Kratki uvod u jezik VHDL

Marko Čupić

12. studenog 2014.

Sadržaj

Predgovor

Ova skripta služi kao popratni materijal laboratorijskim vježbama iz kolegija Digitalna logika. U skripti je dan pregled podskupa jezika VHDL koji se koristi u okviru laboratorijskih vježbi. Ovaj dokument nije konačna verzija skripte (materijal je još u izradi i postupno će se dopunjavati).

Pregledao: Marko Zec

Poglavlje 1

Uvod

Povećanjem složenosti digitalnog sklopovlja pojavila se potreba za omogućavanjem njegovog formalnog opisivanja kako bi se omogućilo obavljanje simulacije funkcionalnosti prije no što se sklopovlje pošalje u proizvodni proces. Ustanove li se pogreške prilikom simulacije, puno je jednostavnije i brže (i *jeftinije*) takvu pogrešku ispraviti.

Daljnjim razvojem tehnologije te razvojem programirljivih sklopova pojavila se još jedna zgodna primjena formalno opisanog sklopovlja: razvijeni su računalom potpomognuti alati koji na temelju formalnog opisa digitalnog sklopovlja mogu programirati prigramirljive sklopove kako bi se oni ponašali u skladu s tim formalnim opisom.

Za potrebe formalnog opisivanja digitalnog sklopovlja razvijeno je mnoštvo jezika za opis sklopovlja. Danas su u najširoj uporabi dva:

- Verilog te
- VHDL.

U okviru kolegija *Digitalna logika* koristit ćemo podskup jezika VHDL koji ćemo opisati u nastavku.

VHDL je skraćenica od *VHSIC Hardware Description Language*, gdje je prvi dio kratica od *Very High Speed Integrated Circuit*. Prevedemo li ovo na hrvatski, VHDL stoji za *Jezik za opis integriranih sklopova vrlo visokih brzina*, što nas, naravno, ne treba sprječavati da ga koristimo i za opisivanje jednostavnijih i sporijih sklopova. Ponekad se u šali kaže i da je VHDL zapravo kratica od *Very Hard and Difficult Language* – no kroz ovaj tekst pokušat ćemo vas uvjeriti da to nije tako.

Poglavlje 2

Načini opisivanja sklopova

U okviru ovog poglavlja upoznat ćemo se s osnovnim načinima opisivanja digitalnog sklopovlja jezikom VHDL. Ovisno što znamo o digitalnom sklopu, opisat ćemo ga ili definiranjem njegove funkcije ili definiranjem njegove građe koja se oslanja na uporabu jednostavnijih sklopova. Stoga ćemo razlikovati dvije vrste opisa, odnosno dva modela sklopa.

Model toka podataka (engl. *dataflow model*) je model sklopa kod kojega pišemo naredbe koje kombiniraju ulazne signale (i eventualno pomoćne signale) uporabom naredbi dodjeljivanja vrijednosti signalima i tako određuju vrijednosti izlaza sklopa. Arhitektura kod ovog modela sastoji se od jedne ili više naredbi pridruživanja vrijednosti signalu.

Strukturni model (engl. *structural model*) je model sklopa koji sklop opisuje pozvajući se na njegovu građu te način na koji su jednostavnije komponente od kojih je sklop izgrađen povezane s ulazima, povezane međusobno te povezane s izlazima modeliranog sklopa. Arhitektura kod ovog modela sastoji se od niza naredbi koje predstavljaju stvaranje primjeraka korištenih komponenata i njihovo međusobno povezivanje.

Funkcijski model (zovemo ga još i **ponašajni model**, engl. *behavioral model*) je model sklopa kod kojega funkcionalnost sklopa najčešće ne opisujemo oslanjajući se logičke operatore koji bi oslikavali način na koji se ulazi transformiraju u izlaze već pišemo algoritam koji nam ne govori kako bi sklop trebalo sintetizirati već samo navodi kako se ulazi preslikavaju u izlaze. Ovo je najbliže što možemo prići klasičnom modelu "crne kutije" kod kojega ni na koji način ne opisujemo građu sklopa već samo njegovu funkcionalnost, koristeći proceduralno programiranje, naredbe poput if, case, for i slično. Arhitektura kod ovog modela sastoji se od jednog ili više blokova process.

2.1 Opis jednog sklopa

Prilikom modeliranja digitalnog sklopa pisat ćemo odgovarajući opis jezikom VHDL. Iako to nije nužno, dobra je praksa opis svakog sklopa staviti u zasebnu datoteku. Prilikom rada sa sustavom VHDLLab2, sam će nas sustav tjerati da pišemo upravo takve opise: unutar jednog projekta, za svaki ćemo sklop morati napraviti novi model koji će biti pohranjen u vlastitu datoteku.

Sam opis sklopa jezikom VHDL sastojat će se od sljedećih dijelova.

1. *Deklaracija korištenih biblioteka i paketa.* Svaki opis započet će navođenjem biblioteka i paketa iz tih biblioteka koje ćemo koristiti u opisu. Biblioteke se uključuju uporabom ključne riječi

library (redak 1 u primjeru u nastavku) a pojedini paketi uporabom ključne riječi use (redak 2 u primjeru u nastavku).

Jedna biblioteka može se sastojati od više paketa, a svaki paket može nuditi definiciju tipova podataka, funkcija i sličnoga. Želimo li iz nekog paketa uključiti sve što je u njemu definirano (kako bismo sve mogli koristiti za opisivanje sklopova), u naredbi kojom se paket uključuje na kraju ćemo napisati .all.

Cjeloviti primjer prikazan je u nastavku. Uočimo da svaka od naredbi završava znakom točka-zarez.

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
```

2. *Deklaracija sučelja sklopa.* Modelirani sklop, gledan izvana, prema svojoj se okolini predstavlja kao crna kutija koja ima određen broj ulaza te određen broj izlaza. Kroz svaki od ulaza odnosno izlaza sklop prima/šalje signale određenog tipa. Definiranjem sučelja sklopa korisnik mora ponuditi detaljan opis svih ulaza i izlaza.

   Definicija sučelja započinje ključnom riječi ENTITY nakon čega slijedi naziv sklopa čije se sučelje modelira, ključne riječi IS PORT(, definicije ulaza i izlaza, potom); i konačno ključna riječ END, naziv modeliranog sklopa pa znak točka-zarez. Svaki od ulaza odnosno izlaza naziva se jednim *portom* sklopa. Definicija porta započinje navođenjem naziva porta, slijedi dvotočka, ključna riječ IN (ako je port ulaz) odnosno OUT (ako je port izlaz), naziv programskog tipa koji opisuje vrstu signala koji se prima/šalje kroz taj port te na kraju znak točka-zarez (ali samo ako to nije definicija posljednjeg porta; ako je, znak točka-zarez ne smije se navesti).

   Pretpostavimo da modeliramo sučelje sklopa koji smo nazvali *SklopS1* i koji ima tri ulaza (A, B i C) te dva izlaza (f i g). Definicija sučelja takvog sklopa bila bi sljedeća.

```
1 ENTITY sklopS1 IS PORT (
2   a: IN std_logic;
3   b: IN std_logic;
4   c: IN std_logic;
5   f: OUT std_logic;
6   g: OUT std_logic
7 );
8 END sklopS1;
```

   Kao tip podataka ovdje smo koristili tip std_logic koji je definiran u biblioteci IEEE. Definicije portova koji su istog smjera (ulaz/izlaz) i kroz koje putuju signali istog tipa mogu se navesti kraće, tako da se nazivi portova nanizaju uz razdvajanje znakom zareza. Tako je, primjerice, ekvivalentna definicija sučelja prethodno navedenoj prikazana u nastavku.

```
1 ENTITY sklopS1 IS PORT (
2   a, b, c: IN std_logic;
3   f, g: OUT std_logic
4 );
5 END sklopS1;
```

   Uočite kako u oba slučaja, nakon posljednje definicije porta nema znaka točka-zarez.

3. *Deklaracija arhitekture sklopa.* U ovom djelu se daje opis unutrašnjosti "crne kutije". Kako smo prethodno napomenuli, taj opis može biti ostvaren na više načina. U određenim slučajevima, u opisu ćemo kombinirati elemente različitih načina opisivanja – takav ćemo opis onda zvati *hibridni*.

   Deklaracija arhitekture sklopa započinje navođenjem ključne riječi ARCHITECTURE, nakon čega slijedi naziv arhitekture, ključna riječ OF, naziv sklopa čija je to arhitektura te ključna riječ IS. Slijedi dio u kojem po potrebi možemo definirati pomoćne (unutarnje odnosno interne) signale, konstante i nove tipove podataka. Nakon toga dolazi ključna riječ BEGIN, jedna ili više naredbi

koje čine model toka podataka, strukturni model ili ponašajni model (svaka naredba završava s točkom-zarezom), i konačno dolazi ključna riječ END, naziv arhitekture i točka-zarez.

Primjer jedne arhitekture dan je u nastavku.

```
1 ARCHITECTURE arch1 OF sklopS1 IS
2   -- ovdje bi došle deklaracije pomoćnih signala
3 BEGIN
4   -- ovdje dolaze naredbe ponašajnog
5   -- ili strukturnog opisa
6   f <= a AND b AFTER 10 ns;
7   g <= (a OR c) AND b AFTER 10 ns;
8 END arch1;
```

U prethodnom primjeru sve što je navedeno iza -- čini komentar. Naziv arhitekture može biti bilo kakav valjan identifikator. Treba napomenuti da opći oblik opisa jednog sklopa može imati još neke elemente, ali na ovom mjestu ih nećemo navoditi.

Operator <= (retci 6 i 7 u prethodnom primjeru) je operator kojim se vrijednost izraza navedenog s desne strane pridružuje izlazu (odnosno općenito nekom signalu) navedenom s lijeve strane. Ako prije znaka točka-zarez dođe ključna riječ AFTER, tada se definira da se to pridruživanje događa uz kašnjenje koje je navedeno nakon ključne riječi, čime imamo mogućnost modeliranja kašnjenja u digitalnim sklopovima. Vrijeme koje se navodi ima iznos i mjernu jedinicu. Primjerice, u prethodnom primjeru oba izlaza za svojim ulazima kasne po 10 nanosekundi.

2.2 O jeziku VHDL

Jezik VHDL nije osjetljiv na velika i mala slova. Tako, primjerice, ključnu riječ kojom započinjemo arhitekturu sklopa možemo pisati ARCHITECTURE, architecture, ArChItEcTuRe ili na bilo koji drugi način.

Prilikom rada sa sustavom VHDLLab2 treba se ipak držati jednog ograničenja koje nameće taj sustav: naziv sklopa (tj. naziv modela) mora se uvijek pisati na isti način. Ako smo napravili model sklopS1, onda ga na svim mjestima moramo navoditi upravo tako.

Od logičkih operatora, na raspolaganju nam stoje sljedeći operatori.

not	logičko NE
and	logičko I
or	logičko ILI
nand	logičko NI
nor	logičko NILI
xor	logičko isključivo-ILI
xnor	komplement logičkog isključivo-ILI

Operator NOT višeg je prioriteta od ostalih operatora. Svi preostali binarni operatori navedeni u prethodnoj tablici istog su prioriteta. Posljedica toga je da prilikom pisanja logičkih izraza *moramo* koristiti zagrade za definiranje prioriteta operatora jer ćemo u slučaju da ih ne koristimo a u izrazu imamo više različitih operatora dobiti prijavljenu pogrešku prilikom prevođenja modela. Tako je, primjerice, sljedeći isječak kôda kojim pokušavamo definirati da je $f = \bar{a} \cdot b + c$ pogrešan.

