8.

Tcl programski jezik

Tool Command Language, skraćeno Tcl, je interpreterski programski jezik koji se koristi za pristup i upravljanje radom velikog broja različitih razvojnih alata kao i za manipulaciju nad podacima koji se obrađuju i generišu pomoću tih alata. Sam jezik podržava rad sa promenljivim, procedurama i kontrolnim strukturama, kao i većina programskih jezika. Vrlo lako se proširuje, definisanjem novih funkcija, tako da danas predstavlja industrijski standard koji koristi većina proizvođača softverskih alata za dizajn elektronskih kola (*Cadence*, *Synopsys*, *Mentor Graphics*, *Xilinx*, *Altera*, itd.), za kontrolu i proširivanje mogućnosti njihovih alata.

Kompanija Xilinx odabrala je Tcl jezik kao osnovni programski jezik za upravljanje radom Vivado Design Suite alata. Tcl interpreter koji se nalazi unutar Vivado Design Suite alata podržava sve mogućnosti Tcl jezika, što omogućava potpunu kontrolu Vivado Design Suite alata, pristup dizajn objektima i njihovim osobinama unutar Vivado baze podataka, kao i mogućnost kreiranja korisnički definisanih izveštaja. Pomoću Tcl jezika moguće je na vrlo jednostavan način prilagoditi standardni tok razvoja dizajna specifičnostima tekućeg projekta.

Tcl jezik takođe obezbeđuje i ugrađene komande za pristup datotekama iz lokalnog fajl sistema. Na ovaj način moguće je dinamički kreirati nove direktorijume, otvarati ili kreirati nove FPGA projekte, dodavati ili izbacivati izvorne fajlove iz tekućeg projekta, pokretati procese sinteze i implementacije i analizirati dobijene rezultate. Pored standardnih formi izveštaja o rezultatima sprovedenih koraka u procesu razvoja digitalnog sistema pomoću FPGA tehnologije, moguće je generisati i korisnički

definisane izveštaje koje sadrže samo neophodne podatke, organizovane na najoptimalniji način, u skladu sa zahtevima krajnje aplikacije ili standardima koji postoje unutar organizacije u kojoj se realizuje tekući projekat.

Tcl jezik se takođe može koristiti za razvoj novih postupaka u procesu realizacije dizajna, ili za zaobilaženje postojećih problema, ubacivanjem ili izbacivanjem dizajn objekata i modifikovanjem njihovih osobina.

Pored podrške komandama koje predstavljaju standardni deo Tcl programskog jezika, Vivado Design Suite definiše i veliki broj novih Tcl komandi, pomoću kojih je olakšan razvoj FPGA baziranih digitalnih sistema. Neke od ovih komandi biće predstavljene u poglavlju 9.

Prilikom rada sa Vivado Design Suite alatom, većina akcija koje se selektuju korišćenjem grafičkog korisničkog interfejsa zapravo se mapira na odgovarajuću Tcl komandu. Vivado prikazuje Tcl komande, zajedno sa njihovim parametrima, koje trenutno izvršava, u okviru „*Tcl Console*“ prozora lociranog na dnu glavnog prozora Vivado Design Suite alata. Pored ovoga, Vivado Design Suite alat generiše i poseban tekstualni fajl, takozvani žurnal (*journal*) fajl pod nazivom *vivado.jou*, u okviru direktorijuma iz kojega je pokrenut Vivado Design Suite alat. U ovom fajlu nalaze se sačuvane sve Tcl komande, u redosledu kako su bile izvršavane u toku trenutno aktivne Vivado sesije (od trenutka pokretanja Vivado Design Suite alata). Sadržaj ovog žurnal fajla može biti jako koristan kao polazna tačka prilikom razvoja novih Tcl skripti.

Vivado Design Suite alat takođe kreira i log fajl, *vivado.log*, u kojem se nalaze rezultati izvršavanja Tcl komandi koje su bile izvršene. Žurnal i log fajlovi su vrlo korisni prilikom provere koje su sve Tcl komande zapravo bile izvršene, i šta su bili rezultati njihovog izvršavanja.

Za dodatne informacije u vezi Tcl jezika i osnovnih Tcl komandi, zainteresovani čitalac se upućuje na zvanični *Tcl Developer Xchange* websajt, www.tcl.tk. Za detaljna objašnjenja dodatnih Tcl komandi, uvedenih posebno za Vivado Design Suite alat, zainteresovani čitalac se upućuje na odgovarajuću dokumentaciju kompanije Xilinx, koja se može pronaći na zvaničnom websajtu kompanije, www.xilinx.com.

NAPOMENA: Individualne Tcl komande ili Tcl programi, koji se u Tcl terminologiji nazivaju Tcl skriptama, izvršavaju se unutar „*Tcl Console*“ prozora. Tcl komande izvršavaju se tako što se željena Tcl komanda, zajedno sa svim potrebnim parametrima, navede u okviru Tcl prozora i nakon toga se pritisne taster *Enter*. Rezultat izvršavanja komande odmah je vidljiv unutar „*Tcl Console*“ prozora. Tcl skripte takođe se pokreću unutar „*Tcl Console*“ prozora. Prvo je neophodno da se pređe u direktorijum u kome je smeštena Tcl skripta koju želimo pokrenuti. Ovo se može učiniti promenom direktorijuma direktno u „*Tcl Console*“ prozoru (korišćenjem standardnih komandi operativnog sistema, kao što su *dir*, *pwd*, *cd*, itd.). Nakon što smo se pozicionirali unutar direktorijuma u kojem se nalazi Tcl skripta koju želimo pokrenuti, Tcl skriptu pokrećemo navođenjem sledeće komande unutar „*Tcl Console*“ prozora

source *ime_skripte.tcl*

8.1. Osnove Tcl programskog jezika

U okviru ovog poglavlja biće predstavljene najosnovnije jezičke konstrukcije Tcl programskog jezika. Za više informacija u vezi Tcl programskog jezika zainteresovani čitalac upućuje se na neku od mnogobrojnih knjiga posvećenih Tcl jeziku, ili na neki od websajtova, na primer www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html.

Tcl komanda predstavlja string, sastavljen od reči razdvojenih razmakom ili tabom koji se završava ; karakterom ili novim redom. Tcl interpreter razbija liniju na sastavne reči i vrši evaluaciju komandi ili zamene promenljivih njihovim vrednostima. Tcl interpreter interpretira liniju sa leva na desno, evaluira svaku reč na koju naiđe u potpunosti, pre nego što pređe na sledeću reč.

8.1.1. Deklaracija promenljivih i dodela vrednosti

U Tcl jeziku svi objekti reprezentovani su u formi liste, iako ih Tcl interpreter interno može reprezentovati u obliku koji obezbeđuje najefikasnije izvršavanje Tcl programa.

Tcl komanda za deklaraciju promenljive i dodelu vrednosti promenljivoj je *set* komanda. Sintaksno pravilo *set* Tcl komande ima sledeći oblik

set *ime_promenljive* [*vrednost*]

Argumenti *set* komande imaju sledeće značenje:

- *ime_promenljive* – ime promenljive koja se kreira ili čija vrednost se želi pročitati. Ukoliko ime promenljive sadrži samo alfanumeričke karaktere bez zagrada, reč je o skalarnoj promenljivoj. Ukoliko ime promenljive ima format *ime_promenljive(indeks)* reč je asocijativnom nizu.
- *vrednost* – vrednost koja se želi dodeliti promenljivoj *ime_promenljive*. Vrednost može biti neposredno navedena vrednost, ili rezultat izvršavanja neke druge Tcl komande koja vraća vrednost.

Na primer, sledeće *set* komande kreiraju dve promenljiva, *A* i *B*. Promenljivoj *A* dodeljuje se vrednost stringa“Ovo je string“, a promenljivoj *B* numerička vrednost 3.14.

```
set A “Ovo je string“
set B 3.14
```

Ukoliko se želi koristiti tekuća vrednost promenljive, potrebno je koristiti znak $ ispred imena promenljive. Na primer, u sledećem Tcl programu deklarišu se promenljive *A* i *B*, a zatim se ispisuju njihove vrednosti korišćenjem *puts* Tcl komande

```
set A "Ovo je string "
set B 3.14

puts $A
puts $B

puts "............................."

set poruka "Vrednost promenljive B je: "
puts "$poruka $B"
```

8.1.2. Evaluacija i supstitucija

U Tcl evaluacija komandi odvija se u dve faze. U prvoj fazi vrše se potrebne zamene argumenata njihovim stvarnim vrednostima. U drugoj fazi vrši se evaluacija rezultujuće komande. Ovim je završen proces izvršavanja Tcl komande. Na primer, u Tcl komandi

puts $prom

u prvoj fazi se *$prom* argument zamenjuje sa stvarnom vrednošću promenljive *prom*, a zatim se izvršava *puts* komanda.

Korišćenjem dvostrukih navodnika “”, uglatih zagrada [] ili vitičastih zagrada {}, moguće je izvršiti dodatne operacije u fazi zamene argumenata.

Grupisanjem reči unutar dvostrukih navodnika “” omogućava se izvođenje zamena unutar navodnika. Kada se izvrše sve zamene unutar navodnika, rezultat se zatim prosleđuje kao jedan argument. Na primer, prilikom izvršavanja sledeće Tcl komande

puts “Trenutna vrednost promeljive je $prom”

prvo se *$prom* argument zamenjuje sa stvarnom vrednošću promenljive *prom*, koja se koristi unutar navodnika i postaje deo stringa, a zatim se čitav string ispisuje na konzolu.

Grupisanjem reči unutar vitičastih zagrada {} **onemogućuje** se supstitucija bilo kog argumenta njegovom stvarnom vrednošću.

Sledeća Tcl skripta ilustruje razliku između grupisanja objekata pomoću dvostrukih navodnika i vitičastih zagrada.

```
set grad {Novi Sad}
set tekst "Glavni grad Vojvodina je: "

puts "$tekst $grad"
puts {$tekst $grad}
```

Ukoliko se želi koristiti rezultat izvršavanja jedne Tcl komande kao argument druge komande, to se može postići uokvirivanjem prve komande u uglate zagrade. Ukoliko Tcl interpreter prilikom faze zamene naiđe na deo koji je oivičen uglatim zagradama, onda se uokvireni deo evaluira kao zasebna komanda od strane Tcl interpretera, a rezultat evaluacije zamenjuje deo oivičen uglatim zagradama. Sledeći primer ilustruje ovaj proces.

```
set x abc
puts "Jednostavna zamena: $x\n"

set y [set x "def"]
puts "Set komanda vraća novu vrednost promenljive: X: $x Y: $y\n"

set z {[set x "Ovo je string unutar navodnika unutar uglatih zagrada"]}
puts "Obratite pažnju na vitičaste zagrade: $z\n"

set a "[set x {Ovo je string unutar uglatih zagrada unutar navodnika}]"
puts "Rezultat izvrsavanja prethodne set komande: $a"
puts "\$x je: $x\n"
```

8.1.3. Izvođenje matematičkih operacija

Tcl komanda za izvođenje matematičkih operacija je *expr*. *Expr* komanda uzima sve svoje argumente, evaluira ih kao Tcl izraz (za razliku od normalne komande), i vraća rezultat evaluacije izraza. Operatori koji su dozvoljeni unutar Tcl izraza uključuju standardne aritmetičke operatore, logičke operatore, bitske operatore, kao i matematičke funkcije kao što su *rand(), sqrt(), cosh(),* itd. Izrazi gotovo uvek rezultuju o odgovarajućoj numeričkoj vrednosti (celobrojnoj ili realnoj). Radi poboljšanja performansi prilikom evaluacije izraza, poželjno ih je uokviriti vitičastim zagradama. Na primer, umesto pisanja *expr $i * 10,* bolje je napisati *expr {$i * 10}.*

Tcl izrazi sastoje se iz kombinacije operanada, operatora i zagrada. Razmaci se smeju koristiti za razdvajanje operanada, operatora i zagrada. Oni se ignorišu od strane Tcl interpretera. Kada god je to moguće, operandi se interpretiraju kao celobrojne vrednosti. Celobrojne vrednosti mogu se zadati u decimalnom formatu (prostim navođenjem vrednosti), oktalnom formatu (ukoliko je prvi karakter operanda 0) ili heksadecimalnom formatu (ako su prva dva karaktera operanda 0x). Ukoliko operand nije zadat u jednom od prethodno navedenih formata, on se tretira kao realan broj, ukoliko je to moguće. Realni brojevi mogu se specificirati u bilo kom formatu koji je prihvatljiv za ANSI kompatibilni C kompajler. Ukoliko numerička interpretacija operanda nije moguća, on se interpretira kao string.

Operand može biti specificiran ili kao numerička vrednost, koristeći neki od prethodno navedenih formata, ili kao Tcl promenljiva, koristeći standardnu *$promenljiva* Tcl notaciju. U ovom slučaju u operaciji se koristi vrednost navedene promenljive.

Lista dozvoljenih operatora koji se mogu koristiti u izrazima uključuje:

- **-**, **+**, **~**, **!** : unarni operator negacije, unarni operator sabiranja, operator bitske negacije, operator logičke negacije.
- ****** : operator stepenovanja.
- *****, **/**, **%** : operator množenja, deljenja i ostatka.
- **+**, **-** : operator sabiranja i oduzimanja.
- <<, >> : operatori pomeranja u levo i u desno.
- <, >, <=, >= : relacioni operatori manje od, veće do, manje ili jednako od, veće ili jednako od.
- **eq**, **ne**, **in**, **ni** : operatori poređenja stringova, jednakosti (**eq**), nejednakosti (**ne**), i dva operatora koja proveravaju da li se navedeni string nalazi u listi (**in**) ili ne (**ni**). Svi operatori vraćaju vrednost 0 ili 1.
- **&**, ^, | : bitski logički operatori I, EX-ILI i ILI.
- **&&**, || : logički operatori I i ILI.
- **x?y:z** : if-then-else operator, identičan operatoru koji postoji u C programskom jeziku

Lista dozvoljenih matematičkih funkcija koje se mogu koristiti u izrazima uključuje:

abs	acos	asin	atan	atan2	bool	ceil	cos	cosh	double
entier	exp	floor	fmod	hypot	int	isqrt	log	log10	max
min	pow	rand	round	sin	sinh	sqrt	srand	tan	tanh
wide									

Pored ovih matematičkih funkcija, dozvoljeno je izvođenje Tcl komandi unutar Tcl izraza, kao što je prikazano u sledećem primeru

```
% set x 1
% set w "Abcdef"
% expr { [string length $w]-2*$x }
```

Rezultat izvršavanja prethodnog Tcl programa jednak je 4, jer se prvo izvršava Tcl komanda *length*, koja kao svoj rezultat vraća dužinu stringa *w* (u našem slučaju dužina stringa iznosi 6), a zatim ta numerička vrednost učestvuje u ostatku izraza koji se nakon svih supstitucija svodi na sledeći matematički izraz 6-2*1 = 4.

Tcl jezik takođe podražava sledeće konverzione funkcije:

- **double()** – konverzija iz celobrojnog u realni format.
- **int()** – konverzija iz realnog u celobrojni format, prostim odsecanjem razlomljenog dela realnog broja.

Sledeći primer ilustruje rad sa Tcl izrazima.

```
set X 100
set Y 256
set Z [expr {$Y + $X}]
set Z_LABEL "$Y plus $X je "

puts "$Z_LABEL $Z"
puts "Kvadratni koren broja $Y je [expr { sqrt($Y) }]\n"

puts "Usled redosleda izvođenja operacije \"5 + -3 * 4\"    je: [expr {-3 * 4 + 5}]"
puts "Usled postojanja zagrada                \"(5 + -3) * 4\" je: [expr {(5 + -3) * 4}]"

set A 3
set B 4
puts "Hipotenuza trougla je: [expr {hypot($A,$B)}]"

#
# Trigonometrijske funkcije ocekuju radijane ...
#
set pi6 [expr {3.1415926/6.0}]
puts "Sinus i kosinus od pi/6: [expr {sin($pi6)}] [expr {cos($pi6)}]"

#
# Rad sa nizovima
#
set a(1) 10
set a(2) 7
set a(3) 17
set b     2
puts "Suma: [expr {$a(1)+$a($b)}]"
```

8.1.4. Naredbe grananja

Kao i većina programskih jezika i Tcl jezik sadrži *if* naredbu. Sintaksa *if* Tcl naredbe ima sledeći oblik

if *izraz1* [**then**]
  telo1
elseif *izraz2* [**then**]
  telo2
elsif *izraz3* [**then**]
...
[**else**]
  teloN

Ključne reči *then* i *else* su opcione, kao što se može videti iz sintaksnog pravila. *Telo1*, *telo2*, ..., *teloN* su blokovi Tcl naredbi, koji se izvršavaju u zavisnosti od rezultata evaluacije testova u svakoj od grana *if* naredbe.

Izrazi *izraz1*, *izraz2*, ... *izrazN-1*, predstavljaju testove od čijih ishoda zavisi koji blok Tcl naredbi će biti izvršen. Rezultat evaluacije svakog od ovih izraza mora biti vrednost koja se može interpretirati kao „tačno“ ili „netačno“. Sledeći Tcl kod ilustruje rad sa *if* Tcl naredbom.

```
set x 1

if {$x == 2} {puts "$x je 2"} else {puts "$x nije 2"}

if {$x != 1} {
   puts "$x je != 1"
} else {
   puts "$x je 1"
}

if $x==1 {puts "X ima vrednost 1"}
```

Tcl jezik takođe sadrži i *switch* naredbu grananja. Tcl *switch* naredba slična je C *switch* naredbi ali je fleksibilnija, jer dozvoljava da selektorska promenljiva bude i tipa string, a ne samo celobrojnog tipa, kao što je slučaj u C programskom jeziku. String će biti poređen sa skupom šablona i kada se naiđe na šablon koji se poklapa sa stringom, asocirani blok Tcl komandi će biti izvršen.

Sintaksno pravilo *switch* naredbe ima sledeći oblik

switch *string* *šablon1* *telo1* *šablon2* *telo2* ... *šablonN* *teloN*	ili	**switch** *string* { *šablon1* *telo1* *šablon2* *telo2* ... *šablonN* *teloN* }

Obratite pažnju da se u svakom prolazu kroz *switch* naredbu može izvršiti samo jedno telo, i to ono telo koje pripada prvom šablonu sa kojim se string poklapa. Ako poslednji šablon ima vrednost *default*, on će se poklopiti sa svakim stringom. Ukoliko unutar *switch* naredbe ne postoji *default* šablon, a tekući string se ne poklapa ni sa jednim od postojećih šablona, neće se izvršiti ni jedno telo.

Ukoliko se koristi verzija *switch* naredbe sa vitičastim zagradama, prilikom evaluacije šablona neće se vršiti supstitucije koje postoje. Međutim, postojeća tela sastavljena od Tcl komandi biće evaluirana kao i svake druge Tcl komande, uključujući i potrebne supstitucije.

Sledeći primer ilustruje različite varijante korišćenja Tcl *switch* naredbe.

```
set x "ONE"
set y 1
set z ONE

# Najrazumljivija forma switch komande
switch $x {
   "$z" {
      set y1 [expr {$y+1}]
      puts "POKLAPANJE \$z. $y + $z is $y1"
   }
   ONE {
      set y1 [expr {$y+1}]
      puts " POKLAPANJE ONE. $y + jedan je $y1"
   }
   TWO {
      set y1 [expr {$y+2}]
      puts " POKLAPANJE TWO. $y + dva je $y1"
   }
   THREE {
      set y1 [expr {$y+3}]
      puts " POKLAPANJE THREE. $y + tri je $y1"
   }
   default {
      puts "$x se ne poklapa ni sa jednim sablonom"
   }
}

switch $x "$z" {
   set y1 [expr {$y+1}]
```

```
   puts "MATCH \$z. $y + $z je $y1"
} ONE {
   set y1 [expr {$y+1}]
   puts "MATCH ONE. $y + jedan je $y1"
} TWO {
   set y1 [expr {$y+2}]
   puts "MATCH TWO. $y + dva je $y1"
} THREE {
   set y1 [expr {$y+3}]
   puts "MATCH THREE. $y + tri je $y1"
} default {
   puts "$x se ne poklapa ni sa jednim od tri sablona"
}

switch $x "ONE" "puts ONE=1" "TWO" "puts TWO=2" "default" "puts NO_MATCH"

switch $x \
"ONE"     "puts ONE=1"  \
"TWO"     "puts TWO=2" \
"default"     "puts NO_MATCH";
```

8.1.5. Naredbe ponavljanja

Tcl sadrži dve standardne naredbe ponavljanja, *while* i *for*. Sintaksno pravilo *while* naredbe ima sledeći oblik

while *test*
telo

Sve dok je *test* test *while* naredbe zadovoljen, izvršava se njeno telo *telo*, koje predstavlja blok Tcl naredbi. Korišćenjem *continue* naredbe unutar tela *while* naredbe, prekida se izvršavanje tela *while* naredbe i započinje nova evaluacija testa. Korišćenjem *break* naredbe, prekida se izvršavanje tela *while* naredbe i ona se napušta.

Sledeći primer ilustruje korišćenjem Tcl *while* naredbi.

```
set x 1

# Ovo je normalan način da se napise Tcl while petlja
while {$x < 5} {
   puts "x is $x"
   set x [expr {$x + 1}]
}

puts "Napustena prva while petlja kada je X jednako $x\n"

# Sledeci primer ilustruje razliku izmedju navodjenja test unutar viticastih
# zagrada i ne
# Koliko puta se prolazi kroz while petlju?
# Zašto se tekuca vrednost promenljive X ne stampa u svakom prolazu?
set x 0
```

```
while "$x < 5" {
   set x [expr {$x + 1}]
   if {$x > 7} break
   if "$x > 3" continue
   puts "x je $x"
}

puts "Napustena druga while petlja kada je X jednako $x"
```

Obratite pažnju da se *test* unutar *while* naredbe **obavezno** mora okružiti sa vitičastim zagradama. Ukoliko to nije slučaj, kao u drugoj *while* petlji iz prethodnog primera, zapravo će se vratiti povratni string koji će rezultovati da se zauvek ostane u *while* petlji. U gornjem primeru se iz petlje ipak izlazi, kada promenljiva *x* postane veća od 7, korišćenjem *break* naredbe.

Sintaksno pravilo za Tcl *for* petlju ima sledeći oblik

for *start test next*
telo

Tokom evaluacije *for* naredbe, *start* deo evaluira se samo jednom, na početku izvršavanja petlje. Nakon evaluacije *start* dela evaluira se *test* deo. Ukoliko je rezultat evaluacije *test* dela tačan, izvršava se *telo* petlje. Ukoliko je rezultat evaluacije *test* dela netačan, petlja se završava i napušta. Nakon što se izvrši telo petlje, evaluira se *next* deo. Nakon evaluacije *next* dela, ponovo se evaluira *test* deo, i čitav proces započinje ponovo. Kao što se može videti, struktura Tcl *for* petlje podseća na strukturu C *for* petlje. *Start* deo najčešće se koristi za inicijalizaciju iteratora petlje. *Test* deo služi za proveru uslova pod kojim se završava izvršavanje petlje. *Next* deo najčešće se koristi za uvećavanje vrednosti iteratora petlje.

Kao i u slučaju *while* petlje, *start*, *test* i *next* izraze potrebno je oivičiti vitičastim zagradama.

Unutar tela *for* petlje, mogu se koristi *continue* i *break* naredbe, sa istim efektom kao i u slučaju *while* petlje.

Tcl jezik ima posebnu naredbu pomoću koje se može izvesti uvećavanje iterator petlje za željeni faktor

incr *prom inkrement*

Tcl komanda *incr* uvećava tekuću vrednost promenljive *prom* za vrednost specificiranu pomoću *inkrement* argumenta. Ukoliko se *inkrement* argument izostavi, vrednost promenljive uvećava se za 1.

Sledeći primer ilustruje način korišćenja Tcl *for* naredbi.

```
for {set i 0} {$i < 10} {incr i 2} {
   puts "Vrednost i unutar prve petlje: $i"
}

for {set i 3} {$i < 2} {incr i} {
   puts "Vrednost i unutar druge petlje: $i"
}

puts "Start"
set i 0
while {$i < 10} {
   puts "Vrednost i unutar trece petlje: $i"
   incr i
   puts "Vrednost i nakon inkrementacije: $i"
}

set i 0
incr i
# Ovo je ekvivalentno sa
set i [expr {$i + 1}]
```

8.1.6. Potprogrami

Pomoću *proc* Tcl naredbe može se kreirati nova Tcl komanda. *Proc* naredba se u širem smislu može posmatrati kao način da se u Tcl definišu potprogrami. Sintaksno pravilo *proc* naredbe ima sledeći oblik

proc *ime argumenti*
telo

Kada se *proc* naredba evaluira, kao rezultat kreira se nova naredba sa imenom *ime* i listom argumenata *argumenti*. Prilikom izvršavanja naredbe *ime*, zapravo se izvršava njeno *telo*, kome su prosleđene **vrednosti** argumenata iz tekućeg poziva.

