            cr_next <= '1';
            cy_next <= '1';
            cg_next <= '0';
            wr_next <= '0';
            reset_next <= '1';
            ELSIF t2='0' AND t4='0' AND t8='0' AND t16='0' AND t32='0' THEN state_next<=ST_INIT;
            cr_next <= '1';
            cy_next <= '0';
            cg_next <= '0';
            wr_next <= '0';
            reset_next <= '0';
            ELSIF t2='0' AND t4='1' AND t8='0' AND t16='0' AND t32='0' THEN state_next<=ST_INIT;
            cr_next <= '1';
            cy_next <= '0';
    
```

Slika 4.18 VHDL opis automata

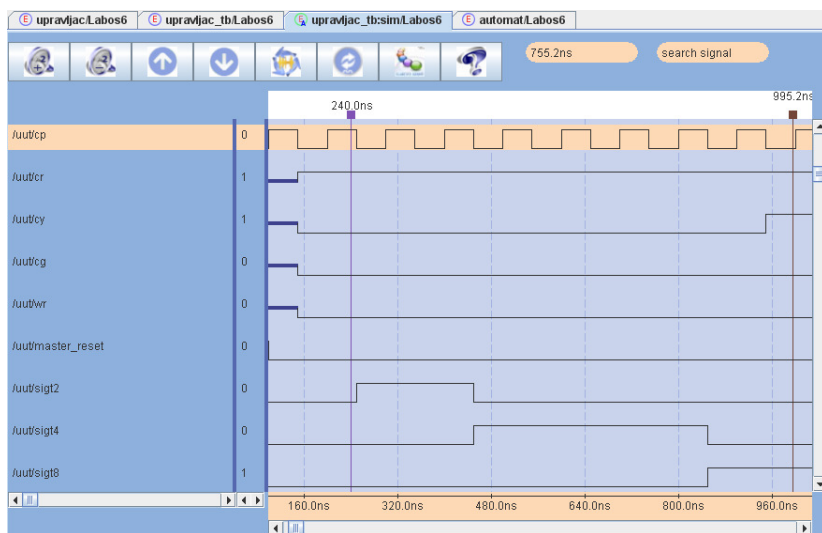
U ovom su trenutku izmodelirani svi sklopovi nužni za izradu upravljača semafora. Nužno je još strukturnim opisom izmodelirati sam upravljač, tako da su dosada modelirani sklopovi njegovi podsklopovi. Ponovno ćemo koristiti uređivač shema kako bismo ovo postigli. Postupak je sličan onome kod modeliranja *timer*-a, pa ćemo prikazati samo krajnji rezultat. Izrada ispitnog sklopa i simulacija je slična onima opisanim u prijašnjim koracima.

Treba primijetiti da se u projektu sad nalazi veći broj hijerarhijski organiziranih datoteka, kao što je vidljivo slijeva u pregledniku projekata. Hijerarhijski prikaz je također koristan studentima kako bi bolje shvatili smisao hijerarhijskom oblikovanja digitalnog sklopa koji smo koristili u izradi ove laboratorijske vježbe.



Slika 4.19 Shema upravljača semafora

Postigli smo naš cilj – modelirali smo upravljač semafora koji se ponaša kako je zadano u laboratorijskoj vježbi, koristeći se pritom nizom pomoćnih alata poput uređivača sheme i uređivača automata, čija je prednost kako i ubrzanje posla, tako i obrazovni karakter. Prošli smo kroz većinu funkcionalnosti VHDLLab-a pokazavši kako se koristi uređivač koda, uređivač ispitnih sklopova, preglednik simulacija, te uređivači sheme i automata. Jedan pogled na VHDL kod koji stvara uređivač automata ili shema je dovoljan da se uvidi koliko taj alat skraćuje vrijeme oblikovanja sklopa. S druge strane, studentu koji tek uči pojmove poput strukturnog opisa, automata ili ispitnog sklopa, ovi su alati izuzetno korisni.



Slika 4.20 Simulacija upravljača semafora

5. Tehnički opis sustava VHDLLab

VHDLLab je pisan u programskom jeziku Java¹⁰ što omogućuje dostupnost na velikom broju operacijskih sustava (*Microsoft Windows, Linux, Solaris* itd.) kao i jednostavniju izradu zbog bogate standardne biblioteke (*J2SE, Java 2 Platform Standard Edition*). Java je objektno orijentirani programski jezik čiji se kôd izvršava, za razliku od većine programskih jezika, u virtualnom stroju (*JVM, Java virtual machine*), što u konačnici i omogućava neovisnost o operacijskom sustavu. Sigurno jedno od najboljih značajki Jave je i automatsko vođenje računa o memorijskom modelu (*engl. garbage collection*) tako da je programeru uvelike olakšano programiranje aplikacije. Sam jezik posuđuje veliki dio svoje sintakse od programskih jezika C i C++, ali za razliku od njih ima jednostavniji objektni model i manje sredstava na nižim razinama.

Da bi se automatizirale zajedničke akcije (npr. prevođenje i pakiranje kôda, pokretanje testova i generiranje rezultata, generiranja dokumentacije kôda, itd.), koristi se program kojemu je upravo to i namjena: *Apache Ant*¹¹. Također, tijekom izrade projekta, korištene su mnogobrojne biblioteke trećih strana koje su ubrzale izradu. Većina njih su proizvodi neprofitne korporacije *Apache Software Foundation*¹² koja je do sada razvila veliki broj biblioteka za Java programski jezik. Sve biblioteke koje VHDLLab koristi su slobodnog i otvorenog kôda, tako da je VHDLLab u konačnici besplatan i prikladan fakultetima za korištenje (što je i jedan od ciljeva!).

Pošto je projekt razvijalo više ljudi bila je potrebno nekakva mogućnost integracije različitih kôdova u jedan projekt. Za to je korišten popularni sustav za verzioniranje izvornog kôda, ali i drugih datoteka: *Subversion*¹³. On koristi središnji repozitorij u kojeg se sprema izvorni kôd ili binarne datoteke projekta. Na taj se način rješava problem više inačica koda te jedinstvene lokacije projekta.

VHDLLab je trenutno projekt na engleskom jeziku, ali je napisan tako da je jednostavno dodati potporu za hrvatski ili bilo koji drugi jezik. Izvorni kôd se pritom ne mijenja, već samo dodaje datoteka s prijevodom, podrazumijevanog, engleskog jezika.

5.1. Izgradnja sustava VHDLLab

Kako su neki od ciljeva, pri izgradnji VHDLLab sustava, bili jednostavna instalacija te još jednostavnije korištenje od strane korisnika, VHDLLab projekt ne obuhvaća jednu aplikaciju već koristi klijent-poslužitelj (*engl. client-server*) arhitekturu te definira dvije aplikacije:

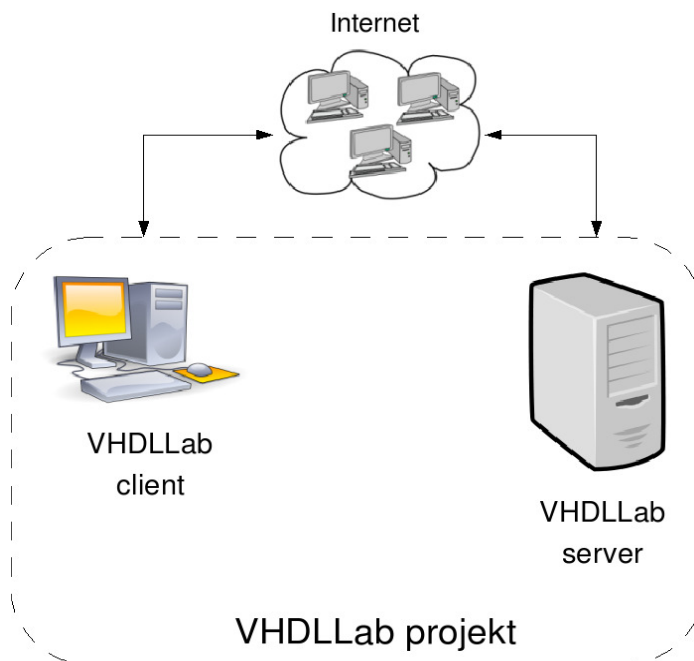
¹⁰ <http://java.sun.com/>

¹¹ <http://ant.apache.org/>

¹² <http://www.apache.org/>

¹³ <http://subversion.tigris.org/>

klijentska (*engl. VHDLLab client*) i poslužiteljska (*engl. VHDLLab server*). Pritom se kao medij za komunikaciju između tih dviju aplikacija koristi Internet.

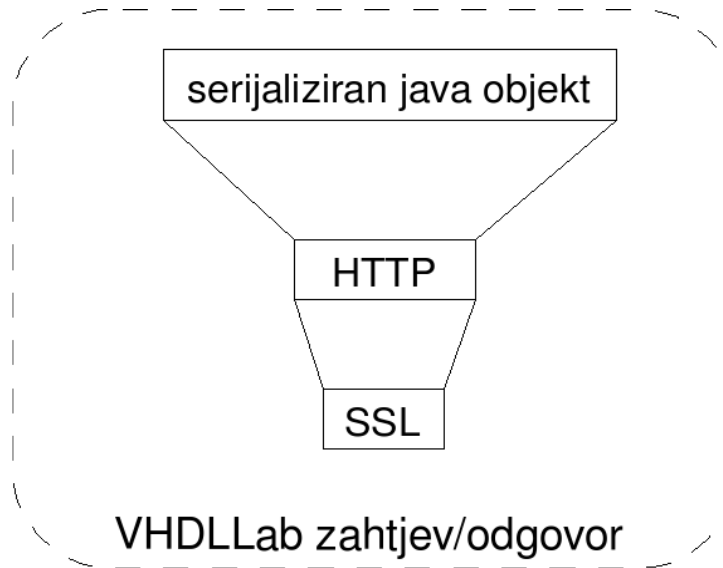


Slika 5.1 Prikaz komunikacije klijent-poslužitelj arhitekture

Klijentska je aplikacija vidljiva krajnjem korisniku i vrlo ju je jednostavno instalirati (najčešće je potrebno u web-preglednik samo unijeti odgovarajuću adresu). Ona mu omogućuje da piše VHDL kôd, modelira sklopove pomoću sheme ili automata, definira ispitni sklop te u konačnici i vidi rezultat simulacije. Sama klijentska aplikacija se više-manje bavi samo grafičkim sučeljem (*GUI, engl. graphical user interface*), dok je sva ostala funkcionalnost implementirana na VHDLLab poslužitelju (npr. generiranje VHDL kôda, prevođenje i simulacija kôda, spremište podataka, rješavanje međuzavisnosti, izvlačenje opisnika digitalne komponente, itd.).

Kao komunikacijski protokol, koji se koristi u komunikaciji klijenta i poslužitelja, koristi se sveprisutni HTTP (*engl. Hypertext Transfer Protocol*) unutar kojeg se zapisuju zahtjevi i odgovori koje poslužitelj treba obraditi. Iako mu to nije namjena, HTTP se sve češće koristi za bilo kakvu razmjenu informacija na Internetu pa je s toga i odabran. Isto tako koristeći HTTP izbjegnuti su neki problemi koji bi se inače javljali te bi ih korisnik trebao riješiti prije pokretanja klijentske aplikacije. Jedan od njih je vezan uz sigurnosne stijene (*engl. firewall*). Naime, HTTP protokol je najčešće dopušten protokol u sigurnosnim stijenama, što ne bi bio slučaj da se koristi neki drugi protokol. Drugi riješeni problemi su već postojeće rješenje za autentifikaciju i upravljanjem sjednicama, jednostavna implementacija nad već postojećim poslužiteljem za Javu, itd. Za sigurnu komunikaciju se dodatno koristi SSL (*engl. Secure*

Sockets Layer) kriptografski protokol, a sami zahtjevi i odgovori s VHDLLab poslužitelja se zapisuju unutar HTTP-a koristeći serijalizaciju Java objekta za koju Java ima podršku.



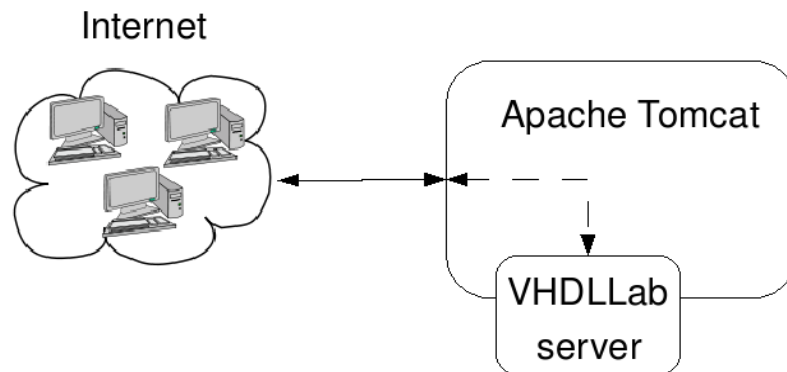
Slika 5.2 Prikaz korištenih protokola u komunikaciji

5.2. VHDLLab poslužitelj

VHDLLab poslužitelj je napravljen korištenjem *Servlet*¹⁴ tehnologije koja omogućuje jednostavan i konzistentan mehanizam proširenja mrežnog poslužitelja (*engl. web server*). Mrežni poslužitelj koji se pritom koristi je Apache Tomcat¹⁵. Servlet, u većini slučaja, predstavlja jednu mrežnu stranicu (*engl. web page*), no u slučaju VHDLLab poslužitelja on definira standardno programsko sučelje (*API, engl. application programming interface*) VHDLLab poslužitelja, tj. definira metode koje je on u mogućnosti napraviti (npr. prevođenje i simuliranje VHDL kôda, spremanje podataka, itd.), a koje onda klijentska aplikacija može koristiti. Takav poslužitelj, kao mrežna aplikacija, se onda pakira u WAR (*engl. web archive*) te se takav postavlja u Tomcat.

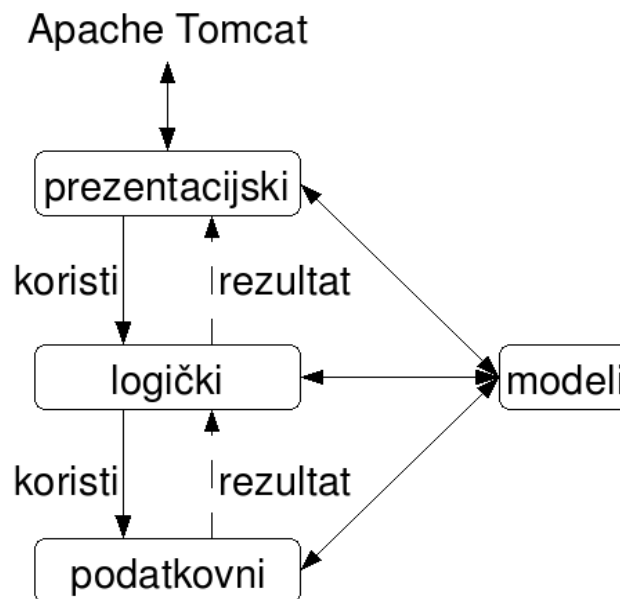
¹⁴ <http://java.sun.com/products/servlet/>

¹⁵ <http://tomcat.apache.org/>



Slika 5.3 VHDLLab poslužitelj je proširenje mrežnog poslužitelja

Modularan kôd je puno lakše oblikovati, zamijeniti, proširiti i u konačnici održavati. Da bi se postigla što veća modularnost, VHDLLab poslužitelj je napravljen koristeći troslojnu arhitekturu. Poredani od višeg prema nižem to su slojevi: prezentacijski (*presentation*), logički (*service*) i podatkovni (*DAO*). Slojevi su konceptualno linearni (organizirani okomito) tako da prezentacijski i podatkovni sloj nikada ne komuniciraju (traže usluge) izravno, već isključivo prezentacijski sloj traži usluge logičkog sloja, dok on traži usluge podatkovnog sloja. Komunikacija između slojeva odvija se isključivo preko sučelja koje pojedini sloj definira (viši sloj koristi mogućnosti nižeg sloja). Sama implementacija nižeg sloja je nedostupna višem sloju. S druge strane modeli (objekti koji sadrže podatke nad kojima se zasniva poslužitelj) koriste se na svim slojevima tako da su oni dostupni kroz cijeli VHDLLab poslužitelj.