```
1   f <= NOT a AND b OR c;
```

Prilikom prevođenja ovog modela, sustav će nas upozoriti da ne zna jesmo li htjeli najprije izračunati logičko I između $\bar{a}$ i b pa s tim rezultatom napraviti logičko ILI s c, ili je najprije trebalo izračunati logičko ILI između b i c pa s tim rezultatom napraviti logičko I s $\bar{a}$. Uz pretpostavku da smo htjeli da se izraz protumači na nama uobičajen način (I je većeg prioriteta od ILI), stavili bismo zagrade kako je prikazano u nastavku.

```
1  f <= (NOT a AND b) OR c;
```

2.3 Tip podataka std_logic

Tip std_logic definira vrijednosti signala koje jedan digitalni sklop može slati drugom digitalnom sklopu. Razmislite li, brzo ćete doći do zaključka da smo i mi na kolegiju osim logičke nule i logičke jedinice već spominjali i neke druge vrijednosti. Tako smo, primjerice, spomenuli da izlaz nekog sklopa može biti u stanju visoke impedancije (Z). Spomenuli smo i da nas ponekad uopće nije briga u kojem je stanju izlaz sklopa (*don't care*).

Prisjetite se sada cjeline s predavanja na kojoj smo govorili o implementacijama logičkih sklopova. Na izlazu logičkog sklopa visoka naponska razina može biti ostvarena na dva načina: jedan je ostvaren tako da je izlaz *pull-up*-otpornikom pritegnut prema napajanju (a tranzistor koji izlaz priteže prema masi je blokiran), a drugi je ostvaren tako da od izlaza prema napajanju imamo uključen tranzistor a od izlaza prema masi isključen tranzistor (*pasivno* vs. *aktivno* generiranje). Koja je posljedica ovih dvaju načina? Izlaze dva sklopa koja imaju izlaz pritegnut na napajanje preko *pull-up* otpornika možemo spojiti zajedno, i time ćemo dobiti novu Booleovu funkciju. To isto s izlazima koji imaju aktivno generiranje napona ne smijemo napraviti jer ćemo uništiti sklopove.

Kako bi omogućio opisivanje svih takvih vrsta sklopova, tip std_logic definira čak 9 različitih vrijednosti signala. Vrijednosti su pobrojane u nastavku.

1. `1`: predstavlja aktivno generiranu logičku jedinicu (engl. *forcing 1*). Ako se na izlazu sklopa pojavi ovakav signal, možemo zamisliti da sklop ima izlaz tranzistorom pritegnut na napajanje.
2. `0`: predstavlja aktivno generiranu logičku nulu (engl. *forcing 0*). Ako se na izlazu sklopa pojavi ovakav signal, možemo zamisliti da sklop ima izlaz tranzistorom pritegnut na masu.
3. `H`: predstavlja pasivno generiranu logičku jedinicu (engl. *weak 1*). Ako se na izlazu sklopa pojavi ovakav signal, možemo zamisliti da sklop ima izlaz pritegnut na napajanje preko *pull-up* otpornika.
4. `L`: predstavlja pasivno generiranu logičku nulu (engl. *weak 0*). Ako se na izlazu sklopa pojavi ovakav signal, možemo zamisliti da sklop ima izlaz pritegnut na masu *pull-down* otpornikom.
5. `X`: obično predstavlja stanje koje se dobije ako se dva izlaza koja aktivno generiraju izlaz spoje zajedno a sklopovi generiraju komplementarne izlaze (u praksi, označava nastanak kratkog spoja od napajanja prema masi). Naziv ove vrijednosti je *forcing Unknown*.
6. `W`: obično predstavlja stanje koje se dobije ako se dva izlaza koja pasivno generiraju izlaz spoje zajedno a sklopovi generiraju komplementarne izlaze (u praksi, označava situacije kada je kroz izlaze sklopova otvoren put od napajanje prema masi, ali koje obično nemaju fatalne posljedice jer je struja ograničena sumom iznosa *pull-up* i *pull-down* otpornika). Napon koji se dobije bit će vjerojatno negdje u zabranjenom području pa sljedeći sklop neće dobro protumačiti što mu se šalje. Naziv ove vrijednosti je *weak Unknown*.

7. `Z`: predstavlja izlaz koji je i od napajanja i od mase izoliran velikim otporom; to je izlaz koji je u trećem stanju (stanju visoke impedancije).

8. `-`: predstavlja situaciju u kojoj nas nije briga što je na izlazu (engl. *don't care*).

9. `U`: predstavlja situaciju u kojoj simulator ne zna što je na izlazu (engl. *uninitialized*). Primjerice, ako modeliramo sklop ILI tako da ima kašnjenje od 10 nanosekundi i na početku simulacije (od t=0) na ulaze dovedemo vrijednosti koje odgovaraju logičkoj nuli, što će biti na izlazu tog sklopa? Kako god da radimo simulaciju, simulator će temeljem našeg opisa moći zaključiti da će od 10. nanosekunde na izlazu biti vrijednost nula; međutim, što će biti od trenutka $t = 0ns$ do $t = 10ns$? Na to ne možemo odgovoriti, pa ćemo takve situacije označavati s vrijednosti `U`.

Literali tipa std_logic pišu se pod jednostrukim navodnicima. Primjerice:

```
1   f <= '1';
```

Uz tip std_logic koji predstavlja jedan signal, biblioteka IEEE nudi nam i tip std_logic_vector koji se može koristiti za opisivanje više signala istog imena i tipa. Sjetimo se primjera multipleksora 8/1: to je sklop koji ima 8 podatkovnih ulaza koje obično označavamo slovom D i indeksima od 0 do 7, ima 3 adresna ulaza koje obično označavamo slovom A i indeksima od 2 do 0 te ima jedan izlaz (obično y). Uporabom tipa std_logic_vector u VHDL-u možemo napisati upravo takvu definiciju. Primjerice, sučelje ovakvog multipleksora definirali bismo na sljedeći način.

```
1  ENTITY mux81 IS PORT (
2    d: IN std_logic_vector(0 to 7);
3    a: IN std_logic_vector(2 downto 0);
4    y: OUT std_logic
5  );
6  END mux81;
```

Uočite kako se nakon naziva tipa std_logic_vector u oblim zagradama navodi raspon indeksa; ako je početni broj manji od završnog, raspon se piše s TO a ako je početni broj veći od završnog, dolazi ključna riječ DOWNTO. Pojedinim signalima može se pristupati indeksiranjem; primjerice možemo zatražiti vrijednost d(6) što je predzadnji od signala ulaza d. Možemo uzimati i podraspone; primjerice: d(1 to 4) predstavlja signale d(1), d(2), d(3) i d(4), i to je po tipu opet std_logic_vector.

Konstante koje su tipa std_logic_vector navode se pod dvostrukim navodnicima. Primjerice, ako je izlaz f definiran kao std_logic_vector(0 to 3), primjer naredbe kojom ćemo vrijednosti dodijeliti svim četirima signalima je:

```
1   f <= "001U";
```

što je ekvivalentno kao da smo dali četiri naredbe:

```
1   f(0) <= '0';
2   f(1) <= '0';
3   f(2) <= '1';
4   f(3) <= 'U';
```

2.3.1 Booleove operacije nad tročlanim skupom {U, 0, 1}

Osim djelovanja Booleovih operatora nad dvočlanim skupom {0, 1}, za potrebe laboratorijskih vježbi moram se upoznati s djelovanjem tih operatora nad nešto širim skupom. Biblioteka IEEE u okviru paketa std_logic_1164 definira djelovanje Booleovih operatora nad svih devet vrijednosti signala std_logic. Mi ćemo se upoznati s minimalnim proširenjem kod kojeg ćemo dvočlani skup {0, 1} proširiti na tročlani skup {0, 1, U} uključivanjem još i vrijednosti U.

Nad tako definiranim tročlanim skupom, vrijednost `U` će i dalje označavati situaciju u kojoj ne znamo koja je vrijednost signala (ali ako su jedine druge moguće vrijednosti `0` i `1`, znamo da signal mora biti nešto od toga – samo ne znamo što).

Imajući to u vidu, proširenje djelovanja operatora je trivijalno, i dano je u nastavku.

Logičko I

- `0` I `0` = `0`
- `0` I `1` = `0`
- `0` I `U` = `0`
- `1` I `0` = `0`
- `1` I `1` = `1`
- `1` I `U` = `U`
- `U` I `0` = `0`
- `U` I `1` = `U`
- `U` I `U` = `U`

Logičko ILI

- `0` ILI `0` = `0`
- `0` ILI `1` = `1`
- `0` ILI `U` = `U`
- `1` ILI `0` = `1`
- `1` ILI `1` = `1`
- `1` ILI `U` = `1`
- `U` ILI `0` = `U`
- `U` ILI `1` = `1`
- `U` ILI `U` = `U`

Logičko NE

- NE `0` = `1`
- NE `1` = `0`
- NE `U` = `U`

Pogledajmo zašto su rezultati uz vrijednost `U` takvi kakvi jesu. Prema aksiomima Booleove algebre, $0 \cdot A = 0$; stoga, ako je A bilo nula, bilo jedan, rezultat ne ovisi o njemu i uvijek je nula. Stoga je i $0 \cdot U = 0$ jer nam `U` samo govori da ne znamo je li konkretna vrijednost nula ili jedan.

Oslanjajući se na jednako razmišljanje, dolazimo i do toga da je $1 \cdot U = U$; naime, ako je signal (čiju vrijednost ne znamo) zapravo u logičkoj jedinici, tada bi rezultat logičkog I bio 1; ako je pak u logičkoj nuli, rezultat bi bio nula. Kako nismo sigurni u rezultat (odnosno rezultat ovisi o konkretnoj vrijednosti tog signala), moramo i za sam rezultat reći da njegovu vrijednost ne znamo.

Tragom jednakog razmišljanja slijedi i da je $U \cdot U = U$. Preostale situacije nećemo razmatrati jer su ili podudarne s dvočlanom Booleovom algebrom, ili su rješive oslanjajući se na komutativnost operatora I.

Pogledajmo u nastavku dva primjera.

Primjer 1.
Što će biti dodijeljeno u signal f nakon izvođenja naredbe pridruživanja f <= (a and b) or c; ako je a='0', b='U', c='0'?

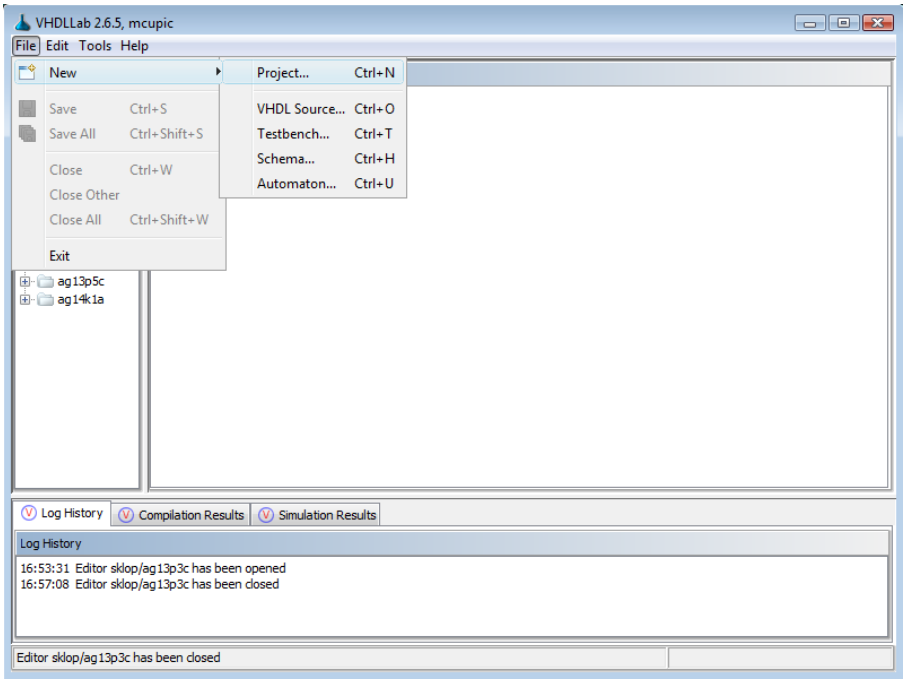
Rješenje:
Prema prethodno definiranom djelovanju operatora, najprije računamo a and b = '0' and 'U' = '0'. Potom računamo '0' or '0' = '0'. Signalu f bit će pridjeljena vrijednost '0'.

Primjer 2.
Što će biti dodijeljeno u signal f nakon izvođenja naredbe pridruživanja f <= a and not a; ako je a='U'?

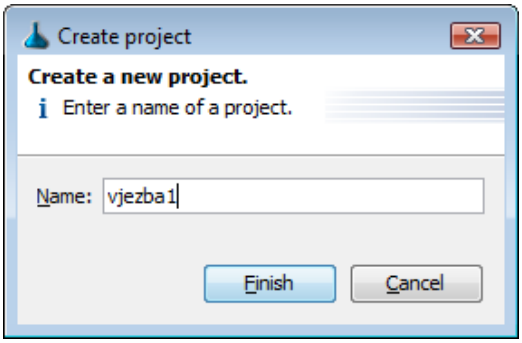
Rješenje:
Prema prethodno definiranom djelovanju operatora, najprije računamo not a = not 'U' = 'U'. Potom računamo 'U' and 'U' = 'U'. Signalu f bit će pridjeljena vrijednost 'U'. Uočite da u ovako definiranom tročlanom skupu ne vrijedi da je $A \cdot \bar{A} = 0$ ili da je $A + \bar{A} = 1$.

2.4 Modeliranje jednostavnog sklopa

U nastavku ćemo proći kroz modeliranje jednostavnog sklopa i to uporabom modela toka podataka, potom strukturnog modela i konačno ponašajnog modela. Sve primjere isprobajte i u sustavu VHDLLab2. Kao prvi korak, u sustavu VHDLLab2 napravite novi projekt (`vjezba1`): iz izbornika `File` odaberite `New` pa `Project`.



U dijalogu koji se otvori unesite kao ime projekta `vjezba1` i pritisnite `Finish`.



S lijeve strane u popisu projekata pojavit će se novostvoreni projekt. Sve što je opisano u nastavku ovog poglavlja radite u tom projektu.

2.4.1 Opis sklopa modelom toka podataka

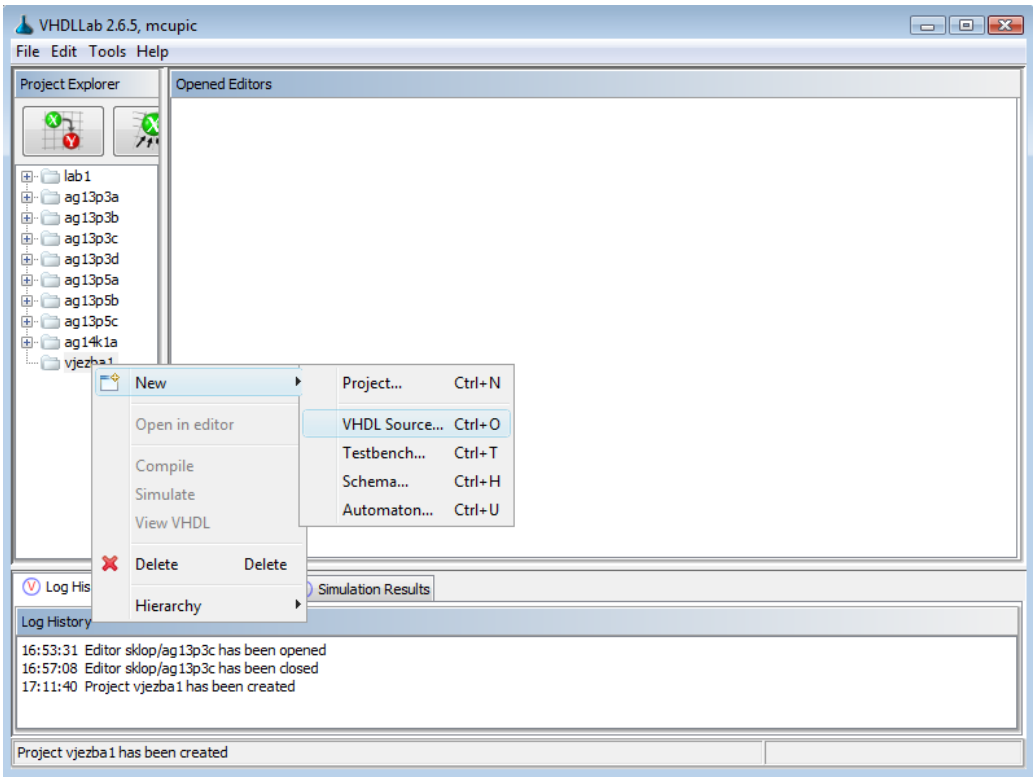
Pretpostavimo da je potrebno napisati model sklopa koji ima četiri ulaza a, b, c i d te jedan izlaz f, i koji ostvaruje sljedeću Booleovu funkciju: $f = (a \cdot b + c) \cdot d$. Također, neka sklop kao "crna kutija" ima kašnjenje od 10 nanosekundi.

Model toka podataka takvog sklopa dan je u nastavku. Sklop smo nazvali sklop1. Model se sastoji od jedne naredbe pridruživanja vrijednosti signalu.

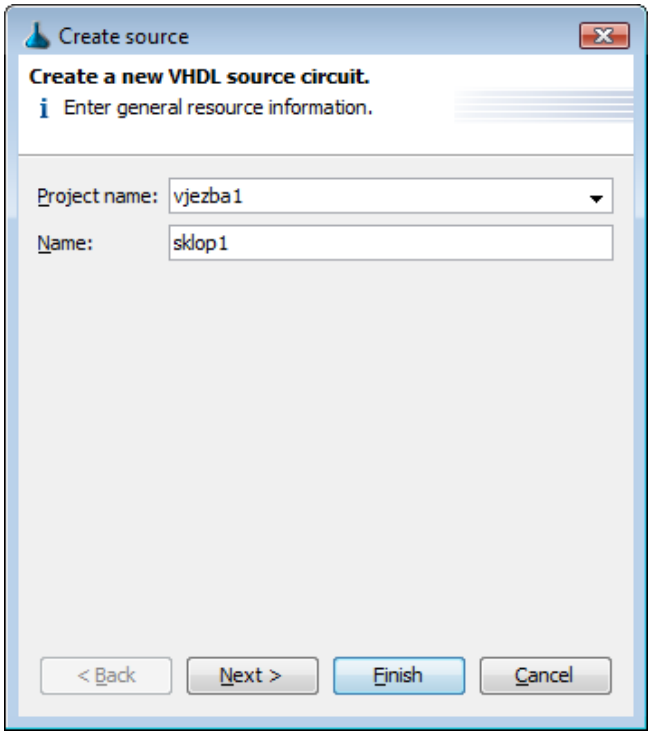
Listing 2.1 : Opis sklopa modelom toka podataka

```
-- uključivanje korištenih biblioteka
library IEEE;
use IEEE.std_logic_1164.all;

-- definicija sučelja sklopa
ENTITY sklop1 IS PORT (
   a: IN std_logic;
   b: IN std_logic;
   c: IN std_logic;
   d: IN std_logic;
   f: out std_logic);
END sklop1;