Povratna vrednost nove naredbe može se specificirati korišćenjem *return* naredbe. *Return* naredba vraća svoj argument pozivaocu potprograma. Ukoliko u proceduri ne postoji *return* naredba, pozivaocu se vraća povratna vrednost poslednje izvršene Tcl komande iz tela procedure.

Sledeći primer ilustruje način definisanja i primer korišćenja dve nove komande, prve sa imenom *sum* i sa dva argumenta, *arg1* i *arg2*, i druge sa imenom *for* (obratite da Tcl jezik nema rezervisane reči, tako da se ime *for* može koristiti i kao ime procedure) i tri argumenta, *a*, *b* i *c*.

```
proc sum {arg1 arg2} {
   set x [expr {$arg1 + $arg2}];
   return $x
}

puts "Suma 2 + 3 je: [sum 2 3]\n\n"
```

```
proc for {a b c} {
   puts "For naredba je zamenjena sa puts komandama!";
   puts "Argumenti koji su prosledjeni su: $a\n$b\n$c\n"
}

for {set i 1} {$i < 10} {incr i}
```

8.1.7. Složene strukture podataka

Pored skalarnih tipova podataka, Tcl podržava rad i sa složenijim tipovima podataka od kojih se najčešće koriste liste i asocijativni nizovi.

Tcl lista predstavlja kolekciju objekata: brojeva, reči, stringova, ili drugih listi. Lista je osnovna struktura podataka unutar Tcl jezika, čak se i svaka komanda može posmatrati kao lista, u kojoj je prvi član liste ime procedure koju je potrebno izvršiti, a preostali članovi su zapravo argumenti komande.

Tcl lista se može kreirati na nekoliko načina:

- Dodeljujući promenljivoj koja se kreira listu vrednosti

  set lista {{element 1} {element 2} {element 3}}

- Korišćenjem *split* komande

  set lista [**split** “element 1.element 2.element 3“ “.“]

- Korišćenjem *list* komande

  set lista [**list** “element 1“ “element 2“ “element 3“]

Pristup elementima liste može se ostvariti korišćenjem *lindex* Tcl komande

lindex *list index*

Lindex komanda vraća vrednost elementa liste *list* sa indeksom *index*. Lista startuje sa indeksom 0, tako da prvi element u listi ima indeks 0, a ne 1.

Pomoću *llength list* Tcl komande moguće je dobiti podatak o broju elemenata u listi *list*.

Ukoliko se želi proći kroz sve elemente liste, može se koristiti *foreach* Tcl komanda, sa sledećim sintaksnim pravilom

foreach *var list*
telo

Tcl komanda *foreach* izvršiće svoje telo po jednom za svaki element liste *list*. U svakoj iteraciji vrednost promenljive *var* biće jednaka vrednosti sledećeg elementa liste *list*.

Sledeći primer ilustruje rad sa Tcl listama.

```
set x "a b c"
puts "Element sa indeksom 2 liste {$x} je: [lindex $x 2]\n"

set y [split 11/11/1918 "/"]
puts "Dan primirja je [lindex $y 1]-ti dan [lindex $y 0]-og meseca u godini\n"

set z [list puts "arg 2 is $y" ]
puts "Komanda podseca: $z\n"

set i 0
foreach j $x {
   puts "$j je vrednost elementa sa indeksom $i u listi x"
   incr i
}
```

Sledeće Tcl komande mogu se koristiti za dodavanje i uklanjanje elemenata iz liste, pretragu i sortiranje elemenata liste: *concat, lappend, linsert, lreplace, lset, lserach, lsort*.

Za razliku od standardnih programskih jezika (kao što su C, BASIC, FORTRAN, Java, itd.), koji podržavaju rad sa nizovima kod kojih vrednost indeksa niza mora biti ceo broj, Tcl, kao i većina ostalih skripting jezika (kao što su Perl, Python, PHP, itd.) omogućava rad sa asocijativnim nizovima (poznatim i pod nazivom heš tabele, *hash tables*), kod kojih je vrednost indeksa niza string.

Sintaksno pravilo za deklaraciju asocijativnog niza zahteva da se vrednost indeksa tekućeg elementa niza stavi u zagrade, kao što je prikazano u sledećem primeru

```
set name(first) "Mary"
set name(last)  "Poppins"

puts "Full name: $name(first) $name(last)"
```

Osim običnog pristupa elementima niza navođenjem njihovog indeksa, Tcl jezik pruža nekoliko komandi za manipulaciju nizovima:

- **array exists** *ime_niza* – vraća vrednost 1 ako je *ime_niza* promenljiva tipa niz, u protivnom vraća 0.
- **array names** *ime_niza* <*šablon*> - vraća listu indeksa koji postoje u nizu *ime_niza*. Ukoliko je opcioni argument *šablon* prisutan, lista sadrži samo indekse čije ime u sebi sadrži *šablon*.
- **array size** *ime_niza* – vraća broj elemenata niza *ime_niza*.

Sledeći Tcl program ilustruje rad sa asocijativnim nizovima.

```
proc addname {first last} {
   # Povezi lokalni objekat name povezan je sa globalnim objektom name
   global name

   # Kreiraj novi ID (smesten unutar niza imena radi lakseg pristupa)
   incr name(ID)
   set id $name(ID)

   # niz imena je zapravo 2D niz
   set name($id,first) $first
   set name($id,last)  $last
}

# Inicijalizuj niz imena i dodaj nekoliko imena
global name
set name(ID) 0

addname Mary Poppins
addname Uriah Heep
addname Rene Descartes
addname Leonardo "da Vinci"

# Proverimo sadrzaj niza, prikazimo elemente koriscenjem parray Tcl komande
parray name

# Ilustracija koriscenja komandi nad nizovima
array set array1 [list {123} {Abigail Aardvark} \
                  {234} {Bob Baboon} \
                  {345} {Cathy Coyote} \
                  {456} {Daniel Dog} ]

puts "Niz1 ima [array size array1] elemenata\n"
puts "Niz1 ima sledeće indekse: \n [array names array1] \n"
puts "ID broj 123 pripada $array1(123)\n"

if {[array exist array1]} {
   puts "array1 je niz!"
} else {
   puts "array1 nije niz!"
}

if {[array exist array2]} {
   puts "array2 je niz!"
} else {
   puts "array2 nije niz!"
}
```

8.2. Osnove Tcl skripti

Tcl skripta predstavlja seriju Tcl komandi, razdvojenih novim redom i tačka zarez simbolom, ;. Tcl skripta se može posmatrati kao Tcl program, čijim izvršavanjem se postiže željeni rezultat. Pomoću Tcl skripti moguće je objediniti sve korake i sva

podešavanja koja su neophodna za uspešno završavanje razvoja željenog digitalnog sistema pomoću odabrane tehnologije, na jednom mestu.

Do sada smo videli kako se korišćenjem grafičkog korisničkog interfejsa Vivado Design Suite alata može sprovesti postupak realizacije digitalnog sistema pomoću odabrane FPGA komponente. Iako je ovaj način razvoja digitalnog sistema potpuno legitiman, on ima ozbiljni nedostatak da je sam proces realizacije nedovoljno jasan, i kao takav lako podložan greškama. Sva podešavanja opcija različitih alata, sam redosled izvođenja akcija koji dovodi do uspešne implementacije u ovom slučaju nigde nije jasno definisan, pošto se zapravo sastoji iz ručnog izvođenja potrebnih podešavanja, pomoću grafičkog korisničkog interfejsa. Ovakvim pristupom vrlo je lako zaboraviti šta treba i kako treba da bude podešeno, i u kojem redosledu je potrebno izvršiti transformacije dizajna, kako bi se dobili željeni rezultati.

Korišćenjem Tcl skripti prevazilaze se svi ovi problemi. Tcl skripta omogućava da se sva potrebna podešavanja različitih alata, kao i sam redosled izvođenja potrebnih akcija jasno navede kao sekvenca odgovarajućih Tcl komandi. Na ovaj način je tok realizacije digitalnog sistema jasno dokumentovan, i ne može doći do pogrešnih interpretacija, ili zaboravljanja nekog od potrebnih podešavanja, što obezbeđuje ponovljivost i stabilnost krajnjih rezultata procesa realizacije. Upravo iz ovih razloga svi ozbiljni projekti su bazirani na korišćenju Tcl skripti, a ne na korišćenju grafičkog korisničkog interfejsa.

Ne postoji standardna procedura razvoja Tcl skripti. Praksa je da svaka kompanija razvija svoj set Tcl skripti, u okviru kojih je definisan tok razvoja digitalnog sistema koji se primenjuje u datoj kompaniji.

U ovom poglavlju biće prikazana struktura vrlo jednostavne Tcl skripte, pomoću koje je moguće proći kroz sve neophodne korake prilikom realizacije HDL modela digitalnog sistema, korišćenjem Vivado Design Suite alata.

Pošto Tcl skripta na automatizovani način kreira projekat, podešava potrebne opcije različitih alata, generiše međurezultate i smešta ih na odgovarajuće lokacije, i na kraju generiše krajnji rezultat, konfiguracioni fajl, da bi sve ovo teklo bez problema, potrebno je organizovati sve ove podatke u vidu poznate strukture direktorijuma na disku. I ova struktura projektnih fajlova može se birati proizvoljno, tako da unutar svake kompanije postoji definisana specifična struktura fajlova koja se koristi unutar te kompanije, ili za svaki projekat pojedinačno.

U našem primeru, potrebna struktura fajlova prikazana je u sledećoj tabeli, u slučaju projekta „*Naive*“ množača matrica sa AXI interfejsom.

Ime direktorijuma	Opis
matrix_multiply	Osnovni direktorijum čitavog projekta, unutar koga se nalaze svi fajlovi za dati projekat.
release	Direktorijum koji sadrži fajlove koji predstavljaju rezultat procesa implementacije i koji se šalju krajnjem korisniku. U našem slučaju ovde će biti smešten konfiguracioni fajl za programiranje FPGA komponente, ali bi pored njega tu mogli da budu smešteni i dokumenti koji predstavljaju

	uputstvo za korišćenje, itd.
result	Radni direktorijum u okviru koga je smešten Vivado projekat, sa svim automatski generisanim fajlovima u procesu realizacije.
src	Osnovni direktorijum u okviru kojega se smeštaju svi izvorni fajlovi koji čine tekući projekat. Na osnovu ovih fajlova, korišćenjem različitih alata, generišu se svi fajlovi koji će na kraju biti smešteni u *release* direktorijum.
c	Direktorijum u koji se smeštaju C fajlovi. Ovo mogu biti fajlovi koji sadrže algoritamski model digitalnog sistema koji se realizuje (ako se u postupku realizacije koristi sinteza visokog nivoa), ali takođe, na primer, mogu biti i izvorni fajlovi koji definišu drajver (ako je krajnji rezultat realizacije IP jezgro).
script	Direktorijum u kome se nalaze svi Tcl skript fajlovi, koji se koriste u procesu realizacije.
vhdl	Direktorijum u kome se nalaze svi VHDL izvorni fajlovi koji čine model digitalnog sistema koji se realizuje.
tb	Direktorijum u kome se nalaze svi VHDL fajlovi koji čine verifikaciono okruženje za sistem koji se realizuje.
xdc	Direktorijum u kome se nalaze XDC fajlovi sa ograničenjima koja se moraju zadovoljiti u procesu realizacije digitalnog sistema.

Tabela 8.1. Potrebna struktura direktorijuma u slučaju korišćenja Tcl skripte za realizaciju digitalnog sistema

Kada imamo jasno definisanu strukturu direktorijuma, možemo napisati odgovarajuću Tcl skriptu. Tcl skripta sada tačno zna gde se nalaze koji izvorni fajlovi, i gde je potrebno smestiti različite fajlove, koji nastaju u procesu realizacije digitalnog sistema. Primer jedne moguće Tcl skripte, pomoću koje je moguće proći kroz čitav proces realizacije digitalnog sistema pomoću Vivado Design Suite alata, prikazan je u nastavu.

```
# KORAK#1: Definisanje direktorijuma u kojima ce biti smesteni projekat i konfiguracioni fajl
set resultDir ../../result/matrix_multiply
set releaseDir ../../release/matrix_multiply
file mkdir $resultDir
file mkdir $releaseDir

create_project matrix_multiply ../../result/matrix_multiply \
-part xc7z020clg484-1 -force

# KORAK#2: Ukljucivanje svih izvornih fajlova u projekat
add_files -norecurse ../vhdl/utils_pkg.vhd
add_files -norecurse ../vhdl/matrix_mult.vhd
add_files -norecurse ../vhdl/dp_bram.vhd
add_files -norecurse ../vhdl/mem_subsystem.vhd
add_files -norecurse ../vhdl/axi_matrix_multiply_v1_0_S00_AXI.vhd
add_files -norecurse ../vhdl/axi_matrix_multiply_v1_0_S01_AXI.vhd
```

```
add_files -norecurse ../vhdl/axi_matrix_multiply_v1_0.vhd

add_files -fileset constrs_1 ../xdc/matrix_multiply.xdc

update_compile_order -fileset sources_1

# KORAK#3: Pokretanje procesa sinteze
launch_runs synth_1
wait_on_run synth_1
puts "*****************************************************"
puts "*                    Sinteza zavrsena!                    *"
puts "*****************************************************"

# KORAK#4: Pokretanje procesa implementacije i generisanja konfiguracionog fajla
set_property STEPS.WRITE_BITSTREAM.TCL.PRE [pwd]/pre_write_bitstream.tcl [get_runs impl_1]

launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1

puts "*****************************************************"
puts "*                    Implementacija zavrsena!                    *"
puts "*****************************************************"

# KORAK#5: Kopiranje konfiguracionog fajla u release folder
file copy -force ../../result/matrix_multiply/matrix_multiply.runs/impl_1/axi_matrix_multiply_v1_0.bit
../../release/matrix_multiply/matrix_multiply.bit
```

Analizom prethodne Tcl skripte možemo primetiti da se ona sastoji iz nekoliko celina:

- **Korak 1: Kreiranje Vivado projekta** – u prvom koraku podešavaju se putanje do direktorijuma u kojima će biti smešteni konačni fajlovi, koji predstavljaju rezultat realizacije digitalnog sistema (*releaseDir* promenljiva), kao i do direktorijuma u kojem će se nalaziti svi automatski generisani Vivado projektni fajlovi ((*releaseDir* promenljiva). Nakon toga, pomoću *create_project* Tcl komande (koja je specifična za Vivado Design Suite alat) kreira se novi Vivado projekat.
- **Korak 2: Dodavanje svih potrebnih izvornih fajova u projekat** – pomoću *add_files* Tcl komandi (specifičnih za Vivado Design Suite alat), u novokreirani projekat uključuju se svi potrebni izvorni fajlovi (VHDL fajlovi kao i XDC fajlovi) koji čine model digitalnog sistema koji će biti realizovan na odabranoj FPGA komponenti.
- **Korak 3: Izvođenje procesa sinteze** – pomoću *launch_runs synth_1* Tcl komande (specifične za Vivado Design Suite alat), pokreće se postupak sinteze HDL modela digitalnog sistema. Tcl skripta zatim čeka dok Vivado Design Suite alat ne završi postupak sinteze (ovo se postiže korišćenjem *wait_on_run synth_1* Tcl komande), pre nego što se pređe na sledeći korak u procesu realizacije.
- **Korak 4: Izvođenje procesa implementacije i generisanja konfiguracionog fajla** – nakon završenog postupka sinteze, pomoću *launch_runs impl_1* Tcl komande pokreće se proces implementacije dizajna (faze raspoređivanja i

povezivanja) i generisanja konfiguracionog fajla. Korišćenjem još jedne *wait_on_run impl_1* Tcl komande ovaj put se čeka da se proces implementacije završi.

- **Korak 5: Kopiranje konfiguracionog fajla u *release* direktorijum** – nakon što je generisan konfiguracioni fajl, kao poslednji korak u procesu realizacije, on se kopira u *release* folder (originalno se on nalazi unutar *result* foldera u kojem su smešteni svi fajlovi koji se generišu od strane Vivado Design Suite alata). Kopiranje se postiže korišćenjem *file copy* Tcl komande. Ovim je čitav postupak realizacije digitalnog sistema pomoću odabrane FPGA komponente završen.

NAPOMENA: Obratite pažnju da je u koraku 4, pomoću *set_property* Tcl komande specificirana dodatna Tcl skripta (*pre_write_bitstream.tcl*) koju je neophodno izvršiti pre započinjanja faze generisanja konfiguracionog fajla. Sadržaj ove skripte prikazan je u nastavku

```
set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
set_property SEVERITY {Warning} [get_drc_checks UCIO-1]
```

Prethodne dve *set_property* Tcl komande spuštaju nivo ozbiljnosti provera NSTD-1 i UCIO-1 na nivo *Warning*, kako bi se dozvolilo završavanje generisanja konfiguracionog fajla. Ove dve provere Vivado Design Suite alat automatski sprovodi, kako bi proverio da li unutar XDC fajla postoji ograničenje u pogledu lokacije svakog ulaznog i izlaznog porta koji čini interfejs digitalnog sistema koji se implementira, i da li je za svaki od portova definisan I/O standard koji je potrebno koristiti. Ukoliko ovo nije zadovoljeno, proces generisanja konfiguracionog fajla se prekida. U našem primeru unutar XDC fajla (*matrix_multiply.xdc*) ne postoje ove definicije ni za jedan od ulaznih i izlaznih portova (jer to trenutno nije bitno, o tome će biti reči u glavi 9). Kako bi se proces generisanja konfiguracionog fajla ipak sproveo do kraja, moraju se spustiti nivoi ozbiljnosti za ove dve provere na nivo *Warning*. Obratite pažnju da ukoliko bi se stvarno želelo isprogramirati FPGA komponentu sa konfiguracionim fajlom, ova ograničenja bi se morala definisati. Više reči o ovim ograničenjima biće u sledećoj, glavi 9.

9.

Analiza dizajna i definisanje ograničenja

Vrlo važan korak u procesu implementacije dizajna jeste faza analize rezultata, dobijenih nakon procesa elaboracije, sinteze, implementacije i generisanja konfiguracionog fajla. Nakon svakog od ovih procesa Vivado Design Suite alat automatski transformiše tekući model dizajna u novu reprezentaciju. Kako je ovaj proces transformacije automatizovan, neophodno je proveriti da li je on izvršen u skladu sa očekivanjima dizajnera. Vivado Design Suite alat pruža odlične mogućnosti za analizu dizajna u svakoj od faza razvoja, koja se bazira na interakciji sa jednom, jedinstvenom, bazom podataka u kojoj se čuvaju sve informacije o stanju dizajna nakon svake od faza realizacije. Korišćenjem Tcl jezika dizajner može da pretražuje ovu bazu podataka, analizira karakteristike dizajna i vrši potrebne modifikacije, ili definiše instrukcije koje će Vivado Design Suite alat zatim koristiti kako bi generisao željene rezultate.

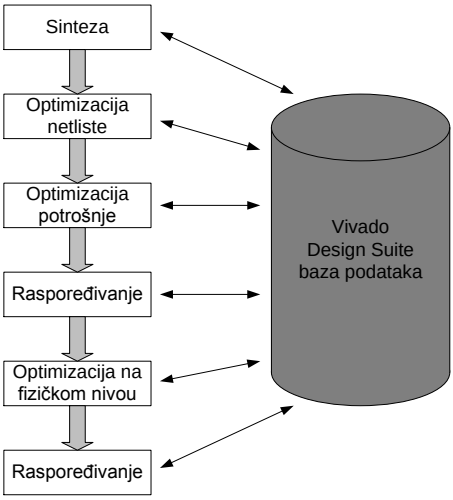
U ovoj glavi biće reči o načinima kako se može sprovesti analiza dizajna pomoću Vivado Design Suite alata, statičkoj vremenskoj analizi, kao i o vrstama i načinima definisanja različitih ograničenja dizajna, pomoću koji dizajner može uticati na način na koji će inicijalni dizajn biti realizovan na odabranoj FPGA komponenti.

9.1. Analiza dizajna pomoću Vivado Design Suite alata

HDL model sistema koji se želi implementirati pomoću FPGA tehnologije prilikom procesa transformacije unutar Vivado Design Suite alata prolazi kroz niz faza (faza

elaboracije, faza sinteze, faza raspoređivanja, faza povezivanja, itd.). Nakon svake od ovih faza, Vivado Design Suite alat generiše novi, unutrašnji model sistema koji se realizuje. Ovi unutrašnji modeli sistema poznati su pod nazivom netliste (*netlist*). Svaki od procesa transformacije vrši neku modifikaciju tekuće netliste ili generiše potpuno novu netlistu. Sve ove informacije Vivado Design Suite alat čuva u okviru jedne, jedinstvene baze podataka, takozvane Vivado Design Suite baze podataka, kojoj zatim pristupaju i ažuriraju različiti procesi, kao što je prikazano na slici 9.1.

Kao što je već rečeno, Vivado Design Suite alat smešta tekući projekat, različite reprezentacije dizajna, kao i informacije o odabranoj FPGA komponenti, u jednu jedinstvenu bazu podataka. Ova baza podataka se zatim koristi prilikom izvršavanja procesa sinteze, implementacije, vremenske analize i generisanja konfiguracionog fajla. Baza podataka se ažurira nakon svakog koraka u procesu realizacije dizajna pomoću ciljne FPGA komponente, tako da u svakom trenutku sadrži najsvežije informacije. Tekući sadržaj baze podataka može se u svakom trenutku zapisani da tvrdi disk, u formi dizajn čekpoint (*Design Checkpoint, .dcp*) fajla.



Slika 9.1. Vivado Design Suite baza podataka i način komunikacije sa bazom od strane različitih Vivado Design Suite procesa

Netlista predstavlja model dizajna, koji je najčešće rezultat izvršavanja nekog od Vivado Design Suite procesa. Netlista se sastoji od ćelija, pinova, portova i mreža.

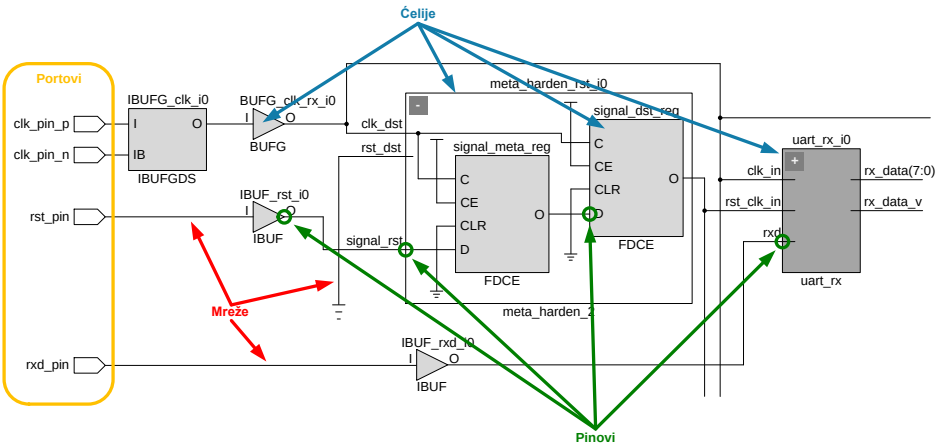
Ćelije predstavljaju dizajn objekte i mogu biti:

- Instance korisnički definisanih modula, odnosno entiteta.
- Instance osnovnih hardverskih resursa koji postoje na odabranoj FPGA komponenti. U okviru Vivado Design Suite alata ove instace poznate su pod

oznakom *BELs* (*Basic Elements*). Neki od standarnih hardverskih resursa su: lookup tabele (*LUTs*), flip flopovi (*FFs*), memorije (*RAMs*), DSP ćelije, itd.

- Takozvane crne kutije, blokovi čija trenutna struktura nije poznata, ali je poznat njihov interfejs sa okruženjem.
- Klok signale, kao posebne vrste mreža, koje će biti posebno diskutovane kasnije.