Slika 5.4: Slojeva struktura VHDLLab poslužitelja

Tablica 5.2: Opis podataka koje sadrži model File

tip	Ime	opis
Long	Id	jedinstveni identifikator datoteke
String	Name	ime datoteke
String	Type	tip datoteke
String	Content	sadržaj datoteke
Project	Project	projekt kojem datoteka pripada



Slika 5.6: Odnos File i Project modela

Model *UserFile* je vrlo sličan modelu *File*. Razlika je što *UserFile* ne pripada nekom projektu već korisniku. Ovaj se model koristi za postavljanje korisničkih postavki. Na poslužiteljskoj strani koristi se samo za persistenciju, dok se sadržaj ovog modela prije svega koristi na klijentskoj strani. Primjerice, u ovaj model se može zapisati podrazumijevani jezik, tako da korisnik ne treba pri svakom pokretanju klijentskog sučelja postavljati u kojem jeziku da mu se VHDLLab klijentska aplikacija prikazuje.

Tablica 5.3: Opis podataka koje sadrži model UserFile

tip	Ime	opis
Long	Id	jedinstveni identifikator korisničke datoteke
String	Name	ime korisničke datoteke
String	Type	tip korisničke datoteke
String	Content	sadržaj korisničke datoteke
String	userId	identifikator korisnika (vlasnika) korisničke datoteke



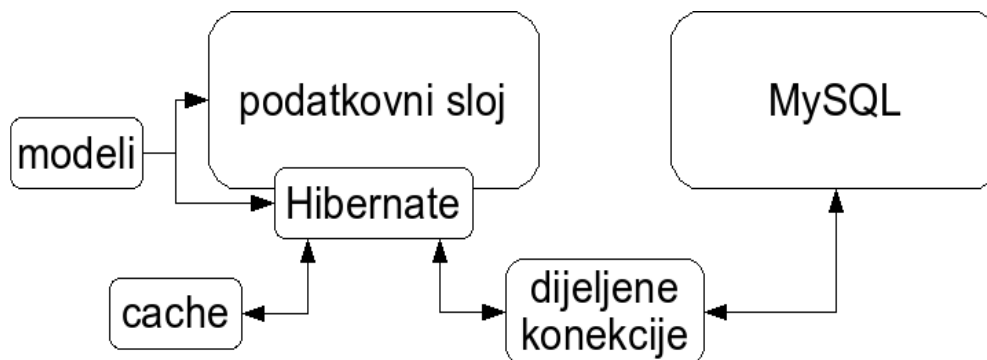
Slika 5.7: Odnos korisnika i UserFile modela

5.2.2. Podatkovni sloj

Podatkovni sloj služi kao spremište i pronalaženje (učitavanje) modela (podataka). Pri tome se kao spremište koristi baza podataka, tj. podatkovni sloj koristi najpoznatiji sustav za upravljanje bazom podataka otvorenog kôda: *MySQL*¹⁶. No kako je *MySQL* relacijska baza

¹⁶ <http://www.mysql.com/>

podataka, a ne objektna, korištena je biblioteka za objekt-relacijsko preslikavanje – *Hibernate*¹⁷. *Hibernate* dakle, preslikava Java objekte u relacijske tablice baze podataka i obratno. Da bi on znao koje svojstvo (*engl. property*) objekta treba preslikati u stupac relacijske tablice potrebno je napisati konfiguracijske datoteke. No pisanje tih datoteka je izbjegnuto tako da se koristi Xdoclet¹⁸ - generator konfiguracijskih ili izvršnih datoteka. U slučaju podatkovnog sloja Xdoclet se koristi da generira konfiguracijske datoteke za *Hibernate*.



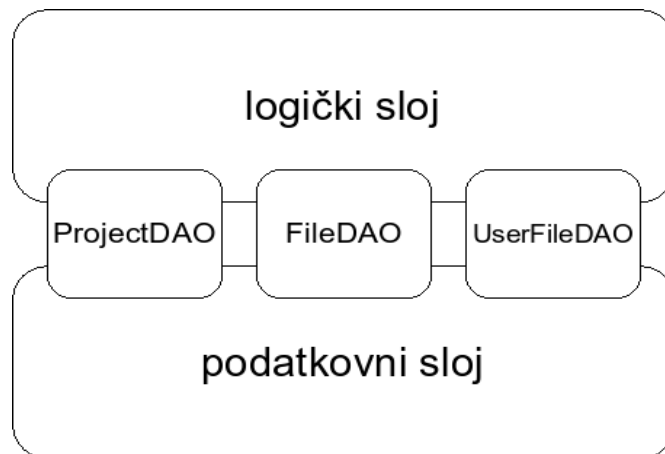
Slika 5.8: Organizacija i korištene tehnologije u podatkovnom sloju

VHDLLab poslužitelj i sustav za upravljanje bazom podataka mogu biti smješteni na različitim računalima opremljenim mrežnim karticama, a kao standardno programsko sučelje za spajanje na sustav za upravljanje bazom podataka *Hibernate* koristi JDBC (*Java Database Connectivity*). Pošto je otvaranje nove JDBC konekcije prilično skupo, a potrebno ju je otvoriti za svaki zahtjev, otvara se nekoliko dijeljenih konekcija. To znači da se jedna konekcija koristi za više zahtjeva. Time je skraćeno vrijeme inicijaliziranja i zatvaranja konekcija, tj. povećane su performanse za malu cijenu. Dodatno se podaci, jednom kad se dovedu iz baze podataka, privremeno spremaju u memoriju (u *cache*) pa se za ponovljeni upit istih podataka ne stvara zahtjev već se odmah vraćaju iz memorije.

Podatkovni sloj pruža uslugu spremanja i pronalaženja modela kroz tri sučelja vidljiva logičkom sloju. Po jedno sučelje za svaki model. Dakako, svaka implementacija tih sučelja koristi *Hibernate*. Ta sučelja su: *ProjectDAO*, *FileDAO* i *UserFileDAO*.

¹⁷ <http://www.hibernate.org/>

¹⁸ <http://xdoclet.codehaus.org/>



Slika 5.9: Sučelja podatkovnog sloja korištena u logičkom sloju

Sučelje *ProjectDAO* definira metode za manipuliranje *Project* modelom, sučelje *FileDAO* definira metode za manipuliranje *File* modelom, dok sučelje *UserFileDAO* definira metode za manipuliranje *UserFile* modelom.

Tablica 5.4: Definirane metode ProjectDAO sučelja

tip povratne vrijednosti	ime metode	parametri metode	opis
Project	Load	id	za odgovarajući jedinstveni identifikator vraća Project model
	Save	project	sprema Project model
	Delete	project	briše Project model
boolean	Exists	id	vraća istinu ako projekt s odgovarajućim identifikatorom postoji
boolean	Exists	userId, name	vraća istinu ako projekt ima navedeno ime i pripada korisniku
List<Project>	findByUser	userId	vraća sve Project modele koji pripadaju navedenom korisniku

Tablica 5.5: Definirane metode FileDAO sučelja

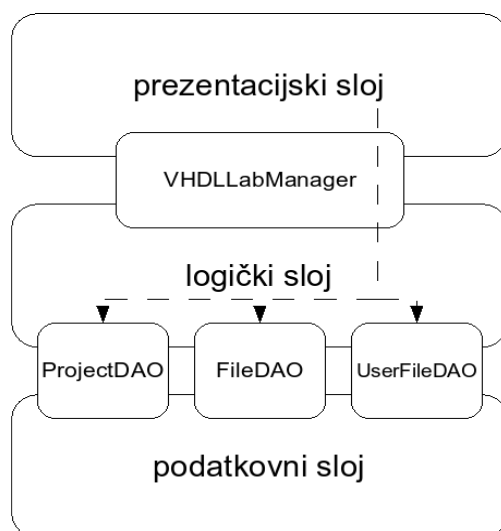
tip povratne vrijednosti	ime metode	parametri metode	opis
File	load	id	za odgovarajući jedinstveni identifikator vraća File model
	save	file	sprema File model
	delete	file	briše File model
boolean	exists	id	vraća istinu ako datoteka s odgovarajućim identifikatorom postoji
boolean	exists	projectId, name	vraća istinu ako datoteka ima navedeno ime i pripada projektu
List<File>	findByName	projectId, name	vraća File model koji ima navedeno ime i pripada projektu

Tablica 5.6: Definirane metode UserFileDAO sučelja

tip povratne vrijednosti	ime metode	parametri metode	opis
UserFile	Load	id	za odgovarajući jedinstveni identifikator vraća UserFile model
	Save	file	sprema UserFile model
	Delete	file	briše Project model
boolean	Exists	id	vraća istinu ako datoteka s odgovarajućim identifikatorom postoji
boolean	Exists	userId, name	vraća istinu ako datoteka ima navedeno ime i pripada korisniku
UserFile	findByName	userId, name	vraća File model koji ima navedeno ime i pripada korisniku
List<UserFile>	findByUser	userId	vraća UserFile modele koji pripadaju korisniku

5.2.3. Logički sloj

Logički sloj pruža mogućnosti manipuliranja s modelima (pogotovo s *File* modelom), tj. koristeći te modele generira razne informacije. No kako su slojevi linearni (kao što je objašnjeno ranije), a logički sloj je srednji sloj, tj. smješten između prezentacijskog i podatkovnog, ponekad su prezentacijskom sloju potrebne mogućnosti podatkovnog (npr. ako korisnik jednostavno želi spremiti jednu VHDL datoteku). Zbog toga logički sloj djelomično implementira mogućnosti podatkovnog sloja tako da jednostavno koristi sučelja podatkovnog sloja. Na taj način prezentacijski sloj koristi logički kao most do podatkovnog. Za te potrebe postoji jedno sučelje koje objedinjuje sve mogućnosti logičkog sloja: *VHDLLabManager*.



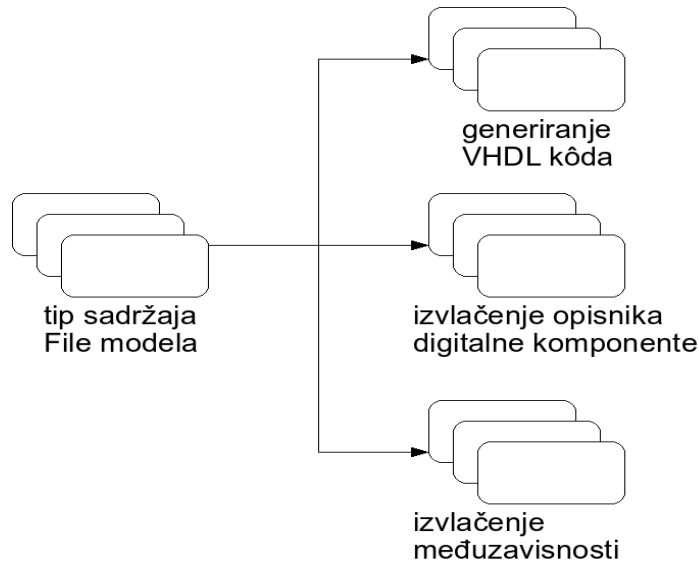
Slika 5.10: Prezentacijski sloj koristi mogućnosti logičkog sloja te preko njega pristupa podatkovnom sloju

Većina informacija koje logički sloj pruža se tiču *File* modela, a kako on može imati više tipova sadržaja (polje *content*), u ovisnosti o tipu (polje *fileType*) se bira implementacija koja zna raditi s tim sadržajem. Trenutno logički sloj prepoznaje 5 tipova sadržaja *File* modela, tj. navedena namjena je odraz različitih sadržaja u modelu:

- normalan VHDL kôd
- VHDL prikazan automatom
- VHDL prikazan shemom
- testbench (testna komponenta)
- rezultat simulacije

Svi ovi tipovi (osim rezultat simulacije jer se za taj tip nema što generirati) imaju posebnu implementaciju za generiranje informacija u logičkom sloju:

- generiranje VHDL kôda
- izvlačenje opisnika digitalne komponente, tj. informacije o ulaznim i izlaznim žicama komponente
- izvlačenje međuovisnosti o drugim komponentama



Slika 5.11: Za svaki tip sadržaja postoje posebne implementacije koje rade naveden posao

Dodatne informacije koje *VHDLLabManager* može generirati su:

- rezultat prevođenje VHDL kôda, tj. vraćanje oznake da je prevođenje uspješno, odnosno liste popratnih grešaka u suprotnom
- rezultat simulacije testnog sklopa (*engl. testbench*), tj. vraćanje oznake da je simulacija uspješna, odnosno liste popratnih grešaka u suprotnom
- izgradnja hierarhijskog stabla za zadani projekt, tj. razrješavanje međuovisnosti između svih datoteka u projektu

Za prevođenje i simulaciju digitalnih komponenti koristi se GHDL¹⁹ program treće strane otvorenog kôda. On nije pisan u Javi pa se poziva kao novi proces te se pomoću cjevovoda čitaju standardni izlazi tog procesa.

¹⁹ <http://ghdl.free.fr/>

5.2.4. Prezentacijski sloj

Prezentacijski sloj je najviši sloj u slojevitoj strukturi VHDLLab poslužitelja i on, koristeći Servlet standardno programsko sučelje, omogućuje mrežnim aplikacijama korištenje mogućnosti koje pruža VHDLLab poslužitelj. Kako se Servlet-i mogu izvoditi samo unutar spremnika Servlet-a (*engl. servlet container*) koristi se *Apache Tomcat* koji je dodatno i HTTP poslužitelj. Servlet je jednostavan i konzistentan mehanizam proširenja mrežnog poslužitelja, tj. jednostavan način da se iz Java programskog jezika implementira servis odnosno mrežna aplikacija. To je upravo i jedan od razloga što Servlet tehnologija izabrana za potrebe VHDLLab poslužitelja.

Dva su Servlet-a definirana u prezentacijskom sloju:

- VHDLLab Servlet koji omogućuje VHDLLab klijentska aplikacija da pristupa i koriste mogućnosti VHDLLab poslužitelja
- JNLP Servlet koji omogućuje jednostavnu instalaciju VHDLLab klijentske aplikacije (objašnjeno kasnije)

VHDLLab Servlet se oslanja na HTTP i svi podaci se prenose upravo preko tog protokola. Sve što klijentska aplikacija treba navesti u zahtjevu je koju informaciju želi dohvatiti ili modificirati te parametre zahtjeva. Parametri se pohranjuju u jednostavnu mapu "ključ = vrijednost" (*engl. Java properties*), dok je zahtjev za informacijom jedan niz znakova koji jednoznačno definira kakvu se informaciju želi dohvatiti ili modificirati. Objekti koji pritom obrađuju takve informacije zovu se registrirane metode i trenutno ih u VHDLLab poslužitelju ima 55. To znači da klijentska aplikacija može iz VHDLLab poslužitelja dohvatiti ili modificirati 55 informacija. Sve te metode uglavnom omogućuju sve što logički sloj omogućuje (izravno ili neizravno preko podatkovnog sloja). U to je dakle uključeno dohvat i pohrana podataka iz svih modela kao i dohvat rezultata prevođenja, simulacije, generiranja VHDL kôda, razrješavanje međuzavisnosti itd. Dodatno je moguće dohvatiti bilo koji podatak iz modela, ali potreban je poseban zahtjev za svaki podatak jer modeli ne postoje izvan VHDLLab poslužitelja. Koja će se registrirana metoda koristiti ovisi o zahtjevu i to je vrlo elegantno riješeno u ovom sloju – koristeći *factory method* oblikovni obrazac (*engl. design pattern*).