-- definicija arhitekture sklopa
-- dan je funkcijski model
ARCHITECTURE arch1 OF sklop1 IS
BEGIN
   f <= ((a AND b) OR c) AND d AFTER 10 ns;
END arch1;
```

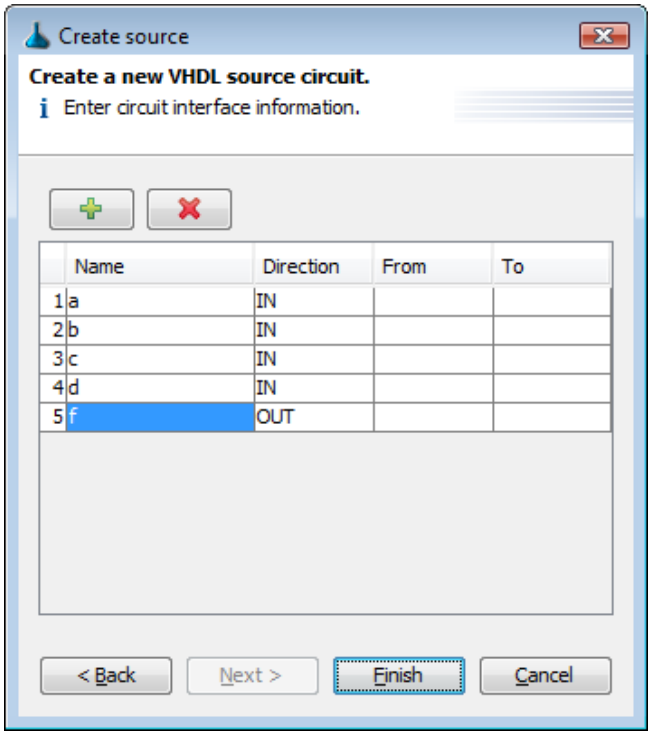
U projektu koji ste pripremili otiđite na `New` pa `VHDL source`.



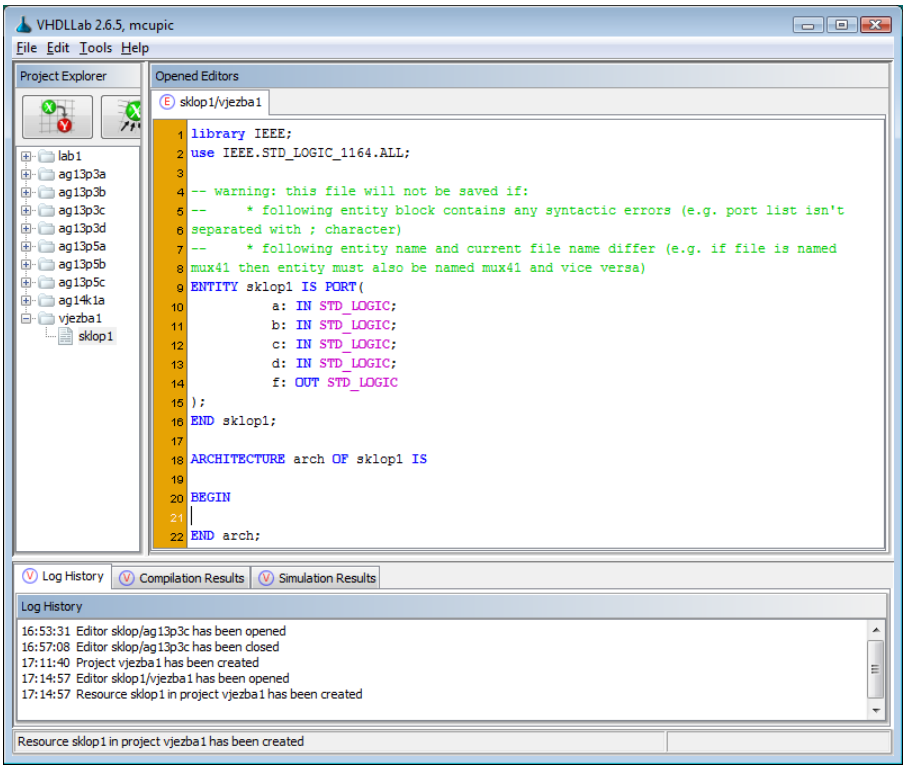
U dijalogu koji će se otvoriti unesite naziv sklopa (`sklop1`) i pritisnite `Next`.



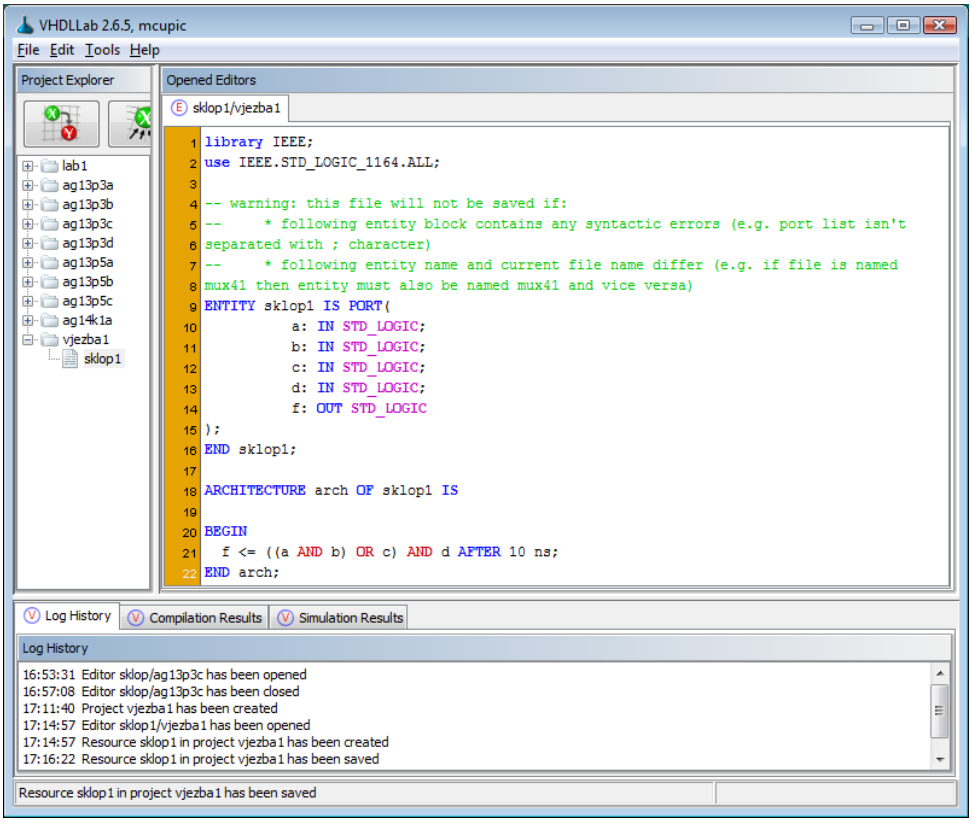
Na sljedećoj stranici dijaloga definirajte sučelje (dodajte četiri ulaza i jedan izlaz pritiskom na gumb + i podesite imena). Kad ste gotovi, pritisnite gumb `Finish`.



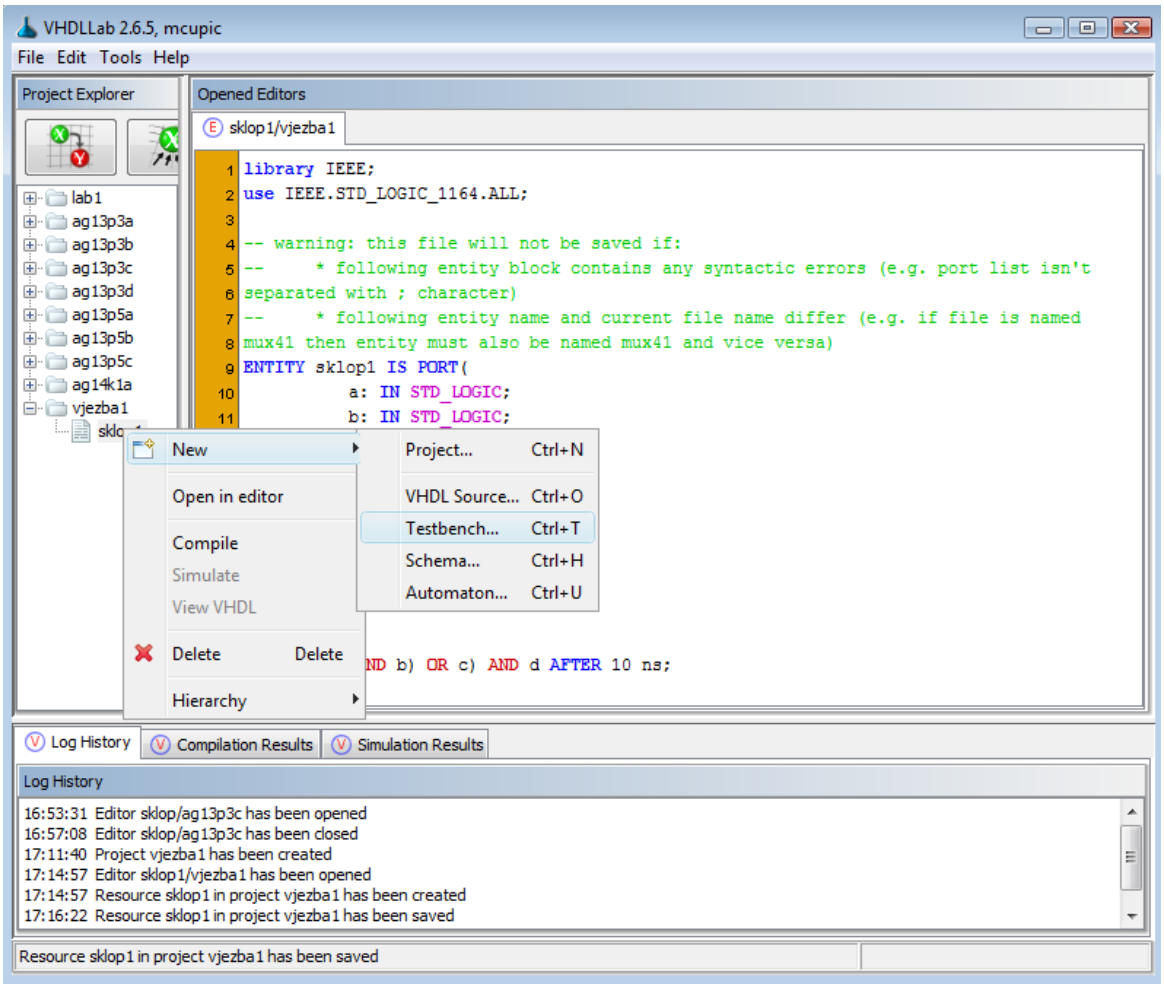
Otvorit će se uređivač VHDL modela, kako je prikazano na sljedećoj slici.



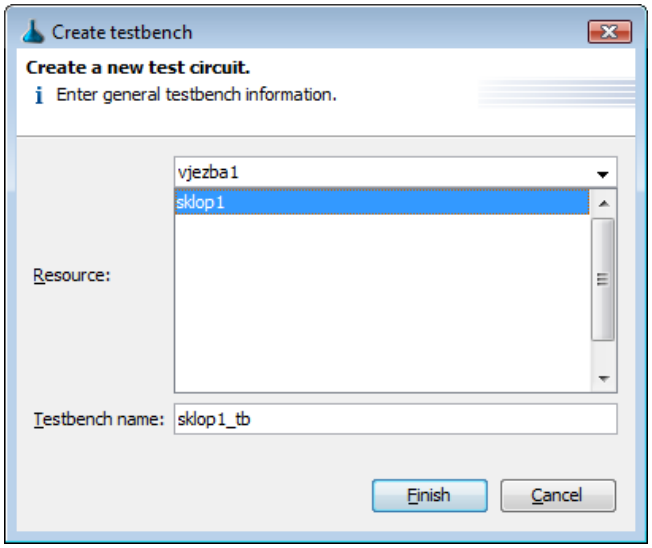
Otiđite u arhitekturu sklopa i tamo definirajte izraz prema kojem se računa vrijednost izlaza f uz zadano kašnjenje od 10 nanosekundi.



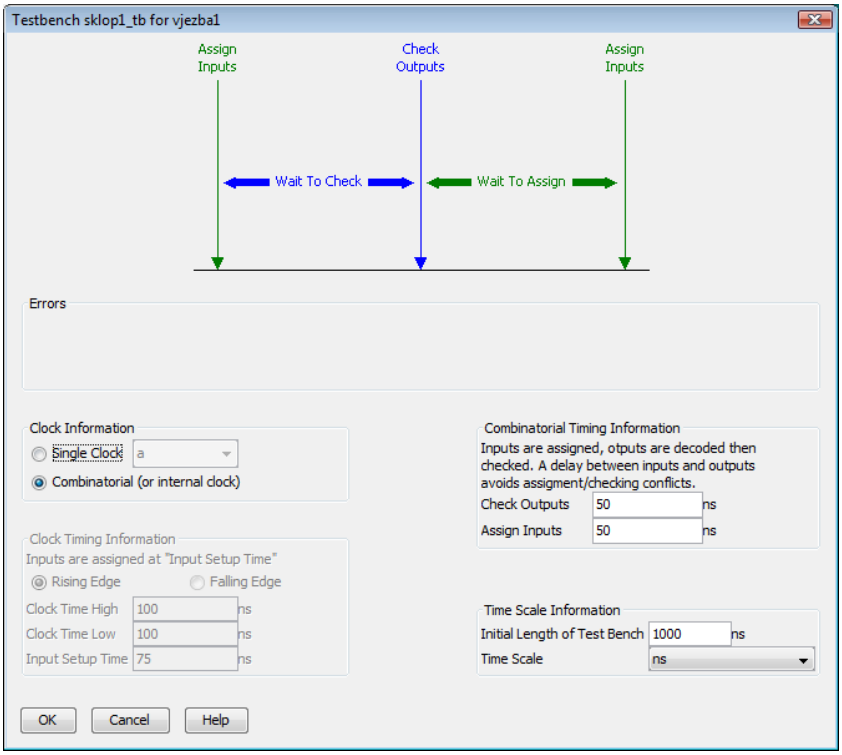
Jednom kada smo napravili model sklopa, isti bi trebalo i ispitati. Stoga u istom projektu napravite novi ispitni sklop: `New`, `Testbench`.



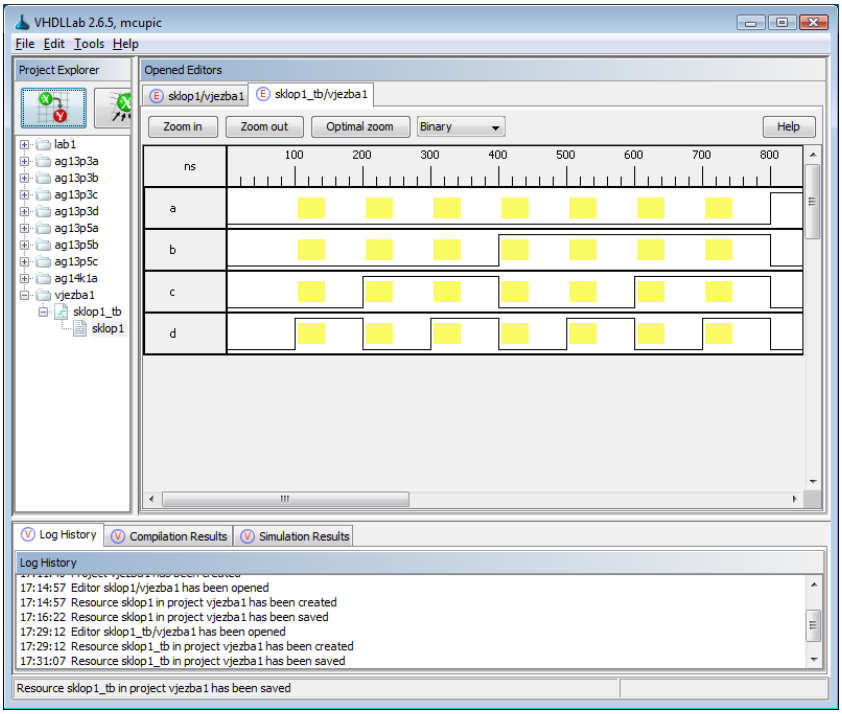
U dijalogu koji se otvori provjerite da je kao odabrani projekt doista označen `vjezba1` (ako nije, odaberite ga), i potom odaberite da želite ispitati sklop čije je ime `sklop1`. Na dnu dijaloga sustav će automatski predložiti da se ispitni sklop zove `sklop1_tb`, što je uobičajena konvencija koje se valja držati. Pritisnite `Finish`.



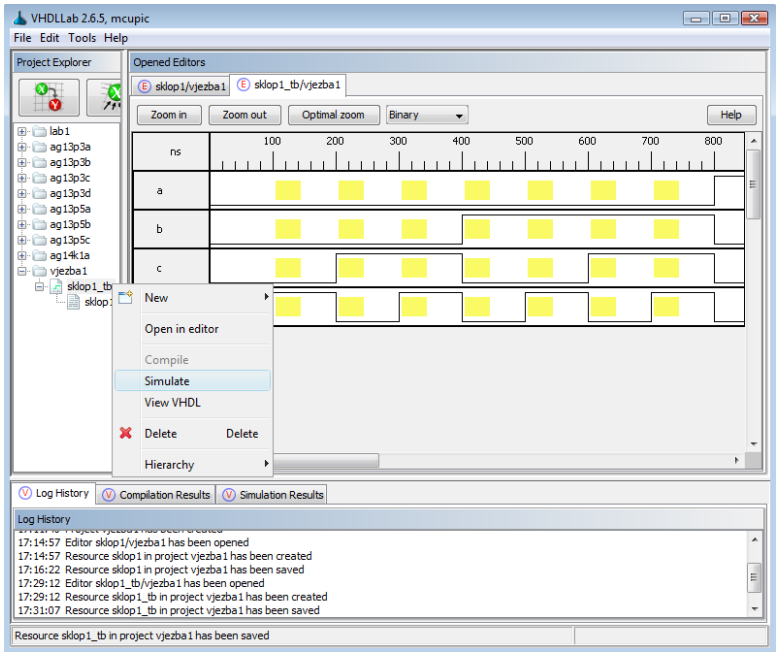
Potom će Vas sustav pitati je li to kombinacijski ili sekvencijski sklop, i koje vremenske parametre želite koristiti. Ostavite sve stavke kako su podešene, i samo pritisnite `OK`.



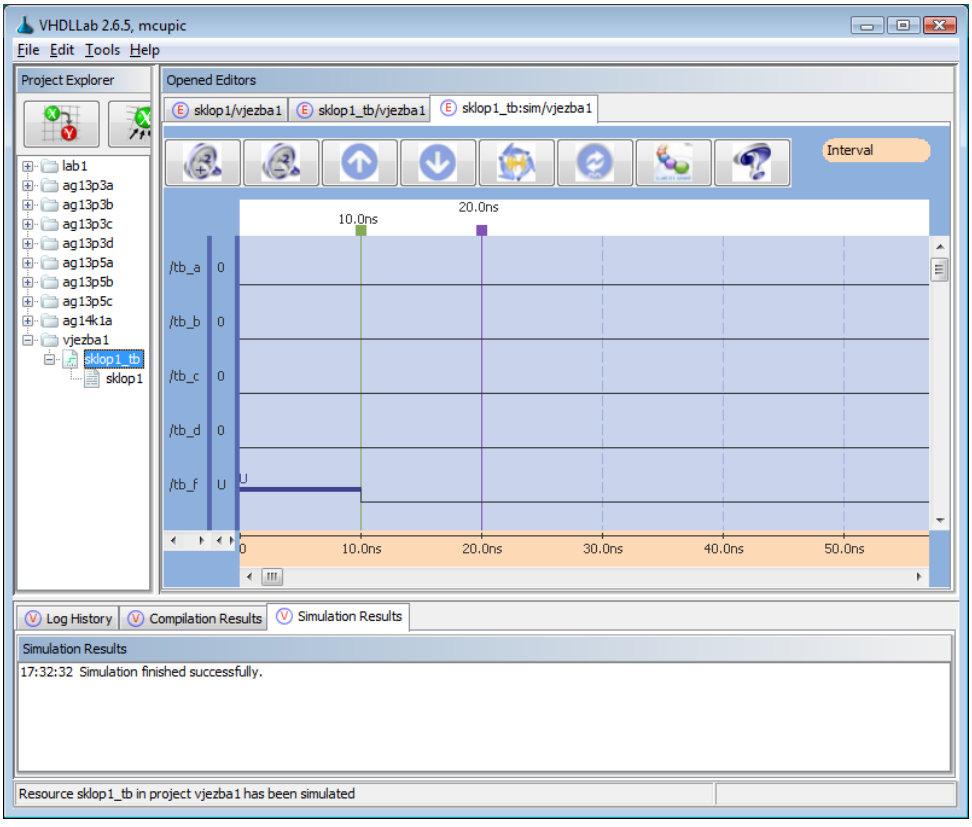
Pred Vama će se otvoriti uređivač ispitnog sklopa. Uređivač će Vam omogućiti da svakih 100 nanosekundi promijenite pobudu koja će biti dovedena na ispitivani sklop. Vaš je zadatak osigurati da rad sklopa ispitate u cijelosti: to znači da u trenutku t=0 ns na ulaze a, b, c i d dovodite vrijednosti 0, 0, 0, 0; u trenutku t=100 ns vrijednosti 0, 0, 0, 1, i tako redom sve do trenutka t=1500 ns kada na ulaze treba dovesti 1, 1, 1, 1. Kad ste gotovi, snimite ispitni sklop.



Nakon što je ispitni sklop napravljen i snimljen, možemo ga pokrenuti. Napravite desni klik na naziv ispitnog sklopa u stablu projekta s lijeve strane, i odaberite stavku `Simulate`.



Sustav će na poslužitelju napraviti simulaciju. Otvorit će se novi prozor u kojem će biti prikazani rezultati simulacije.

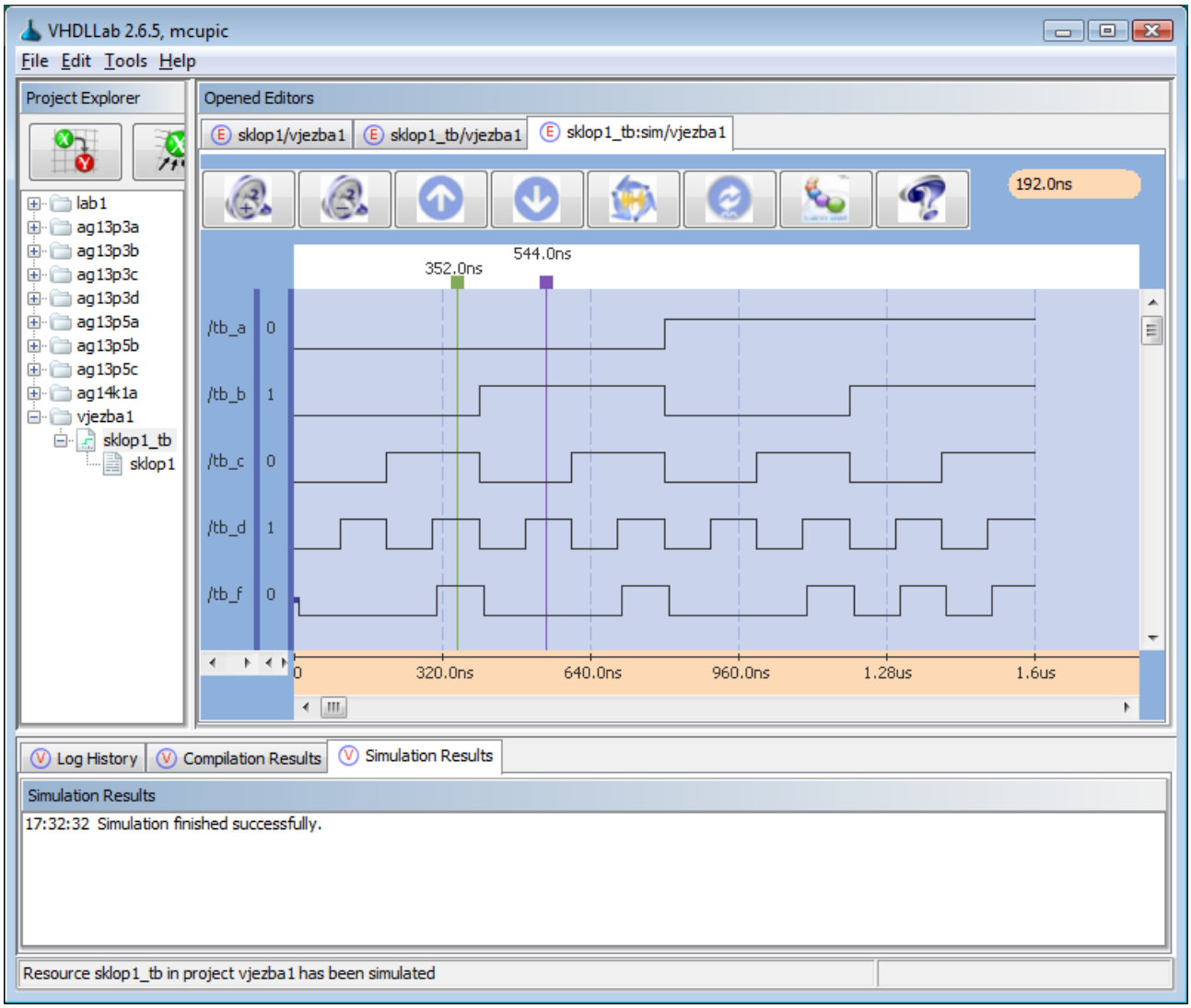


Pogledajmo malo bolje te rezultate. Inicijalno je odabran takav faktor uvećanja uz koji se na zaslonu prikazuje prvih pedesetak nanosekundi simulacije. Naime, analizom rezultata, sustav je zaključio da

se već u tom dijelu počinje događati nešto zanimljivo. Prva četiri retka prikazuju kako se kroz vrijeme mijenjaju signali a, b, c i d: mijenjaju se upravo onako kako smo ih u uređivaču ispitnog sklopa podesili. U t=0 ns sva četiri signala postavljena su na vrijednost '0', i čeka se odgovor našeg ispitivanog sklopa da postavi nešto na izlaz f.