Pod pinovima smatraju se portovi koji postoje na svakoj od ćelija. Sa druge strane, pod portovima se smatraju samo portovi modela na najvišem nivou hijerarhije. Mreže predstavljaju veze koje spavaju pinove sa drugim pinovima ili portovima. Na slici 9.2 ilustrovani su gradivni objekti netliste na jednom konkretnom primeru.



Slika 9.2. Objekti netliste

Kao što je već rečeno, Vivado Design Suite alat prilikom svog rada generiše čitav niz različitih netlisti, koje predstavljaju opise modela koji se implementira u različitim fazama njegove implementacije. Sve ove netliste, kao i svi objekti od kojih se one sastoje, nalaze se u jednoj, zajedničkoj bazi podataka, Vivado Design Suite bazi podataka. Vivado Design Suite pruža dizajneru mogućnost da analizira sve ove netliste kako bi utvrdio da su procesi transformacije korektno izvedeni, ili eventualno uočio i otklonio nastale greške. Ovaj postupak u procesu implementacije dizajna naziva se faza analize dizajna (*Design Analysis*), i biće detaljnije diskutovan u nastavku.

Vivado Design Suite alat dozvoljava dizajneru da analizira strukturu sledećih netlisti:

- **Netliste koja nastaje nakon procesa elaboracije dizajna** – ova netlista predstavlja reprezentaciju dizajna pre njegove sinteze pomoću RTL kompajlera. Sastoji se iz:

- instanci modula, odnosno entiteta, koje je definisao dizajner u HDL modelu sistema,
- generičkih reprezentacija standardnih kombinacionih i sekvencijalnih mreža (logičke kapije, baferi, multiplekseri, sabirači, komparatori, registri).

Ova netlista zapravo prikazuje kako je Vivado Design Suite alat „razumeo“ HDL model sistema koji se implementira, i preveo ga u mrežu standardnih kombinacionih i sekvencijalnih mreža. Struktura ove netliste može se videti pritiskom na komandu „*Open Elaborate Design*“, u okviru „*RTL Analysis*“ grupe unutar „*Flow Navigator*“ prozora.

- **Netliste koja nastaje nakon procesa RTL sinteze dizajna** – ova netlista predstavlja reprezentaciju dizajna nakon što se izvrši proces RTL sinteze. Sastoji se iz:
  - instanci modula, odnosno entiteta, koje je definisao dizajner u HDL modelu sistema,
  - osnovnih hardverskih resursa prisutnih na odabranoj FPGA komponenti: lookup tabela, flip flopova, multipleksera, BlockRAM memorija, DSP blokova, elemenata od kojih se formiraju klok mreže (BUFG, BUFR, MMCM), ulazno/izlaznih elemenata (IBUF, OBUF, I/O flip flopovi), itd.

  Struktura ove netliste može se videti pritiskom na komandu „*Open Synthesized Design*“, u okviru „*Synthesis*“ grupe unutar „*Flow Navigator*“ prozora.

- **Netliste koja nastaje nakon procesa implementacije** (raspoređivanja i povezivanja) dizajna – ova netlista predstavlja reprezentaciju dizajna nakon što je završen proces pakovanja, raspoređivanja i povezivanja. Kod ove netliste sve ćelije imaju tačno određenu poziciju na FPGA komponenti, a sve mreže su mapirane u odgovarajuće kanale za rutiranje. Struktura ove netliste može se videti pritiskom na komandu „*Open Implemented Design*“, u okviru „*Implementation*“ grupe unutar „*Flow Navigator*“ prozora.

Postoji veliki broj razloga zbog kojih postoji potreba za pristupom individualnim objektima, ili grupama objekata, koji čine netlistu, a nalaze se unutar Vivado Design Suite baze podataka:

- Definisanje vremenskih i lokacijskih ograničenja za dizajn,
- Generisanje kastomizovanih vremenskih izveštaja,
- Pronalaženje objekata unutar šematskog prikaza netliste,
- Generisanje kastomizovanih šematskih prikaza,
- Definisanje i modifikacija različitih osobina (*Property*) objekata, kao što su različiti atributi, I/O naponski standardi, itd.,

- Ručno raspoređivanje objekata po FPGA komponenti,
- Izvršavanju manjih modifikacija na transformisanom dizajnu (*Engineering Change Orders*, ECOs), koje predstavljaju manje funkcionalne promene na dizajnu koji je raspoređen i povezan.

Većina ovih akcija može se unutar Vivado Design Suite alata ostvariti korišćenjem grafičkog korisničkog interfejsa i odgovarajućih komandi, ali je daleko moćniji način za njihovo sprovođenje korišćenjem odgovarajućih Tcl komandi, podržanih od strane Vivado Design Suite alata.

Korišćenjem Tcl komadni može se ostvariti interakcija sa Vivado Design Suite bazom podataka, mogu se selektovati objekti unutar baze podataka, čitati i modifikovati njihove osobine, a sami objekti se lako mogu koristiti unutar složenijih Tcl skripti. Svaki od objekata koji se nalazi unutar baze podataka ima niz osobina (*Property*), čije vrednosti je moguće pročitati, a ponekad i modifikovati, pomoću odgovarajućih Tcl komandi.

Najčešće korišćene Tcl komande koje se koriste u svrhu pristupa i modifikacije objekata unutar Vivado Design Suite baze podataka su:

- **Komande za selekciju odgovarajućih objekata** – *get_cells, get_nets, get_ports, get_pins*
- **Komande za selekciju i modifikaciju osobina objekata** – *report_property, list_property_value, get_property, set_property*

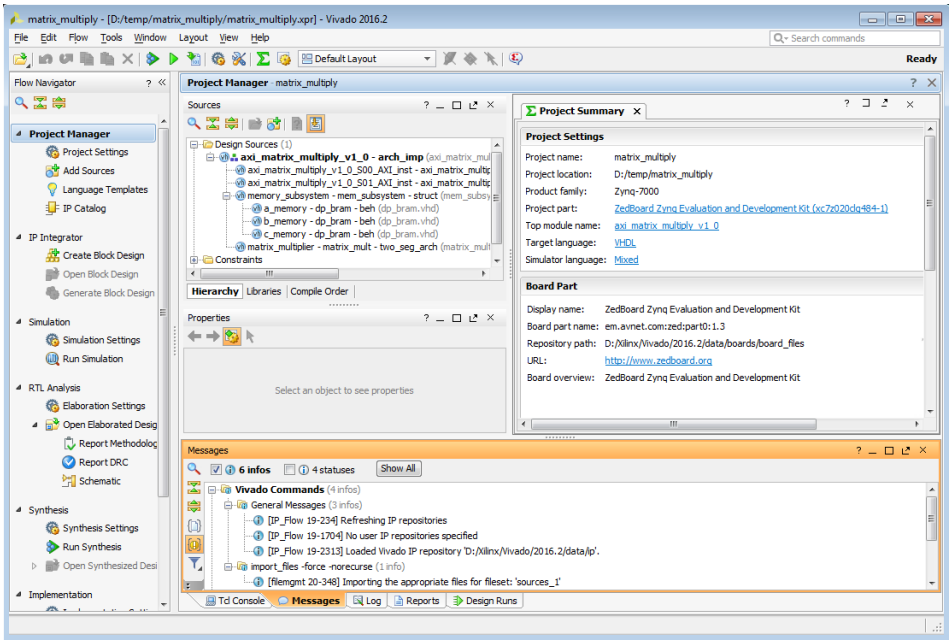
Pre nego što pređemo na detaljno objašnjavanje korišćenja ovih Tcl komandi, potrebno je ući u Vivado Design Suite bazu podataka i otvoriti odgovarajuću netlistu. U našem primeru otvorićemo netlistu IP jezgra sa AXI interfejsom „*Naive*“ množača matrica, razvijenog u glavi 6, koja nastaje nakon elaboracije HDL modela.

Prvi korak jeste da se u okviru Vivado Design Suite alata kreira novi projekat, i da se u njega uključe svi VHDL modeli koji čine IP jezgro sa AXI interfejsom „*Naive*“ množača matrica. Sprovodeći postupak opisan u poglavljima 1.3 i 1.4, potrebno je kreirati novi projekat, sa imenom *matrix_multiply*, i u njega uključiti sledeće VHDL modele:

- *utils_pkg.vhd* – paket u kome se nalazi definisana *log2c* funkcija, prikazan na strani 155.
- *axi_matrix_multiply_v1_0.vhd* – model čitavog sistema, prikazan na strani 275.
- *axi_matrix_multiply_v1_0_S00_AXI.vhd* – model AXI-*Lite* kontolera, prikazan na strani 261.
- *axi_matrix_multiply_v1_0_S01_AXI.vhd* – model AXI-*Full* kontolera, prikazan na strani 267.

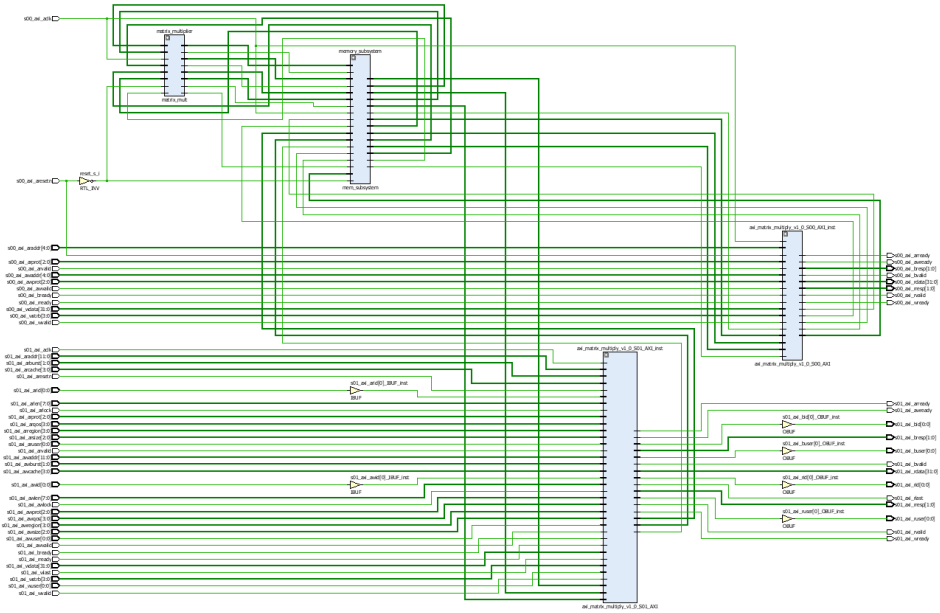
- *mem_subsystem.vhd* – model memorijskog podsistema, prikazan na strani 256.
- *dp_bram.vhd* – model dvopristupne memorije, prikazan na strani 260.
- *matrix_mult.vhd* – RTL model matričnog množača matrica, prikazan na strani 153.

Nakon što je projekat kreiran, izgled glavnog prozora Vivado Design Suite alata trebalo bi da odgovara onom prikazanom na slici 9.3.



Slika 9.3. Vivado Design Suite glavni prozor sa kreiranim matrix_multiply projektom

Pritiskom na komandu „*Open Elaborate Design*“, u okviru „*RTL Analysis*“ grupe, unutar „*Flow Navigator*“ prozora, Vivado Design Suite alat prelazi u režim analize netliste koja nastaje nakon elaboracije HDL modela. Sama netlista prikazuje se u vidu šematskog prikaza, kao što se može videti na slici 9.4.



Slika 9.4. Šematski prikaz netliste koja nastaje nakon elaboracije HDL modela „Naive“ množača matrica

Na slici 9.4 mogu se videti objekti od kojih se sastoji VHDL model IP jezgra „*Naive*“ množača matrica na najvišem nivou hijerarhije:

- Četiri modula od kojih je sastavljen strukturni model IP jezgra:
  - *axi_matrix_multiply_v1_0_S00_AXI_inst*,
  - *axi_matrix_multiply_v1_0_S01_AXI_inst*,
  - *matrix_multiplier*,
  - *memory_subsystem.*
- Odgovarajući broj ulaznih portova, prikazanih na levoj strani šematskog prikaza, i izlaznih portova, prikazanih na desnoj strani šemastkog prikaza.
- Odgovarajući broj bafera.

Svaki od modula prikazan je pomoću pravougaonika na čijim ivicama se nalaze pinovi. U gornjem levom uglu se nalazi znak „+“, koji omogućava da se prikaže unutrašnja struktura odabranog modula, ukoliko nije reč o primitivnom bloku. Portovi čitavog dizajna i pinovi na svakom od modula, povezani su pomoću odgovarajućih mreža, prikazanih zelenim linijama na slici 9.4., čineći tako netlistu koja reprezentuje dizajn.

9.1.1. Tcl komande za selekciju objekata iz baze padataka

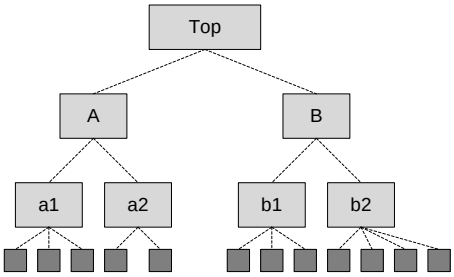
Pre nego što izvršimo bilo kakvu drugu operaciju nad nekim objektom iz Vivado Design Suite baze podataka, moramo ga selektovati. Objekti se mogu selektovati korišćenjem grafičkog korisničkom interfejsa, ili pretragom objekata unutar odgovarajućih *Netlist* prozora, koji se pojavljuju kada se otvore elaborirani, sintetizovani ili implementirani dizajn. Međutim, daleko efikasniji i korisniji način da se ova selekcija izvrši je korišćenjem odgovarajućih Tcl komandi. Vivado Design Suite alat u cilju selekcije objekata iz baze podataka obezbeđuje sledeće Tcl komande, koje će u nastavku detaljnije biti opisane:

- ***get_cells*** – pomoću ove komande moguće je selektovati željenu ćeliju, ili ćelije, iz netliste,
- ***get_nets*** – pomoću ove komande moguće je selektovati željenu mrežu, ili mreže, iz netliste,
- ***get_ports*** – pomoću ove komande moguće je selektovati željeni port, ili portove, iz netliste,
- ***get_pins*** – pomoću ove komande moguće je selektovati željeni pin, ili pinove, iz netliste.

Navedene komande kao rezultat svog izvršavanja vraćaju listu dizajn objekata, sa kojom se zatim može dalje manipulisati pomoću drugih Tcl komandi.

Kao što je poznato, većina dizajna je struktuirana u vidu hijerahijski organizovanih međusobno povezanih blokova, odnosno modula. U procesu analize dizajna, prvo je neophodno locirati dizajn objekat koji je od interesa, selektovati ga, a zatim analizirati i eventualno modifikovati neke od njegovih osobina. Tcl *get_** komande imaju zadatak da lokalizuju traženi objekat unutar dizajna i da ga vrate pozivaocu. Podrazumevani način rada svih *get_** Tcl komandi rezultuje u pretrazi i vraćanju dizajn objekata koji se nalaze na najvišem nivou hijerarhije unutar modela dizajna koji se pretražuje. Korišćenjem *current_instance* <*ime_modula*> Tcl komande pre korišćenja *get_** Tcl komande, moguće je da se prostor pretrage ograniči na odabrani blok/modul unutar hijerahijske strukture modela dizajna. Vraćanje na najviši nivo hijerarhijske strukture postiže se pozivanjem *current_instance* komande bez ikakvog argumenta.

Na primer, na slici 9.5 prikazana je hijerarhijska strutkura modela hipotetičkog dizajna koji se analizira. Moduli *A* i *B* instancionirani su u okviru najvišeg hijerarhijskog nivoa. Modul *A* sastoji se iz dva bloka *a1* i *a2*, dok se modul *B* sastoji iz blokova *b1* i *b2*. Blokovi *a1*, *a2*, *b1* i *b2* instancioniraju samo primitivne hardverske resurse, tako da se hijerarhijska struktura modela hipotetičkog dizajna završava na ovom nivou.



Slika 9.5. Hijerarhijska strutkura modela dizajna koji se analizira

Sledeći Tcl fragment ilustruje kako se korišćenjem *current_instance* Tcl komande može kontrolisati prostor pretrage prilikom korišćenja *get_** Tcl komandi.

```
# Selektuj modul B kao prostor pretrage
current_instance B
# Kao rezultat izvrsavanja sledeće Tcl komande vratiće se lista
# objekata koja sadrži samo objekte b1 i b2, jer su to svi objekti
# koji postoje unutar modula B
get_cells *
# Izvrsavanjem sledeće komande vratiće se lista koja sadrži
# samo mreže koje povezuju objekte b1 i b2
get_nets *
# Resetuj prostor pretrage na najvisi nivo hijerarhije
current_instance
# Izvršavanjem sledeće komande vraća se lista objekata koja
# sadrži objekte A i B, jer su to jedini objekti koji postoje na
# najvišem nivou hijerarhije
get_cells *
```

Iako *get_** komande pretražuju samo najviši hijerahijski nivo od trenutne pozicije u hijerahiji dizajna (što zavisi od toga da li je pre poziva *get_** komande pozvana *current_instance* komanda), moguće je specificirati šablon pretrage koji uključuje hijerahijsko ime objekta (ili objekata) koji se traži, relativno od trenutne pozicije u hijerarhiji dizajna na kojoj se nalazimo. Na primer, da bismo selektovali *b1* objekat iz prethodnog primera, dovoljno je *get_cell* komandu pozvati sa sledećim parametrom

```
get_cells B/b1
```

Ponekad je neophodno pretraživati više hijerarhijskih nivoa istovremeno, počevši od nivoa na kome se trenutno nalazimo. Većina *get_** Tcl komadni ima *–hieararchical* opciju, pomoću koje se od Vivado Design Suite alata traži da prilikom potrage za objektima pretražuje i sve hijerahijske nivoe koji se nalaze od tekućeg nivoa na kome se nalazimo (nivo na kome se nalazimo zavisi od parametra korišćenog prilikom poziva *current_instance* komande, a ako ova komanda nije pozvana onda se nalazimo na najvišem hijerarhijskom nivou).

```
# Sledeća komanda vraća sve objekte koji postoje u modelu dizajna
get_cells –hierarchical *
# Sledeća komanda vraća sve objekte čije ime sadrži string *1*
get_cells –hierarchical *1*
```

NAPOMENA: Kada se koristi *–hieararchical* opcija, Vivado Design Suite alat pokušava da upari specificirani šablon pretrage sa imenima objekata koji postoje na svakom od hijerarhijskih nivoa, a ne sa punim hijerarhijskim imenima objekata. Prilikom korišćenja *–hieararchical* opcije šablon pretrage **ne sme** sadržiti hijerarhijske separatore (simbol /), jer će se u protivnom kao rezultat izvršavanja *get_** Tcl komande vratiti prazna lista. Jedini izuzetak od ovog pravila je slučaj kada je unutar modela dizajna koji se pretražuje uklonjena hijerarhija, jer tada imena objekata u modelu uključuju i hijerarhijske separatore. Sledeće dve *get_cells* Tcl komande ilustruju gorenavedeno ograničenje.

```
# Sledeća komanda vraća praznu listu
get_cells –hierarchical B/*
# Sledeća komanda vraća objekte B/b1 i B/b2
get_cells –hierarchical b*
```

Vrlo često imamo potrebu da koristimo samo podskup iz skupa svih objekata koji predstavljaju rezultat izvršavanja *get_** Tcl komandi. Vivado Design Suite alat nudi opciju *–filter,* pomoću koje je moguće definisati dodatne testove koje moraju ispuniti objekti koji se nalaze u povratnoj listi objekata. *Filter* opcija preko svog argumenta specificira dodatne testove na odabranim osobinama koje moraju ispuniti objekti, originalno vraćeni od strane *get_** Tcl komande, da bi bili uključeni u povratnu listu objekata. Dodatni uslov potrebno je navesti unutar dvostrukih navodnika. U slučaju da dodatni uslov sadrži operacije poređenja nizova, mogu se koristiti sledeći operatori: jednaki (==), različiti (!=), sadrži string (=~), ne sadrži string (!~). U slučaju da dodatni uslov sadrži poređenje numeričkih vrednosti koriste se standardni operatori <, >, <=, >=.
Na primer sledeća *get_pins* Tcl komanda vraća listu pinova koji su ulazni i ne sadrže "RESET" string u svom imenu

```
get_pins –filter {DIRECTION == IN && NAME !~ "RESET*"}
```

U prethodnom primeru testira se vrednost DIRECTION osobine pina, i zahteva se da ona ima vrednost IN. Istovremeno, NAME osobina pina mora ne sme da sadrži "RESET" string. Od svih pinova koji postoje unutar modela dizajna, biće vraćeni samo oni koji zadovoljavaju ova dva dodatna uslova.