Do sada je opisano da se preko HTTP-a prenose zahtjevi i odgovori, no nije opisano u kojem se formatu oni prenose. Rješenje koje je korišteno prilično je jednostavno. Koristi se serijalizacija koju omogućuje Java platforma. Jedan objekt je bazni generički objekt koji se koristi pri komunikaciji klijentske i poslužiteljske aplikacije. Svi ostali objekti su nadogradnja na bazni tako da korisnik ne treba znati točan znakovni niz za svaki pojedini zahtjev već samo instancira objekt koji opisuje traženi zahtjev, a taj objekt onda postavlja točan znakovni niz. Bazni objekt se zove *Method* i sadrži tri polja koja odgovaraju već poznatoj razmjeni podataka.

Tablica 5.7: Polja objekta Method

Tip	ime	Opis
String	methodName	znakovni niz koji označava pojedini zahtjev (upisuje se automatski kada se koriste spomenuto instanciranje objekta koji opisuje traženi zahtjev)
Properties	parameters	parametri zahtjeva
T	result	rezultat zahtjeva; upisuje ga jedna od registriranih metoda kad obradi zahtjev ili se uopće ne upisuje ako je zahtjev isključivo modificirajuće namjene

Prva dva polja popunjava klijentska aplikacija, dok se rezultat postavlja isključivo u VHDLLab poslužitelju (u registriranoj metodi). Tip rezultata je ovisan o informaciji koja se treba vratiti pa je on generički. Evo jedan primjer korištenja sa strane korisničke aplikacije:

```
// učitaj sadržaj datoteke s identifikatorom 1065
Method m = new LoadFileContentMethod(Long.valueOf(1065));

... // serijaliziraj objekt i pošalji ga VHDLLab poslužitelju

String content = m.getResult(); // rezultat zahtjeva
... // učini nešto sa sadržajem
```

Primjer 5.1 Primjer korištenja VHDLLab api-a sa strane korisničke aplikacije

Drugi Servlet koji prezentacijski sloj definira je *JNLP* Servlet. On je donekle vezan uz klijentsku aplikaciju jer ona koristi *JNLP* (*engl. Java Network Launching Protocol*) i na taj način izbjegne bilo kakve instalacijske procedure i dodatne komplikacije. Sve što je potrebno da se podrži takav protokol je jedna XML datoteka u *JNLP* formatu. *JNLP* Servlet stvara i vraća takvu datoteku s odgovarajućim parametrima.

5.2.5. Sigurnost

Tijekom razvoja VHDLLab projekta (klijentske i poslužiteljske aplikacije) razmišljalo se o sigurnosti. Zbog toga je VHDLLab poprilično siguran iako se sa sigurnošću nikad ne zna. To je osjetljiva tema i nikad nije moguće postići stopostotnu sigurnost. Barem ne dok ste priključeni na Internet, a pogotovo ne za nekakav poslužitelj.

Prvo što VHDLLab poslužitelj zahtjeva od korisnika je autentifikacija. Bez nje korisnik neće moći pristupiti poslužitelju. Pri tome se stvarna baza korisnika ne treba držati zajedno s VHDLLab poslužiteljem. Npr. za probni rok na kojem se VHDLLab koristio za predmet Digitalna logika na Fakultetu elektrotehnike i računarstva autentifikacija se vršila preko glavnog fakultetskog poslužitelja te namjene. Nije se zahtijevala ponovna registracija studenata samo da bi koristili VHDLLab.

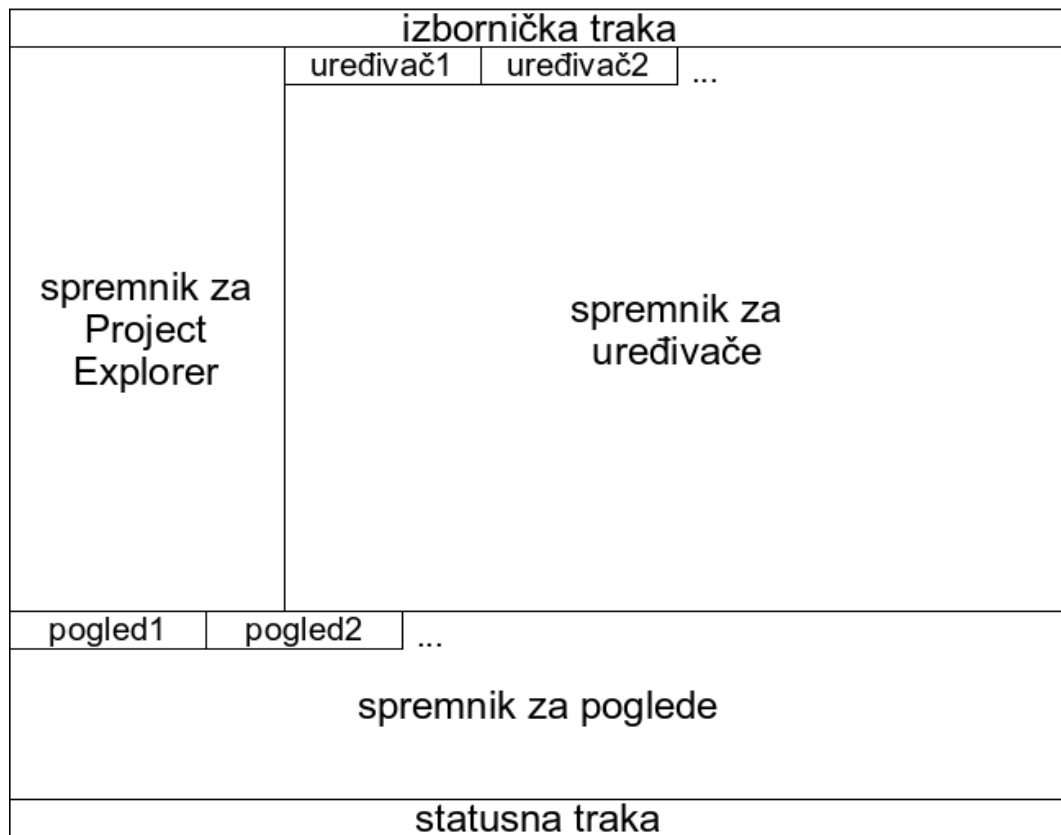
Pored autentifikacije koristi se i postupak autorizacije, tj. obični korisnici ne mogu obavljati zahtjeve namijenjene administratoru sustava. Nadalje, korisnik može mijenjati i čitati isključivo podatke koje pripadaju njemu, a ne nekom drugom korisniku.

Dodatno se provjeravaju i serijalizirani objekti koji prenose podatke između klijentske i poslužiteljske aplikacije. Da bi se ustanovilo da li je netko izvana pokušao mijenjati sadržaj tog objekta koristi se obrambeno kopiranje.

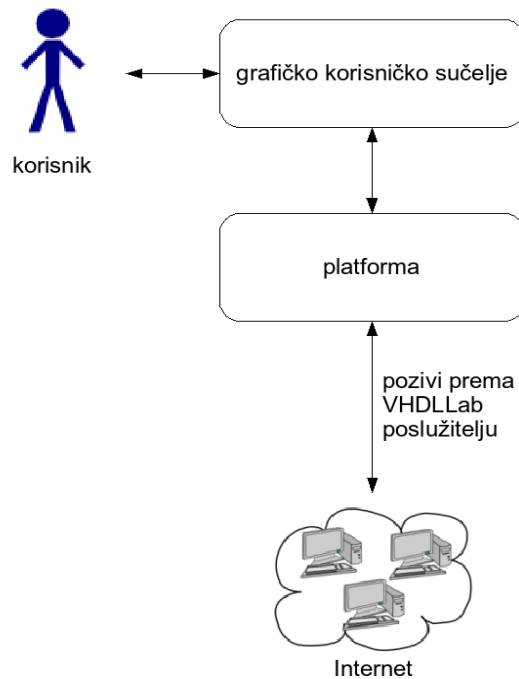
I za kraj se kroz cijelu sjednicu (od autentifikacije do kraja rada s poslužiteljskom aplikacijom) kriptiraju svi podaci poslani prema poslužitelja i od njega. Za to se koristi *SSL* kriptografski protokol na transportnom sloju *OSI* modela računalne mreže koji je korišten od velikog broja drugih aplikacija (nepovezanih sa sustavom VHDLLab). Korištenjem *SSL* protokola sprečava se mnogo vrsta napada kao npr. napad s čovjekom u sredini, krađa sjedničkog ključa, krađa bilo kakvih privatnih informacija uključujući i šifru pri pokušaju autentifikacije, kao i bilo kakvo njuškanje mrežnog prometa.

5.3. VHDLLab klijent

Klijentska aplikacija je jedini dio VHDLLab sustava vidljiv korisniku. Koristeći grafičko sučelje (*GUI*, *engl. graphical user interface*) korisnik može obaviti sve što VHDLLab poslužitelj omogućuje, tj. jednostavno i efikasno obavljati dizajniranje i testiranje digitalnog sustava (sastavljenoga od više digitalnih sklopova). Korisnik pritom koristi dvije komponente grafičkog sustava (što je i neka gruba podjela istog): *Editor* (uređivač) i *View* (pogled). Razlika između te dvije komponente je što uređivači mogu mijenjati nekakav sadržaj (npr. *VHDL* kôd) dok pogledi služe isključivo za pregledavanje sadržaja (npr. pregledavanje rezultata prevođenja). Pozicija tih komponenti je precizno određena pa se tako uređivači prikazuju u središnjem (najvećem) spremniku dok se pogledi prikazuju ispod uređivača osim u slučaju jednog posebnog koji se nalazi lijevo od uređivača. Taj se poseban pogled zove preglednik projekata (*engl. Project Explorer*) koji se koristi da prikaže sve projekte i hijerarhiju datoteka unutar pojedinog projekta. Pritom oba spremnika imaju mogućnost istovremenog otvaranja više uređivača/pogleda, ali prikazuju samo jedan.

**Slika 5.12: Organizacija prostora u glavnom prozoru**

Klijentska aplikacija je napravljena tako da gotovo transparentno koristi usluge VHDLLab poslužitelja. Da bi se postigla transparentnost, kao i razne razine apstrakcije napravljena je platforma – kompletan sustav koji definira standardno aplikacijsko sučelje za uređivače i poglede. Svi razredi koji su povezani s grafičkim korisničkim sučeljem koriste platformu za komunikaciju s ostatkom sustava.

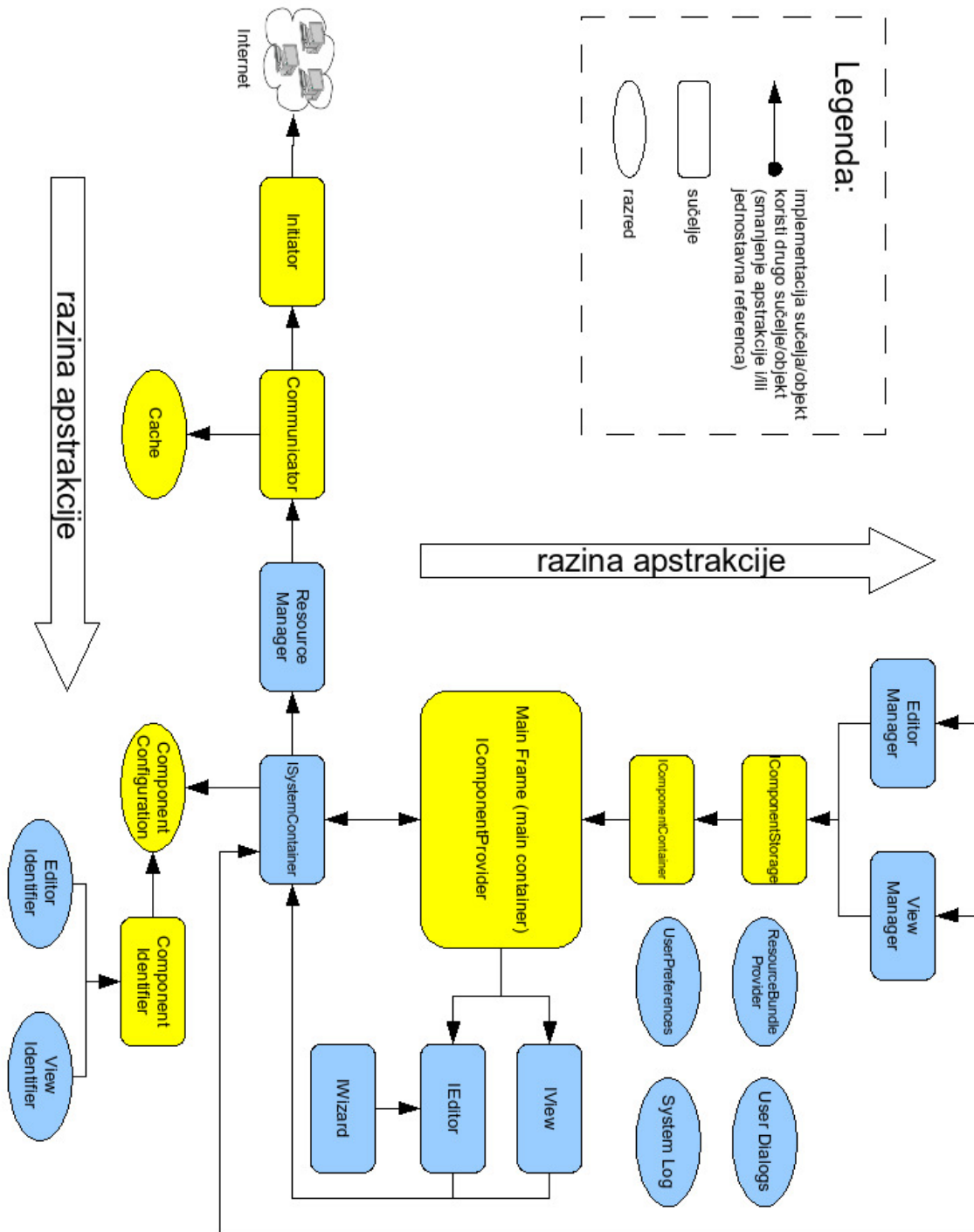


Slika 5.13: Organizacija klijentske aplikacije

5.3.1. Platforma

Pored toga što osigurava transparentnost, platforma također omogućuje da se koristi standardno aplikacijsko sučelje koje ima veliku apstrakciju u odnosu na ono što VHDLLab poslužitelj nudi. Na taj način uređivači i pogledi ne trebaju znati ništa o primjerice VHDLLab poslužitelju, načinu na koji se podaci prenose, formatu prijenosa, sigurnosti i nizu ostalih aspekata za koje se brine platforma. Implementacija platforme uključuje niz sučelja koja, gledajući od standardnog aplikacijskog sučelja prema samoj konekciji na VHDLLab poslužitelj, smanjuju razinu apstrakcije. Također platforma je napravljena tako da koristi arhitekturu baziranu na događajima, tj. kada se dogodi neka akcija o tome će se obavijestiti svi registrirani slušatelji (*engl. listeners*). Na taj se način platforma lakše koristi i lakše ju je unaprijediti.

Napomena: Riječ *resurs* u danjem tekstu označava datoteku, projekt ili korisničku datoteku dok riječ *komponenta* označava uređivač ili pogled.



Slika 5.14: Arhitektura platforme i veze između pojedinih dijelova

* plavi oblici predstavljaju standardno aplikacijsko sučelje, dok su žuti dio implementacije platforme

Initiator sučelje ima najnižu razinu apstrakcije u platformi i pruža mogućnost da se obavi zahtjev za dohvatom ili modifikacijom neke informacije VHDLLab poslužitelju. Pri čemu je zahtjev objekt *Method* koji kroz standardno aplikacijsko sučelje nudi VHDLLab poslužitelj. U tom istom objektu će se nakon obrade zahtjeva nalaziti rezultat izvođenja. Platforma implementira ovo sučelje tako da koristi *Apache HTTP Client*²⁰ biblioteku za potrebe uspostave HTTP konekcije. Dodatno će *Method* objekt serijalizirati, te takav niz bajtova dodatno komprimirati koristeći *GZIP* kompresiju. Implementacija također vodi računa o autentifikaciji korisnika koja se vrši preko HTTP-a i koristi *SSL* da bi se podaci kriptirali.