Pogledajte što se događa na izlazu f. U trenutku t=0 ns izlaz f postavljen je na vrijednost U, i tako ostaje još neko vrijeme prije no što se promijeni i postane '0'. Vidite li sa slike koliko je vremena na izlazu vrijednost 'U'? Možete li to povezati s načinom na koji smo modelirali naš sklop? Koja je točno naredba odgovorna za to?

Ako ste odgovorili na prethodna pitanja, sada možete malo odzumirati simulaciju. U tu svrhu koristite alat za odzumiravanje (malo plavo povećalo sa znakom minusa u alatnoj traci). Odzumiravajte sve dok ne dobijete prikaz kao na sljedećoj slici.



Na prikazanoj slici jedan od kursora je prebačen na t=544 ns a drugi (zeleni, aktivan) na t=352 ns. Vrijednosti svih signala u trenutku u kojem se nalazi zeleni kursor ujedno su prikazana (očitana) uz lijevi rub simulacije, između naziva signala i njihovih valnih oblika.

Iz sinulacije vidimo za funkcija postaje '1' u 5 slučajeva: kada su a, b, c i d redom 0,0,1,1, zatim kada su 0,1,1,1, zatim kada su 1,0,1,1, zatim kada su 1,1,0,1 i konačno kada su 1,1,1,1. Je li to u redu?

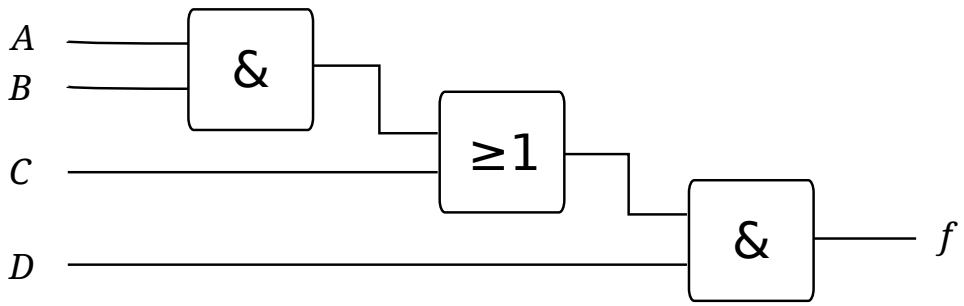
Definirate li ispitni sklop na prikazani način (tako da navedete sve kombinacije ulaza i to redosljedom kako biste ih inače pisali u tablici istinitosti, u određenom smislu simulacija Vam daje prikaz stvarne tablice istinitosti kroz vrijeme: svakih 100 nanosekundi izmjenjuje se jedan redak tablice). Ako ste funkciju prethodno i sami tabelirali, tada bi se provjera radi li sklop ispravno trebala svesti na usporedbu Vaše tablice i tablice koju je dala simulacija.

2.4.2 Strukturni opis sklopa

Strukturni opis sklopa polazi od pretpostavke da ćemo sklop opisivati uporabom jednostavnijih gotovih komponenata i njihovim povezivanjem. Strukturni model sklopa konceptualno je upravo ono što radimo kada sklop definiramo uporabom shematica.

Za ilustraciju ćemo u nastavku napraviti strukturni model sklopa koji ostvaruje već prethodno razmatranu funkciju: $f = (a \cdot b + c) \cdot d$. Sklop ćemo nazvati sklop2.

Shematski prikaz ovog sklopa dan je u nastavku.



Stoga ćemo pretpostaviti da u projektu već imamo napravljene gotove jednostavnije komponente koje odgovaraju sklopovima od kojih se sastoji prikazana shema. Konkretno, pretpostavit ćemo da u projektu imamo napravljen model `SklopAND` koji predstavlja sklop koji ima dva ulaza i jedan izlaz i računa logičko I, te da imamo napravljen model `SklopOR` koji predstavlja sklop koji ima dva ulaza i jedan izlaz i računa logičko ILI.

S obzirom da te sklopove još nemamo u projektu, evo njihovih definicija u nastavu (uočite, te smo sklopove opisali modelom toka podataka). Dodajte ih u projekt (napravite dva nova VHDL modela).

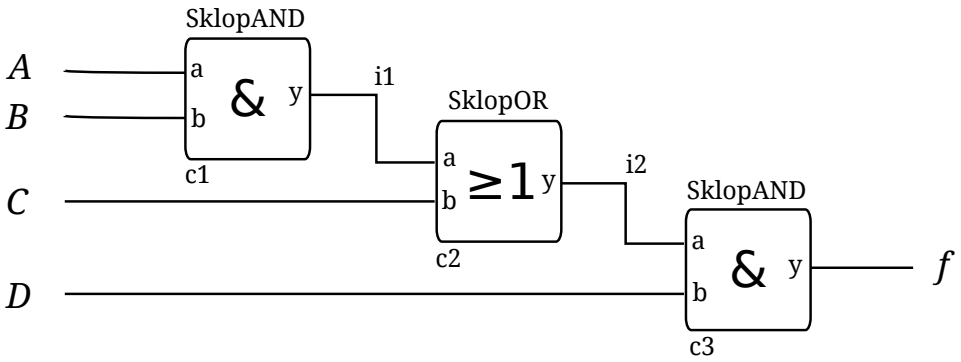
Listing 2.2 : Ponašajni model sklopa SklopAND

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  ENTITY SklopAND IS PORT (
5    a, b: IN std_logic;
6    y: out std_logic);
7  END SklopAND;
8
9  ARCHITECTURE arch1 OF SklopAND IS
10 BEGIN
11   y <= a AND b AFTER 10 ns;
12 END arch1;
```