U slučaju da osobina objekta koja se testira može da uzme Bulove vrednosti, može se direktno koristiti unutar testa, na primer

```
get_cells –filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

Tcl komanda *get_cells*

Pomoću *get_cells* Tcl komande moguće je dobiti listu ćelija iz Vivado Design Suite baze podataka koji zadovoljavaju uslove navedene u komandi. Uprošćeno sintaksno pravilo za *get_cells* Tcl komandu glasi

get_cells [-hierarchical] [-filter <test>] [-of_objects <args>] [<šablon pretrage>]

Povratna vrednost *get_cells* komande je lista ćelija iz dizajna.

Opcije *get_cells* komande imaju sledeće značenje:

- [*-hierarchical*] – ako je navedena, ova opcija znači da će prilikom pretrage dizajna biti pretraženi i svi hijerarhijski nivoi koji se nalaze ispod tekućeg, sa koga je pozvana komanda. Ukoliko se ne navede, biće vraćena lista ćelija koje se nalaze samo na tekućem nivou hijerarhije.
- [*-filter* <*test*>] – ako je navedena, ova opcija omogućava da se prilikom generisanja povratne liste ćelija u nju smeste ćelije čije ime sadrži <*šablon pretrage*> i ispunjava dodatni test <*test*>, koji predstavlja Bulov izraz sastavljen od elementarnih testova osobina ćelija.
- [*-of_objects* <*args*>] – omogućava da se pretražuju samo ćelije koje su povezane na specificirane pinove, vremenske putanje, mreže, BEL-ove, klok regione, itd. Argument <*args*> *of_objects* opcije mora biti rezultat pozivanja neke of *get_** Tcl komandi, a ne može prosto biti lista objekata. U slučaju kada se koristi ova opcija ne može se koristiti opcija [<*šablon pretrage*>].
- [<*šablon pretrage*>] – string koji se mora nalaziti unutar imena svake ćelije da bi ona bila selektovana. U okviru šablona pretrage može se navesti više od jednog stringa, koji moraju biti navedeni unutar vitičastih zagrada, {}, ili dvostrukih navodnika, “”. Prilikom definisanja stringa mogu se koristiti sledeći džoker simboli:
    - * – zamena za proizvoljni niz karaktera,
    - ? – zamena za proizvoljni karakter,
    - ukoliko se prilikom poziva komande iskoriti –*regexp* opcija onda se umesto džokera mogu koristiti regularni izrazi.

  U slučaju da se ne navede, to je ekvivalentno kao da šablon pretrage ima vrednost *.

U suštini, sve opcije *get_cells* Tcl komande vrše restrikciju liste ćelija koja predstavlja povratnu vrednost komande. Kao što se iz sintaksnog pravila može videti, sve opcije *get_cells* komande su neobavezne. Ukoliko se *get_cells* komanda pozove bez ijednog argumenta, rezultat će biti lista svih ćelija na tekućem nivou hijerarhije. Ukoliko se kao šablon pretrage navede puno ime neke ćelije, povratna vrednost će biti lista koja sadrži

samo tu ćeliju. Ukoliko se unutar šablona pretrage koriste džokeri, povratna vrednost će biti lista ćelija koje zadovoljavaju navedene džokere.

Na primer, u slučaju da analiziramo netlistu koja nastaje nakon elaboracije VHDL modela IP jezgra „*Naive*“ množača matrica, primenom narednih *get_cells* Tcl komandi dobijaju se sledeće liste ćelija.

Primer 9.1: Selekcija svih ćelija na najvišem hijerarhijskom nivou dizajna

Listu svih ćelija na najvišem nivou hijerarhije dobijamo ako unutar „*Tcl Console*“ prozora izvršimo sledeću Tcl komandu

get_cells

Kao rezultat izvršavanja ove komande u „*Tcl Console*“ prozoru dobijamo sledeću listu ćelija

axi_matrix_multiply_v1_0_S00_AXI_inst axi_matrix_multiply_v1_0_S01_AXI_inst matrix_multiplier memory_subsystem reset_s_i s01_axi_arid[0]_IBUF_inst s01_axi_awid[0]_IBUF_inst s01_axi_bid[0]_OBUF_inst s01_axi_buser[0]_OBUF_inst s01_axi_rid[0]_OBUF_inst s01_axi_ruser[0]_OBUF_inst

Unutar liste ćelija možemo prepoznati module za implementaciju:

- AXI-*Lite* interfejsa (*axi_matrix_multiply_v1_0_S00_AXI_inst*),
- AXI-*Full* interfejsa (*axi_matrix_multiply_v1_0_S01_AXI_inst*),
- Memorijskog podsistema (*memory_subsystem*),
- „*Naive*“ algoritma množenja matrica (*matrix_multiplier*)

Pored ovih ćelija lista sadrži i određeni broj hardverskih primitava (BELs), u našem slučaju ulaznih bafera (IBUF) i izlaznih bafera (OBUF).

NAPOMENA: Iz prethodnog primera može se primetiti da Vivado Design Suite alat vraća listu objekata unutar jedne linije, što u znatnoj meri otežava pregledanje liste. Korišćenjem *join* Tcl komande i specijanog karaktera za novi red, \n, možemo kreirati listu kod koje će svaki element biti ispisan u novom redu

join [**get_cells**] \n

Nakon izvršavanja prethodne komande dobijamo sledeću listu

axi_matrix_multiply_v1_0_S00_AXI_inst
axi_matrix_multiply_v1_0_S01_AXI_inst

matrix_multiplier
memory_subsystem
reset_s_i
s01_axi_arid[0]_IBUF_inst
s01_axi_awid[0]_IBUF_inst
s01_axi_bid[0]_OBUF_inst
s01_axi_buser[0]_OBUF_inst
s01_axi_rid[0]_OBUF_inst
s01_axi_ruser[0]_OBUF_inst

Primer 9.2: Selekcija svih ćelija koje se nalaze unutar *memory_subsystem* ćelije

Pretpostavimo da želimo da selektujemo sve ćelije koje se nalaze unutar memorijskog podsistema. Jedan način da se to uradi je da se izvrše sledeće dve Tcl komande. Prvo ćemo da pređemo unutar *memory_subsystem* modula izvršavajući sledeću komandu

current_instance memory_subsystem

Nakon što smo prešli unutar *memory_subsystem* modula, listu svih ćelija koje se nalaze instancionirane unutar njega na najvišem nivou hijerarhije, dobijamo izvršavanjem sledeće Tcl komande

get_cells

Rezultat izvršavanja prethodne komande je sledeća lista

memory_subsystem/GND
memory_subsystem/VCC
memory_subsystem/a_memory
memory_subsystem/b_memory
memory_subsystem/c_memory
memory_subsystem/cmd_s_reg
memory_subsystem/en_a_s_i
memory_subsystem/en_b_s_i
memory_subsystem/en_c_s_i
memory_subsystem/m_s_reg[0]
memory_subsystem/m_s_reg[1]
memory_subsystem/n_s_reg[0]
memory_subsystem/n_s_reg[1]
memory_subsystem/p_s_reg[0]
memory_subsystem/p_s_reg[1]
memory_subsystem/status_s_reg

Isti rezultat dobija se ako na najvišem hijerarhijskom nivou izvršimo sledeću Tcl komandu

get_cells memory_subsystem/*

NAPOMENA: Pre izvršavanja prethodne Tcl komande potrebno je vratiti se iz *memory_subsystem* modula u *top* modul. Ovo se postiže izvršavanje *current_instance* Tcl komande bez ikakvog argumenta.

Primer 9.3: Selekcija svih ćelija koje se nalaze unutar netliste

Korišćenjem *–hierarchical* opcije možemo rekurzivno selektovati sve ćelije koje postoje unutar netliste „*Naive*“ množača matrica, koja nastaje nakon procesa elaboracije VHDL modela. Sledeća Tcl komanda ostvaruje željeni rezultat

get_cells –hierarchical

Primer 9.4: Selekcija svih registara koji se nalaze unutar *memory_subsystem* modula

Korišćenjem opcije *filter*, možemo definisati dodatne uslove koje ćelije moraju da ispune kako bi se našle u povratnoj listi. Na primer, recimo da želimo da dobijemo listu svih registara koji postoje unutar *memory_subsystem* modula. U tu svrhu možemo iskoristiti PRIMITIVE_GROUP osobinu ćelije, koja specificira kom RTL tipu data ćelija pripada. U našem slučaju vrednost ove osobine treba da bude RTL_REGISTER. Kompletna Tcl komanda koja vraća željenu listu ima sledeći oblik

get_cells -filter {PRIMITIVE_GROUP == "RTL_REGISTER"} memory_subsystem/*

Kao rezultat izvršavanja prethodne Tcl komande dobijamo sledeću listu ćelija, koje su sve iz grupe „RTL_REGISTER“.

memory_subsystem/cmd_s_reg
memory_subsystem/m_s_reg[0]
memory_subsystem/m_s_reg[1]
memory_subsystem/n_s_reg[0]
memory_subsystem/n_s_reg[1]
memory_subsystem/p_s_reg[0]
memory_subsystem/p_s_reg[1]
memory_subsystem/status_s_reg

Tcl komanda *get_nets*

Pomoću *get_nets* Tcl komande moguće je dobiti listu mreža iz Vivado Design Suite baze podataka koje zadovoljavaju uslove navedene u komandi. Uprošćeno sintaksno pravilo za *get_nets* Tcl komandu glasi

get_nets [-hierarchical] [-filter <test>] [-of_objects <args>] [<šablon pretrage>]

Povratna vrednost *get_nets* komande je lista mreža iz dizajna.

Opcije *get_nets* komande imaju sledeće značenje:

- [*-hierarchical*] – ako je navedena, ova opcija znači da će prilikom pretrage dizajna biti pretraženi i svi hijerarhijski nivoi koji se nalaze ispod tekućeg, sa koga je pozvana komanda. Ukoliko se ne navede, biće vraćena lista mreža koje se nalaze samo na tekućem nivou hijerarhije.
- [*-filter* <*test*>] – ako je navedena, ova opcija omogućava da se prilikom generisanja liste povratnih mreža u nju smeste mreže čije ime sadrži <*šablon pretrage*> i ispunjavaju dodatni test <*test*>, koji predstavlja Bulov izraz sastavljen od elementarnih testova osobina mreža.
- [*-of_objects* <*args*>] – omogućava da se pretražuju mreže koje su povezane na specificirane ćelije, pinove, portove, klokove, itd. Argument <*args*> *of_objects* opcije mora biti rezultat pozivanja neke of *get_** Tcl komandi, a ne može prosto biti lista objekata. U slučaju kada se koristi ova opcija ne može se koristiti opcija [<*šablon pretrage*>].
- [<*šablon pretrage*>] – string koji se mora nalaziti unutar imena svake mreže da bi ona bila selektovana. U okviru šablona pretrage može se navesti više od jednog stringa, koji moraju biti navedeni unutar vitičastih zagrada, {}, ili dvostrukih navodnika, “”. Prilikom definisanja stringa mogu se koristiti sledeći džoker simboli:
  - * – zamena za proizvoljni niz karaktera,
  - ? – zamena za proizvoljni karakter,
  - ukoliko se prilikom poziva komande iskoriti –*regexp* opcija onda se umesto džokera mogu koristiti regularni izrazi.

  U slučaju da se ne navede, to je ekvivalentno kao da šablon pretrage ima vrednost *.

U suštini, sve opcije *get_nets* Tcl komande vrše restrikciju liste mreža koja predstavlja povratnu vrednost. Kao što se iz sintaksnog pravila može videti, sve opcije *get_nets* komande su neobavezne. Ukoliko se *get_nets* komanda pozove bez ijednog argumenta, rezultat će biti lista svih mreža na tekućem nivou hijerarhije. Ukoliko se kao šablon pretrage navede puno ime neke mreže, povratna vrednost će biti lista koja sadrži samo tu mrežu. Ukoliko se unutar šablona pretrage koriste džokeri, povratna vrednost će biti lista mreža koje zadovoljavaju navedene džokere.

Na primer, u slučaju da analiziramo netlistu koja nastaje nakon elaboracije VHDL modela IP jezgra „*Naive*“ množača matrica, primenom narednih *get_nets* Tcl komandi dobijaju se sledeće liste mreža.

Primer 9.5: Selekcija svih mreža unutar *matrix_multiplier* modula

Sledeća Tcl komanda kao rezultat vraća sve mreže koje postoje unutar *matrix_multiplier* modula, ne uključujući mreže koje se nalaze na nižim hijerarhijskim nivoima unutar ovog modula.

get_nets matrix_multiplier/*

Primer 9.6: Selekcija svih klok mreža na najvišem nivou hijerarhije

Korišćenjem opcije *filter* u okviru *get_nets* komanda mogu se filtrirati samo mreže koje imaju tražene osobine. Na primer, sledeća *get_nets* komanda pomoću *filter* opcije proverava vrednost TYPE osobinu svake mreže koja je prisutna na najvišem nivou hijerarhije u okviru elaborirane netliste „*Naive*“ množača matrica. Samo mreže kod kojih je vrednost ove osobine jednaka stringu „LOCAL_CLOCK“ biće vraćena.

get_nets -filter {TYPE == "LOCAL_CLOCK"}

Osobina TYPE pruža informaciju o tipu mreže kojoj tekuća mreža pripada. Vrednost „LOCAL_CLOCK“ označava da je reč o mreži koja razvodi klok signal. Kao rezultat izvršavanja prethodne Tcl komande dobija se lista svih klok mreža koje postoje unutar navišeg hijerarhijskog nivoa unutar „*Naive*“ množača matrica

s00_axi_aclk
s01_axi_aclk

Primer 9.7: Selekcija svih mreža koje su povezane na *memory_subsystem* modul

Korišćenjem opcije *of_objects* moguće je pretraživati mreže koje su povezane samo sa selektovanim objektima unutar dizajna. Na primer, sledeća Tcl komanda kao rezultat vraća sve mreže koje su vezane za pinove *memory_subsystem* modula

get_nets -of_objects [**get_cells** memory_subsystem]

Pomoću *get_cells* komande selektuje se željeni modul, a zatim se pomoću opcije *of_objects* on prosleđuje kao objekat koji učestvuje u pretrazi mreža.

Primer 9.8: Selekcija svih višebitnih mreža koje su povezane na *matrix_multiplier* modul

Korišćenjem kombinacije *of_objects* i *filter* opcija mogu se definisati vrlo specifični kriterijumi koji mreže moraju zadovoljiti da bi se našle u povratnoj listi. Na primer, sledeća Tcl komanda kao rezultat vraća listu svih višebitnih mreža koje su povezane na *matrix_multiplier* modul

get_nets -of_objects [**get_cells** matrix_multiplier] -filter {BUS_WIDTH > 1}

Pomoću *of_objects* opcije selektuje se samo *matrix_multiplier* modul kao jedini objekat čije mreže će učestovati u procesu selekcije. Ovaj deo komande je sličan komandi iz prethodnog primera. Dodatnom opcijom *filter*, iz skupa svih mreža koje su povezane na *matrix_multiplier* modul, biće selektovane samo one čija BUS_WIDTH osobina ima vrednost veću od 1. BUS_WIDTH osobina govori o tome kolika je širina mreže, izražena u bitima. Kao rezultat izvršavanja prethodne Tcl komande dobija se sledeća lista mreža

```
a_addr_s[0]  a_addr_s[1]  a_addr_s[2]  a_addr_s[3]  a_data_s[0]  a_data_s[1]
a_data_s[2]  a_data_s[3]  a_data_s[4]  a_data_s[5]  a_data_s[6]  a_data_s[7]
b_addr_s[0]  b_addr_s[1]  b_addr_s[2]  b_addr_s[3]  b_data_s[0]  b_data_s[1]
b_data_s[2]  b_data_s[3]  b_data_s[4]  b_data_s[5]  b_data_s[6]  b_data_s[7]
c_addr_s[0]  c_addr_s[1]  c_addr_s[2]  c_addr_s[3]  c_data_s[0]  c_data_s[10]
c_data_s[11]  c_data_s[12]  c_data_s[13]  c_data_s[14]  c_data_s[15]
c_data_s[16]  c_data_s[17]  c_data_s[18]  c_data_s[1]  c_data_s[2]  c_data_s[3]
c_data_s[4]  c_data_s[5]  c_data_s[6]  c_data_s[7]  c_data_s[8]  c_data_s[9]
p_s[0] p_s[1] n_s[0] n_s[1] m_s[0] m_s[1]
```

Tcl komanda *get_ports*

Pomoću *get_ports* Tcl komande moguće je dobiti listu portova iz Vivado Design Suite baze podataka koji zadovoljavaju uslove navedene u komandi. Uprošćeno sintaksno pravilo za *get_ports* Tcl komandu glasi

get_ports [-hierarchical] [-filter <test>] [-of_objects <args>] [<šablon pretrage>]

Povratna vrednost *get_ports* komande je lista portova.

Opcije *get_ports* komande imaju sledeće značenje:

- [-*hierarchical*] – ako je navedena, ova opcija znači da će prilikom pretrage dizajna biti pretraženi i svi hijerarhijski nivoi koji se nalaze ispod tekućeg, sa koga je pozvana komanda. Ukoliko se ne navede, biće vraćena lista portova koji se nalaze samo na tekućem nivou hijerarhije.
- [-*filter* <*test*>] – ako je navedena, ova opcija omogućava da se prilikom generisanja liste povratnih portova u nju smeste portovi čije ime sadrži <*šablon pretrage*> i ispunjava dodatni test <*test*>, koji predstavlja Bulov izraz sastavljen od elementarnih testova osobina portova.
- [-*of_objects* <*args*>] – omogućava da se pretražuju portovi koji su povezani na specificirane mreže, BEL-ove, vremenske putanje, klokove, itd. Argument <*args*> *of_objects* opcije mora biti rezultat pozivanja neke of *get_** Tcl komandi, a ne može prosto biti lista objekata. U slučaju kada se koristi ova opcija ne može se koristiti opcija [<*šablon pretrage*>].
- [<*šablon pretrage*>] – string koji se mora nalaziti unutar imena svakog porta da bi on bio selektovan. U okviru šablona pretrage može se navesti više od jednog stringa, koji moraju biti navedeni unutar vitičastih zagrada, {}, ili

dvostrukih navodnika, “”. Prilikom definisanja stringa mogu se koristiti sledeći džoker simboli:

- * – zamena za proizvoljni niz karaktera,
- ? – zamena za proizvoljni karakter,
- ukoliko se prilikom poziva komande iskoriti –*regexp* opcija onda se umesto džokera mogu koristiti regularni izrazi.

U slučaju da se ne navede, to je ekvivalentno kao da šablon pretrage ima vrednost *.

U suštini, sve opcije *get_ports* Tcl komande vrše restrikciju liste portova koja predstavlja povratnu vrednost. Kao što se iz sintaksnog pravila može videti, sve opcije *get_ports* komande su neobavezne. Ukoliko se *get_ports* komanda pozove bez ijednog argumenta, rezultat će biti lista svih portova na tekućem nivou hijerarhije. Ukoliko se kao šablon pretrage navede puno ime nekog porta, povratna vrednost će biti lista koja sadrži samo taj port. Ukoliko se unutar šablona pretrage koriste džokeri, povratna vrednost će biti lista portova koji zadovoljavaju navedene džokere.

Na primer, u slučaju da analiziramo netlistu koja nastaje nakon elaboracije VHDL modela IP jezgra „*Naive*“ množača matrica, primenom narednih *get_port* Tcl komandi dobijaju se sledeće liste portova.

Primer 9.9: Selekcija svih portova dizajna

Korišćenjem *get_ports* Tcl komande bez ikakvih opcija i šablona pretrage dobija se lista svih portova (ulaznih, izlaznih i ulazno/izlaznih) koji postoje u „*Naive*“ množaču matrica.

```
get_ports
```

Primer 9.10: Selekcija svih ulaznih portova dizajna

Korišćenjem opcije *filter* moguće je izvršiti dodatnu selekciju portova, prema zadatom kriterijumu, i na taj način redukovati povratnu listu. Na primer, sledeća Tcl komanda kao rezultat svog izvršavanja vraća listu ulaznih portova „*Naive*“ množača matrica.