Puno je lakše korisniku i ostatku platforme identificirati pojedine resurse preko alternativnoga identifikatora nego preko primarnog kojeg čini poprilično velik broj. Cilj *Communicator* sučelja je upravo da pretvara alternativni jedinstveni identifikator u primarni i obrnuto. Na taj način omogućuje ostatku platforme da resurse identificira na višoj razini apstrakcije dok se za pojedini zahtjev koristi primarni identifikator kako to i VHDLLab poslužitelj zahtjeva. Takve mape, primarni - alternativni identifikator, se održavaju u razredu *Cache* da se ne treba svaki put VHDLLab poslužitelj pitati za alternativni identifikator. *Cache* razred dodatno održava tipove podataka u upotrebi jer ostatak platforme često zahtjeva tu informaciju. U implementaciji *Communicator* sučelja se također kreira različite *Method* objekte (ovisno o zahtjevu) i iz njih se vraća rezultat zahtjeva pozivatelju metode sučelja. Neke metode *Communicator* sučelja vraćaju objekte koje sadrže više podataka za koje inače treba više zahtjeva te tako dodatno podiže razinu apstrakcije. Alternativni identifikator za pojedini resurs je slijedeći:

- projekt – ime projekta i identifikator korisnika
- datoteka – ime datoteke i ime projekta
- korisnička datoteka – ime korisničke datoteke i identifikator korisnika

Sučelje *Resource Manager* sadrži sve metode kao i *Communicator* sučelje, ali i dodatne za registraciju slušaoca (*engl. listener*). Ovo sučelje implementira arhitekturu baziranu na događajima za: prije i nakon prevođenje ili simulacije kôda, te prije ili nakon kreiranja ili brisanja projekta ili datoteke. Za događaje prije neke akcije slušaoci mogu postaviti veto te tako zabraniti da se akcija uopće izvede. Npr. jedan registrirani slušaoc postavlja veto na kreiranje projekta ako njegovo ime nije dozvoljeno (takvo da bi VHDLLab poslužitelj to prihvatio). Ovo sučelje je zadnje koje se bavi dohvatom ili mijenjanjem podataka s VHDLLab poslužitelja.

Osim *ISystemContainer* sučelja sva ostala sučelja se bave komponentama (uređivači i pogledi). *ISystemContainer* jednostavno sadrži reference na sve *Manager* sučelja,

²⁰ <http://hc.apache.org/httpclient-3.x/>

Component Configuration, te na *Main Frame* da omogući drugima da ostvare modalne dijaloge. To sučelje se predaje svima koji za dodatne mogućnosti koriste druga sučelja.

Oboje uređivači i pogledi se u platformi vode kao zasebni moduli te je zbog toga vrlo jednostavno dodavati nove. Sve što je potrebno napraviti je dodati opis u posebnu konfiguracijsku datoteku (niti jedan izvorni kôd pritom nije potrebno mijenjati). Takav opis uključuje: jedinstveni identifikator uređivača ili pogleda realiziran kao jednostavan niz znakova, te puno ime razreda koji implementira odgovarajuće sučelje - *IEditor* ili *IView*. Za uređivače je opcionalno moguće navesti: tip datoteke koju je on sposoban prikazati, da li uređivač dopušta da se njegov sadržaj spremi, te da li je sadržaj uređivača moguće promijeniti jednom kada se postavi. Model te konfiguracijske datoteke je upravo *Component Configuration* razred.

Svi uređivači (a isto vrijedi i za poglede) se u platformi tretiraju jednako, tj. platforma ne razlikuje uređivače na temelju njihovih mogućnosti. To je i jedan od razloga što se uređivači mogu vrlo jednostavno dodavati u platformu. No kako je ponekad potrebno dohvatiti točno određenu instancu uređivača, platforma pruža tu mogućnost preko *Editor Identifier* razreda (i *View Identifier* za poglede), a *Component Identifier* definira sučelje za tu mogućnost. Identifikacija se vrši preko tri parametra: grupa kojoj komponenta pripada (uređivač ili pogled), jedinstveni identifikator komponente (definiran u konfiguracijskoj datoteci) i neobavezni objekt koji služi kao modifikator primjerka komponente i koji se primarno koristi za uređivače. Primjer za to je kada su otvorena dva ista uređivača, ali prikazuju sadržaj dviju različitih datoteka. U tom slučaju je modifikator instance upravo identifikator datoteke koju pojedini uređivač prikazuje. Ako ipak modifikator instance nije postavljen tada platforma neće omogućiti da više od jedne instance komponente postoji (biva prikazan korisniku).

Main Frame je glavni prozor klijentske aplikacije u kojeg se svi ostali spremnici ili komponente spremaju, te dodatno sadrži izborničku traku kao i statusnu traku. *IComponentProvider* sučelje koje *Main Frame* implementira definira jednu metodu koja na osnovu parametara vraća spremnik komponenata. Parametar je jednostavna pozicija spremnika, odnosno: centar, lijevo ili dolje.

IComponentContainer sučelje koristi *IComponentProvider* da bi za pojedini spremnik omogućilo otvaranje (prikazivanje korisniku) i zatvaranje komponente, postavljanje naslova i jednostavnog opisa, dohvaćanje trenutno označene (aktivne) komponente, itd. *IComponentStorage* sučelje nadograđuje mogućnosti *IComponentContainer*-a tako što omogućuje identificiranje komponente – nužno zbog *Editor* i *View Identifier* razreda.

Editor Manager i *View Manager* su specijalizirana sučelja za uređivače i poglede. *Editor Manager* dodatno nudi mogućnosti prikazivanja sadržaja pojedinog uređivača u obliku VHDL kôda, lakše otvaranje uređivača na temelju identifikatora datoteke, spremanja sadržaja jednog ili više uređivača, zatvaranja jednog ili svih uređivača, vraćanje svih trenutno

otvorenih uređivača, itd. *View Manager* ima slične mogućnosti kao i *Editor Manager* samo primjenjive na poglede.

Četiri sljedeća razreda nemaju izravnih referenci kao što su sučelja/razredi do sada imali tako da se mogu koristiti od strane komponenata odmah, bez da se njima pristupa preko *ISystemContainer* sučelja. *ResourceBundleProvider* razred je vezan uz lokalizaciju (prirodnog) jezika, tj. prikaz klijentske aplikacije u pojedinom jeziku. Koristeći taj razred komponente ne trebaju znati o kojem se točno jeziku radi već će to on saznati koristeći platformu. *UserPreferences* omogućuje komponentama da koriste korisničke postavke (korisničke datoteke) kako bi prilagodile svoje ponašanje kako to korisniku odgovara. Primjerice, kada se klijentska aplikacija zatvori, a bili su otvoreni neki uređivači, pri ponovnom pokretanju aplikacija ti uređivači će se ponovno otvoriti. *User Dialogs* predstavljaju skup dijaloga (prozori) koje više koristi platforma nego komponente iako je i njima na raspolaganju. Nego od dijaloga su *Save Dialog* i *Run Dialog*, od kojih se prvi koristi da se korisniku javi koje uređivače hoće spremati, a drugi za pokretanje prevođenja ili simulacije kôda. *System Log* je poprilično važan razred baziran na događajima u kojeg se spremaju zadnjih 50 rezultata prevođenja i simulacije, tako da korisnik može vidjeti primjerice koje je datoteke prevodio/simulirao. Ovaj je razred važan jer sadrži poruke sustava kao i poruke o eventualnim greškama. Poruke sustava se generiraju svaki put kada se izvrši neka važnija akcija. Npr. spremanje, kreiranje i brisanje datoteke ili projekta, prevođenje ili simulacija kôda, itd. Takve će poruke korisnik vidjeti zahvaljujući posebnom *Status History* pogledu. Poruke o eventualnim greškama su detaljne poruke ukoliko se dogodi nekakva iznimna situacija (ukoliko postoji *bug*). I za takve poruke postoji posebni pogled koji ih prikazuje, ali je on dostupan samo u posebnom režimu rada klijentske aplikacije koji se uključuje samo dok se aplikacija razvija, dok se u normalnom režimu rada takve poruke šalju VHDLLab poslužitelju gdje se pohranjuju. Normalni korisnici obično nisu svjesni ovih poruka (niti to njima nešto znači), ali je nama (programerima klijentske aplikacije) ovo neprocjenjiva informacija da se što lakše reproduciraju, te u konačnici i poprave *bug*-ovi.

IEditor sučelje definira sve metode koje uređivač mora implementirati da bi se mogao koristiti.

Tablica 5.8: Definirane metode IEditor sučelja

ime metode	Opis
Init	inicijalizira stanje uređivača (npr. grafičko korisničko sučelje)
Dispose	uništava uređivač (npr. uređivač zaustavlja sve pokrenute dretve)
setSystemContainer	postavlja ISystemContainer da uređivač može pristupiti različitim informacijama
getWizard	vraća instancu IWizard implementacije nužne za sve uređivače
addEditorListener	dodaje slušaoc koji je obaviješten o svim promjenama sadržaja uređivača
removeEditorListener	uklanja registriranog slušača
removeAllEditorListeners	uklanja sve registrirane slušače
getListeners	vraća sve registrirane slušače
Undo	pokreće undo akciju (integracija s glavnom izbornom trakom)
Redo	pokreće redo akciju (integracija s glavnom izbornom trakom)
setFileContent	postavlja ime datoteke i projekta kao i sadržaj datoteke koje uređivač prikazuje
getData	vraća sadržaj uređivača
getProjectName	vraća ime projekta kojem pripada datoteka koju uređivač prikazuje
getFileName	vraća ime datoteke koju uređivač prikazuje
setSaveable	ovisno o zastavici postavlja da se sadržaj uređivača može pohraniti
isSaveable	vraća zastavicu da li se sadržaj uređivača može pohraniti
setReadOnly	ovisno o zastavici postavlja da se sadržaj uređivača može mijenjati
isReadOnly	vraća zastavicu da li se sadržaj uređivača može mijenjati
highlightLine	postavlja specificiranu liniju kao označenu (radi samo kod TextEditor-a)

Svi uređivači imaju poseban proces inicijalizacije koji se sastoji od pozivanja odgovarajućih metoda definiranim redoslijedom na koje onda uređivači mogu računati:

1. `setSystemContainer`
2. `init`
3. `setSaveable`
4. `setReadOnly`
5. `setFileContent`

Kada se uređivač uništava (zatvara) jednostavno se poziva metoda `dispose`.

IWizard sučelje definiran samo jednu metodu: *getInitialFileContent* koja ima namjenu da prikaže čarobnjak (*engl. wizard*) koji vodi korisnika kroz proces kreiranja nove datoteke (npr. kreiranje nove VHDL datoteke). Sadržaj datoteke se pritom puni s nekim podrazumijevanim vrijednostima (npr. u slučaju nove VHDL datoteke to može biti unaprijed definirane *library* ili *use* naredbe). Metoda vraća ime nove datoteke, ime projekta kojemu pripada te generirani sadržaj datoteke.

IView sučelje je slično *IEditor* sučelju samo što je puno jednostavnije. Ima samo mogućnosti inicijalizacije i destrukcije, te postavljanja *ISystemContainer* sučelja.

5.3.2. Uređivači

Slijedeći uređivači su definirani (korišteni) u VHDLLab klijentskoj aplikaciji:

- `TextEditor`
- `Testbench`
- `Simulation`
- `Automat`
- `Schema`

TextEditor je vrlo jednostavan uređivač koji se oslanja na *JTextPane* komponentu koja već postoji u Java platformi i omogućuje da se upisuje jednostavan tekst. Ovaj uređivač implementira funkcionalnost *highlightLine* te integracijom s *Compilation* pogledom omogućuje da korisnik klikne na određenu grešku prevođenja te vidi točno u kojoj se liniji dogodila pogreška. Ovaj uređivač može onemogućiti korisniku da mijenja sadržaj tako da mu se preda neistinita zastavica u *setReadOnly* metodi. Primjer za to je kada se samo pregledava VHDL kôd nekih drugih uređivača, jer je takav VHDL kôd generiran automatski, te ga nije moguće mijenjati. *Undo* i *Redo* metode su također implementirane tako da, ukoliko korisnik pogriješi te obriše dio VHDL kôda, može jednostavno vratiti obrisani dio korištenjem *undo*, odnosno *redo*, akcije iz izborničke trake.

Uređivač ispitnih sklopova omogućuje stvaranje ispitnog sklopa korištenog pri simulaciji. Ispitni sklop je ovisan o digitalnom sklopu koji se ispituje (te će se to odraziti u hijerarhiji projekta). Ispitni sklop se gradi tako da se iz ovisne digitalne komponente izvuče opis ulaznih vrata (*engl. port*) te se za ta vrata podešavaju logične vrijednosti u ovisnosti o vremenu. Sadržaj uređivača se uvijek može mijenjati i spremati.

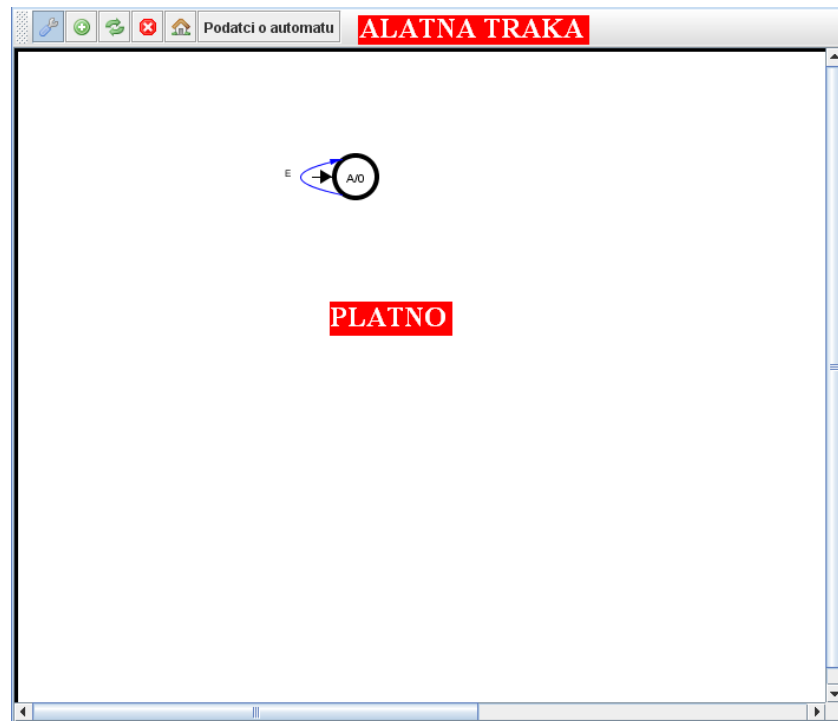
Uređivač simulacija prikazuje rezultat simulacije bazirane na ispitnom sklopu. Slično kao i kod uređivača ispitnih sklopova, uređivač simulacija prikazuje logične vrijednosti pojedinih ulaznih, ali i svih internih signala te izlaznih vrata u ovisnosti o vremenu. Dodatno ima mogućnost uvećanja ili umanjivanja skale za lakši pregled, automatsko pozicioniranje na sljedeći ili prethodni rastući ili padajući brid pojedinog signala te mjerenja intervala vremena. Sadržaj uređivača se nikada ne može mijenjati, a snimati se može samo ako to korisnik eksplicitno zatraži iz glavne izborničke trake. Na taj način se, primjerice, asistentu jednostavno može prikazati rezultat laboratorijske vježbe.