Listing 2.3 : Ponašajni model sklopa SklopOR

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  ENTITY SklopOR IS PORT (
5    a, b: IN std_logic;
6    y: out std_logic);
7  END SklopOR;
8
9  ARCHITECTURE arch1 OF SklopOR IS
10 BEGIN
11   y <= a OR b AFTER 10 ns;
12 END arch1;
```

Da bismo prikazali strukturni model sklopa sklop2, najprije ćemo nadopuniti prethodnu shemu, kako je prikazano u nastavku.



Što smo napravili?

- Iznad svakog sklopa dopisali smo naziv komponente koja u projektu predstavlja model tog sklopa.
- Ispod svakog sklopa naveli smo jedinstveno ime primjerka komponente. U shemi imamo ukupno tri primjerka od dvije komponente: imamo dva primjerka komponente SklopAND (nazvali smo ih c1 i c3) te jedan primjerak komponente SklopOR (nazvali smo ga c2).
- U svaki sklop ucrtali smo "lokalna" imena ulaza i izlaza. U skladu s prethodno danom definicijom, sklopovi SklopAND i SklopOR imaju ulaze a i b te izlaz y.
- Svim žicama koje nisu niti ulazi sklopa niti izlazi sklopa (odnosno koje se izvana ne vide i ne čine sučelje sklopa) dali smo imena. Te će žice postati interni signali: vodiči kojima ćemo povezivati komponente na način koji se izvana ne vidi.

Sada smo spremni za pisanje strukturnog modela. Model je prikazan u nastavku.

Listing 2.4 : Strukturni model sklopa

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  ENTITY sklop2 IS PORT (
5    a,b,c,d: IN std_logic;
6    f: out std_logic);
7  END sklop2;
8
9  ARCHITECTURE arch1 OF sklop2 IS
10   SIGNAL i1, i2: std_logic;
11 BEGIN
12   c1: ENTITY work.SklopAND PORT MAP (a, b, i1);
13   c2: ENTITY work.SklopOR PORT MAP (i1, c, i2);
14   c3: ENTITY work.SklopAND PORT MAP (i2, d, f);
15 END arch1;
```

Usporedite `sklop1` i `sklop2`: prvog smo opisali ponašajno a drugog strukturno. *Sučelja* su u oba slučaja jednaka: radi li se o ponašajnom ili o strukturnom opisu, nije moguće zaključiti iz sučelja sklopa već isključivo iz arhitekture sklopa. Stoga u strukturnom opisu sve do retka 10 nema nikakvih razlika u odnosu na ponašajni opis.

U retku 10 dodali smo deklaraciju dvaju signala: `i1` i `i2`, oba tipa std_logic. Interni signali deklariraju se u arhitekturi sklopa prije ključne riječi BEGIN uporabom ključne riječi SIGNAL.

Nakon ključne riječi BEGIN započinje strukturni opis. Za svaki od primjeraka sklopova koje imamo u shemi naveli smo po jednu naredbu kojom se stvara primjerak komponente. Naredba uvijek započinje nazivom primjerka komponente, slijedi znak dvotočke, ključna riječ ENTITY, potom `work.` pa naziv naše komponente; potom dolaze ključne riječi PORT MAP, otvorena obla zagrada, definicija povezivanja komponente, zatvorena obla zagrada pa točka-zarez.

Prethodni primjer u sva tri retka koristi *pozicijsko povezivanje*: u oblim zagradama redosljedom kojim su ulazi i izlazi definirani u sučelju komponente čiji primjerak stvaramo navodimo ulaze, izlaze odnosno interne signale iz našeg sklopa koje spajamo. Pogledajmo pažljivije redak 13: u oblim zagradama piše (i1,c,i2). Iz sučelja komponente `SklopOR` pak znamo da sklop najprije ima ulaz a, potom ulaz b pa izlaz y. Slijedi da smo na ulaz a komponente `SklopOR` doveli interni signal $i1$, na ulaz b komponente `SklopOR` naš vlastiti ulaz c a na izlaz y komponente `SklopOR` naš interni signal $i2$.

Osim pozicijskog povezivanja, VHDL nam dozvoljava i uporabu *povezivanja putem imena*. U tom slučaju u oblim zagradama navodimo lokalno ime ulaza/izlaza komponente, znak `=>` pa ime ulaza/izlaza/internog signala našeg sklopa koji spajamo na taj port komponente. Stoga smo tijelo arhitekture mogli napisati i ovako.

```
1    c1: ENTITY work.SklopAND PORT MAP (a=>a, b=>b, y=>i1);
2    c2: ENTITY work.SklopOR PORT MAP (a=>i1, b=>c, y=>i2);
3    c3: ENTITY work.SklopAND PORT MAP (a=>i2, y=>f, b=>d);
```

Pri pisanju povezivanja putem imena redosljed navođenja signala više nije bitan. To je ilustrirano u prethodnom primjeru kod primjerka `c3`, gdje je povezivanje izlaza komponente navedeno u sredini. Kako kod ovog povezivanja prevoditelju dajemo sve potrebne informacije, redosljed kojim ih navodimo postaje nebitan.

Valja napomenuti da se primjerci komponenti mogu stvarati na još jedan način. Kako se tu zahtjevaju promjene na dva mjesta, cjelovit modificirani primjer prikazan je u nastavku.