```
get_ports -filter {DIRECTION == "IN"}
```

Osobina DIRECTION govori o tome koji je smer posmatranog porta. Ukoliko je port ulazni, vrednosti osobine DIRECTION biće jednaka IN.

Primer 9.11: Selekcija svih portova koji su povezani na klok mreže unutar dizajna

Kao i kod ranijih komandni, korišćenjem *of_objects* opcije moguće je redukovati skup objekata koji se pretražuju prilikom poziva *get_ports* Tcl komande. Na primer, sledeća Tcl komanda korišćenjem *get_nets* Tcl komande u okviru *of_objects* opcije redukuje skup objekata koji će biti pretraživani prilikom izvršavanja *get_ports* Tcl komande samo na klok mreže.

get_ports -of_objects [**get_nets** -filter {TYPE == "LOCAL_CLOCK"}]

Kao rezultat izvršavanja prethodne Tcl komande dobiće se lista portova na koje su povezane klok mreže, drugim rečima dobiće se lista svih klok portova „*Naive*“ množača matrica.

s00_axi_aclk
s01_axi_aclk

Primer 9.12: Selekcija svih jednobitnih izlaznih portova dizajna

Korišćenjem *filter* opcije mogu se definisati dodatni uslovi koje portovi moraju da zadovolje kako bi se našli u povratnoj listi objekata. Sledeća komanda selektuje samo izlazne jednobitne portove „*Naive*“ množača matrica.

get_ports -filter {BUS_WIDTH == 1 && DIRECTION == "OUT"}

Kao što se analizom ove Tcl komande može videti, uslov unutar *filter* opcije može da bude složen, sastavljen od logičke kombinacije elementarnih uslova. U ovom primeru potrebno je selektovati samo jednobitne portove (to se radi prvim elementarnim uslovom koji proverava da li je vrednost osobine BUS_WIDTH jednaka 1), koji su istovremeno ulazni portovi (to se radi pomoću drugog elementarnog uslova, koji proverava da li je vrednost osobine DIRECTION jednaka stringu „OUT“). Tek ako su oba uslova zadovoljena, dotični port smešta se u povratnu listu. U slučaju „*Naive*“ množača matrica povratna lista sastoji se iz sledećih portova

s01_axi_bid[0]
s01_axi_buser[0]
s01_axi_rid[0]
s01_axi_ruser[0]

Tcl komanda *get_pins*

Pomoću *get_pins* Tcl komande moguće je dobiti listu pinova iz Vivado Design Suite baze podataka koji zadovoljavaju uslove navedene u komandi. Uprošćeno sintaksno pravilo za *get_pins* Tcl komandu glasi

get_pins [-hierarchical] [-filter <test>] [-of_objects <args>] [<šablon pretrage>]

Povratna vrednost *get_pins* komande je lista pinova.

Opcije *get_pins* komande imaju sledeće značenje:

- [-*hierarchical*] – ako je navedena, ova opcija znači da će prilikom pretrage dizajna biti pretraženi i svi hijerarhijski nivoi koji se nalaze ispod tekućeg, sa koga je pozvana komanda. Ukoliko se ne navede, biće vraćena lista pinova koji se nalaze samo na tekućem nivou hijerarhije.
- [-*filter* <*test*>] – ako je navedena, ova opcija omogućava da se prilikom generisanja liste povratnih pinova u nju smeste pinovi čije ime sadrži <*šablon pretrage*> i ispunjava dodatni test <*test*>, koji predstavlja Bulov izraz sastavljen od elementarnih testova osobina pinova.
- [-*of_objects* <*args*>] – omogućava da se pretražuju pinovi koji su povezani na specificirane ćelije, klokove, vremenske putanje, mreže, itd. Argument <*args*> *of_objects* opcije mora biti rezultat pozivanja neke of *get_** Tcl komandi, a ne može prosto biti lista objekata. U slučaju kada se koristi ova opcija ne može se koristiti opcija [<*šablon pretrage*>].
- [<*šablon pretrage*>] – string koji se mora nalaziti unutar imena svakog pina da bi on bio selektovan. U okviru šablona pretrage može se navesti više od jednog stringa, koji moraju biti navedeni unutar vitičastih zagrada, {}, ili dvostrukih navodnika, “”. Prilikom definisanja stringa mogu se koristiti sledeći džoker simboli:
    - * – zamena za proizvoljni niz karaktera,
    - ? – zamena za proizvoljni karakter,
    - ukoliko se prilikom poziva komande iskoriti –*regexp* opcija onda se umesto džokera mogu koristiti regularni izrazi.

  U slučaju da se ne navede, to je ekvivalentno kao da šablon pretrage ima vrednost *.

U suštini, sve opcije *get_pins* Tcl komande vrše restrikciju liste pinova koja predstavlja povratnu vrednost. Kao što se iz sintaksnog pravila može videti, sve opcije *get_pins* komande su neobavezne. Ukoliko se *get_pins* komanda pozove bez ijednog argumenta, rezultat će biti lista svih pinova na tekućem nivou hijerarhije. Ukoliko se kao šablon pretrage navede puno ime nekog pina, povratna vrednost će biti lista koja sadrži samo taj pin. Ukoliko se unutar šablona pretrage koriste džokeri, povratna vrednost će biti lista pinova koji zadovoljavaju navedene džokere.

Na primer, u slučaju da analiziramo netlistu koja nastaje nakon elaboracije VHDL modela IP jezgra „*Naive*“ množača matrica, primenom narednih *get_pins* Tcl komandi dobijaju se sledeće liste pinova.

Primer 9.13: Selekcija svih pinova *matrix_multiply* modula

Selekcija svih pinova koji čine interfejs *matrix_multiply* modula može se ostvariti korišćenjem sledeće *get_pins* Tcl komande.

get_pins matrix_multiplier/*

Primer 9.14: Selekcija svih izlaznih pinova *memory_subsystem* modula

Korišćenjem opcije *filter* mogu se filtrirati selektovani pinovi, prema zadatom kriterijumu. Na primer, pomoću sledeće komande moguće je selektovati samo izlazne pinove *memory_subsystem* modula.

get_pins -filter {DIRECTION == OUT} memory_subsystem/*

Primer 9.15: Selekcija svih ulaznih magistrala čija je širina jednaka 8 bita *matrix_multiply* modula

Korišćenjem kompozitnih testova unutar *filter* opcije moguće je definisati složene uslove koje pinovi moraju da ispune da bi se našli u povratnoj listi objekata *get_pins* Tcl komande. Pomoću sledeće komande selektuju se samo 8-bitni ulazni pinovi *matrix_multiplier* modula.

get_pins -filter {DIRECTION == IN && BUS_WIDTH == 8} matrix_multiplier/*

Test DIRECTION osobine selektuje samo ulazne pinove, a test BUS_WIDTH osobine pinove koji su 8-bitni. Konjunkcijom ova dva test selektuju se samo 8-bitni ulazni pinovi. Šablon pretrage definiše da se prilikom ovog dodatnog testiranja u obzir trebaju uzimati samo pinovi čije ime započinje stringom „matrix_multiplier/“.

Primer 9.16: Selekcija svih pinova koji su povezani na klok signale unutar dizajna

Korišćenjem kobinacije *of_objects* i *filter* opcija može se dodatno suziti prostor pretrage. Na primer, korišćenjem sledeće *get_pins* Tcl komande moguće je selektovati sve pinove modula koji se nalaze na najvišem hijerarhijskom nivou modela „*Naive*“ množača matrica koji su povezani sa svim klok signalima koji postoje na najvišem hijerarhijskom nivou modela „*Naive*“ množača matrica.

get_pins -of_objects [**get_nets** -filter {TYPE == "LOCAL_CLOCK"}]

Pomoću *get_nets* Tcl komande prvo se selektuju samo klok mreže koje postoje na najvišem hijerarhijskom nivou modela „*Naive*“ množača matrica. Zatim se unutar ove liste objekata pronalaze svi pinovi na koje su ove mreže povezane. U slučaju „*Naive*“ množača matrica povratna lista trebalo bi da sadrži sledeće pinove

axi_matrix_multiply_v1_0_S00_AXI_inst/S_AXI_ACLK
matrix_multiplier/clk
memory_subsystem/clk
axi_matrix_multiply_v1_0_S01_AXI_inst/S_AXI_ACLK

Zadaci za vežbu

Zadatak 9.1.

Unutar sintetizovanog dizajna „*Naive*“ množača matrica selektovati sve flip flopove koji čine registar stanja unutar *matrix_multiplier* modula (signal *state_reg*). Prilikom selekcije potrebno je ispitati da li je vrednost osobine PRIMITIVE_GROUP jednaka “FLOP_LATCH“.

Zadatak 9.2.

Unutar sintetizovanog dizajna „*Naive*“ množača matrica selektovati sve mreže koje su povezane na flip flopove koji čine registar stanja unutar *matrix_multiplier* modula (signal *state_reg*).

Zadatak 9.3.

Unutar sintetizovanog dizajna „*Naive*“ množača matrica selektovati sve mreže koje su povezane na izlazne pinove flip flopova koji čine registar stanja unutar *matrix_multiplier* modula (signal *state_reg*).

Zadatak 9.4.

Unutar sintetizovanog dizajna „*Naive*“ množača matrica selektovati sve pinove koji su povezani na mreže koje polaze od izlaznih pinova flip flopova koji čine registar stanja unutar *matrix_multiplier* modula (signal *state_reg*).

9.1.2. Tcl komande za selekciju i modifikaciju osobina objekata

Nakon što smo selektovali željeni objekat, ili objekte, iz Vivado Design Suite baze podataka možemo ga koristiti. Jedan od najčešćih način korišćenja jeste da se selektuju, provere i eventualno modifikuju neke od osobina (*Properties*) koje su definisane za odabrani objekat. Osobine objekata se mogu selektovati korišćenjem grafičkog korisničkom interfejsa, pretragom i selekcijom objekta unutar odgovarajućih *Netlist* prozora, i inspekcijom i modifikacijom raspoloživih osobina unutar *Properties* kartice, koje se pojavljuju kada se otvore elaborirani, sintetizovani ili implementirani dizajn. Međutim, daleko efikasniji i korisniji način da se ove operacije izvrše je korišćenjem

odgovarajućih Tcl komandi. Vivado Design Suite alat obezbeđuje sledeće Tcl komande za manipulaciju osobinama objekata, koje će u nastavku detaljnije biti opisane:

- ***report_property*** – pomoću ove komande moguće je izlistati sve osobine, njihov tip i tekuću vrednost, koje su definisane za odabrani objekat iz baze podataka,
- ***list_property_value*** – pomoću ove komande moguće je izlistati sve dozvoljene vrednosti za odabranu osobinu odabranog objekta iz baze podataka,
- ***get_property*** – pomoću ove komande moguće je dobiti trenutnu vrednost selektovane osobine odabranog objekta iz baze podataka,
- ***set_property*** – pomoću ove komande moguće je dodeliti novu vrednost za selektovanu osobinu odabranog objekta iz baze podataka.

Tcl komanda *report_property*

Pomoću *report_property* Tcl komande moguće je dobiti listu svih osobina, koje su definisane za odabrani objekat iz Vivado Design Suite baze podataka. Za svaku osobinu prikazano je kojem tipu pripada i koja je njena trenutna vrednost. Uprošćeno sintaksno pravilo za *report_property* Tcl komandu glasi

report_property [<*-all*>] [<*objekat*>] [<*šablon pretrage*>]

Povratna vrednost *report_property* komande je lista osobina.

Opcije *report_property* komande imaju sledeće značenje:

- [*-all*] – ako je navedena, ova opcija znači da će biti ispisane sve osobine koje postoje za odabrani objekat, iako neke od njih možda nemaju postavljene vrednosti u tekućem objektu.
- [<*objekat*>] – specificira konkretni objekat čija lista osobina se zahteva. Objekat mora biti jedinstven, ako se prosledi više od jednog objekta dobija se poruka o grešci. Za specifikaciju objekta potrebno je koristiti *get_** Tcl komande.
- [<*šablon pretrage*>] – string koji se mora nalaziti unutar imena oobine da bi ona bila selektovana. U okviru šablona pretrage može se navesti više od jednog stringa, koji moraju biti navedeni unutar vitičastih zagrada, {}, ili dvostrukih navodnika, “”. Prilikom definisanja stringa mogu se koristiti sledeći džoker simboli:
    - * – zamena za proizvoljni niz karaktera,
    - ? – zamena za proizvoljni karakter,
    - ukoliko se prilikom poziva komande iskoriti –*regexp* opcija onda se umesto džokera mogu koristiti regularni izrazi.

U slučaju da se ne navede, to je ekvivalentno kao da šablon pretrage ima vrednost *.

Na primer, u slučaju da analiziramo netlistu koja nastaje nakon elaboracije VHDL modela IP jezgra „*Naive*“ množača matrica, primenom narednih *report_property* Tcl komandi dobijaju se sledeće liste osobina.

Primer 9.17: Lista svih postojećih osobina za *matrix_multiply* modul

Lista svih osobina koje postoje za *matrix_multiply* modul može se dobiti korišćenjem sledeće *report_property* Tcl komande.

report_property -all [**get_cells** matrix_multiplier]

Primer 9.18: Lista svih osobina za *matrix_multiply* modul koje imaju postavljene vrednosti

Lista svih osobina *matrix_multiply* modula za koje su definisane vrednosti može se dobiti korišćenjem sledeće *report_property* Tcl komande.

report_property [**get_cells** matrix_multiplier]

Primer 9.19: Lista svih osobina *a_addr_i[0]* pina *matrix_multiply* modula čije ime počinje sa stringom „BUS“

Sledeća Tcl komanda može se iskoristiti za dobijanje liste svih osobina *a_data_i[0]* pina *matrix_multiply* modula čije ime počinje sa stringom „BUS“. Ove osobine vezane su za karakteristike magistrale kojoj pripada pin *a_data_i[0]*.

report_property -all [**get_pins** matrix_multiplier/a_data_i[0]] BUS*

Nakon izvršavanja prethodne komande dobija se sledeća lista osobina

Property	Type	Read-only	Value
BUS_DIRECTION	enum	true	IN
BUS_NAME	string	true	a_data_i
BUS_START	int	true	7
BUS_STOP	int	true	0
BUS_WIDTH	int	true	8

Tcl komanda *list_property_value*

Pomoću *list_property_value* Tcl komande moguće je dobiti listu dozvoljenih vrednosti za željenu osobinu odabranog objekta unutar Vivado Design Suite baze podataka. Uprošćeno sintaksno pravilo za *list_property_value* Tcl komandu glasi

list_property_value <*osobina*> [<*objekat*>]

Povratna vrednost *list_property_value* komande je lista dozvoljenih vrednosti osobine objekta.

Opcije *list_property_value* komande imaju sledeće značenje:

- <*osobina*> – ime osobine čija lista dozvoljenih vrednosti se traži.
- [<*objekat*>] – specificira konkretni objekat kome pripada osobina čija lista dozvoljenih vrednosti se zahteva. Objekat mora biti jedinstven, ako se prosledi više od jednog objekta dobija se poruka o grešci. Za specifikaciju objekta potrebno je koristiti *get_** Tcl komande.

U slučaju da analiziramo netlistu koja nastaje nakon elaboracije VHDL modela IP jezgra „*Naive*“ množača matrica, sledeći primer ilustruje primenu *list_property_value* Tcl komande.

Primer 9.20: Lista dozvoljenih vrednosti osobine pina koja određuje smer

Pomoću sledeće list_property_value Tcl komande moguće je dobiti listu dozvoljenih vrednosti za osobinu DIRECTION *a_data_i[0]* pina *matrix_multiply* modula. Vrednost osobine DIRECTION pruža informaciju o smeru pina.

list_property_value DIRECTION [**get_pins** matrix_multiplier/a_data_i[0]]

Nakon izvršavanja prethodne Tcl komande dobija se sledeća lista dozvoljenih vrednosti osobine DIRECTION

IN OUT INOUT MIXED UNKNOWN

Tcl komanda *get_property*

Pomoću *get_property* Tcl komande moguće je dobiti tekuću vrednost specificirane osobine za odabrani objekat iz Vivado Design Suite baze podataka. Uprošćeno sintaksno pravilo za *get_property* Tcl komandu glasi

get_property <*osobina*> <*objekat*>

Povratna vrednost *get_property* komande je vrednost odabrane osobine.

Opcije *get_property* komande imaju sledeće značenje:

- <*osobina*> – ime osobine čija trenutna vrednost se traži.

- *<objekat>* – specificira konkretni objekat kome pripada osobina čija trenutna vrednost se zahteva. Objekat mora biti jedinstven, ako se prosledi više od jednog objekta dobija se poruka o grešci. Za specifikaciju objekta potrebno je koristiti *get_** Tcl komande.

U slučaju da analiziramo netlistu koja nastaje nakon elaboracije VHDL modela IP jezgra „*Naive*“ množača matrica, sledeći primer ilustruje primenu *get_property* Tcl komande.

Primer 9.21: Dobijanje informacije o smeru selektovanog pina

Pomoću sledeće Tcl komande moguće je dobiti informaciju o smeru selektovanog pina odabranog modula unutar „*Naive*“ množača matrica. U našem primeru dobićemo informaciju o smeru *a_data_i[0]* pina *matrix_multiply* modula.

get_property DIRECTION [**get_pins** matrix_multiplier/a_data_i[0]]

Izvršavanjem prethodne Tcl komande dobija se sledeća povratna vrednost

IN

Tcl komanda *set_property*

Pomoću *set_property* Tcl komande moguće je postaviti odabranu osobinu selektovanog objekta iz Vivado Design Suite baze podataka na željenu vrednost. Vrednost na koju se postavlja odabrana osobina mora biti iz liste dozvoljenih vrednosti za posmatranu osobinu. Uprošćeno sintaksno pravilo za *set_property* Tcl komandu glasi

set_property *<osobina>* *<vrednost>* *<objekat>*

Povratna vrednost *set_property* komande je lista pinova.

Opcije *set_property* komande imaju sledeće značenje:

- *<osobina>* – ime osobine čija trenutna vrednost se traži.
- *<vrednost>* – vrednost na koju je potrebno postaviti odabranu osobinu. Vrednost na koju se postavlja osobina mora biti iz skupa dozvoljenih vrednosti posmatrane osobine.
- *<objekat>* – specificira konkretni objekat kome pripada osobina čija vrednost se postavlja. Objekat mora biti jedinstven, ako se prosledi više od jednog objekta dobija se poruka o grešci. Za specifikaciju objekta potrebno je koristiti *get_** Tcl komande.

U slučaju da analiziramo netlistu koja nastaje nakon elaboracije VHDL modela IP jezgra „*Naive*“ množača matrica, sledeći primer ilustruje primenu *set_property* Tcl komande.

Primer 9.22: Označavanje odabrane mreže za debagovanje

Da bi potrebni signali bili na raspolaganju prilikom hardverskog debagovanja, korišćenjem integrisanog logičkog analizatora, pre sinteze dizajna moraju se označiti kao mreže koje su potrebne za debagovanje. Na ovaj način označeni signali neće biti optimizovani i potencijalno uklonjeni prilikom sinteze i implementacije dizajna. Označavanje signala kao signala koji je potreban prilikom debagovanja ostvaruje se postavljanjem njegove MARK_DEBUG osobine na vrednost 1. U ovom primeru označićemo signal *b_wr_s* kao signal koji će biti potreban u procesu debagovanja.

Svi signali su inicijalno neoznačeni za debagovanje, odnosno vrednost osobine MARK_DEBUG postavljena je na nulu. Ovo se može proveriti za *b_wr_s* signal, izvršavanjem sledeće *get_property* Tcl komande

get_property MARK_DEBUG [**get_nets** b_wr_s]

Povratna vrednost prethodne Tcl komande jednaka je 0, što znači da signal *b_wr_s* trenutno nije označen za debagovanje. Korišćenjem sledeće *set_property* Tcl komande možemo signal *b_wr_s* označiti kao signal potreban za debagovanje

set_property MARK_DEBUG 1 [**get_nets** b_wr_s]

Ponovnim korišćenjem prethodne *get_property* Tcl komande, možemo se uveriti da je MARK_DEBUG osobina *b_wr_s* signala sada postavljena na vrednost 1, čime je on označen za debagovanje.

Zadaci za vežbu

Zadatak 9.5.

Unutar sintetizovanog dizajna „*Naive*“ množača matrica izlistati listu svih osobina definisanih za port *s00_axi_wdata[0]*.

Zadatak 9.6.

Unutar sintetizovanog dizajna „*Naive*“ množača matrica izlistati listu svih dozvoljenih vrednosti za IOSTANDARD osobinu za port *s00_axi_wdata[0]*.

Zadatak 9.7.

Unutar sintetizovanog dizajna „*Naive*“ množača matrica izlistati tekuću vrednost osobine IOSTANDARD za port *s00_axi_wdata[0]*.

Zadatak 9.8.

Unutar sintetizovanog dizajna „*Naive*“ množača matrica promeniti tekuću vrednost osobine IOSTANDARD na LVCMOS25 za port *s00_axi_wdata[0]*. Nakon promene proveriti da li je nova vrednost osobine IOSTANDARD porta *s00_axi_wdata[0]* LVCOMS25.

9.2. Statička vremenska analiza

Kao što je već napomenuto u glavi 1, ukoliko se digitalni sistem projektuje koristeći metodologiju sinhronog sekvencijalnog dizajna, tada se vremenska simulacija ponašanja sistema može zameniti procesom **statičke vremenske (tajming) analize (Static Timing Analisys, STA)**. Statička vremenska analiza na analitički i formalan način izračunava kašnjenja signala duž svakog od mogućih unutrašnjih puteva u dizajnu i proverava da li kritične putanje (putanje sa najvećim kašnjenjem) zadovoljavaju postavljena vremenska ograničenja, odnosno da li implementirani dizajn može da radi na zahtevanoj učestanosti globalnog sinhronizacionog (klok, *clock*) signala. Statička vremenska analiza je vrlo brza i zahteva vrlo malo računskog vremena. Pomoću statičke tajming analize ne može se utvrditi da li je dizajn funkcionalno korektan, niti da li u njemu postoje gličevi, niti da li će postojanje gličeva uticati na korektno ponašanje sistema. Za odgovore na ova pitanja mora se koristiti vremenska simulacija. Sa druge strane, ukoliko se digitalni sistem projektuje primenom metodologije sinhronog sekvekcijalnog dizajna, onda je korišćenje vremenske simulacije (koja je vrlo „skupa“ u pogledu vremena potrebnog za njeno kompletiranje) suvišno. Dovoljno je samo analizirati izveštaj nakon kompletiranja statičke vremenske analize i, ukoliko je on pozitivan (u smislu da su vremenska kašnjenja signala duž kritičnih putanja kraća od zahtevanih), može se sa sigurnošću garantovati da će projektovani sistem imati očekivano ponašanje, ukoliko bude radio na brzinama koje su manje ili jednake brzinama koje su bile speficirane u fajlu sa ograničenjima.

Pre nego što nastavimo, uvešćemo par termina koji će biti korišćeni u nastavku ovog poglavlja:

- Pod **generišućom ivicom** (*launch edge*) klok signala podrazumevaćemo ivicu klok signala (rastuću, opadajuću ili obe) na kojoj se generiše nova vrednost signala koji predstavlja izlaz sekvencijalnog elementa.

- Pod **upisujućom ivicom** (*capture edge*) klok signala podrazumevaćemo ivicu klok signala (rastuću, opadajuću ili obe) na kojoj se nova vrednost signala upisuje u sekvencijalni element.
- **Izvorni klok signal** predstavlja klok signal koji upravlja radom sekvencijalnog elementa od čijeg izlaza započinje putanja signala.
- **Odredišni klok signal** predstavlja klok signal koji upravlja radom sekvencijalnog elementa na čijem ulazu se završava putanja signala.

Pod **putanjom signala** (*Timing Path*) unutar digitalnog sistema podrazumevamo svaku putanju koja spaja dva sekvencijalna elementa, koji su kontrolisani istim ili različitim klok signalima. Unutar digitalnog dizajna implementiranog pomoću FPGA komponente postoji nekoliko različitih klasa putanja signala (prikazanih i na slici 9.6):

- **Putanja signala od ulazne nožice do unutrašnjeg sekvencijalnog elementa** (putanja 1 sa slike 9.6)

  Kod ove klase putanja signala započinje izvan FPGA komponente, unutar neke druge komponente koja je prisutna u sistemu (recimo na štampanoj ploči na kojoj se nalazi i FPGA komponenta). Ova spoljašnja komponenta kontrolisana je pomoću sistemskog klok signala, koji ne mora nužno da bude i klok signal FPGA komponente. Nailaskom generišuće ivice sistemskog klok signala generiše se nova vrednost signala koji stiže do ulazne nožice FPGA komponente nakon izvesnog vremenskog intervala, koji se naziva **kašnjenje ulaza** (*Input Delay*). Nakon prolaska kroz ulaznu nožicu, signal se propagira kroz FPGA komponentu, sve dok ne stigne na ulaz nekog od unutrašnjih sekvencijalnih elemenata, u koji se upisuje nailaskom upisujuće ivice odredišnog klok signala (koji može ali i ne mora biti sistemski klok signal).

- **Unutrašnja putanja signala od sekvencijalnog elementa do sekvencijalnog elementa** (putanja 2 sa slike 9.6)

  Kod ove klase putanja signala započinje na izlazu nekog sekvencijalnog elementa koji se nalazi unutar FPGA komponente. Novi signal se pojavljuje na izlazu unutrašnjeg sekvencijalnog elementa, nailaskom generišuće ivice izvornog klok signala. Ovaj signal zatim putuje kroz unutrašnju logiku FPGA komponente (na primer kroz LUT tabele) i veze za rutiranje, sve dok ne dođe do ulaza nekog unutrašnjeg sekvencijalnog elementa (koji može ali i ne mora biti isti element sa kojega je krenuo signal) i upisuje se u njega nailaskom upisujuće ivice odredišnog klok signala (koji može ali i ne mora biti jednak sa izvornim klok signalom).

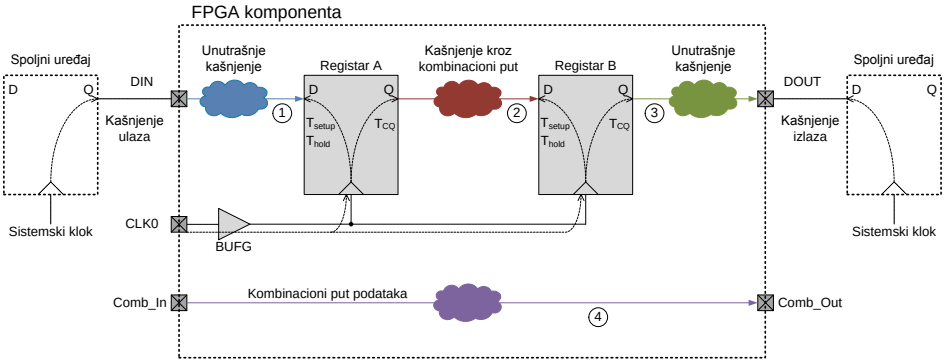
- **Putanja signala od unutrašnjeg sekvencijalnog elementa do izlazne nožice** (putanja 3 sa slike 9.6)

  Kod ove klase putanja signala započinje na izlazu nekog sekvencijalnog elementa koji se nalazi unutar FPGA komponente. Nova vrednost signala se pojavljuje na izlazu unutrašnjeg sekvencijalnog elementa nailaskom generišuće

ivice izvornog klok signala. Ovaj signal zatim putuje kroz unutrašnju logiku FPGA komponente (na primer kroz LUT tabele) i veze za rutiranje, sve dok ne dođe do izlazne nožice FPGA kola. Nakon napuštanja FPGA komponente, signal nakon izvesnog vremenskog intervala, koji se naziva **kašnjenje izlaza** (*Output Delay*) dolazi na ulaz sekvencijalnog elementa koji se nalazi izvan FPGA komponente, u koji se upisuje nailaskom upisujuće ivice sistemskog klok signala.

- **Putanja signala od ulazne nožice do izlazne nožice koja ne prolazi ni kroz jedan unutrašnji sekvencijalni element** (putanja 4 sa slike 9.6)

  U ovoj klasi putanja signal se generiše izvan FPGA komponente, ulazi kroz odgovarajuću ulaznu nožicu i propagira se kroz FPGA komponentu, prolazeći kroz unutrašnju logiku FPGA komponente (LUT tabele) i veze za rutiranje, sve dok ne napusti FPGA komponentu preko neke izlazne nožice.



Slika 9.6. Tipični unutrašnji putevi signala unutar FPGA komponente

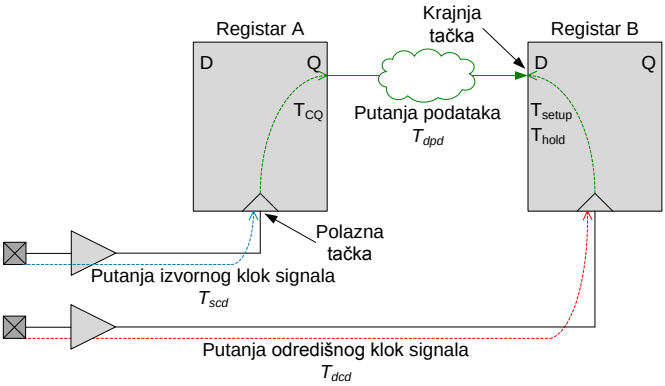
Svaka putanja signala sastoji se iz tri sekcije (prikazane na slici 9.7):

- **Putanje izvornog klok signala** (*Source Clock Path*) – ovo je putanja koju prolazi izvorni klok signal od mesta njegovog generisanja (ovo je tipično neka od ulaznih nožica FPGA komponente ili izlazni port unutašnjeg bloka za generisanje klok signala, takozvanog MMCM (*Mixed-Mode Clock Manager*) bloka) do klok ulaza sekvencijalnog elementa na čijem izlazu započinje putanja podataka. U slučaju da putanja signala počinje na ulaznoj nožici FPGA komponente, putanja izvornog klok signala ne postoji. Kašnjenje signala duž ove putanje označićemo sa T_{scd}.

- **Putanje podataka** *(Data Path)* – ovaj deo putanje signala je deo duž kojega se propagiraju podaci čiji integritet je neophodno očuvati (da bi čitav sistem radio pouzdano). Putanja podataka započinje na izlazu podataka sekvencijalnog elementa kontrolisanog izvornim klok signalom ili na ulaznoj nožici FPGA komponente, a završava na ulazu podataka sekvencijalnog elementa

kontrolisanog odredišnim klok signalom ili na izlaznoj nožici FPGA komponente. Kašnjenje signala duž ove putanje označićemo sa T_{dpd}.

- **Putanje odredišnog klok signala** (*Destination Clock Path*) – ovo je putanja koju prolazi odredišni klok signal od mesta njegovog generisanja (ovo je tipično neka od ulaznih nožica FPGA komponente ili izlazni port unutašnjeg bloka za generisanje klok signala, takozvanog MMCM (*Mixed-Mode Clock Manager*) bloka) do klok ulaza sekvencijalnog elementa na čijem ulazu podataka završava putanja podataka. U slučaju da putanja signala završava na izlaznoj nožici FPGA komponente, putanja odredišnog klok signala ne postoji. Kašnjenje signala duž ove putanje označićemo sa T_{dcd}.

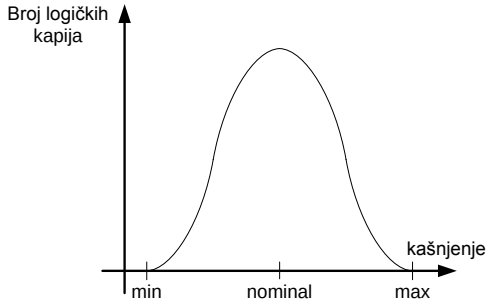


Slika 9.7. Sekcije koje čine putanju signala

Ono što donekle komplikuje izvođenje statičke vremenske analize jeste takozvani „*On-Chip Variation*“ problem, usled kojega vremenske karakteristike (propagaciono kašnjenje, vremena uspostavljanja i držanja, itd.) instanci elementarnih hardverskih resursa iz iste klase (flip flop, logička kapija, memorijska ćelija, kanal za rutiranje, itd.), na istom integrisanom kolu, nikada nisu identične. Obratite pažnju da vremenske karakteristike instanci istih hardverskih resursa na različitim čipovima mogu biti *drastično* drugačije (ovo je takozvani „*Chip-to-Chip Variation*“ problem).

Zašto nastaje „*On-Chip Variation*“ problem? Postoji nekoliko razloga, ali u osnovi je on posledica nesavršenosti fabrikacionog procesa integrisanih kola, tokom kojega nije moguće odražavati različite parametre procesa fabrikacije (na primer, veličine proreza na maski koje definišu širinu vodova i veličinu tranzistora nisu savršeno jednake, postoje nesavršenosti u optičkom sistemu koji se koristi za prenos maske na substrat, postoje varijacije u procesu ecovanja, itd.) konstantnim tokom vremena i prostora. Ovo je takozvani „*Process Variation*“ problem, problem **varijacije procesa fabrikacije (P)**. Kao posledica ovih nesavršenosti u procesu fabrikacije, vremenske karakteristike instanci istog hardverskog resursa će se donekle razlikovati, kao posledica u varijaciji karakteristika tranzistora od kojih su ovi elementi izgrađeni. Na primer, kašnjenja

logičkih kapija neće biti ista, već će poštovati Gausovu raspodelu, kao što je prikazano na slici 9.8.



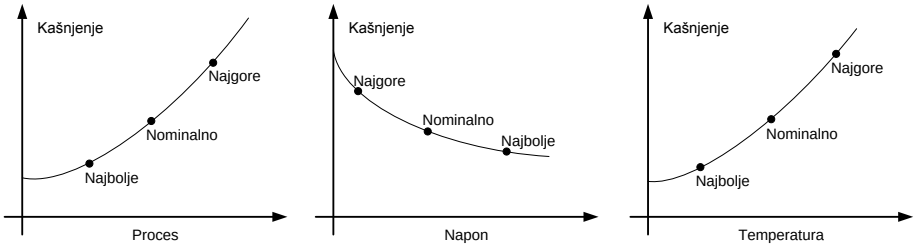
Slika 9.8. Distribucija kašnjenja različitih instanci iste familije logičkih kapija fabrikovanih na istom čipu, kao posledica varijacija u procesu fabrikacije

Pored varijacija koje su posledica nesavršenosti procesa fabrikacije integrisanih kola, varijacije vremenskih karakteristika elementarnih hardverskih resursa postoje i usled:

- **Varijacija u naponu napajanja (V)** – koje su posledice konačne otpornosti veza preko kojih se napon napajanja dovodi do svakog hardverskog resursa, pa je napon napajanja hardverskog resursa koji su udaljeniji od izvora napajanja niži od hardverskog resursa koji mu je bliži. Kako sve vremenske karakteristike predstavljaju funkciju napona napajanja, ova dva hardverska resursa iz iste klase će imati različite vremenske karakteristike.
- **Varijacija temperature (T)** – kao što je poznato vremenske karakteristike električnih kola su i temperaturno zavisne. Temperatura različitih delova integrisanog kola može se razlikovati, usled različitog stepena aktivnosti hardverskih resursa ili usled spoljašnjih faktora. Poznato je da se u CMOS tehnologiji većina električne energije disipira u termičku, prilikom svake promene logičkog stanja u kojem se posmatrani element (logička kapija, flip flop) nalazi. Ukoliko na integrisanom kolu postoje regioni u kojima postoji visoki stepen promene logičkih nivoa (promene sa logičke nule na logičku jedinicu i obrnuto), ovi regioni će se više zagrevati od neaktivnih regiona, odnosno na čipu će se formirari takozvane „vrele tačke“ (*hot spots*). Usled temperaturne zavisnosti vremenskih karakteristika, vremenske karakteristike logičkih elemenata koji se nalaze unutar „vrelih tačaka“ biće značajno drugačije od vremenskih karakteristika logičkih elemenata istog tipa koji se nalaze u neaktivnim regionima integrisanog kola.

Ove tri varijacije (procesa P, napona napajanja V, temperature T) glavni su uzročnici varijacije vremenskih karakteristika identičnih hardverskih resursa na integrisanom kolu

i moraju se uzeti u obzir prilikom izvođenja statičke vremenske analize kako bi ona dala pouzdane rezultate. Kombinovanjem ove tri dimenzije dobija se trodimenzionalni model vremenskih karakteristika hardverskog elementa (logičke kapije, flip flop, metalne veze, itd.) koji modeluje i varijabilnost usled PVT procesa. Obratite pažnju da prilikom statičke vremenske analize nije neophodno prolaziti kroz čitav trodimenzionalni region mogućih kašnjenja svakog od hardverskih elemenata, već je dovoljno analizirati ponašanje sistema u takozvanim graničnim tačkama (*Corners*), koje predstavljaju tačke sa najboljim i najlošijim vrednostima kašnjenja hardverskog elementa u trodimenzionalnom prostoru, a određene su odgovarajućim minimalnim i maksimalnim vrednostima duž svake od tri ose, kao što je prikazano na slici 9.9.

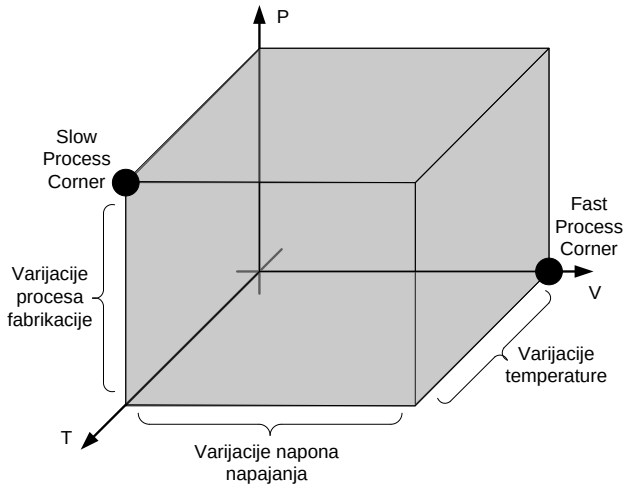


Slika 9.9. Varijacija vremenskih karakteristika usled varijacija procesa fabrikacije (P), napona napajanja (V) i temperature (T)

U procesu statičke vremenske analize uglavnom se koriste dve kritične tačke u PVT prostoru, definisane na sledeći način:

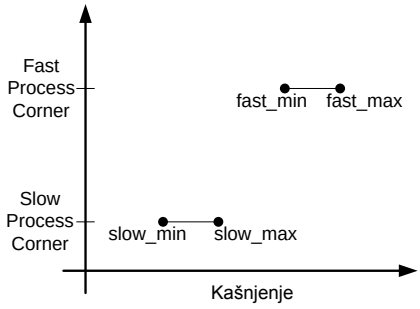
- **Najbolja tačka** (*Fast Process Corner, FPC*) – ova tačka odlikuje se najboljim performansama koje se mogu postići duž svake od tri ose (najmanje kašnjenje na proces osi, pri najvećem naponu napajanja i na najmanjoj radnoj temperaturi)
- **Najlošija tačka** (*Slow Proces Corner, SPC*) – ova tačka odlikuje se najlošijim performansama koje se mogu postići duž svake od tri ose (najveće kašnjenje na proces osi, pri najmanjem naponu napajanja i na najvećoj temperaturi)

Region u PVT prostoru u kome se nalaze moguće vrednosti za kašnjenje elementa usled PVT varijacija, kao i položaji *Fast Process Corner* i *Slow Process Corner* tačaka prikazane na slici 9.10.



Slika 9.10. PVT prostor i položaj FPC i SPC tačaka

Da bi se uzele u obzir i promene u kašnjenjima koje su posledica lokalnih varijacija u procesu fabrikacije, napona napajanja i temperature na jednom integrisanom kolu (koje delom zavise i od samog dizajna koji je trenutno implementiran na integrisanom kolu) obično se za FPC i SPC tačke definišu minimalne i maksimalne vrednosti kašnjenja svakog od elementarnih hardverskih elemenata koji postoje na integrisanom kolu (*fast_min* i *fast_max* za FPC tačku i *slow_min* i *slow_max* za SPC tačku), kao što je prikazano na slici 9.11. Na ovaj način dolazi se do četiri tačke pomoću kojih se definišu kašnjenjanja elementarnih gradivnih blokova unutar integrisanog kola, koje se zatim koriste u procesu statičke vremenske analize.



Slika 9.11. Definicija minimalnih i maksimalnih kašnjenja za FPC i SPC tačke

Prilikom statičke vremenske analize za svaku putanju signala koja postoji unutar dizajna koji se implementira vrše se dve provere:

- **Provera zadovoljivosti vremena uspostavljanja na odredišnom sekvencijalnom elementu** (*Setup Check*)
- **Provera zadovoljivosti vremena držanja na odredišnom sekvencijalnom elementu** (*Hold Check*)

Za svaku putanju unutar dizajna ove dve provere se vrše i u FPC i u SPC tački unutar PVT prostora, uzimajući najgori slučaj prilikom formiranja putanje signala. Ukoliko su vremena uspostavljanja i držanja ispunjena u obe granične tačke, posmatrana putanja zadovoljava postavljene kriterijume statičke vremenske analize.

9.2.1. Provera vremena uspostavljanja (*Setup Check*)

Prilikom provere vremena uspostavljanja vrši se sledeći proračun. Potrebno je odrediti sledeće dve vrednosti:

- Trenutak kada podatak mora da bude stabilan na ulazu odredišnog sekvencijalnog elementa (*Data Required Time*), T_{DRT}. Na osnovu slike 9.12 T_{DRT} vrednost se može izračunati na sledeći način

$$T_{DRT} = T_{cet} + T_{dcd} - C_{\mathrm{u}} - T_{setup}$$

  gde je sa T_{cet} označen trenutak nailaska upisujuće ivice odredišnog klok signala, T_{dcd} predstavlja kašnjenje odredišnog klok signala, C_u je takozvana nesigurnost klok signala (koja je uglavnom posledica postojanja džitera klok signala, o čemu će više reči biti u poglavlju 9.3.1), a T_{setup} predstavlja vreme uspostavljanja odredišnog sekvencijalnog elementa.

- Trenutak kada podatak stiže na ulaz odredišnog sekvencijalnog elementa (*Data Arrival Time*), T_{DAT}. Na osnovu slike 9.12 T_{DAT} vrednost se može izračunati na sledeći način

$$T_{DAT} = T_{let} + T_{scd} + T_{dpd}$$

  gde je sa T_{let} označen trenutak nailaska generišuće ivice izvornog klok signala, T_{scd} predstavlja kašnjenje izvornog klok signala, a T_{dpd} predstavlja kašnjenje signala duž putanje podataka.

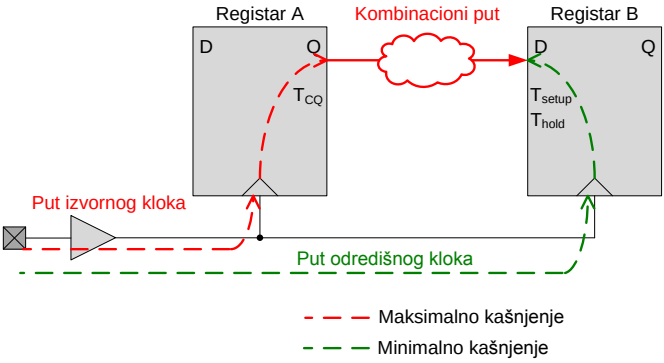
Razlika T_{DRT} i T_{DAT} vrednosti predstavlja marginu koja postoji na posmatranoj putanji podataka kada se posmatra vreme uspostavljanja signala

$$Slack_{setup} = T_{DRT} - T_{DAT}.$$

Prilikom određivanja $Slack_{setup}$ vrednosti potrebno je analizirati najgori slučaj za putanju signala prikazanu na slici 9.12.

$$Slack_{setup} = \min(T_{DRT}) - \max(T_{DAT}) =$$
$$= T_{cet} + \min(T_{dcd}) - C_u - T_{setup} - [T_{let} + \max(T_{scd}) + \max(T_{dpd})].$$

Za kašnjenje signala duž putanje izvornog klok signala (T_{scd} parametar) i duž putanje podataka (T_{dpd} parametar) uzimaju se najgora vremena (*max* vrednosti u FPC i SPC tačkama sa slike 9.11), dok se za kašnjenje signala duž putanje odredišnog klok signala (T_{dcd} parametar) uzima najbolje vreme (*min* vrednosti u FPC i SPC tačkama sa slike 9.11).



Slika 9.12. Definicija putanja koje učestvuju prilikom provere vremena uspostavljanja

NAPOMENA: Provera vremena uspostavljanja u okviru Vivado Design Suite alata poznata je pod nazivom **Max Delay Analysis**.

Margina vremena uspostavljanja se računa u dve tačke PVT prostora, FPC i SPC

$$Slack_{setup\_fast} = \text{fast\_min}(T_{DRT}) - \text{fast\_max}(T_{DAT}),$$

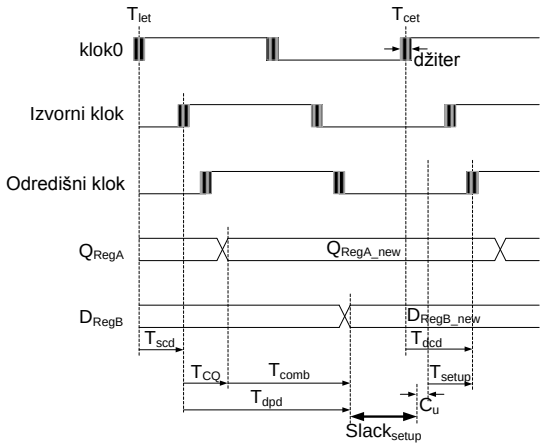
$$Slack_{setup\_slow} = \text{slow\_min}(T_{DRT}) - \text{slow\_max}(T_{DAT}).$$

Ukoliko su ovako izračunate vrednosti margina pozitivne, to znači da je vreme uspostavljanja za posmatranu putanju signala zadovoljeno. Ukoliko je barem jedna od njih negativna, vreme uspostavljanja za posmatranu putanju se ne može zadovoljiti.

Na slici 9.13 prikazan je vremenski dijagram karakterističnih signala iz sistema sa slike 9.12, zajedno sa označenim parametrima koji se koriste prilikom provere vremena uspostavljanja. Signal *klok0* predstavlja talasni oblik klok signala na mestu njegovog generisanja (kao što je već rečeno ovo je tipično neka od ulaznih nožica FPGA

komponente ili izlazni port unutašnjeg bloka za generisanje klok signala) koji se zatim razvodi po FPGA komponenti. Izvorni i odredišni klok signali su zapravo replike *klok0* signala, ali zakašnjenje T_{scd} i T_{dcd} sekundi. Nakon nailaska generišuće ivice na izvornom klok signalu (na slici 9.13 generišuća ivica izvornog klok signala je rastuća ivica), nakon T_{CQ} vremena na Q izlazu registra A pojavljuje se nova vrednost, Q_{RegA_new}. Ova vrednost onda putuje kroz unutrašnju logiku FPGA komponente i kanale za rutiranje i nakon T_{comb} sekundi stiže na D ulaz registra B, D_{RegB_new}. Zbir vrednosti T_{CQ} i T_{comb} zapravo predstavlja kašnjenje duž puta podataka, T_{dpd}. Da bi se obezbedio pouzdan upis, nova vrednost signala na D ulazu registra B, D_{RegB_new}, mora biti stabilna barem T_{setup} sekundi pre nailaska upisujuće ivice odredišnog klok signala (sa slike 9.13 možemo videti da je u našem primeru upisujuća ivica odredišnog klok signala takođe rastuća ivica), koja u odnosu na *klok0* signal kasni T_{dcd} sekundi. Kako periode *klok0*, izvorišnog i odredišnog klok signala variraju, usled potojanja klok džitera (o kome će više reči biti u poglavlju 9.3.1), signal D_{RegB} zapravo mora biti stabilan dodatnih C_u sekundi pre nailaska *setup* prozora. Parameter C_u zapravo meri ovu nesigurnost u tačnim trenucima nailaska generišućih i upisujućih ivica izvornog i odredišnog klok signala.

Obratite pažnju da su trenuci generisanja i upisivanja signala podataka, T_{let} i T_{cet}, u slučaju provere vremena uspostavljanja razdvojeni jednu periodu klok signala, jer nova vrednost signala na ulazu registra B, D_{RegB_new}, generisana kao posledica generisanja nove vrednosti na izlazu registra A, Q_{RegA_new}, na tekućoj generišućoj ivici izvornog klok signala, mora biti stabilna dovoljno dugo pre nailaska naredne upisujuće ivice odredišnog klok signala.



Slika 9.13. Margina u vremena uspostavljanja signala

9.2.2. Provera vremena držanja (*Hold Check*)

Prilikom provere vremena držanja vrši se sledeći proračun. Potrebno je odrediti sledeće dve vrednosti:

- Trenutak do kada podatak mora da bude stabilan na ulazu odredišnog sekvencijalnog elementa (*Data Required Time*), T_{DRT}. Na osnovu slike 9.14 T_{DRT} vrednost se može izračunati na sledeći način

$$T_{DRT} = T_{cet} + T_{dcd} - C_{\mathrm{u}} + T_{hold}$$

  gde je sa T_{cet} označen trenutak nailaska upisujuće ivice odredišnog klok signala, T_{dcd} predstavlja kašnjenje odredišnog klok signala, C_u je takozvana nesigurnost klok signala (koja je uglavnom posledica postojanja džitera klok signala, o čemu će više reči biti u poglavlju 9.3.1), a T_{hold} predstavlja vreme držanja odredišnog sekvencijalnog elementa.

- Trenutak kada podatak stiže na ulaz odredišnog sekvencijalnog elementa (*Data Arrival Time*), T_{DAT}. Na osnovu slike 9.14 T_{DAT} vrednost se može izračunati na sledeći način

$$T_{DAT} = T_{let} + T_{scd} + T_{dpd}$$

  gde je sa T_{let} označen trenutak nailaska generišuće ivice izvornog klok signala, T_{scd} predstavlja kašnjenje izvornog klok signala, a T_{dpd} predstavlja kašnjenje signala duž putanje podataka.

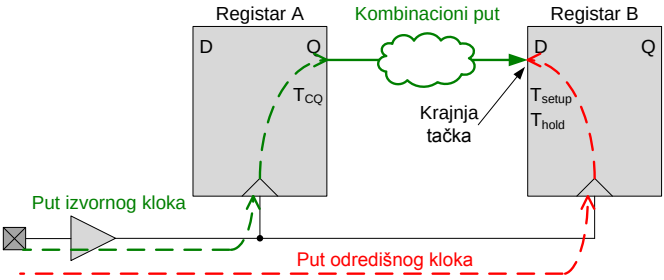
Razlika T_{DAT} i T_{DRT} vrednosti predstavlja marginu koja postoji na posmatranoj putanji podataka kada se posmatra vreme držanja signala

$$Slack_{hold} = T_{DAT} - T_{DRT}.$$

Prilikom određivanja $Slack_{hold}$ vrednosti potrebno je analizirati najgori slučaj za putanju signala prikazanu na slici 9.14.

$$\begin{aligned} Slack_{hold} &= \min(T_{DAT}) - \max(T_{DRT}) = \\ &= T_{let} + \min(T_{scd}) + \min(T_{dpd}) - [T_{cet} + \max(T_{dcd}) - C_{\mathrm{u}} + T_{hold}]. \end{aligned}$$

Za kašnjenje signala duž putanje izvornog klok signala (T_{scd} parametar) i duž putanje podataka (T_{dpd} parametar) ovoga puta uzimaju se najbolja vremena (*min* vrednosti u FPC i SPC tačkama sa slike 9.11), dok se za kašnjenje signala duž putanje odredišnog klok signala (T_{dcd} parametar) uzima najgore vreme (*max* vrednosti u FPC i SPC tačkama sa slike 9.11).



Slika 9.14. Definicija putanja koje učestvuju prilikom provere vremena držanja

NAPOMENA: Provera vremena uspostavljanja u okviru Vivado Design Suite alata poznata je pod nazivom **Min Delay Analysis**.

Margina vremena uspostavljanja se računa u dve tačke PVT prostora, FPC i SPC

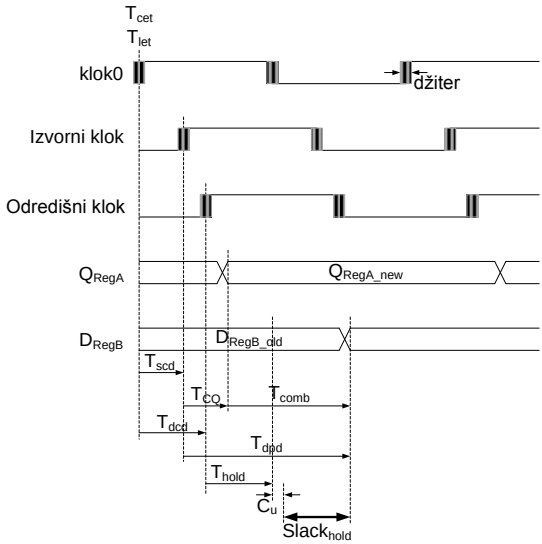
$$Slack_{hold\_fast} = \text{fast\_min}(T_{DAT}) - \text{fast\_max}(T_{DRT}),$$

$$Slack_{hold\_slow} = \text{slow\_min}(T_{DAT}) - \text{slow\_max}(T_{DRT}).$$

Ukoliko su ovako izračunate vrednosti margina pozitivne, to znači da je vreme držanja za posmatranu putanju signala zadovoljeno. Ukoliko je barem jedna od njih negativna, vreme držanja za posmatranu putanju se ne može zadovoljiti.

Na slici 9.15 prikazan je vremenski dijagram karakterističnih signala iz sistema sa slike 9.14, zajedno sa označenim parametrima koji se koriste prilikom provere vremena držanja. Kao i u slučaju provere vremena uspostavljanja, signal *klok0* predstavlja talasni oblik klok signala na mestu njegovog generisanja, koji se zatim razvodi po FPGA komponenti. Izvorni i odredišni klok signali su zapravo replike *klok0* signala, ali zakašnjenje T_{scd} i T_{dcd} sekundi. Nakon nailaska generišuće ivice na izvornom klok signalu (na slici 9.15 generišuća ivica izvornog klok signala opet je rastuća ivica), nakon T_{CQ} vremena na Q izlazu registra A pojavljuje se nova vrednost, Q_{RegA_new}. Ova vrednost onda putuje kroz unutrašnju logiku FPGA komponente i kanale za rutiranje i nakon T_{comb} sekundi stiže na D ulaz registra B. Zbir vrednosti T_{CQ} i T_{comb} opet predstavlja kašnjenje duž puta podataka, T_{dpd}. Da bi sistem radio pouzdano, stara vrednost signala na D ulazu registra B, D_{RegB_old}, mora biti stabilna barem T_{hold} sekundi nakon nailaska upisujuće ivice odredišnog klok signala (sa slike 9.15 možemo videti da je upisujuća ivica odredišnog klok signala ponovo rastuća ivica), koja u odnosu na *klok0* signal kasni T_{dcd} sekundi. Kako periode *klok0*, izvorišnog i odredišnog klok signala variraju, usled potojanja klok džitera (o kome će više reči biti u poglavlju 9.3.1), signal D_{RegB} zapravo mora biti stabilan dodatnih C_u sekundi nakon prolaska *hold* prozora. Kao i u slučaju provere vremena uspostavljanja, parameter C_u meri nesigurnost u tačnim trenucima nailaska generišućih i upisujućih ivica izvornog i odredišnog klok signala.

Obratite pažnju da se trenuci generisanja i upisivanja signala podataka, T_{let} i T_{cet}, u slučaju provere vremena držanja dešavaju istovremeno, jer stara vrednost signala na ulazu registra *B*, D_{RegB_old}, generisana na prethodnoj generišućoj ivici izvornog klok signala, mora biti stabilna dovoljno dugo posle nailaska naredne upisujuće ivice odredišnog klok signala.



Slika 9.15. Margina u vremena držanja signala

9.2.3. Sprovođenje statičke vremenske analize korišćenjem Vivado Design Suite alata

Pomoću Vivado Design Suite alata moguće je sprovesti i statičku vremensku analizu nakon što se dizajn implementira (završe se faze raspoređivanja i povezivanja). Pre nego što prikažemo koje sve mogućnosti Vivado Design Suite alat pruža za sprovođenje statičke vremenske analize, moramo završiti fazu implementacije VHDL modela „*Naive*“ množača matrica. U poglavlju 9.1 kreirali smo Vivado projekat „*Naive*“ množača matrica i uključili sve potrebne izvorne VHDL fajlove koji predstavljaju model množača. Pre nego što započnemo faze sinteze i implementacije „*Naive*“ množača matrica potrebno je definisati potrebne performanse koje implementirani dizajn treba da poseduje. Vivado Design Suite alat koristi ove zadate performanse kao ciljeve koje treba dostići prilikom sinteze i implementacije dizajna. Takođe, prilikom statičke vremenske analize Vivado Design Suite alat koristi specificirane performanse prilikom provere zadovoljivosti vremena uspostavljanja i držanja na svim unutrašnjim putanjama signala koje postoje unutar dizajna. Način kako se definišu željene performanse sistema koje je potrebno zadovoljiti prilikom njegove implementacije

predmet je poglavlja 9.3. U ovom trenutku potrebno je dodati još jedan izvorni fajl u Vivado projekat, takozvani fajl sa ograničenjima (*Constraints File*), koji je običan tekstualni fajl u kojem se nalaze sve ograničenja i potrebne performanse koje dizajn koji se implementira treba da zadovolji. U našem primeru, za sada ćemo unutar fajla sa ograničenjima smestiti samo dva ograničenja, u pogledu željenih minimalnih perioda dva ulazna klok signala u model „*Naive*“ množača matrica, *s00_axi_aclk, s01_axi_aclk*. Pretpostavimo da želimo da implementirani dizajn radi na 100 MHz. Tada je u fajl sa ograničenjima potrebno smestiti sledeća dva ograničenja