5.3.2.1. Uređivač automata

Uređivač automata zamišljen je kao zamjena za zamorno pisanje VHDL koda koji modelira Mooreov i Mealyjev stroj s konačnim brojem stanja. Ovaj uređivač u sklopu VHDLLab sustava nasljeđuje razred *AbstractEditor* i implementira sučelje *IWizard*. Glavni razred ovog uređivača je *Automat*. Ovaj razred služi kao spremnik za sve komponente koje se koriste u uređivaču. Preko implementacije *AbstractEditora* opisanog ranije komunicira s ostatkom aplikacije. Najvažniji razred u implementaciji je razred *AutoDrawer*. Ovaj razred predstavlja implementaciju platna za crtanje i modela automata u kojem su pohranjeni svi podatci o njemu. Model automata sastoji se tri bitna dijela. Prvi dio su podatci automata za čiju pohranu se koristi. Za ovaj je posao zadužen razred *AUTPodatci*. Slijedeći dio modela je lista stanja automata. Jedno stanje modelirano je razredom *Stanje*, a sva stanja zajedno definirani su kao povezana lista (*LinkedList<Stanje>*). Posljednji dio modela čine prijelazi. Jedan prijelaz modeliran je razredom *Prijelaz*, a svi prijelazi pohranjeni su u skup realiziran raspršenim adresiranjem (*HashSet<Prijelaz>*). Razred *AutoDrawer* nasljeđuje Javinu *Swing* komponentu *JPanel*. Ovu smo komponentu modificirali da služi kao platno za crtanje. Ovo se ponašanje postiže preopterećivanjem metode *paintComponent* koju svaka *Swing* komponenta ima. Samo se crtanje, kod promjene uzrokovane unosom, izvodi na slici tipa *BufferedImage* koja se u *paintComponent* metodi iscrtava preko komponente. Na ovaj način izveden je tzv. *double-buffering* kojim se izbjegava efekt treperenja (*engl. flicker effect*). Prilikom crtanja matematičkim transformacijama i uporabom Bézierovih krivulja dobiva se efekt zakrivljenosti linija koje predstavljaju prijelaze te položaj teksta na prijelazima. Iscrtavanje radi vizualnog dojma koristi algoritme *antialiasing*-a. Ovim algoritmima postižu se fini prijelazi između područja ispunjenih različitim bojama i umanjuje vidljivost osnovnog elementa displaya (*engl. pixel*). Razred *AutoDrawer* ima šest stanja rada kojima upravljaju gumbi deklarirani u razredu *Automat*.

1. Uređivanje objekata (stanja i prijelaza) u dizajnu
2. Dodavanje novog stanja
3. Dodavanje novog prijelaza I. dio (određivanje stanja iz kojeg se prelazi)
4. Dodavanje novog prijelaza II. dio (određivanje stanja u koje prijelaz vodi)
5. Brisanje postojećih objekata.
6. Određivanje početnog stanja

Alatna traka uređivača služi za mijenjanje trenutnih stanja rada sustava. Sastoji se od petoro gumbi za manipuliranje stanjem rada uređivača i gumba za ispis podataka o automatu koji radimo i promjenu nazva signala sučelja. Sučelje uređivača prikazano je na slici 5.15.



Slika 5.15 Sučelje uređivača automata

Spomenuti razredi *Stanje*, *Prijelaz* i *AUTPodatci* služe kao strukture za spremanje podataka. U strukturi *AUTPodatci* pohranjeni su ime sklopa, tip automata, opis sučelja i još par podataka o dimenzijama platna. Struktura *Stanje* sadrži ime stanja, izlaz za stanje (ako je riječ o Mooreovom automatu), stanje u koje se prelazi za slučaj da ne postoji definiran prijelaz za neki ulaz (*else* prijelaz). Tu su još sadržani i pozicija i boja stanja potrebni za iscrtavanje. Zadnja struktura je prijelaz. Ona sadrži naziv stanja iz kojeg se prelazi, naziv stanja u koje se prelazi, te pobude i izlaze za taj prijelaz.

Podatci o automatu pohranjuju se u XML datoteku specificiranog formata. Ta se datoteka pri obnavljanju automata iz baze podataka parsira alatom *Digester*. *Digester* je moćan alat u obliku Java biblioteke koji omogućuje efikasnu i jednostavnu pretvorbu XML datoteke u Java objekte. Ti objekti su već opisanog tipa *Stanje*, *Prijelaz* i *AUTPodatci*. Razred zadužen za ovu funkcionalnost je *AUTParser*. Generiranje VHDL koda radi poseban razred u sklopu sloja usluga. Način na koji se kod generira opisan je u poglavlju 2.2.5.

5.3.2.2. Uređivač shema digitalnih sklopova

U izradi uređivača za sheme digitalnih sklopova, korišten je oblikovni obrazac model-pogled-kontroler (*MVC*²¹), kako bi se na apstraktnoj razini odvojili podaci, upravljanje podacima i prikaz podataka. Kao što je ranije navedeno, uređivač sheme sklopova modelira digitalni sklop pomoću drugih digitalnih sklopova i žica koje ih povezuju.

Prednosti korištenja ovog obrasca su višestruke. Prije svega, svi podaci vezani uz shemu nalaze se grupirani na jednom logičkom mjestu u programu – modelu. Model je, dakle, dio programa koji sadrži podatke vezane uz program i omogućava pristup tim podacima putem dobro definiranog sučelja²², a to su ovom slučaju naziv sklopa, njegovi ulazni i izlazni signali, sklopovi od kojih je on izgrađen, njihovi nazivi i položaji na platnu, žice koje povezuju sklopove, njihovi nazivi i razmještaji, popis sklopova na koje je pojedina žica priključena, itd. S obzirom da se ovdje radi o velikoj količini podataka, ima smisla sve te podatke držati na jednom mjestu, kako bi se olakšalo programiranje, ali i ispravljanje pogrešaka u programu, održavanje programskog koda i dodavanje novih funkcionalnosti. Treba napomenuti da su svi podaci vezani uz shemu digitalnog sklopa tijekom rada u uređivaču spremljeni u memoriji, te da se u jednom trenutku moraju pohraniti u datoteku, tj. u niz okteta. Ovaj postupak naziva se serijalizacija podataka, i dodatna je prednost izrade modela olakšana serijalizacija, s obzirom da se svi podaci koje treba serijalizirati nalaze na jednom logičkom mjestu u programu. Naposljetku, zahvaljujući modelu, svi su podaci dostupni preko unaprijed dogovorenog sučelja, a detalji pohrane, organizacije i postupaka za dohvat podataka su za ostale slojeve sakriveni.

Pogled je dio ovog obrasca koji prikazuje model u obliku korisničkog sučelja, koje je pogodno za interakciju s korisnikom. U slučaju uređivača shema, to se korisničko sučelje sastoji od platna za izradu sheme, popisa svojstava izabranog sklopa ili žice, te popisa sklopova koje je moguće dodati na platno. Korisničko sučelje uređivača shema je detaljno opisano u prethodnim poglavljima.

Kontroler je dio obrasca koji ima ulogu posrednika između pogleda i modela. On zaprima korisničke zahtjeve, i odgovara na njih, te obavlja promjene na modelu. U izradi uređivača shema digitalnih sklopova, on je izveden u obliku sučelja preko kojeg se iz korisničkog

²¹ Model-view-controller

²² Pod pojmom sučelje se ovdje misli na programsko sučelje, a ne na sučelje digitalnog sklopa kao ranije. Konkretno, u Javi je to programski konstrukt *interface*.

sučelja šalju zahtjevi prema modelu. Prednost tog međusloja sastoji se u tome što on omogućava grupiranje svih operacija i zahtjeva nad podacima na jednom mjestu, što omogućava lakše programiranje i održavanje programskog koda. Ako se obrazac komanda (*engl. command*) koristi u kombinaciji s kontrolerom, on omogućava i sustavno praćenje prethodno obavljenih operacija, i u skladu s tim operacije *undo* i *redo*. Nadalje, u tandemu s obrascem promatrač (*engl. observer*), on omogućava obavješćavanje elemenata pogleda u trenutku kad se obavi promjena nad modelom. Ovo će biti opisano detaljnije nešto kasnije.

Nećemo ulaziti u detalje i implementaciju uređivača shema digitalnih sklopova. Umjesto toga, opisat ćemo isključivo najbitnija sučelja vezana uz model i kontroler uređivača shema, opisati njihovu ulogu i povezanost, te na primjeru opisati funkcioniranje uređivača shema.

Temeljno sučelje preko kojeg se upravlja modelom sheme naziva se *ISchemaInfo*. Preko ovog sučelja moguće je dohvatiti reference na sučelja za upravljanje komponentama, žicama, i svojstvima sklopa koji se modelira. Ovo sučelje omogućava pristup svim podacima koji su za shemu bitni, pa je pomoću njega pogodno činiti i serijalizaciju u tekstualni oblik. Serijalizaciju provodi klasa *SchemaSerializer* koja pomoću ovog sučelja dobiva pristup kolekciji komponentata, kolekciji žica i kolekciji svojstava sklopa, te iterirajući kroz njih zapisuje sve podatke u XML formatu u datoteku. Kasnije, klasa *SchemaDeserializer* dobiva tu datoteku i na temelju nje stvara i vraća *ISchemaInfo* objekt. Svaki *ISchemaInfo* objekt ima metodu koja vraća referencu na *ISchemaComponentCollection*, *ISchemaWireCollection* i *ISchemaEntity*.

ISchemaComponentCollection je sučelje koje služi za dohvat sklopova po imenu i položaju, iteriranje po skupu sklopova, dodavanje, preimenovanje, micanje i brisanje sklopova. Svaki je sklop na platnu predstavljen jednim objektom koji implementira sučelje *ISchemaComponent*, te se u nekoj implementaciji sučelja *ISchemaComponentCollection* nalazi na popisu objekata. Zadaća je ovog sučelja omogućiti pristup skupu sklopova bez znanja o internoj pohrani objekata. Neka implementacija bi mogla objekte naprosto skladištiti u jednostruko povezanu listu, za što bi eventualni dohvat po koordinatama bio linearna operacija. Međutim, ako se kasnije želi implementirati efikasan dohvat elemenata po koordinatama, moguće je napraviti novu implementaciju za skladištenje objekata koja koristi *quad-tree*²³. Treba napomenuti da nakon te promjene nije nužno mijenjati kod u drugim dijelovima programa, s obzirom da je sučelje ostalo isto – upravo je u tome prednost sučelja.

Kao što je već spomenuto, sklopovi na platnu predstavljeni su objektom sučelja *ISchemaComponent*, koje služi za dohvat i postavljanje imena instance sklopa, dohvat imena tipa sklopa, veličine sklopa na platnu, dohvat ulaza i izlaza sklopa, pinova sklopa (nožica na platnu na koje je moguće spojiti žice), i parametara sklopa. Parametri sklopa

²³ Quad-tree je struktura podataka koja se često koristi za skladištenje objekata razmještenih u dvodimenzionalnom prostoru, te koja nudi manju složenost dohvata objekta po koordinatama od npr. jednostruko povezane liste.

opisani su sučeljem *IParameterCollection* pomoću kojeg je moguće vršiti iteriranje, dodavanje, dohvat i brisanje parametara. Svaki je parametar predstavljen sučeljem *IParameter*, koje vraća ime parametra, tip parametra i vrijednost parametra. Primjer parametara su broj ulaza sklopa *I* ili kašnjenje digitalnog sklopa. U prvom slučaju je tip parametra cijeli broj, a u drugom slučaju je tip parametra vrijeme. Svaki parametar može imati pridružen objekt sučelja *IParameterConstraint* koji ograničava skup vrijednosti parametra, te objekt sučelja *IParameterEvent* koji u slučaju promjene vrijednosti parametra pokreće promjene na modelu sklopa. Tako će *IParameterEvent* vezan uz broj ulaza sklopa *I* nakon promjene vrijednosti parametara dodati na sklop *I* dodatne pinove.

ISchemaWireCollection je sučelje preko kojeg je moguće dohvaćati žice na platnu po imenu ili koordinatama, dodavati žice na platno i micati žice s platna. Svaka žica na platnu predstavljena je objektom sučelja *ISchemaWire*, preko kojeg je moguće dohvatiti i postaviti ime žice, dohvatiti listu svih segmenata žice, dodavati ili brisati segmente žice, te dohvatiti parametre žice opisane kao i ranije sučeljem *IParameterCollection*. Naposljetku, *ISchemaEntity* sadrži parametre vezane uz sam sklop koji se modelira – to su ulazi i izlazi, te ime tipa sklopa.

Što se tiče kontrolera, temeljno sučelje koje ga opisuje naziva se *ISchemaController*. Preko njega viši slojevi upućuju upite o modelu i zatražuju promjene nad modelom. Također, njegova je uloga obavijestiti sve registrirane poglede o eventualnim promjenama na modelu sheme, kako bi oni mogli pravovremeno osvježiti svoj sadržaj.

Svaku operaciju nad modelom moguće je ostvariti tako da se sučelju *ISchemaInfo* doda metoda koja obavlja pripadnu operaciju. Primjerice, za povezivanje žice na sklop moguće je napisati metodu koja kao parametar prima ime sklopa i ime žice, te u njenoj implementaciji spojiti žicu na taj sklop. Nedostatak ovakvog pristupa je dvojak. Najprije, svaki put kad želimo dodati neku novu operaciju nad modelom, npr. brisanje ili dodavanje žice, proširivanje žice, promjena imena sklopa, promjena nekog parametra sklopa ili žice itd., neophodno je implementaciju sučelja *ISchemaInfo* mijenjati – dakle, jednom napisani i provjereni odsječak koda, trebalo bi mijenjati. Dodatno, ovaj bi pristup zahtijevao da svaka implementacija sučelja *ISchemaInfo* implementira nanovo metodu za spajanje žice na sklop. Ako u nekom trenutku odlučimo da trenutna implementacija *ISchemaInfo* sučelja nije dovoljno efikasna, te krenemo pisati novu, u interesu nam je da ne ponavljamo sličan kod vezan uz spajanje žice na sklop, kad je tu operaciju moguće obaviti pomoću već postojećeg *ISchemaInfo* sučelja dohvaćanjem žice i sklopa, te dohvaćanjem pina na sklopu i pridruživanju žice pinu (operacija koje su već podržane u sučelju *ISchemaInfo*).

Zaključak gornjeg razmatranja je da operaciju za spajanje žice na sklop, kao i ostale složene operacije (one koje je moguće obaviti pomoću osnovnih operacija sučelja *ISchemaInfo*) treba smjestiti negdje drugdje – u nekoj drugoj klasi. Moguće je imati klasu koja sadrži metode za sve složenije operacije nad modelom, ali u tom slučaju ostaje prvi

problem. Bolje je rješenje svaku operaciju nad shemom uskladištiti u zasebnu klasu koja implementira neko opće sučelje za objekte koji obavljaju operacije nad modelom. Ovakvo ućahurivanje operacije u objekt je oblikovni obrazac koji se naziva komanda (*engl. command pattern*), i njegove su prednosti višestruke. Osim što rješava oba navedena problema, također omogućuje i jedinstven način obrade rezultata operacije, omogućuje jednostavno praćenje povijesti obavljenih komandi, te dodatno omogućuje jednostavnu izvedbu *undo/redo* mehanizma. Općenito, omogućuje jedinstven način serijalizacije komandi, međutim, u našem slučaju to nije bilo potrebno.

U slučaju uređivača shema, sučelje za obrazac komande je *ICommand*, i sadrži metodu *performCommand*, koja prima referencu na objekt sučelja *ISchemaInfo*, dakle referencu na sam model, i vraća objekt tipa *ICommandResponse*, koji sadrži rezultate obavljanja operacije. Svi objekti koji implementiraju ovo sučelje nazivaju se komande, i zamisao je da se pozivom njihove implementacije metode *performCommand* obave promjene na modelu koristeći sučelje *ISchemaInfo*, te se u objektu sučelja *ICommandResponse* vrate rezultati obavljanja operacije – da li je ona uspješna, a ako nije, poruka koja objašnjava zašto nije. Pritom, sučelje *ICommand* zahtijeva da operacije čiji je rezultat izvođenja neuspješan ne promijene ništa u modelu sheme.

Dodatno, sučelje *ICommand* sadrži metode *isUndoable* i *undoCommand*. Smisao ovih metoda je omogućiti *undo/redo* mehanizam. Ako je komanda reverzibilna, odnosno, moguće je poništiti posljedice izvođenja pripadne operacije, tad metoda *isUndoable* mora vratiti *true*, u protivnom vraća *false*. Ako je komanda reverzibilna, tad je definirana i njena metoda *undoCommand*, u protivnom će *undoCommand* baciti iznimku. Ugovor (*contract*) sučelja *ICommand* je sljedeći: ako je komanda reverzibilna, onda će nakon izvođenja metoda *performCommand* i *undoCommand* nad istim *ISchemaInfo* objektom tim redoslijedom od strane iste komande, *ISchemaInfo* objekt biti u istom stanju kao i prije izvođenja metoda. Svaka implementacija ovog sučelja mora ovo poštovati.