Listing 2.5 : Strukturni model sklopa uz deklaraciju komponenti

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  ENTITY sklop2 IS PORT (
5    a,b,c,d: IN std_logic;
6    f: out std_logic);
7  END sklop2;
8
9  ARCHITECTURE arch1 OF sklop2 IS
10   component SklopAND is port (
11     a : IN STD_LOGIC;
12     b : IN STD_LOGIC;
13     y : OUT STD_LOGIC
14   );
15   end component;
16   component SklopOR is port (
17     a : IN STD_LOGIC;
18     b : IN STD_LOGIC;
19     y : OUT STD_LOGIC
20   );
21   end component;
22   SIGNAL i1, i2: std_logic;
23 BEGIN
24   c1: SklopAND PORT MAP (a, b, i1);
25   c2: SklopOR PORT MAP (i1, c, i2);
26   c3: SklopAND PORT MAP (i2, d, f);
27 END arch1;
```

U čemu su razlike? Počev od retka 10 u arhitekturi smo uz definiciju potrebnih internih signala ponudili i definiciju sučelja svih korištenih komponenata. Ova se definicija radi uporabom ključne riječi COMPONENT, nakon čega je ostatak jednak kao kod definicije sučelja komponente. Tako u retcima 10 do 15 dajemo definiciju sučelja komponente `SklopAND` a u retcima 16 do 21 definiciju sučelja komponente `SklopOR`.

Uz te definicije sada više nema potrebe da se u arhitekturi sklopa prilikom stvaranja primjeraka komponenata piše ENTITY work.; dovoljno je samo navesti naziv komponente (ilustrirano u retcima 24–26).

Postoji još razlika između ova dva načina opisivanja, ali te nam razlike na ovom kolegiju nisu značajne pa ih nećemo niti navoditi.

2.4.3 Ponašajni opis sklopa

Kod ponašajnog modela sklopa njegovu funkcionalnost ne opisujemo oslanjajući se isključivo na logičke operacije, već koristimo proceduralno programiranje i naredbe za kontrolu toka izvođenja. U arhitekturi sklopa, za to se koriste blokovi process. Svaki blok process može imati svoje ime (piše se ispred ključne riječi process i odvaja dvotočkom), može imati *listu osjetljivosti* (to je popis signala koji simulatoru kažu da se pri promjenama tih signala program naveden u bloku process mora ponovno izvesti), može definirati vlastite varijable (varijable su slične signalima samo što se deklariraju unutar bloka process prije ključne riječi begin, vidljive su samo u tom bloku, ne omogućavaju modeliranje kašnjenja i vrijednost im se pridružuje znakom := a ne kao kod signala znakom <=), i konačno unutar tijela bloka koje je omeđeno ključnim riječima begin i end process dolazi programski opis.

Primjer ponašajnog opisa sklopa za koji smo već prikazali i model toka podataka i strukturni model prikazan je u nastavku.

Listing 2.6 : Ponašajni model sklopa

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  ENTITY sklop3 IS PORT (
5    a: IN std_logic;
6    b: IN std_logic;
7    c: IN std_logic;
8    d: IN std_logic;
9    f: out std_logic);
10 END sklop3;
11
12 ARCHITECTURE arch1 OF sklop3 IS
13 BEGIN
14
15 fja: process(a,b,c,d)
16        variable tmp: std_logic;
17      begin
18        tmp := '0';
19        if d='1' then
20          if a='1' and b='1' then
21            tmp := '1';
22          elsif c='1' then
23            tmp := '1';
24          end if;
25        end if;
26        f <= tmp after 10 ns;
27      end process;
28
29 END arch1;
```

Sve naredbe napisane unutar bloka process na početku simulacije izvedu se jednom i potom se njegovo izvođenje suspendira tako dugo dok se ne promijeni neki od signala unutar navedene liste osjetljivosti (u našem primjeru čine je signali a, b, c i d). Kada se bilo koji od tih signala promijeni, naredbe u bloku process ponovno se izvedu i blok se opet zamrzne na kraju. Blok process s listom osjetljivosti ponaša se kao beskonačna petlja koja na dnu ima naredbu koja zamrzava izvođenje sljedeće iteracije sve dok se ne detektira promjena vrijednosti na nekom od signala iz liste osjetljivosti. Ekvivalentni blok process onome iz prethodnog primjera prikazan je u nastavku: sada nemamo listu osjetljivosti već eksplicitnu naredbu koja zamrzava prelazak u novu iteraciju sve do promjene na bilo kojem od navedenih signala.

Listing 2.7 : Blok process bez liste osjetljivosti

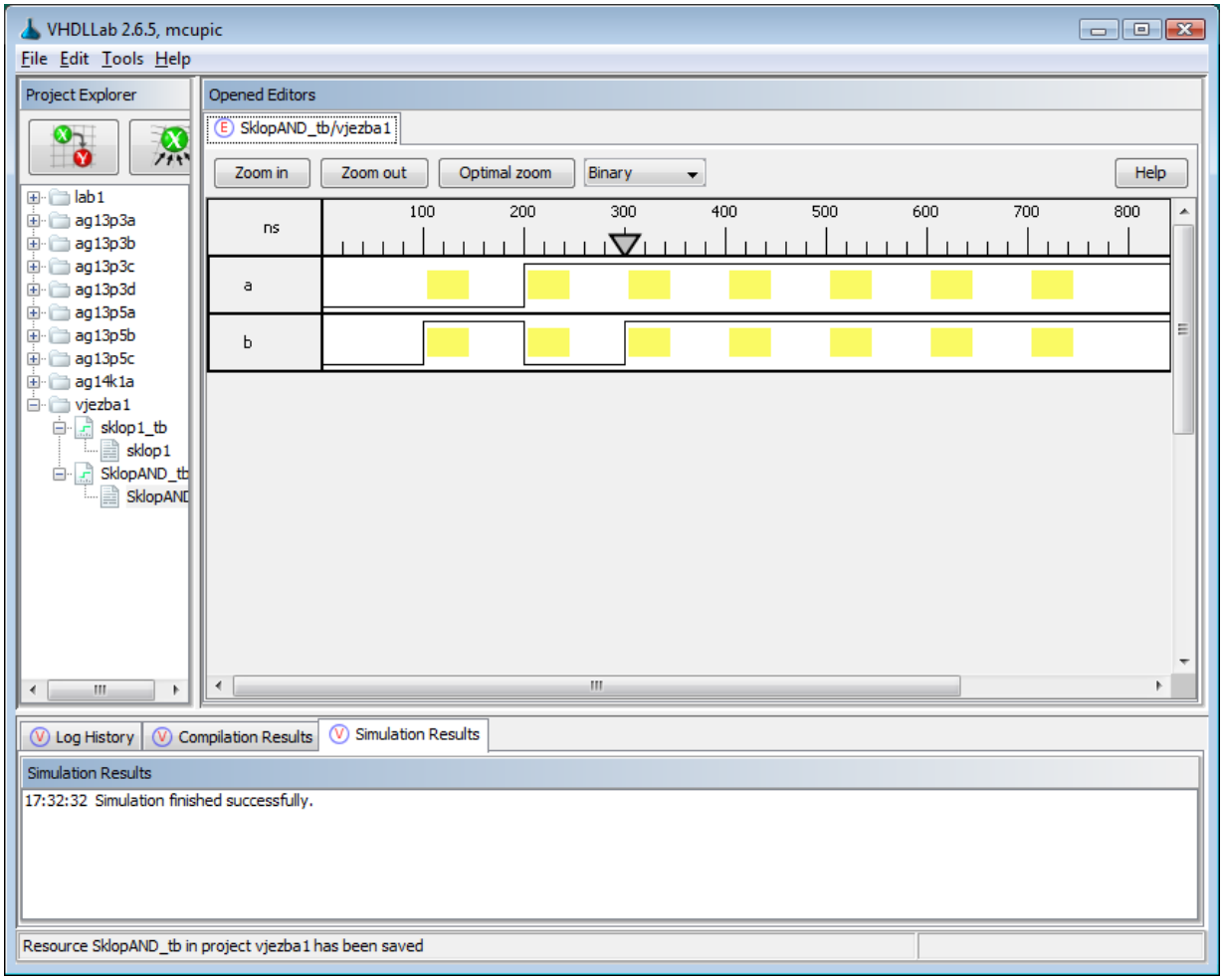
```
1  fja: process
2         variable tmp: std_logic;
3       begin
4         tmp := '0';
5         if d='1' then
6           if a='1' and b='1' then
7             tmp := '1';
8           elsif c='1' then
9             tmp := '1';
10          end if;
11        end if;
12        f <= tmp after 10 ns;
13        wait on a, b, c, d;
14      end process;
```

Pogledajmo sada ovaj posljednji opis. U bloku process definirali smo pomoćnu varijablu tmp (redak 2). Potom je u tijelu bloka najprije inicijaliziramo na vrijednost '0' pa nizom if naredbi provjeravamo i

po potrebi korigiramo njezinu vrijednost na '1' (retci 5 do 11). Konačno, u retku 12 imamo naredbu koja izlazu f pridružuje izračunatu vrijednosti uz kašnjenje od 10 nanosekundi. U retku 13 imamo naredbu koja suspendira prelazak u sljedeću iteraciju sve dok se ne detektira promjena na bilo kojem od signala *a*, *b*, *c* ili *d*. Kod verzije bloka process koji ima eksplicitno navedenu listu osjetljivosti ovaj redak nije potreban.

2.5 Ispitni sklop

Ostalo nam je za odgovoriti na još jedno pitanje: što je zapravo ispitni sklop? Do sada smo se s ispitnim sklopovima upoznali koristeći "čarobnjake" sustava VHDLLab2 koji su nam nudili lijepo grafičko sučelje za njihovo uređivanje, i kada smo pokrenuli simulaciju, magija se dogodila i mi smo dobili rezultate. Da bismo demistificirali ispitne sklopove, vratimo se osnovama. Kroz prethodne primjere napravili ste i u projekt dodali ponašajni model sklopa `SklopAND`. Napravimo za njega ispitni sklop pomoću ugrađenog čarobnjaka, i podesimo pobudu tako da u t=0 ns postavi ulaze *a* i *b* na '0', u t=100 ns postavi *a* na '0' a *b* na '1', u t=200 ns postavi *a* na '1' a *b* na '0' te u t=300 ns postavi *a* na '1' i *b* na '1'.



U stablu projekta s lijeve strane odaberite sada mišem sam ispitni sklop `SklopAND_tb` i iz iskočnog izbornika odaberite stavku `View VHDL`. Iznenađenje: ispitni sklop nije ništa drugo doli još jedan sklop koji je opisan u VHDL-u. VHDL kôd koji ćete dobiti prikazan je i ovdje u nastavku.