```
create_clock -period 10.000 -name s00_axi_aclk -waveform {0.000 5.000} [get_ports s00_axi_aclk]
create_clock -period 10.000 -name s01_axi_aclk -waveform {0.000 5.000} [get_ports s01_axi_aclk]
```

Kao što se može videti, željena učestanost klok signala na kojoj sistem koji se projektuje treba da pouzdano radi, specificira se korišćenjem *create_clock* Tcl komande, o kojoj će više reči biti u poglavlju 9.3.1.

Nakon što smo u tekst editoru kreirali fajl sa gornja dva ograničenja, potrebno ga je sačuvati na disku, recimo pod imenom *matrix_multiply.xdc*. Ekstenzija *.xdc* je oznaka pomoću koje Vivado Design Suite alat zna da je tekući fajl zapravo fajl koji sadrži listu ograničenja koja je potrebno ispoštovati prilikom implementacije dizajna.

Nakon što smo kreirali fajl sa ograničenjima potrebno ga je uključiti u tekući projekat. To se radi pritiskom na komandu „*Add Sources*“ u okviru „*Project Manager*“ grupe komandi u okviru „*Flow Navigator*“ prozora. Kada se pojavi novi prozor, potrebno je odabrati opciju „*Add or create constraints*“ i u sledećem prozoru uključiti *matrix_multiply.xdc* fajl sa ograničenjima, izborom komande “*Add Files*”. Pritiskom na dugme *Finish* odabrani fajl sa ograničenjima se uključuje u tekući projekat, što se može potvrditi ukoliko se pogleda sadržaj *constrs_1* grupe u okviru *Sources/Hierarchy* prozora. Unutar ove grupe fajlova treba da se nalazi *matrix_multiply.xdc* fajl.

Sada smo spremni da pokrenom proces RTL sinteze dizajna, pritiskom na „*Run Synthesis*“ komandu u okviru *Synthesis* grupe iz „*Flow Navigator*“ prozora. Kada se proces sinteze uspešno završi, možemo pokrenuti proces implementacije, pritiskom na „*Run Implementation*“ komandu u okviru *Implementation* grupe iz „*Flow Navigator*“ prozora. Nakon što se proces iplementacije završi, Vivado Design Suite alat će automatski generisati i izveštaj o sprovedenoj statičkoj vremenskoj analizi, koji se može pronaći u okviru *Reports* kartice, na dnu glavnog Vivado prozora, unutar grupe „*Routed Design*“ izveštaja koji se generišu nakon završene faze povezivanja, pod nazivom „*Timing Summary Report*“. Dvoklikom na ovaj izveštaj prikazuje se njegov sadržaj u okviru glavnog Vivado prozora.

„*Timing Summary Report*“ izveštaj predstavlja jedan od načina koji stoje na raspolaganju unutar Vivado Design Suite alata za analizu rezultata sprovedene statičke vremenske analize. „*Timing Summary Report*“ izveštaj sadrži rezultate sprovedene statičke vremenske analize na implementiranom dizajnu i sastoji se iz sledećih sekcija.

„*Timing Summary*“ sekcija

Ova sekcija izveštaja, prikazana na slici 9.16, sadrži osnovne informacije o generisanom izveštaju, kao što su dizajn za koji je generisan izveštaj, ciljna FPGA komponenta koja se koristi, verzija Vivado alata koja je korišćena za sprovođenje statičke vremenske analize i generisanje izveštaja, datum generisanja izveštaja i odgovarajuća Tcl komanda koja je korišćena za pokretanje ove akcije.