Vratimo se na sučelje *ISchemaController*. Ono sadrži metodu *send*, kojoj se predaje referenca na objekt sučelja *ICommand*. Ideja je da se predanom objektu pozove metoda *performCommand*, kojoj se preda referenca na model, te se na taj način obave promjene nad modelom. Metoda *performCommand* pritom vraća objekt tipa *ICommandResponse* koji se potom vraća iz metode *send*. Dodatno, *ISchemaController* sadrži metode *undo* i *redo*. Ideja ovih metoda je poništavanje prethodno obavljene operacije, pa opišimo kako one rade. Naime, neka implementacija *ISchemaController*-a trebala bi sadržavati dva stoga – zovimo ih *undo* i *redo* stog. Prilikom svakog poziva metode *send* dodatno se na vrh *undo* stoga stavlja komanda ako je ona izvedena uspješno, i ako je reverzibilna. Ako komanda nije reverzibilna, *undo* stog se prazni. Ako netko kasnije pozove metodu *undo* sučelja *ISchemaController*, najprije će se s vrha *undo* stoga skinuti jedna komanda, te će se pozvati njena metoda *undoCommand*, koja će vratiti model u stanje prije izvršavanja metode *performCommand* te komande. Dodatno, ta će se komanda staviti na *redo* stog. Ovo se

može ponoviti više puta, dok god *undo* stog nije prazan. Pozivom metode *redo* skida se jedna komanda s *redo* stoga, poziva njena metoda *performCommand* i stavlja na *undo* stog. *Redo* stog se prazni svakim pozivom metode *send*. Na ovaj je način pomoću oblikovnog obrasca komande ostvaren *undo/redo* mehanizam.

Zadaća je kontrolera da obavijesti sve poglede kad god se dogodi neka promjena na modelu. U tu svrhu *ISchemaController* ima metode *addListener* i *removeListener* za dodavanje i brisanje referenci na poglede. Neka implementacija ovog sučelja će tipično imati listu na kojoj se nalaze sve te reference. Pri uspješnom obavljanju neke komande, kontroler će obavijestiti sve registrirane poglede, nakon čega će oni moći osvježiti svoj sadržaj. Primjerice, promjenom položaja sklopa na platnu, platno (koje je pogled) će biti obaviješteno od strane kontrolera i ponovno se iscrtati. Ovaj oblikovni obrazac, kod kojeg se objekt registrira kod nekog drugog objekta kako bi bio obaviješten o promjenama kod tog objekta kada se one dogode, naziva se promatrač (*engl. observer pattern*).

Opišimo sada način funkcioniranja čitavog uređivača sheme. Pri pokretanju uređivača model se stvara ili deserijalizira iz datoteke, stvara se kontroler i pogledi (platno za shemu, popis digitalnih sklopova, popis svojstava izabranog digitalnog sklopa) koji se registriraju kod kontrolera metodom *addListener*. Nakon toga korisnik uređuje shemu, a na temelju svake njegove radnje šalje se kontroleru jedan objekt sučelja *ICommand*. Neka korisnik pokuša pomaknuti digitalni sklop s jednog mjesta na drugo. Stvara se objekt klase *MoveComponentCommand*, kojem se u konstruktoru predaju ime objekta i željene odredišne koordinate. Potom se taj objekt šalje kontroleru pozivom metode *send*. Kontroler poziva metodu *performCommand* od tog objekta i pamti povratni objekt sučelja *ICommandResponse*. Ako je u tom objektu zapisano da je operacija uspješna, kontroler stavlja taj objekt na *undo* stog za eventualno poništavanje kasnije, iterira kroz listu registriranih pogleda pozivajući njihove metode *propertyChange*, na što oni osvježavaju svoj sadržaj, te naposljetku vraća objekt sučelja *ICommandResponse* pozivatelju metode *send*, na temelju čega on saznaje je li operacija uspjela ili ne, i zašto.

5.3.3. Pogledi

Slijedeći pogledi su definirani (korišteni) u VHDLLab klijentskoj aplikaciji:

- ProjectExplorer
- Status History
- Error History
- Compilation
- Simulation

ProjectExplorer je pogled koji prikazuje sve projekte koje je korisnik kreirao te sve datoteke koje im pripadaju. Pretpostavljeni prikaz datoteka je hijerarhijski, pri čemu su datoteke koje koriste druge datoteke na vrhu hijerarhije. *ProjectExplorer* dodatno može prikazivati obrnuti hijerarhijski prikaz, kao i prikaz spljoštene (flat) hijerarhije, tj. bez prikaza međuovisnosti. Ovaj pogled također prikazuje ikone pored svake datoteke kao i za svaki projekt i to tako da svaki tip datoteke ima svoju ikonu. Implementacija koristi *JTree* komponentu koja je dio Java platforme.

Status History i *Error History* su prilično slični. Prvi prikazuje poruke sustava, dok drugi prikazuje poruke o greškama koje su se dogodile u sustavu. Oboje se u inicijalizaciji registriraju kao slušači *System Log* razredu i primaju obavijesti koje onda prikazuju korisniku. Dodatno se *Error History* ne može pokrenuti kada VHDLLab koristi normalan korisnik već isključivo dok se aplikacija razvija.

Compilation i *Simulation* pogledi su također vrlo slični. Prvo prikazuje poruke prevođenja kôda, dok drugi prikazuje poruke simulacije, tj. javljaju da je prevođenje/simulacija uspješno završena, odnosno javljaju poruke o grešci ukoliko dođe do pogreške (npr. pogrešno napisan VHDL kôd). Isto kao i *Status History*, i *Compilation* i *Simulation* pogledi se u inicijalizaciji registriraju kao slušači *System Log* razredu i primaju obavijesti koje onda prikazuju korisniku. *Compilation* pogled dodatno pruža mogućnost integracije s *TextEditor* uređivačem tako da korisnik samo dvaput klikne na poruku o grešci i *TextEditor* pokaže o kojoj se liniji radi.

5.3.4. Instalacija


Uvjet da korisnik obavi instalaciju klijentske aplikacije je da već ima instaliran *Java 6 JRE*²⁴. Nakon toga korisnik treba samo kliknuti određenu adresu u svojem web pregledniku. Obično je ta adresa prigodno označena prepoznatljivom "*Launch*" slikom.

Slika 5.16 prikazuje uputu za instaliranje klijentske aplikacije kakva već postoji za kolegij Digitalna logika na Fakultetu elektrotehnike i računarstva.

VHDLLab

Upute za instalaciju

Kako biste instalirali VHDLLab, potrebno je obaviti jednostavnu instalaciju opisanu u nastavku. Animirani prikaz intalacije dostupan je [ovdje](#).

1. Instalirati Java Runtime Environment (JRE) 6. Do JRE-a možete doći [ovdje](#).
2. Pokrenuti aplikaciju po prvi puta, klikom na .


Korak 1: instalacija JRE 6

Otidite na stranice [proizvođača](#), skinite i instalirajte Java Runtime Environment (JRE). Kako bi sustav VHDLLab radio ispravno, potrebno je skinuti JRE (barem) verzije 6. Detaljne upute za instalaciju na operacijskim sustavima Windows / Linux dostupne su na stranicama Sun-a. Najjednostavnije je skinuti samoraspakiravajuće binarne datoteke (exe / bin). Po instalaciji provjerite sljedeće:

- Je li postavljena varijabla okruženja JAVA_HOME?
- Ako nije, postavite je tako da pokazuje na punu stazu do direktorija u kojem se nalazi JRE.
- Je li modificirana varijabla okruženja PATH tako da sadrži \$JAVA_HOME/bin direktorij?

Ako nije, modificirajte je.

Korak 2: pokretanja VHDLLab aplikacije po prvi puta

VHDLLab se distribuira pomoću tehnologije Java Web Start. Stoga je dovoljno kliknuti na  kako bi se aplikacija skinula s Interneta i pokrenula. Na operacijskom sustavu MS Windows tijekom pokretanja bit će upitani želite li stvoriti *shortcut* na *Desktopu*. Ako odgovorite potvrdno, kasnije ćete aplikaciju moći pokretati direktno dvoklikom na *shortcut*, bez potrebe za pokretanjem preglednika. U suprotnom, VHDLLab uvijek možete pokrenuti preko preglednika klikom na prethodni link.

Nakon pokretanja sustava, bit će upitani za vaše korisničko ime i zaporku. Koristite isto ime i zaporku kao i za sustav *Nescume*.

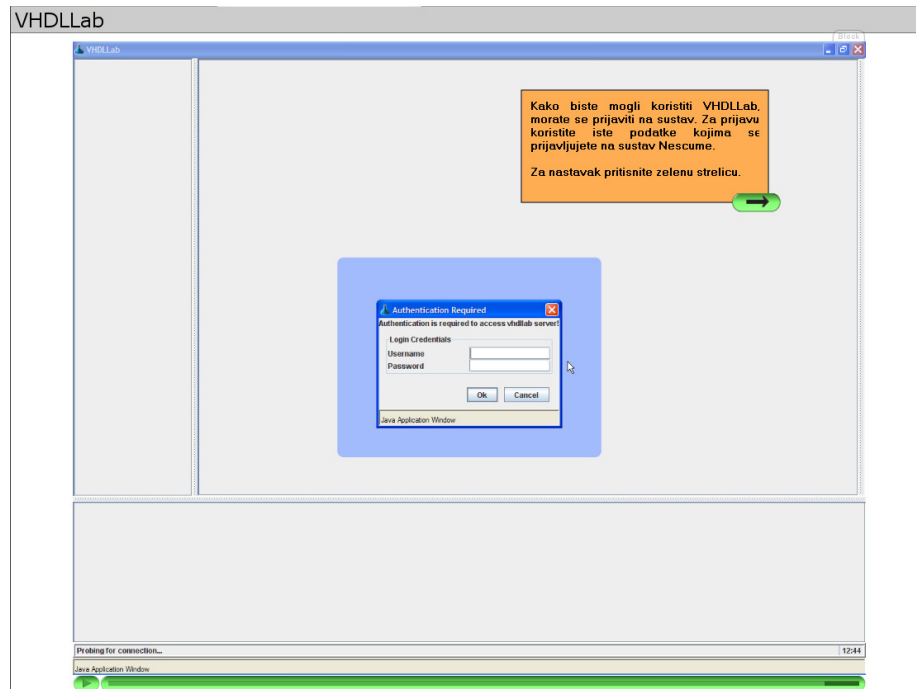
Slika 5.16 Postojeća uputa za instaliranje klijentske aplikacije

Da se omogući takva jednostavna instalacijska procedura, klijentska aplikacija napravljena je uporabom tehnologije JNLP (*engl. Java Network Launching Protocol*), što uključuje pisanje samo jedne datoteke XML formata. Klikom na određenu adresu (*launch*) otvorit će se Java Web Start aplikacija koja će automatski preuzeti izvršni kôd klijentske aplikacije, te svih ostalih potrebnih biblioteka. Nakon što preuzme sve potrebno pokrenuti će se klijentska aplikacija i čim se korisnik autentificira moći će započeti s radom. U konačnici čitava instalacijska procedura svedena je na jedan klik iz web preglednika.

Korisnik ima mogućnosti postavljanja prečaca (*engl. shortcut*) na svoju radnu površinu (*engl. desktop*), te ukoliko to izabere, može klijentsku aplikaciju ubuduće pokretati preko tog prečaca. Svaki put kada se pokrene klijentska aplikacija Java Web Start će pogledati da li ima kakvih poboljšanja (novih verzija) te ako ima, automatski preuzeti noviju verziju i nju pokrenuti. Na taj se način osigurava da korisnik uvijek ima najnoviju i najbolju verziju VHDLLab klijentske aplikacije.

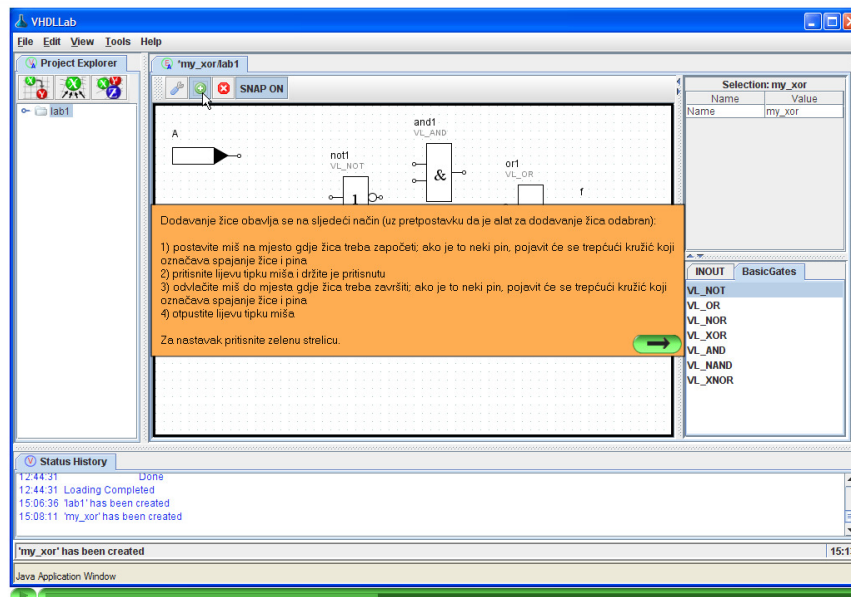
²⁴ JRE (*engl. Java Runtime Edition*) je minimalno potrebno za pokretanje bilo kakve aplikacije pisane u Java programskom jeziku

Kako bi se dodatno olakšalo upoznavanje sa sustavom VHDLLab, na raspolaganju su animirani prikazi instalacije (Slika 5.17), te izrade jedne laboratorijske vježbe uporabom schematic uređivača (Slika 5.18).



Slika 5.17 Kraj instalacijske procedure i prijava na sustav

Digitalna logika: 1. laboratorijska vježba



Slika 5.18 Animirani prikaz izrade laboratorijske vježbe

6. Zaključak

Sustav VHDLLab rješava niz problema vezanih uz provođenje laboratorijskih vježbi iz Digitalne logike i srodnih kolegija na Fakultetu elektrotehnike i računarstva. Sustav je usmjeren na obrazovanje studenta i u potpunosti je ispunjen niz zahtjeva obrazovnog karaktera koji su postavljeni kad smo se odlučili za njegovu izradu. Osim olakšanog upoznavanja s osnovnim pojmovima digitalne logike, modeliranjem digitalnih sklopova i provjerom rada istih, sustav pruža mogućnost eksperimentiranja i rada kod kuće. Studenti su dobili kvalitetan programski sustav kojim kod kuće mogu raditi obavljati laboratorijske vježbe i učiti koncepte modeliranja digitalnih sklopova bez velike muke oko nabave i instalacije profesionalnih alata. Izrazita prednost sustava su kvalitetni uređivači za sheme sklopova i dijagrame stanja digitalnih automata, jer studentu jasno pokazuju povezanost jezika VHDL s modeliranjem digitalnih sklopova. Naposljetku, kao jedna od značajnih prednosti sustava pokazala se prenosivost i jednostavnost instalacije, te lakoća i intuitivnost uporabe.

Strukturno modeliranje sklopova pomoću uređivača shema omogućuje brzo i pregledno modeliranje čak i u slučaju kompleksnih sklopova. Iako *Web/SE* sustav nudi uređivač shema, pokazalo se da je uređivač shema sustavu VHDLLab praktičniji i jednostavniji za uporabu. Ova je tvrdnja poduprta činjenicom da smo kao uzor za VHDLLab uzeli *Web/SE* i ispravili određene stvari koje smo smatrali nefunkcionalnima.