Listing 2.8 : Ispitni sklop

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity SklopAND_tb is
5  end SklopAND_tb;
6
7  architecture structural of SklopAND_tb is
8    component SklopAND is port (
9      a : IN STD_LOGIC;
10     b : IN STD_LOGIC;
11     y : OUT STD_LOGIC);
12   end component;
13
14   signal tb_a : STD_LOGIC;
15   signal tb_b : STD_LOGIC;
16   signal tb_y : STD_LOGIC;
17 begin
18
19     -- primjerak sklopa koji se ispituje
20     uut: SklopAND port map ( tb_a, tb_b, tb_y );
21
22     -- sklop koji generira zadane ispitne uzorke
23   process
24   begin
25     tb_a <= '0';
26     tb_b <= '0';
27     wait for 100 ns;
28
29     tb_b <= '1';
30     wait for 100 ns;
31
32     tb_a <= '1';
33     tb_b <= '0';
34     wait for 100 ns;
35
36     tb_b <= '1';
37     wait for 100 ns;
38     wait;
39   end process;
40
41 end structural;
```

Uočimo: ispitni sklop je sklop čije je sučelje prazno; on nema niti ulaza, niti izlaza (retci 4 i 5). U arhitekturi sklopa prije ključne riječi BEGIN dana je deklaracija sučelja komponente `SklopAND_tb`: to je *ispitivana komponenta*. Potom je još deklarirano upravo onoliko internih signala koliko ispitivani sklop ima ulaza i izlaza. Tijelo arhitekture sastoji se od samo dvije naredbe (redak 21 te redak 24).

U retku 21 dana je naredba koja stvara jedan primjerak ispitivane komponente i interne signale spaja na njezine ulaze i izlaze. Tom je primjerku dano ime `uut` (što je kratica od engleske fraze *Unit Under Test*).

Druga naredba je naredba process: ona se proteže od retka 24 pa sve do retka 40, i VHDL čitav njezin sadržaj tretira kao jednu naredbu, odnosno ta čitava naredba modelira novi sklop koji se također nalazi u našem ispitnom sklopu. Specifičnost ove naredbe (bloka process) je da nam omogućava da dio funkcionalnosti sklopa opisujemo gotovo kao u klasičnom programskom jeziku: naredbe koje se navedu unutar bloka process sustav izvršava slijedno. Blok process kakav je napisan u ovom primjeru ponaša se kao beskonačna petlja čije bi se naredbe neprestano ponavljale (više ćemo naučiti nešto kasnije).

Pogledajmo sada malo detaljnije naredbe unutar bloka process. Retci 26 i 27 na interne signale `tb_a` i `tb_b` postavljaju vrijednost '0'. No ti isti interni signali spojeni su na ulaze `a` i `b` naše ispitivane komponente, čime ovaj blok process postavlja vrijednosti koje se kao ulazi dovode na ispitivanu komponentu koja će odreagirati na odgovarajući način i svoj izlaz `y` postaviti na odgovarajuću vrijednost (uz zadano kašnjenje); ta će vrijednost biti vidljiva i na internom signalu `tb_y` jer je izlaz `y` spojen na njega.

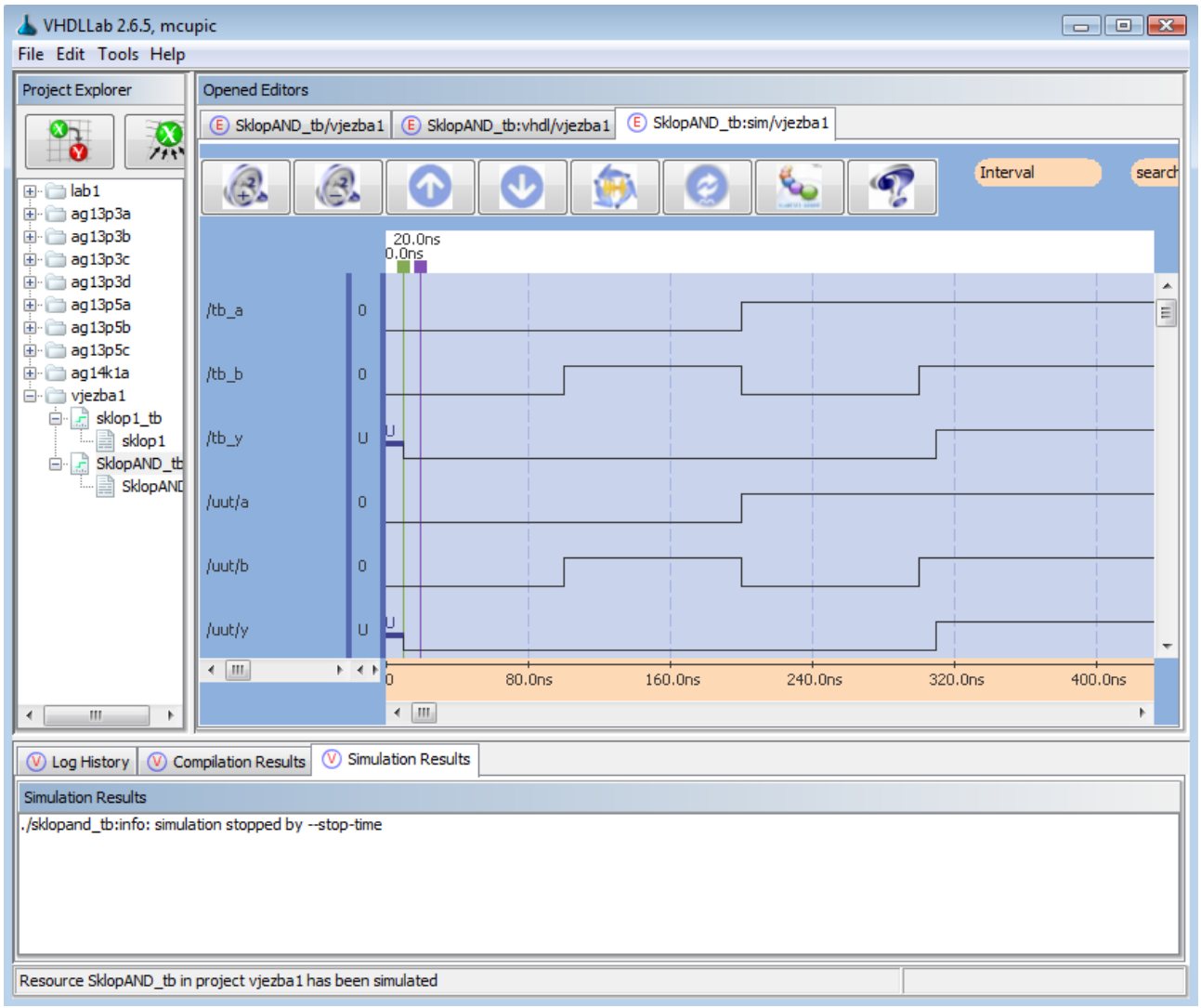
Izvođenje ovog bloka potom se zamrzava u trajanju od 100 ns (redak 28, naredba wait for), nakon čega naredba u retku 30 mijenja vrijednost internog signala `tb_b` na '1' (`tb_a` ostaje kakav je bio) i blok process se opet zaglavljuje na sljedećih 100 ns (redak 31). Postupak se ponavlja još dva puta (retci 33 do 35, pa retci 37 do 38). Na ovaj način blok process na interne je signale postavio ulazne signale upravo u skladu s onime kako smo ih mi naklikali u grafičkom uređivaču. Kako bi se spriječilo da se ova pobuda sada ciklički ponavlja, blok process terminira se u retku 39 naredbom bezuvjetnog čekanja wait; iz koje blok process više nikada ne može izaći.

Ako u simuliranom sustavu u tom trenutku više nema nikakvih promjena, ovo je i trenutak u kojem će završiti i simulacija.

Još nam je ostalo da pokrenemo simulaciju i promotrimo rezultate (slika u nastavku). Pogledajte sada malo pažljivije popis svih signala koje prikazuje simulator. Uočite da postoji 6 signala.

- `/tb_a`: to je interni signal koji se nalazi u ispitnom sklopu; vrijednosti mu postavlja blok process a čita naš ispitivani sklop.
- `/tb_b`: to je interni signal koji se nalazi u ispitnom sklopu; vrijednosti mu postavlja blok process a čita naš ispitivani sklop.
- `/tb_y`: to je interni signal koji se nalazi u ispitnom sklopu; vrijednosti mu postavlja naš ispitivani sklop.
- `/uut/a`: to je ulaz sklopa `SklopAND`; vrijednosti mu izvana postavlja naš ispitni sklop, a vrijednost čita naredba u arhitekturi sklopa `SklopAND` koja određuje vrijednost izlaza `y`.
- `/uut/b`: to je ulaz sklopa `SklopAND`; vrijednosti mu izvana postavlja naš ispitni sklop, a vrijednost čita naredba u arhitekturi sklopa `SklopAND` koja određuje vrijednost izlaza `y`.
- `/uut/y`: to je izlaz sklopa `SklopAND`; vrijednosti mu postavlja naredba u arhitekturi sklopa `SklopAND` koja određuje vrijednost tog izlaza.

Da smo imali bogatije strukturne modele (sklop koji sadrži sklop koji sadrži sklop) popis signala bio bi veći i bogatiji. U rezultatima simulacije za svaki od postojećih signala mogli biste vidjeti vrijednosti u bilo koje trenutku.



Primjetimo: ispitni sklop modeliran je hibridnim modelom.

2.6 O kašnjenjima

Jezik VHDL uz niz ključnih riječi koje služe za modeliranje funkcionalnosti sklopova nudi i neke jezične konstrukte koji omogućavaju specificiranje kašnjenja. Primjerice, već smo se upoznali s naredbom oblika:

```
1   f <= a AND b AFTER 10 ns;
```

Treba napomenuti da ti jezični konstrukti nisu sintetizabilni: opišete li digitalni sklop na takav način i potom zatražite od sintetizatora digitalnog sklopovlja da takav opis "utoči" u Vaš programirljivi sklop (primjerice, FPGA), sintetizator će naredbe kašnjenja zanemariti. Naime, prilikom sinteze sklopa iz VHDL opisa sintetizator na raspolaganju ima programirljiv sklop koji ste mu zadali i koji je ostvaren u određenoj tehnologiji. U toj tehnologiji, komponente će imati kašenjenja koja su određena samom tehnologijom i na koja Vi (niti sintetizator) ne možete utjecati.

Zapitate li se zašto onda takve naredbe uopće postoje, odgovor je jednostavan: kako bi se omogućilo izvođenje što je moguće realnijih simulacija. Naime, svaki ozbiljniji alat za modeliranje i sintezu

digitalnog sklopovlja korisnicima nudi mogućnost izvođenja nekoliko različitih vrsta simulacija. Najjednostavnija simulacija je simulacija ponašajnog modela (engl. *Behavioral model simulation*) kod koje simulator simulira Vaš opis upravo onako kako ste ga definirali. Međutim, takav alat možete zatražiti i provođenje simulacije koja uzima u obzir fizički razmještaj komponenata u programirljivom sklopu i način na koji su komponente međusobno povezane (engl. *Post place and route simulation*). Ono što će se u tom trenutku dogoditi jest da će alat zanemariti Vaš polazni opis i generirati novi VHDL opis koji koji u obzir uzima građu programirljivih komponenata i njihova stvarna kašnjenja koja će opisivati koristeći naredbe poput prethodne navedene uz AFTER 10 ns. Takav opis potom će poslati simulatoru na simulaciju čime ćete dobiti vjerniju sliku onoga što će se u stvarnosti događati jednom kada se konačni sklop sintetizira. Ciljani korisnik ovih primitiva niste dakle Vi kao dizajner sklopa (iako Vam nitko ne brani da ih koristite, tako dugo dok znate da se to ne može tako sintetizirati) već je to sam alat u kojem radite razvoj i sintezu.

Iz upravo opisanog razloga niti sljedeći isječak kôda napisanog u VHDL-u koji se u ispitnim sklopovima često koristi za generiranje signala takta nije sintetizabilan.

Listing 2.9 : Nesintetizabilan generator signala takta

```
process
begin
  cp <= '1';
  wait for 50 ns;
  cp <= '0';
  wait for 50 ns;
end process;
```

Naime, rad opisanog sklopa temelji se na pretpostavci da je sintetizator sposoban osigurati upravo navedena kašnjenja, što naravno nije slučaj. Stoga će ovakvi dijelovi kôda često biti korišteni u ispitnim sklopovima koje će simulator moći simulirati, ali ih nikada nećemo imati u opisima sklopova koje fizički treba sintetizirati: programirljivi sklopovi za ovu će svrhu morati imati unaprijed proizveden sklop čija je zadaća generiranje signala takta, ili će u jednostavnijoj varijanti imati zaseban ulaz preko kojeg će korisnik izvana morati dovesti prikladno generiran signal takta (nekim primjerice relaksacijskim oscilatorom ili kvarcnim oscilatorom).