```
------------------------------------------------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2016.2 (win64) Build 1577090 Thu Jun  2 16:32:40 MDT 2016
| Date         : Sun Aug 14 11:20:36 2016
| Host         : Rastislav-PC running 64-bit Service Pack 1  (build 7601)
| Command      : report_timing_summary -warn_on_violation -max_paths 10 -file axi_matrix_multiply_v1_0_timing_summary_routed.rpt -rpx axi_matrix_multiply_v1_0_timing_summary_routed.rpx
| Design       : axi_matrix_multiply_v1_0
| Device       : 7z020-clg484
| Speed File   : -1  PRODUCTION 1.11 2014-09-11
------------------------------------------------------------------------------------------------------------------------------------------------------
```

Slika 9.16. „Timing Summary“ sekcija izveštaja o sprovedenoj statičkoj vremenskoj analizi

„*Timer Settings*“ sekcija

Sledeća sekcija u izveštaju o sprovedenoj statičkoj vremenskoj analizi je „*Timer Settings*“ sekcija, prikazana na slici 9.17, u okviru koje se mogu videti podešavanja alata za statičku vremensku analizu koja su korišćena prilikom statičke vremenske analize.

```
------------------------------------------------------------------------------------------------
| Timer Settings
| --------------
------------------------------------------------------------------------------------------------

  Enable Multi Corner Analysis               :  Yes
  Enable Pessimism Removal                   :  Yes
  Pessimism Removal Resolution               :  Nearest Common Node
  Enable Input Delay Default Clock           :  No
  Enable Preset / Clear Arcs                 :  No
  Disable Flight Delays                      :  No
  Ignore I/O Paths                           :  No
  Timing Early Launch at Borrowing Latches   :  false

  Corner  Analyze    Analyze
  Name    Max Paths  Min Paths
  ------  ---------  ---------
  Slow    Yes        Yes
  Fast    Yes        Yes
```

Slika 9.17. „Timer Settings“ sekcija izveštaja o sprovedenoj statičkoj vremenskoj analizi

„*Check Timing*“ sekcija

U okviru „*Check Timing*“ sekcije prikazani su rezultati većeg broja provera koje su izvršene prilikom statičke vremenske analize. Izgled ove sekcije izveštaja u slučaju „*Naive*“ množača matrica prikazan je na slici 9.18.

```
1. checking no_clock
--------------------
 There are 0 register/latch pins with no clock.


2. checking constant_clock
--------------------------
 There are 0 register/latch pins with constant_clock.


3. checking pulse_width_clock
-----------------------------
 There are 0 register/latch pins which need pulse_width check


4. checking unconstrained_internal_endpoints
--------------------------------------------
 There are 0 pins that are not constrained for maximum delay.

 There are 0 pins that are not constrained for maximum delay due to constant clock.


5. checking no_input_delay
--------------------------
 There are 80 input ports with no input delay specified. (HIGH)

 There are 0 input ports with no input delay but user has a false path constraint.


6. checking no_output_delay
---------------------------
 There are 32 ports with no output delay specified. (HIGH)

 There are 0 ports with no output delay but user has a false path constraint

 There are 0 ports with no output delay but with a timing clock defined on it or propagating through it


7. checking multiple_clock
--------------------------
 There are 0 register/latch pins with multiple clocks.


8. checking generated_clocks
----------------------------
 There are 0 generated clocks that are not connected to a clock source.


9. checking loops
-----------------
 There are 0 combinational loops in the design.


10. checking partial_input_delay
--------------------------------
 There are 0 input ports with partial input delay specified.


11. checking partial_output_delay
---------------------------------
 There are 0 ports with partial output delay specified.


12. checking latch_loops
------------------------
 There are 0 combinational latch loops in the design through latch input
```

Slika 9.18. „Check Timing“ sekcija izveštaja o sprovedenoj statičkoj vremenskoj analizi

„*Check Timing*“ sekcija izveštava o sledećem:

1. Broju registara ili lečeva koji nemaju definisan klok. Za ove registre neće se sprovesti provera vremena uspostavljanja i držanja.
2. Broju registara ili lečeva koji imaju konstantan klok. Za ove registre neće se sprovesti provera vremena uspostavljanja i držanja.
3. Broju registara ili lečeva čiji klok zahteva proveru odnosa signal/pauza.
4. Broju unutrašnjih putanja signala koje nisu pokrivene nekim od vremenskih ograničenja koja su prisutna u fajlu sa ograničenjima.
5. Broju ulaznih nožica FPGA komponente koje se koriste u tekućoj implementaciji a nemaju definisano ograničenje po pitanju maksimalnog dozvoljenog propagacionog kašnjenja od ulazne nožice do prvog unutrašnjeg sekvencijalnog elementa.
6. Broju izlaznih nožica FPGA komponente koje se koriste u tekućoj implementaciji a nemaju definisano ograničenje po pitanju maksimalnog dozvoljenog propagacionog kašnjenja od poslednjeg unutrašnjeg sekvencijalnog elementa do izlazne nožice.
7. Broju registara ili lečeva do kojih stiže više od jednog klok signala. U slučaju da više od jednog klok signala stiže do registra nije jasno koji od njih je potrebno koristiti u procesu provere vremena uspostavljanja i držanja.
8. Proveru da li interno generisani klokovi (pomoću MMCM bloka) koriste neki od primarnih klokova kao referencu.
9. Proveru da li u dizajnu postoje kombinacione petlje.
10. Broju ulaznih nožica FPGA komponente koje se koriste u tekućoj implementaciji, a imaju delimično definisano ograničenje po pitanju maksimalnog dozvoljenog propagacionog kašnjenja od ulazne nožice do prvog unutrašnjeg sekvencijalnog elementa.
11. Broju izlaznih nožica FPGA komponente koje se koriste u tekućoj implementaciji, a imaju delimično definisano ograničenje po pitanju maksimalnog dozvoljenog propagacionog kašnjenja od poslednjeg unutrašnjeg sekvencijalnog elementa do izlazne nožice.
12. Proveru da li u dizajnu postoje kombinacione petlje sa lečevima.

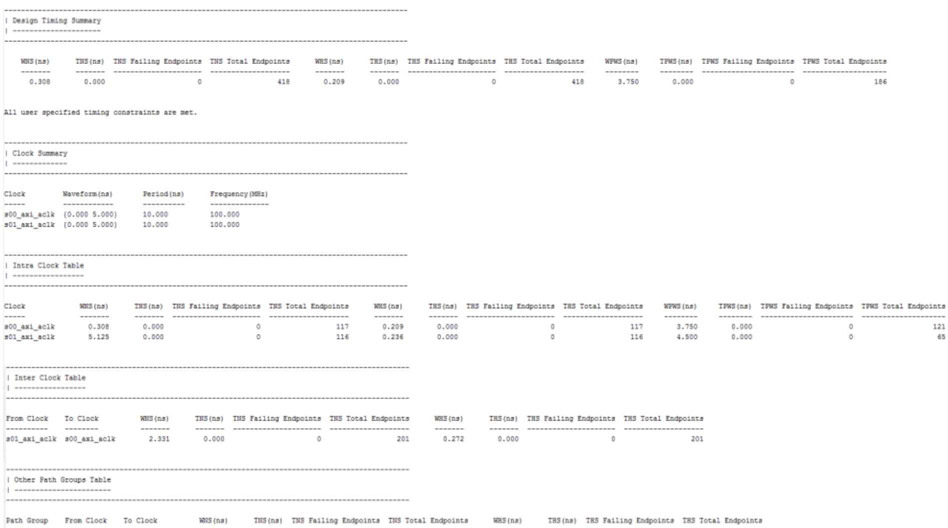
„*Design Timing*“ sekcija

„*Design Timing*“ sekcija, prikazana na slici 9.19, sadrži sledeće informacije predstavljene u vidu odvojenih tabela:

- „*Design Timing Summary*“ tabela – prikazuje podatke o ukupnom broju analiziranih unutrašnjih putanja signala i o broju onih koji zadovoljavaju i ne

zadovoljavaju odgovarajuća vremenska ograničenja. Iz ove tabele može se videti koliko putanja ne zadovoljava zahtevana vremenska ograničenja.

- „*Clock Summary*“ tabela – prikazuje informacije o svim klokovima za koje postoje definisana vremenska ograničenja u pogledu minimalno potrebne učestanosti.
- „*Intra Clock*“ tabela – prikazuje informacije o broju putanja signala, koje se nalaze unutar istog klok domena, koje zadovoljavaju i ne zadovoljavaju zahtevana vremenska ograničenja.
- „*Inter Clock*“ tabela – prikazuje informacije o broju putanja signala, koje počinju u jednom a završavaju u drugom klok domenu, koje zadovoljavaju i ne zadovljavaju zahtevana vremenska ograničenja.
- „*Other Path Groups*“ tabela – prikazuje informacije o putanjama signala koje pripadaju posebnim grupama putanja koje je definisao dizajner, i o broju putanja koje zadovoljavaju i ne zadovoljavaju zahtevana vremenska ograničenja.



```
------------------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------  ----------------------  --------------------
      0.308        0.000                      0                  418        0.209        0.000                      0                  418        3.750        0.000                       0                   186

All user specified timing constraints are met.

------------------------------------------------------------------------------------------------
| Clock Summary
| -------------
------------------------------------------------------------------------------------------------

Clock        Waveform(ns)       Period(ns)      Frequency(MHz)
-----        ------------       ----------      --------------
s00_axi_aclk  {0.000 5.000}      10.000          100.000
s01_axi_aclk  {0.000 5.000}      10.000          100.000

------------------------------------------------------------------------------------------------
| Intra Clock Table
| -----------------
------------------------------------------------------------------------------------------------

Clock             WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
-----             -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------  ----------------------  --------------------
s00_axi_aclk        0.308        0.000                      0                  117        0.209        0.000                      0                  117        3.750        0.000                       0                   121
s01_axi_aclk        5.125        0.000                      0                  116        0.236        0.000                      0                  116        4.500        0.000                       0                    65

------------------------------------------------------------------------------------------------
| Inter Clock Table
| -----------------
------------------------------------------------------------------------------------------------

From Clock    To Clock          WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints
----------    --------          -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------
s01_axi_aclk  s00_axi_aclk        2.331        0.000                      0                  201        0.272        0.000                      0                  201

------------------------------------------------------------------------------------------------
| Other Path Groups Table
| -----------------------
------------------------------------------------------------------------------------------------

Path Group    From Clock    To Clock          WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints
----------    ----------    --------          -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------
```

Slika 9.19. „Design Timing“ sekcija izveštaja o sprovedenoj statičkoj vremenskoj analizi

„*Timing Details*“ sekcija

Sledeća sekcija u izveštaju o sprovedenoj statičkoj vremenskoj analizi je „*Timing Details*“ sekcija. U okviru ove sekcije prikazani su detaljni izveštaji sprovedenim proverama vremena uspostavljanja i držanja na 10 najkritičnijih putanja signala za svaku od kombinacija „izvorni klok“-„odredišni klok“ koja postoji u dizajnu. Za svaku

od postojećih kombinacija izvornih i odredišnih klokova na početku detaljnog izveštaja navedena je informacija koji su izvorni i odredišni klok za sve putanje iz ove grupe, a zatim sledi tabela u kojoj su sumarno prikazani rezultati provere vremena uspostavljanja, držanja i širine impulsa za sve putanje iz ove grupe, kao što se može videti na slici 9.20. Na slici 9.20 vidimo da je reč o grupi putanja signala kojima je klok *s00_axi_aclk* izvorni klok, a istovremeno i odredišni klok. Takođe vidimo da statička vremenska analiza nije identifikovala niti jedno narušavanje vremena uspostavljanja i držanja na ovoj grupi putanja signala, a takođe možemo videti i kolike su minimalne vrednosti margina (*Worst Slack*) prilikom provera vremena uspostavljanja i držanja.

```
------------------------------------------------------------------------------------------------
| Timing Details
| --------------
------------------------------------------------------------------------------------------------


--------------------------------------------------------------------------------------------------
From Clock:  s00_axi_aclk
  To Clock:  s00_axi_aclk

Setup :            0  Failing Endpoints,  Worst Slack        0.308ns,  Total Violation        0.000ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.209ns,  Total Violation        0.000ns
PW    :            0  Failing Endpoints,  Worst Slack        3.750ns,  Total Violation        0.000ns
--------------------------------------------------------------------------------------------------
```

Slika 9.20. „Timing Summary“ sekcija izveštaja o sprovedenoj statičkoj vremenskoj analizi

Nakon tabele sa slike 9.20, slede detaljni izveštaji koji pokazuju kakvi su rezultati sprovedenih provera vremena uspostavljanja i držanja, koje su opisane u poglavljima 9.2.1 i 9.2.2, za 10 najkritičnijih putanja signala iz posmatrane grupe, prvo za proveru vremena uspostavljanja, a zatim za proveru vremena držanja.

Struktura izveštaja o proveri vremena uspostavljanja

Svaki izveštaj o sprovedenoj proveri vremena uspostavljanja za individualnu putanju signala ima strukturu prikazanu na slici 9.21 i sastoji se iz pet delova:

1. **Sažetak izveštaja** – sadrži informacije o:
    - da li je vreme uspostavljanja zadovoljeno, *Slack (MET)*, ili nije, *Slack (FAIL)*,
    - izvornom sekvencijalnom elementu na kojem započinje putanja signala (*Source*),
    - odredišnom sekvencijalnom elementu na kojem završava putanja signala (*Destionation*),
    - grupi putanja signala kojoj ova konkretna putanja pripada (*Path Group*),

- tipu provere koja je izvršena (*Path Type*), u primeru se slike 9.21 reč je o proveri vremena uspostavljanja (*Setup*),
- vremenskom intervalu između upisujuće i generišuće ivice odredišnog i izvornog klok signala (*Requirement*),
- ukupnom kašnjenju dela putanje koja se odnosi na propagaciju podataka (*Data Path Dealy*), sa prikazom koliki deo ovog kašnjenja je posledica prolaska kroz logičke elemnete (*logic*), a koliki je posledica prolaska kroz kanale za rutiranje (*route*),
- kroz koliko nivoa logičkih elemenata prolazi putanja (*Logic Levels*),
- koliko je smicanje između izvornog i odredišnog klok signala (*Clock Path Skew*) i kako je ono izračunato,
- kolika je vrednost nesigurnosti klok signala (*Clock Uncertainty*) i kako je ona izračunata.

2. **Kalkulacija kašnjenja duž putanje izvornog signala** – u ovom delu prikazan je način na koji je izračunata vrednost kašnjenja putanje izvornog signala, parametar T_{SCD}.

   Analizom ovog dela izveštaja možemo videti kojom putanjom putuje izvorni klok signal, od mesta svog nastanka do ulaza u sekvencijalni element na kojem započinje putanja podataka posmatrane putanje signala. Za svaki element kroz koji prolazi izvorni klok, naveden je njegov doprinos ukupnom kašnjenju.

3. **Kalkulacija kašnjenja duž putanje podataka** - u ovom delu prikazan je način na koji je izračunata vrednost kašnjenja putanje podataka, parametar T_{DPD}.

   Analizom ovog dela izveštaja možemo videti kroz koje logičke elemente i kanale za rutiranje prolazi signal podataka, na svom putu od izvornog do odredišnog sekvecijalnog elementa. Takođe se može videti i koliki je doprinos svakog logičkih elemenata i kanala za rutiranje kroz koje prolazi signal ukupnom kašnjenju duž putanje podataka.

4. **Kalkulacija kašnjenja duž putanje odredišnog signala** – u ovom delu prikazan je način na koji je izračunata vrednost kašnjenja putanje odredišnog signala, parametar T_{DCD}.

   Analizom ovog dela izveštaja možemo videti kojom putanjom putuje odredišni klok signal, od mesta svog nastanka do ulaza u sekvencijalni element na kojem završava putanja podataka posmatrane putanje signala. Za svaki element kroz koji prolazi odredišni klok naveden je njegov doprinos ukupnom kašnjenju.

5. **Kalkulacija margine vremena uspostavljanja signala za datu putanju signala** – u ovom delu prikazan je način na koji je izračunata margina vremena uspostavljanja za posmatranu putanju signala. Ukoliko je vrednost

margine pozitivna, znači da na posmatranoj putanji signala nije narušeno vreme uspostavljanja. Ukoliko je vrednost margine negativna, vreme uspostavaljanja na posmatranoj putanji signala je narušeno i dizajn neće raditi pouzdano.

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             0.308ns  (required time - arrival time)
  Source:                 memory_subsystem/a_memory/dob_reg[5]/C
                            (rising edge-triggered cell FDRE clocked by s00_axi_aclk  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            memory_subsystem/c_memory/RAM_reg/DIBDI[15]
                            (rising edge-triggered cell RAMB36E1 clocked by s00_axi_aclk  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             s00_axi_aclk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (s00_axi_aclk rise@10.000ns - s00_axi_aclk rise@0.000ns)
  Data Path Delay:        9.313ns  (logic 3.786ns (40.653%)  route 5.527ns (59.347%))
  Logic Levels:           10  (CARRY4=4 LUT2=1 LUT5=1 LUT6=4)
  Clock Path Skew:        -0.103ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    4.594ns = ( 14.594 - 10.000 )
    Source Clock Delay      (SCD):    5.154ns
    Clock Pessimism Removal (CPR):    0.457ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

1

```
    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)

                         (clock s00_axi_aclk rise edge)
                                                    0.000     0.000 r
    Y19                                             0.000     0.000 r  s00_axi_aclk (IN)
                         net (fo=0)                 0.000     0.000    s00_axi_aclk
    Y19                  IBUF (Prop_ibuf_I_O)       0.984     0.984 r  s00_axi_aclk_IBUF_inst/O
                         net (fo=1, routed)         2.205     3.189    s00_axi_aclk_IBUF
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)       0.101     3.290 r  s00_axi_aclk_IBUF_BUFG_inst/O
                         net (fo=120, routed)       1.864     5.154    memory_subsystem/a_memory/s00_axi_aclk_IBUF_BUFG
    SLICE_X108Y60        FDRE                                      r  memory_subsystem/a_memory/dob_reg[5]/C
```

2

```
    SLICE_X108Y60        FDRE (Prop_fdre_C_Q)       0.478     5.632 r  memory_subsystem/a_memory/dob_reg[5]/Q
                         net (fo=22, routed)        1.098     6.730    memory_subsystem/b_memory/dob_reg[7]_0[5]
    SLICE_X110Y62        LUT6 (Prop_lut6_I2_O)      0.296     7.026 r  memory_subsystem/b_memory/temp_next1__0_carry__0_i_2/O
                         net (fo=2, routed)         0.828     7.854    matrix_multiplier/dob_reg[1][2]
    SLICE_X107Y62        CARRY4 (Prop_carry4_DI[2]_CO[3])
                                                    0.398     8.252 r  matrix_multiplier/temp_next1__0_carry__0/CO[3]
                         net (fo=1, routed)         0.000     8.252    matrix_multiplier/temp_next1__0_carry__0_n_0
    SLICE_X107Y63        CARRY4 (Prop_carry4_CI_CO[2])
                                                    0.228     8.480 r  matrix_multiplier/temp_next1__0_carry__1/CO[2]
                         net (fo=9, routed)         0.700     9.180    memory_subsystem/a_memory/CO[0]
    SLICE_X106Y61        LUT5 (Prop_lut5_I3_O)      0.313     9.493 f  memory_subsystem/a_memory/temp_next1__60_carry__1_i_13/O
                         net (fo=1, routed)         0.605    10.098    memory_subsystem/a_memory/temp_next1__60_carry__1_i_13_n_0
    SLICE_X106Y61        LUT6 (Prop_lut6_I2_O)      0.124    10.222 r  memory_subsystem/a_memory/temp_next1__60_carry__1_i_3/O
                         net (fo=2, routed)         0.675    10.897    memory_subsystem/a_memory/temp_reg_reg[15]_3[0]
    SLICE_X104Y64        LUT6 (Prop_lut6_I0_O)      0.124    11.021 r  memory_subsystem/a_memory/temp_next1__60_carry__1_i_7/O
                         net (fo=1, routed)         0.000    11.021    matrix_multiplier/dob_reg[7]_1[1]
    SLICE_X104Y64        CARRY4 (Prop_carry4_S[1]_O[2])
                                                    0.578    11.599 r  matrix_multiplier/temp_next1__60_carry__1/O[2]
                         net (fo=1, routed)         0.459    12.058    matrix_multiplier/temp_next1[13]
    SLICE_X103Y65        LUT2 (Prop_lut2_I1_O)      0.301    12.359 r  matrix_multiplier/RAM_reg_i_47/O
                         net (fo=1, routed)         0.000    12.359    matrix_multiplier/RAM_reg_i_47_n_0
    SLICE_X103Y65        CARRY4 (Prop_carry4_S[1]_O[3])
                                                    0.640    12.999 r  matrix_multiplier/RAM_reg_i_36/O[3]
                         net (fo=2, routed)         0.582    13.581    matrix_multiplier/temp_next0[15]
    SLICE_X104Y67        LUT6 (Prop_lut6_I1_O)      0.306    13.887 r  matrix_multiplier/RAM_reg_i_9/O
                         net (fo=1, routed)         0.579    14.467    memory_subsystem/c_memory/DIBDI[15]
    RAMB36_X5Y13         RAMB36E1                                  r  memory_subsystem/c_memory/RAM_reg/DIBDI[15]
    -------------------------------------------------------------------    -------------------
```

3