Uređivač dijagrama stanja digitalnog automata osmišljen je u potpunosti u sklopu ovog projekta. Ostali alati za modeliranje sustava pomoću jezika VHDL nemaju uređivače tog tipa te samim time ovu opciju dizajniranja smatramo velikom prednošću sustava VHDLLab. Ovaj se način ponašajnog opisivanja sekvencijskih sklopova pokazao kao najjednostavniji i najbrži uz najmanje pogrešaka pri dizajnu.

Uređivač koda i uređivač ispitnih sklopova također ispunjavaju uvjete koje smo zadali. U budućnosti se planira nadogradnja tih uređivača dodatnom funkcionalnošću i samim time dodatno poboljšanje kvalitete aplikacije.

S programerskog stajališta napomenuli bismo da se programsko ostvarenje sustava sastoji od 589 razreda, 83 sučelja i 4603 metode koji zajedno sačinjavaju 47860 linija koda. Projekt se sastoji od 861 datoteke, a samo izvorni kod velik je 3.4 megabajta.

Najveći pokazatelj uspjeha projekta je anketa održana među studentima koji su ove akademske godine u sklopu kolegija Digitalna logika eksperimentalno odlučili koristiti sustav VHDLLab. VHDLLab odabralo je oko 45 studenata, a od toga ih je točno 30 ispunilo anketu. Rezultati ankete su veoma pozitivni i može se očekivati skora primjena sustava na sve studente prve godine. Također, treba napomenuti da je anketa bila na dobrovoljnoj bazi i anonimna, te da je pisana nakon što su studenti završili s vježbama i polaganjem predmeta.

Od 30 studenata koji su ispunili anketu samo je jedan izjavio da bi umjesto VHDLLab-a radije koristio neki komercijalni program. Obrazloženje je bilo: „WebISE je ipak standardiziran i koristi se u industriji, dok je VHDLLab ipak lakše shvatiti i odraditi od kuće“. Ovo nam ide u prilog, s obzirom da je sustav VHDLLab ionako predviđen prvenstveno za obrazovne svrhe, i služi kako bi studentima olakšao prvi susret sa modeliranjem digitalnih sklopova. Anketni listić i kratka analiza ankete dani su u dodatku A.

Modularnost izvedbe ove aplikacije te njena slojevitost čine dodavanje novih funkcija jednostavnim. U bliskoj se budućnosti planira dodati jednostavniju, edukacijski primjerenu, inačicu alata za sintezu sklopova u tehnologiji FPGA. Isto tako, sustavu se može dodati čitav niz funkcionalnosti vezan uz automatizirano generiranje laboratorijskih vježbi za pojedine studente, kao i njihovu provjeru, čime se smanjuje opterećenje na fakultetsko osoblje, a ujedno i poboljšava kvaliteta studija.

Uz male promjene može služiti kao razvojna okolina za Verilog ili neki drugi jezik nevezan za modeliranje sklopova. Smatramo da se alati kao uređivač automata mogu koristiti i za generiranje C++ ili Java koda za strojeve s konačnim brojem stanja. Naša je želja aktivirati mlađe studente i polaznike Java tečaja da se bave ovim područjem i razvijaju ovaj i slične sustave.

Sve u svemu, smatramo da sustav VHDLLab ima veliki značaj za savladavanje tema iz područja kolegija Digitalne logike, ali i za Bolonjski studij uopće. Ublažujući krivulju učenja i omogućujući studentima samostalni rad kod kuće, VHDLLab pruža bolje prilike za obrazovanje studenata, olakšavajući istovremeno posao voditeljima laboratorijskih vježbi. Uz proširenja planirana u budućnosti, VHDLLab će dodatno individualizirati pristup studentima, što je i jedan od glavnih ciljeva Bolonjskog studija.

VHDLLab je sustav koji obrazovanju pruža mnoge prednosti, i njegov je značaj teško zanemariti.

7. Literatura

- [1] M. Čupić, *Digitalna elektronika i Digitalna logika – zbirka riješenih zadataka*. KIGEN, Zagreb, 2006.
- [2] U. Peruško, *Digitalna elektronika – logičko i električko projektiranje*, III. prošireno izdanje. Školska Knjiga, Zagreb, 1996.
- [3] S. Brown, Z. Vranešić, *Fundamentals of digital logic with VHDL design*. McGraw-Hill Companies, Inc., 2000.
- [4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [5] S. Srbljić, *Jezični procesori 1 – Uvod u teoriju formalnih jezika, automata i gramatika*, Element, Zagreb, 2004.
- [6] D. Žubrinić, *Diskretna matematika*, Element, Zagreb, 2002.
- [7] B. Eckel, *Thinking in Java*. Prentice-Hall, December 2002.
- [8] S. McConnell, *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press, June 2004.
- [9] R. Sedgewick, *Algorithms in Java, Parts 1-4*, Addison-Wesley, 2002.
- [10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, MIT Press, 2001.
- [11] B. Collins-Sussman, B. W. Fitzpatrick, C. M. Pilato, *Version Control with Subversion*, O'Reilly Media, 2004.
- [12] H. Maruyama, K. Tamura, N. Uramoto, *XML and Java: Developing Web Applications*, Addison-Wesley, 1999.
- [13] E. Burnette, *Eclipse IDE Pocket Guide*, O'Reilly Media, 2005.
- [14] K. Beck, *Test Driven Development: By Example*, Addison-Wesley, 2002.
- [15] T. Downey, *Web Development with Java: Using Hibernate, JSPs and Servlets*, Springer, 2007.
- [16] C. Bauer, *Hibernate in Action*, Manning Publications, 2004.
- [17] B. Dalbelo Bašić, *Bilješke s predavanja: Inteligentni sustavi*, Fakultet elektrotehnike i računarstva, Zagreb, 2001.
- [18] Java dokumentacija: <http://java.sun.com/javase/6/docs/>
- [19] „Java tutorial“: <http://java.sun.com/docs/books/tutorial/>
- [20] Internet stranice projekta Ant: <http://ant.apache.org/>
- [21] Internet stranice projekta Tomcat: <http://tomcat.apache.org/>
- [22] Internet stranice projekta Hibernate: <http://www.hibernate.org/>
- [23] Internet stranice simulatora GHDL: <http://ghdl.free.fr/>
- [24] Internet stranice proizvoda Xilinx ISE: <http://www.xilinx.com/ise/>

8. Dodatak A: Anketa

Anketa: sustav VHDLLab

1. Što mislite o mogućnosti izradi laboratorijskih vježbi kod kuće?

Jako sam zadovoljna/zadovoljan	1	2	3	4	5	Apsolutno nezadovoljna/nezadovoljan
Vrlo korisno	1	2	3	4	5	Apsolutno beskorisno

2. Koliko ste vremena prosječno po vježbi radili s VHDLLabom?

3. Na skali od 1 do 5, koliko je teško napraviti instalaciju sustava VHDLLab?

Vrlo lagano	1	2	3	4	5	Vrlo teško
-------------	---	---	---	---	---	------------

4. Ocijenite sučelje prema korisniku sustava VHDLLab.

Vrlo lagano za korištenje	1	2	3	4	5	Vrlo teško za korištenje
Intuitivno	1	2	3	4	5	Zbunjujuće

5. Pitanja vezana uz funkcionalnost sustava.

Jeste li ikada pogledali kako izgleda VHDL kod ispitnog sklopa koji nacrtate?	JESAM	NISAM
Smatrate li da je ta mogućnost korisna s edukacijskog aspekta?	DA	NE
Jeste li ikada pogledali kako izgleda VHDL kod sklopa kojeg nacrtate u obliku sheme?	JESAM	NISAM
Smatrate li da je ta mogućnost korisna s edukacijskog aspekta?	DA	NE
Jeste li koristili mogućnost da automat definirate crtanjem?	JESAM	NISAM
Jeste li ikada pogledali kako izgleda VHDL kod sklopa kojeg tako definirate?	JESAM	NISAM
Smatrate li da je ta mogućnost korisna s edukacijskog aspekta?	DA	NE

6. Kada bi kolegij imao još 6 laboratorijskih vježbi, i kada bi Vam se ponovno ponudila mogućnost izbora hoćete li nastaviti raditi u VHDLLabu ili u alatu Xilinx ISE WebPACK, što biste odabrali, i zašto?

Xilinx ISE WebPACK		1		2		VHDLLab
--------------------	--	---	--	---	--	---------

Odgovor:

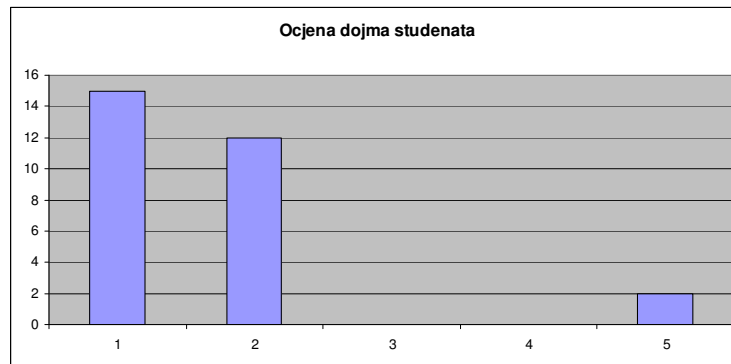
7. Što Vam se najviše nije dopalo u vezi sustava VHDLLab?

8. Što Vam se je najviše dopalo u vezi sustava VHDLLab?

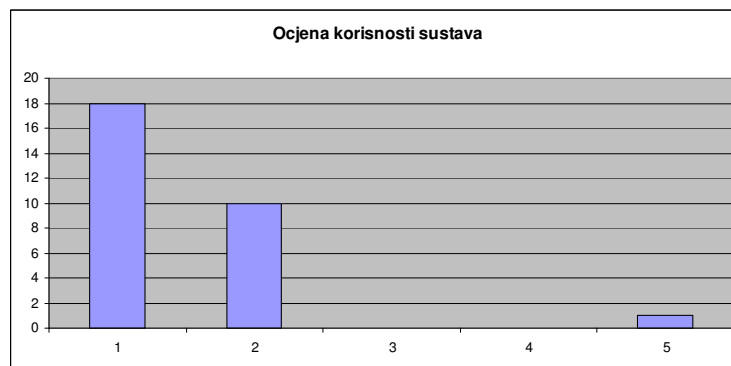
9. Imate li kakvu primjedbu ili prijedlog za kraj?

Slika 8.1 Anketni obrazac

1. Što mislite o mogućnosti izrade laboratorijske vježbe od kuće?

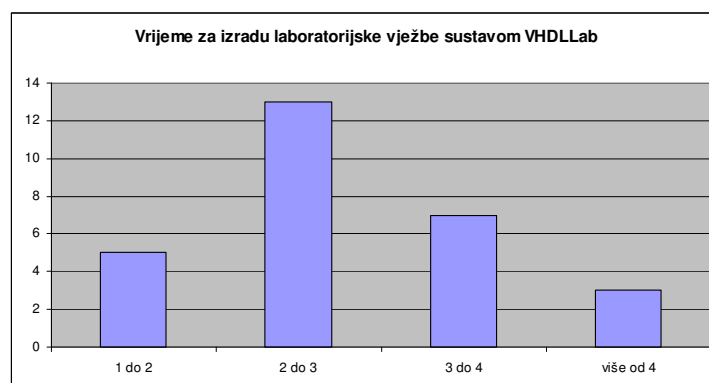


Graf 8.1 Dojam koliko je student zadovoljan. 1-jako zadovoljan, 5-apsolutno nezadovoljan.
Srednja vrijednost 1.6897



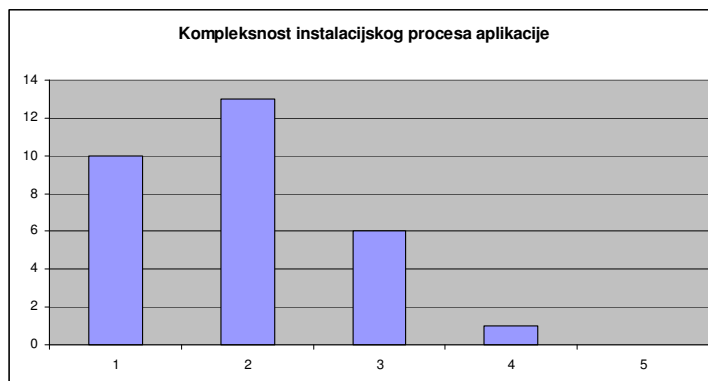
Graf 8.2 Koliko studenti smatraju sustav korisnim. 1- jako koristan, 5-apsolutno beskoristan.
Srednja vrijednost 1.4828

2. Koliko vam je u prosjeku trebalo vremena za izradu vježbe?



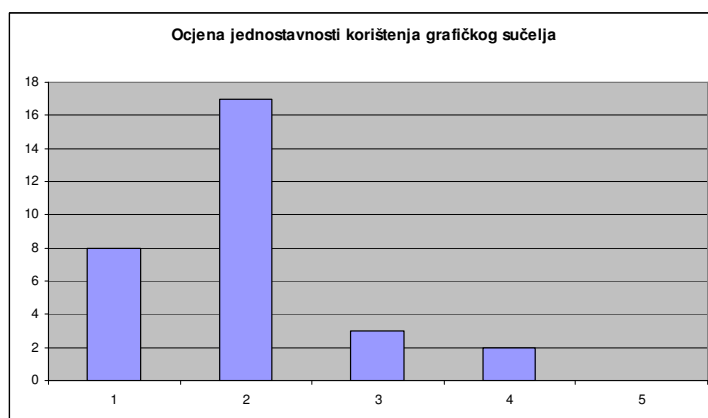
Graf 8.3 Procjena vremena potrebnog za izradu laboratorijske vježbe u satima

3. Koliko je teško napraviti instalaciju sustava VHDLLab?

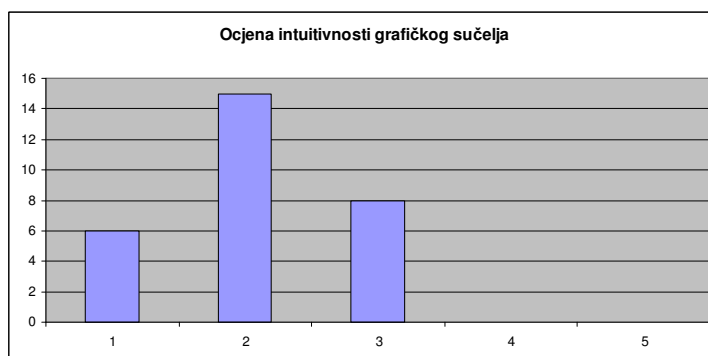


Graf 8.4 Procjena težine instalacijskog procesa. 1-vrlo lagano, 5-vrlo teško. Srednja vrijednost 1.9333

4. Ocjenite sučelje prema korisniku sustava VHDLLab.

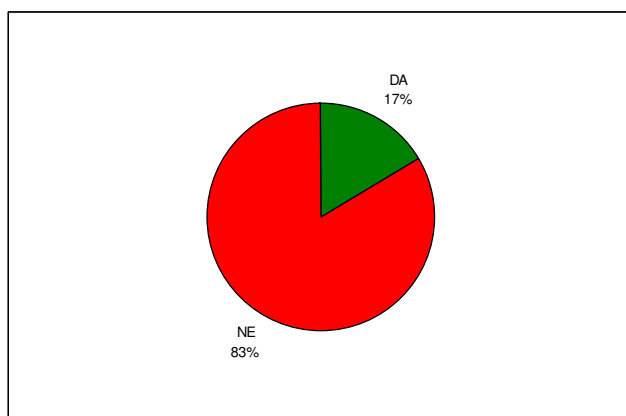


Graf 8.5 Jednostavnost za korištenje. 1-vrlo jednostavno, 5-vrlo teško. Srednja vrijednost 1.9667

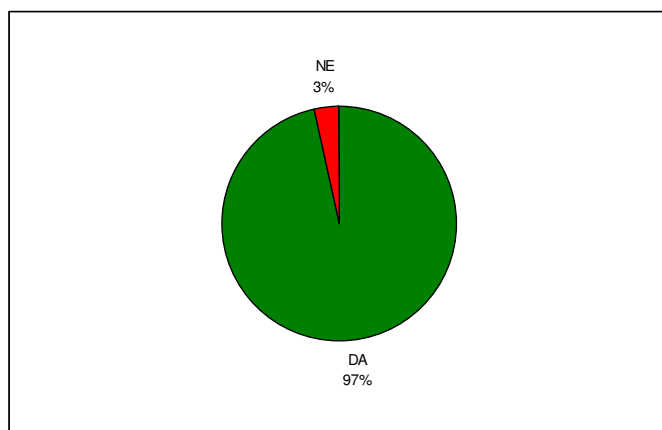


Graf 8.6 Intuitivnost sučelja. 1-vrlo intuitivno, 5-zbunjujuće. Srednja vrijednost 2.069

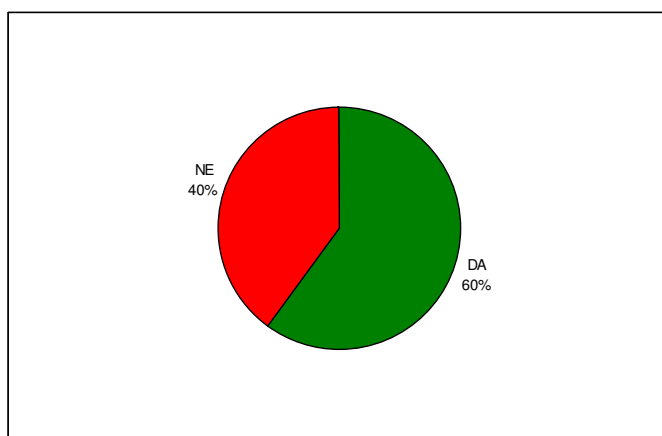
5. Pitanja vezana uz funkcionalnost sustava:



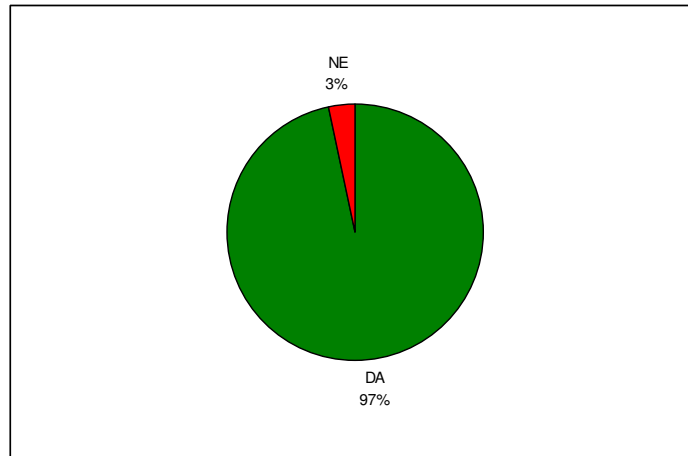
Graf 8.7 Jeste li ikada pogledali kako izgleda VHDL kod ispitnog sklopa?



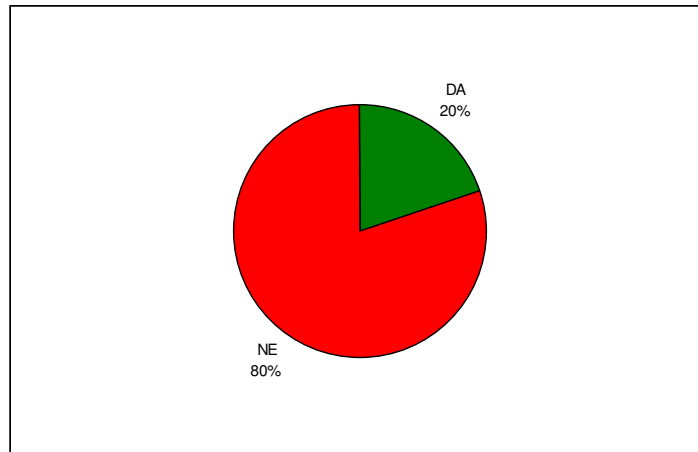
Graf 8.8 Smatrate li da je ta mogućnost korisna s edukacijskog aspekta?



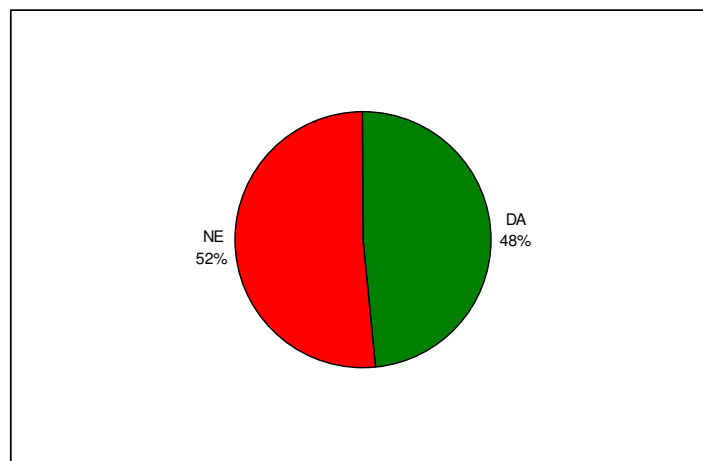
Graf 8.9 Jeste li ikad pogledali kako izgleda VHDL kod sklopa kojeg ste nacrtali u obliku sheme?



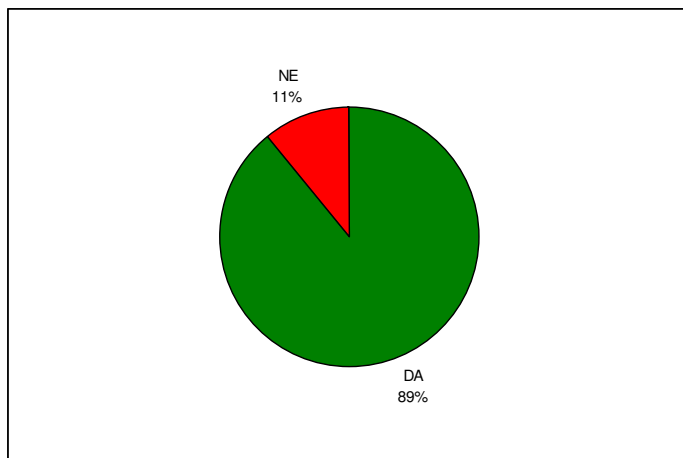
Graf 8.10 Smatrate li da je ta mogućnost korisna s edukacijskog aspekta?



Graf 8.11 Jeste li koristili mogućnost da automat definirate crtanjem?

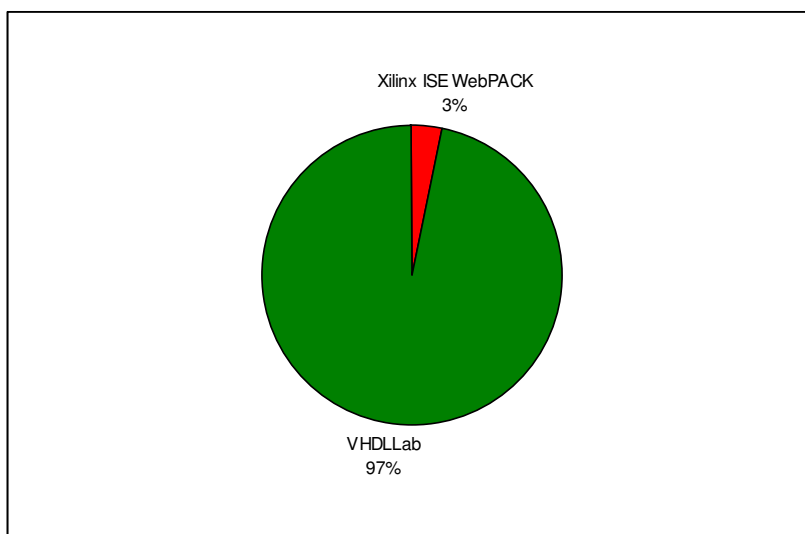


Graf 8.12 Jeste li ikad pogledali kako izgleda VHDL kod sklopa kojeg tako definirate?



Graf 8.13 Smatrate li da je ta mogućnost korisna s edukacijskog aspekta?

6. Kada bi kolegij imao još 6 laboratorijskih vježbi, i kada bi vam se ponovo nudila mogućnost izbora hoćete li nastaviti raditi u VHDLLabu ili u alatu Xilinx ISE Web PACK, što biste odabrali i zašto?



Graf 8.14 U čemu bi studenti htjeli nastaviti raditi laboratorijske vježbe

9. Dodatak B: Indeks slika, tablica i primjera

9.1. Indeks slika

Slika 2.1 Jednostavni električni krug (I).....	4
Slika 2.2 Jednostavni električni krug (ILI).....	5
Slika 2.3 Jednostavni logički sklopovi (1).....	6
Slika 2.4 Jednostavni logički sklopovi (2).....	7
Slika 2.5 Simbol za dekodner	9
Slika 2.6 Simbol za multipleksor	10
Slika 2.7 ROM 4×4	11
Slika 2.8 Blok FPGA sklopa.....	13
Slika 2.9 Primjer FPGA sklopa.....	13
Slika 2.10 Bistabil izveden invertorima	14
Slika 2.11 Simboli bistabila	16
Slika 2.12 Opća struktura strojeva stanja	17
Slika 2.13 Sučelje programa WebISE	27
Slika 2.14 Sučelje programa ModelSim	28
Slika 3.1 Glavni izbornik.....	34
Slika 3.2 Preglednik projekata	36
Slika 3.3 Lista prethodnih radnji.....	36
Slika 3.4 Uređivač koda.....	37
Slika 3.5 Uređivač automata.....	38
Slika 3.6 Uređivač sheme digitalnog sklopa.....	40
Slika 3.7 Uređivač ispitnih sklopova.....	42
Slika 3.8 Preglednik simulacija.....	43
Slika 4.3 Stvaranje novog projekta	48
Slika 4.4 Dijalog za opis sučelja sklopa	48
Slika 4.5 Opis brojila u uređivaču koda	49
Slika 4.6 Odabir sklopa za izradu ispitnog sklopa	50
Slika 4.7 Ispitni signali za brojilo	50
Slika 4.8 VHDL kod ispitnog sklopa	51
Slika 4.9 Pokretanje simulacija sklopa.....	51
Slika 4.10 Rezultati simulacije brojila	52
Slika 4.11 Pohrana rezultata simulacije.....	52
Slika 4.12 Ponašajni opis dekodera.....	53
Slika 4.13 Unos opisa sučelja <i>timer-a</i>	54
Slika 4.14 Inicijalno stvorena shema <i>timer-a</i>	54
Slika 4.15 Shema <i>timer-a</i>	55
Slika 4.16 VHDL kod stvoren na temelju sheme sklopa.....	56
Slika 4.17 Dijalog za odabir vrste automata	57
Slika 4.18 Stanja automata	57
Slika 4.19 Automat sa stanjima i prijelazima	58
Slika 4.20 VHDL opis automata	59
Slika 4.21 Shema upravljača semafora.....	60
Slika 4.22 Simulacija upravljača semafora.....	60

Slika 5.15 Sučelje uređivača automata	84
Slika 5.16 Postojeća uputa za instaliranje klijentske aplikacije.....	91
Slika 5.17 Kraj instalacijske procedure i prijava na sustav	92
Slika 5.18 Animirani prikaz izrade laboratorijske vježbe	92
Slika 8.1 Anketni obrazac.....	96

9.2. Indeks tablica

Tablica 2.1 Sklop <i>I</i>	5
Tablica 2.2 Sklop <i>ILI</i>	5
Tablica 2.3 Sklop <i>NE</i>	6
Tablica 2.4 Sklop isključivo <i>ILI</i>	7
2.5 Tablica dekodera 2/4	9
2.6 Tablica dekodera 2/4	10
2.7 Tablica SR-bistabila	15
2.8 Tablica JK-bistabila.....	15
2.9 Tablica T-bistabila.....	16
2.10 Tablica D-bistabila	16
Tablica 4.1 Stanja upravljača semafora.....	46
Tablica 5.1 Opis podataka koje sadrži model Project	65
Tablica 5.2: Opis podataka koje sadrži model File	66
Tablica 5.3: Opis podataka koje sadrži model UserFile.....	66
Tablica 5.4: Definirane metode ProjectDAO sučelja.....	68
Tablica 5.5: Definirane metode FileDAO sučelja.....	69
Tablica 5.6: Definirane metode UserFileDAO sučelja	69
Tablica 5.7: Polja objekta Method	73
Tablica 5.8: Definirane metode IEditor sučelja	81

9.3. Indeks primjera

Primjer 2.1 Sučelje multipleksora 2 na 1.....	20
Primjer 2.2 Skica architecture bloka	20
Primjer 2.3 Sintaksa process bloka	21
Primjer 2.4 Implementacija multipleksora 2 na 1	21
Primjer 2.5 Sintaksa naredbe za stvaranje komponenti.....	22
Primjer 2.6 Opis sučelja za sklop <i>I</i>	22
Primjer 2.7 Opis sučelja za D-bistabil.....	23
Primjer 2.8 D-bistabil s ulazom za omogućavanje.....	23
Primjer 2.9 VHDL model za dekodera 2/4.....	24
Primjer 2.10 Opis Mooreova digitalnog automata.....	26
Primjer 2.11 VHDL model „Hello world!“	29
Primjer 2.12 Ponašajni opis bistabila jezikom Verilog	30
Primjer 5.1 Primjer korištenja VHDLLab api-a sa strane korisničke aplikacije	73

10. Sažetak

Miro Bezjak, Davor Delač, Aleksandar Prokopec, *VHDLLab, Obrazovni programski sustav za modeliranje i simuliranje digitalnih sklopova*

Ovaj rad opisuje programski sustav VHDLLab za modeliranje i simuliranje digitalnih sklopova primijenjen na Bolonjskom studiju na Fakultetu elektrotehnike i računarstva u Zagrebu. Programski sustav je napravljen u skladu s ciljevima i težnjama Bolonjskog procesa, i služi za unaprijeđenje kvalitete obrazovanja. Primjenjiv je na područje digitalne elektronike i digitalne logike. Osmišljen je kako bi ublažio krivulju učenja studentu koji se prvi put susreće s temama poput digitalnog sklopa, njegovog modeliranja i simuliranja.

Sustav VHDLLab je uspješno primijenjen na kolegiju Digitalna logika na prvom semestru Bolonjskog studija Fakulteta elektrotehnike i računarstva (akademska godina 2007./2008.). U okviru rada obrađeni su i dojmovi i iskustva studenata koji su koristili sustav. Kvalitetu sustava najbolje opisuju njihove pozitivne kritike.

Sustav je napravljen slojevito poštujući opće prihvaćene programske modele i arhitekture, pa je u potpunosti proširiv s novim mogućnostima i usklađen s postojećim sustavima. Ovo omogućava proširenja u budućnosti, kao i mogućnost njegove uporabe na drugim kolegijima.

Ključne riječi: VHDL, IDE, simulacija, modeliranje, digitalni sklop

11. Summary

Miro Bezjak, Davor Delač, Aleksandar Prokopec, *VHDLLab, Educational software for modelling and simulation of digital circuits*

This paper describes the VHDLLab system used for modelling and simulation of digital circuits. This system is used on Bologna studies on the Faculty of Electrotechnics and Computer Science in Zagreb. The system is made according to goals and aspirations of the Bologna process, such as upgrading the quality of education. It is applicable to the field of digital electronics and digital logic. It is devised to soften the learning curve of students who are being introduced to topics such as digital circuit, modelling and simulation.

VHDLLab system has been successfully applied to the course Digital logic on the first semester of the Bologna study of Faculty of Electrotechnics and Computer Science (academic year 2007./2008.). Impressions and experiences of students who used this system have been described within this paper. The quality of the system is best described by their positive critics.

The system has been made in layers abiding the commonly accepted programming models and architectures, and is therefore completely expandable with new abilities and compatible with existing systems. This allows expansions in future, as well as the possibility of use on other courses.

Keywords: VHDL, IDE, simulation, modelling, digital circuit