```
                         (clock s00_axi_aclk rise edge)
                                                   10.000    10.000 r
    Y19                                             0.000    10.000 r  s00_axi_aclk (IN)
                         net (fo=0)                 0.000    10.000    s00_axi_aclk
    Y19                  IBUF (Prop_ibuf_I_O)       0.850    10.850 r  s00_axi_aclk_IBUF_inst/O
                         net (fo=1, routed)         2.006    12.856    s00_axi_aclk_IBUF
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)       0.091    12.947 r  s00_axi_aclk_IBUF_BUFG_inst/O
                         net (fo=120, routed)       1.647    14.594    memory_subsystem/c_memory/s00_axi_aclk_IBUF_BUFG
    RAMB36_X5Y13         RAMB36E1                                  r  memory_subsystem/c_memory/RAM_reg/CLKBWRCLK
                         clock pessimism            0.457    15.051
                         clock uncertainty         -0.035    15.016
    RAMB36_X5Y13         RAMB36E1 (Setup_ramb36e1_CLKBWRCLK_DIBDI[15])
                                                   -0.241    14.775    memory_subsystem/c_memory/RAM_reg
    -------------------------------------------------------------------
```

4

```
                         required time                        14.775
                         arrival time                        -14.467
    -------------------------------------------------------------------
                         slack                                 0.308
```

5

Slika 9.21. Struktura izveštaja o proveri vremena uspostavljanja za jednu putanju signala

Struktura izveštaja o proveri vremena držanja

Svaki izveštaj o sprovedenoj proveri vremena držanja za individualnu putanju signala ima strukturu koja je identična strukturi izveštaja o proveri vremena upostavljanja. Izveštaj o proveri vremena držanja prikazan je na slici 9.22 i sastoji se iz pet delova:

1. **Sažetak izveštaja** – sadrži informacije o:
    - da li je vreme držanja zadovoljeno, *Slack (MET)*, ili nije, *Slack (FAIL)*,
    - izvornom sekvencijalnom elementu na kojem započinje putanja signala (*Source*),
    - odredišnom sekvencijalnom elementu na kojem završava putanja signala (*Destionation*),
    - grupi putanja signala kojoj ova konkretna putanja pripada (*Path Group*),
    - tipu provere koja je izvršena (*Path Type*), u primeru se slike 9.22 reč je o proveri vremena držanja (*Hold*),
    - vremenskom intervalu između upisujuće i generišuće ivice odredišnog i izvornog klok signala (*Requirement*),
    - ukupnom kašnjenju dela putanje koja se odnosi na propagaciju podataka (*Data Path Dealy*), sa prikazom koliki deo ovog kašnjenja je posledica prolaska kroz logičke elemnete (*logic*), a koliki je posledica prolaska kroz kanale za rutiranje (*route*),
    - kroz koliko nivoa logičkih elemenata prolazi putanja (*Logic Levels*),
    - koliko je smicanje između izvornog i odredišnog klok signala (*Clock Path Skew*) i kako je ono izračunato,
    - kolika je vrednost nesigurnosti klok signala (*Clock Uncertainty*) i kako je ona izračunata, ovaj deo nije prisutan ukoliko su izvorni i odredišni klok signali isti.
2. **Kalkulacija kašnjenja duž putanje izvornog signala** – u ovom delu prikazan je način na koji je izračunata vrednost kašnjenja putanje izvornog signala, parametar T_{SCD}.

    Analizom ovog dela izveštaja možemo videti kojom putanjom putuje izvorni klok signal, od mesta svog nastanka do ulaza u sekvencijalni element na kojem započinje putanja podataka posmatrane putanje signala. Za svaki element kroz koji prolazi izvorni klok naveden je njegov doprinos ukupnom kašnjenju.
3. **Kalkulacija kašnjenja duž putanje podataka** - u ovom delu prikazan je način na koji je izračunata vrednost kašnjenja putanje podataka, parametar T_{DPD}.

Analizom ovog dela izveštaja možemo videti kroz koje logičke elemente i kanale za rutiranje prolazi signal podataka, na svom putu od izvornog do odredišnog sekvecijalnog elementa. Takođe se može videti i koliki je doprinos svakog logičkih elemenata i kanala za rutiranje kroz koje prolazi signal ukupnom kašnjenju duž putanje podataka.

4. **Kalkulacija kašnjenja duž putanje odredišnog signala** – u ovom delu prikazan je način na koji je izračunata vrednost kašnjenja putanje odredišnog signala, parametar T_{DCD}.

   Analizom ovog dela izveštaja možemo videti kojom putanjom putuje odredišni klok signal, od mesta svog nastanka do ulaza u sekvencijalni element na kojem završava putanja podataka posmatrane putanje signala. Za svaki element kroz koji prolazi odredišni klok naveden je njegov doprinos ukupnom kašnjenju.

5. **Kalkulacija margine vremena držanja signala za datu putanju signala** – u ovom delu prikazan je način na koji je izračunata margina vremena uspostavljanja za posmatranu putanju signala.

   Ukoliko je vrednost margine pozitivna, znači da na posmatranoj putanji signala nije narušeno vreme držanja. Ukoliko je vrednost margine negativna vreme držanja na posmatranoj putanji signala je narušeno i dizajn neće raditi pouzdano.

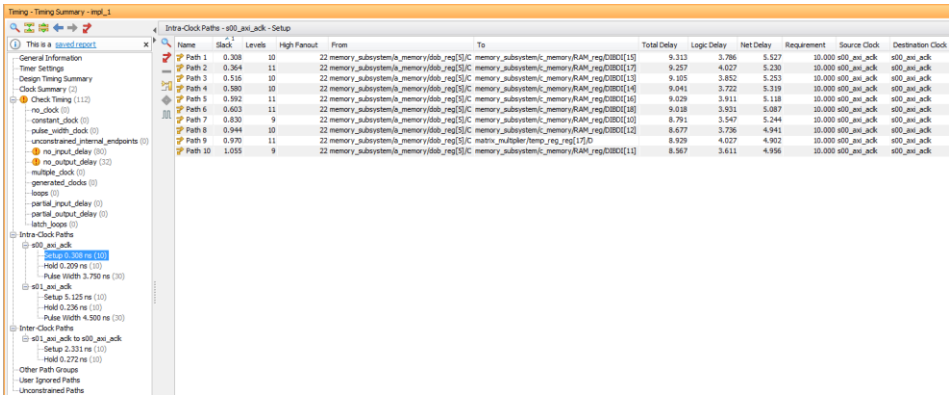
```
Min Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             0.209ns  (arrival time - required time)
  Source:                 axi_matrix_multiply_v1_0_S00_AXI_inst/axi_araddr_reg[2]/C
                            (rising edge-triggered cell FDSE clocked by s00_axi_aclk  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            axi_matrix_multiply_v1_0_S00_AXI_inst/axi_rdata_reg[0]/D
                            (rising edge-triggered cell FDRE clocked by s00_axi_aclk  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             s00_axi_aclk
  Path Type:              Hold (Min at Fast Process Corner)
  Requirement:            0.000ns  (s00_axi_aclk rise@0.000ns - s00_axi_aclk rise@0.000ns)
  Data Path Delay:        0.313ns  (logic 0.209ns (66.707%)  route 0.104ns (33.293%))
  Logic Levels:           1  (LUT5=1)
  Clock Path Skew:        0.013ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    2.069ns
    Source Clock Delay      (SCD):    1.548ns
    Clock Pessimism Removal (CPR):    0.508ns
                                                                                  1

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------  -------------------
                         (clock s00_axi_aclk rise edge)
                                                    0.000     0.000 r
    Y19                                             0.000     0.000 r  s00_axi_aclk (IN)
                         net (fo=0)                 0.000     0.000    s00_axi_aclk
    Y19                  IBUF (Prop_ibuf_I_O)       0.212     0.212 r  s00_axi_aclk_IBUF_inst/O
                         net (fo=1, routed)         0.674     0.886    s00_axi_aclk_IBUF
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)       0.026     0.912 r  s00_axi_aclk_IBUF_BUFG_inst/O                                  2
                         net (fo=120, routed)       0.636     1.548    axi_matrix_multiply_v1_0_S00_AXI_inst/s00_axi_aclk_IBUF_BUFG
    SLICE_X112Y60        FDSE                                      r  axi_matrix_multiply_v1_0_S00_AXI_inst/axi_araddr_reg[2]/C
  -------------------------------------------------------------------  -------------------
    SLICE_X112Y60        FDSE (Prop_fdse_C_Q)       0.164     1.712 f  axi_matrix_multiply_v1_0_S00_AXI_inst/axi_araddr_reg[2]/Q
                         net (fo=3, routed)         0.104     1.816    axi_matrix_multiply_v1_0_S00_AXI_inst/Q[0]
    SLICE_X113Y60        LUT5 (Prop_lut5_I3_O)      0.045     1.861 r  axi_matrix_multiply_v1_0_S00_AXI_inst/axi_rdata[0]_i_1/O
                         net (fo=1, routed)         0.000     1.861    axi_matrix_multiply_v1_0_S00_AXI_inst/reg_data_out[0]      3
    SLICE_X113Y60        FDRE                                      r  axi_matrix_multiply_v1_0_S00_AXI_inst/axi_rdata_reg[0]/D
  -------------------------------------------------------------------  -------------------

                         (clock s00_axi_aclk rise edge)
                                                    0.000     0.000 r
    Y19                                             0.000     0.000 r  s00_axi_aclk (IN)
                         net (fo=0)                 0.000     0.000    s00_axi_aclk
    Y19                  IBUF (Prop_ibuf_I_O)       0.401     0.401 r  s00_axi_aclk_IBUF_inst/O
                         net (fo=1, routed)         0.731     1.132    s00_axi_aclk_IBUF                                              4
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)       0.029     1.161 r  s00_axi_aclk_IBUF_BUFG_inst/O
                         net (fo=120, routed)       0.908     2.069    axi_matrix_multiply_v1_0_S00_AXI_inst/s00_axi_aclk_IBUF_BUFG
    SLICE_X113Y60        FDRE                                      r  axi_matrix_multiply_v1_0_S00_AXI_inst/axi_rdata_reg[0]/C
                         clock pessimism           -0.508     1.561
    SLICE_X113Y60        FDRE (Hold_fdre_C_D)       0.091     1.652    axi_matrix_multiply_v1_0_S00_AXI_inst/axi_rdata_reg[0]
  -------------------------------------------------------------------
                         required time                        -1.652
                         arrival time                          1.861
  -------------------------------------------------------------------                                                                5
                         slack                                 0.209
```

Slika 9.22. Struktura izveštaja o proveri vremena držanja za jednu putanju signala

Pored tekstualnog prikazana izveštaja o izvršenoj statičkoj vremenskoj analizi, Vivado Design Suite alat pruža mogućnost i interaktivnog pregleda ovog izveštaja. Da bismo ušli u interaktivni mod, prvo je potrebno pokrenuti komandu „*Open Implemented Design*“, u okviru *Implementation* grupe unutar „*Flow Navigator*“ prozora. Nakon što Vivado Design Suite alat pređe u režim analize implementiranog dizajna, pritiskom na *Timing* karticu u donjem delu Vivado glavnog prozora dobija se izveštaj o sprovedenoj statičkoj vremenskoj analizi koji može da se interaktivno pregleda. Struktura interaktivnog izveštaja praktično je identična strukturi tekstualnog izveštaja i prikazana je na slici 9.23.



Slika 9.23. Izgled interaktivnog izveštaja o sprovedenoj statičkoj vremenskoj analizi

Sa slike 9.23 može se videti da se u levom prozoru nalaze linkovi ka svakoj od sekcija koja postoji untuar izveštaja. Klikom na odgovarajući link u desnom prozoru se prikazuje sadržaj odabrane sekcije. Na slici 9.23 odabrana je sekcija izveštaja koja prikazuje 10 najkritičnijih putanja signala unutar *s00_axi_aclk* grupe putanja, kada se posmatra provera vremena uspostavljanja. Duplim klikom na neku od putanja prikazanih u desnom prozoru otvara se novi prozor, prikazan na slici 9.24, koji sadrži detaljan izveštaj o načinu sprovedene provere vremena uspostavljanja za odabranu putanju signala.

Sa slike 9.24 možemo primetiti da je izveštaj ponovo organizovan u nekoliko celina:

1. ***Summary*** – deo koji sadrži sažetak izveštaja, prezentujući najvažnije informacije o proveri koja je izvršena za odabranu putanju signala
2. ***Source Clock Path*** – deo koji detaljno prikazuje način na koji je izračunata vrednost kašnjenja putanje izvornog signala, parametar T_{SCD}.
3. ***Data Path*** – deo koji detaljno prikazuje način na koji je izračunata vrednost kašnjenja putanje podataka, parametar T_{DPD}.
4. ***Destination Clock Path*** – deo koji detaljno prikazuje način na koji je izračunata vrednost kašnjenja putanje odredišnog signala, parametar T_{DCD}.

Summary

Name	Path 1
Slack	0.308ns
Source	memory_subsystem/a_memory/dob_reg[5]/C (rising edge-triggered cell FDRE clocked by s00_axi_aclk {rise@0.000ns fall@5.000ns period=10.000ns})
Destination	memory_subsystem/c_memory/RAM_reg/DIBDI[15] (rising edge-triggered cell RAMB36E1 clocked by s00_axi_aclk {rise@0.000ns fall@5.000ns period=10.000ns})
Path Group	s00_axi_aclk
Path Type	Setup (Max at Slow Process Corner)
Requirement	10.000ns (s00_axi_aclk rise@10.000ns - s00_axi_aclk rise@0.000ns)
Data Path Delay	9.313ns (logic 3.786ns (40.653%) route 5.527ns (59.347%))
Logic Levels	10 (CARRY4=4 LUT2=1 LUT5=1 LUT6=4)
Clock Path Skew	-0.103ns
Clock Uncertainty	0.035ns

Source Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock s00_axi_aclk rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000	Site: Y19	s00_axi_aclk
net (fo=0)	0.000	0.000		s00_axi_aclk
IBUF (Prop_ibuf_I_O)	(r) 0.984	0.984	Site: Y19	s00_axi_aclk_IBUF_inst/O
net (fo=1, routed)	2.205	3.189		s00_axi_aclk_IBUF
BUFG (Prop_bufg_I_O)	(r) 0.101	3.290	Site: BUFGCTRL_X0Y0	s00_axi_aclk_IBUF_BUFG_inst/O
net (fo=120, routed)	1.864	5.154		memory_subsystem/a_memory/s00_axi_aclk_IBUF_BUFG
FDRE			Site: SLICE_X108Y60	memory_subsystem/a_memory/dob_reg[5]/C

Data Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
FDRE (Prop_fdre_C_Q)	(r) 0.478	5.632	Site: SLICE_X108Y60	memory_subsystem/a_memory/dob_reg[5]/Q
net (fo=22, routed)	1.098	6.730		memory_subsystem/b_memory/dob_reg[7]_0[5]
LUT6 (Prop_lut6_I2_O)	(r) 0.296	7.026	Site: SLICE_X110Y62	memory_subsystem/b_memory/temp_next1__0_carry__0_i_2/O
net (fo=2, routed)	0.828	7.854		matrix_multiplier/dob_reg[1][2]
CARRY4 (Prop_carry4_DI[2]_CO[3])	(r) 0.398	8.252	Site: SLICE_X107Y62	matrix_multiplier/temp_next1__0_carry__0/CO[3]
net (fo=1, routed)	0.000	8.252		matrix_multiplier/temp_next1__0_carry__0_n_0
CARRY4 (Prop_carry4_CI_CO[2])	(r) 0.228	8.480	Site: SLICE_X107Y63	matrix_multiplier/temp_next1__0_carry__1/CO[2]
net (fo=9, routed)	0.700	9.180		memory_subsystem/a_memory/CO[0]
LUT5 (Prop_lut5_I3_O)	(f) 0.313	9.493	Site: SLICE_X106Y61	memory_subsystem/a_memory/temp_next1__60_carry__1_i_13/O
net (fo=1, routed)	0.605	10.098		memory_subsystem/a_memory/temp_next1__60_carry__1_i_13_n_0
LUT6 (Prop_lut6_I2_O)	(r) 0.124	10.222	Site: SLICE_X106Y61	memory_subsystem/a_memory/temp_next1__60_carry__1_i_3/O
net (fo=2, routed)	0.675	10.897		memory_subsystem/a_memory/temp_reg_reg[15]_3[0]
LUT6 (Prop_lut6_I0_O)	(r) 0.124	11.021	Site: SLICE_X104Y64	memory_subsystem/a_memory/temp_next1__60_carry__1_i_7/O
net (fo=1, routed)	0.000	11.021		matrix_multiplier/dob_reg[7]_1[1]
CARRY4 (Prop_carry4_S[1]_O[2])	(r) 0.578	11.599	Site: SLICE_X104Y64	matrix_multiplier/temp_next1__60_carry__1/O[2]
net (fo=1, routed)	0.459	12.058		matrix_multiplier/temp_next1[13]
LUT2 (Prop_lut2_I1_O)	(r) 0.301	12.359	Site: SLICE_X103Y65	matrix_multiplier/RAM_reg_j_47/O
net (fo=1, routed)	0.000	12.359		matrix_multiplier/RAM_reg_j_47_n_0
CARRY4 (Prop_carry4_S[1]_O[3])	(r) 0.640	12.999	Site: SLICE_X103Y65	matrix_multiplier/RAM_reg_j_36/O[3]
net (fo=2, routed)	0.582	13.581		matrix_multiplier/temp_next0[15]
LUT6 (Prop_lut6_I1_O)	(r) 0.306	13.887	Site: SLICE_X104Y67	matrix_multiplier/RAM_reg_j_9/O
net (fo=1, routed)	0.579	14.467		memory_subsystem/c_memory/DIBDI[15]
RAMB36E1			Site: RAMB36_X5Y13	memory_subsystem/c_memory/RAM_reg/DIBDI[15]
Arrival Time		14.467		

Destination Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock s00_axi_aclk rise edge)	(r) 10.000	10.000		
	(r) 0.000	10.000	Site: Y19	s00_axi_aclk
net (fo=0)	0.000	10.000		s00_axi_aclk
IBUF (Prop_ibuf_I_O)	(r) 0.850	10.850	Site: Y19	s00_axi_aclk_IBUF_inst/O
net (fo=1, routed)	2.006	12.856		s00_axi_aclk_IBUF
BUFG (Prop_bufg_I_O)	(r) 0.091	12.947	Site: BUFGCTRL_X0Y0	s00_axi_aclk_IBUF_BUFG_inst/O
net (fo=120, routed)	1.647	14.594		memory_subsystem/c_memory/s00_axi_aclk_IBUF_BUFG
RAMB36E1			Site: RAMB36_X5Y13	memory_subsystem/c_memory/RAM_reg/CLKBWRCLK
clock pessimism	0.457	15.051		
clock uncertainty	-0.035	15.016		
RAMB36E1 (Setup_ramb36e1_CLKBWRCLK_DIBDI[15])	-0.241	14.775	Site: RAMB36_X5Y13	memory_subsystem/c_memory/RAM_reg
Required Time		14.775		

Slika 9.24. Interaktivni prikaz izveštaja o proveri vremena uspostavljanja za odabranu putanju signala

9.3. Definisanje ograničenja koja dizajn mora da zadovolji

Kao što je poznato, procesi sinteze i implementacije (raspoređivanja i povezivanja) nisu traktabilni, odnosno za njihovo optimalno rešenje trenutno ne postoje efikasni algoritmi (koji bi se završavali u polinomijalnom vremenu). Usled toga svi savremeni postupci sinteze i implementacije baziraju se na korišćenju odgovarajućih heurističkih algoritama. Heuristički algoritmi kao rezultat svog rada daju suboptimalna rešenja, a sam kvalitet rešenja u velikoj meri zavisi od inicijalnog rešenja, raspoloživog vremena za rešavanje problema, kao i efikasnog korišćenja raspoloživog vremena za optimizaciju kritičnih delova problema (što zahteva dobro poznavanje problema kako bi se unutar njega mogli identifikovati kritični delovi čijom optimizacijom se u najvećoj meri poboljšava kvalitet celokupnog rešenja).

Heuristički algoritmi koji se koriste za izvođenje procesa sinteze i implementacije odlikuju se istim karakteristikama. Ukoliko im se ne pruže dodatne informacije o kritičnim delovima dizajna koje je neophodno dodatno optimizovati, kvalitet rešenja u najvećem broju slučajeva neće biti zadovoljavajući (u smislu maksimalne učestanosti rada sistema i potrebnih hardverskih resursa). Bez dodatnih smernica heuristički algoritmi će se jednako fokusirati i na manje bitne i na više bitne delove dizajna, posvećujući im ravnomernu pažnju prilikom procesa optimizacije. Način na koji dizajner može „usmeriti“ procese sinteze i implementacije na kritične delove dizajna jeste specifikacijom **ograničenja dizajna** (*Design Constraints*).

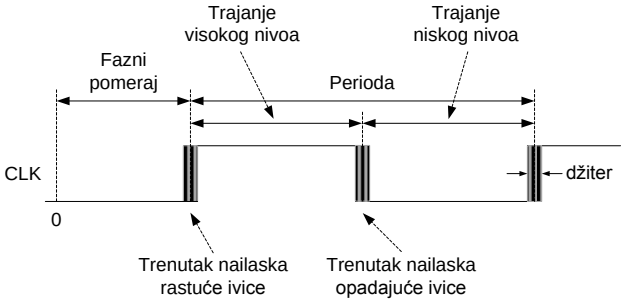
Postoje dve grupe ograničenja koja se mogu definisati za dizajn:

- **Vremenska ograničenja** (*Timing Constraints*) – pomoću ove grupe ograničenja, moguće je definisati potrebne periode klok signala koji su prisutni u dizajnu, maksimalno dozvoljena kašnjenja duž putanja signala koje postoje u dizajnu, putanje signala duž kojih kašnjenja mogu biti proizvoljna, itd.
- **Fizička ograničenja** (*Physical Constraints*) – pomoću ove grupe ograničenja, moguće je definisati pozicije ulaznih i izlaznih portova na nožicama FPGA komponente, zone unutar FPGA komponente unutar kojih je potrebno smestiti odgovarajuće module/blokove koji postoje u dizajnu, itd.

NAPOMENA: Za specifikaciju ograničenja dizajna Vivado Design Suite alat koristi podskup standardnog Tcl jezika, koji je poznat pod nazivom *Xilinx Design Constraints* (*XDC*). *XDC* fajl se interpretira kao i svaki drugi Tcl fajl. Vivado Design Suite alat obezbeđuje posebne Tcl komande pomoću kojih se mogu definisati potrebna ograničenja dizajna, koje će biti predstavljene u narednim poglavljima.

9.3.1. Definisanje vremenskih ograničenja na klok signalima

Osnovno vremensko ograničenje, koje se koristi unutar svakog dizajna koji poseduje barem jedan klok signal, jeste definisanje parametara klok signala. Za svaki klok signal koji je prisutan u dizajnu, moraju se definisati njegove karakteristike: perioda ili učestanost, fazni pomeraj, faktor ispune (odnos trajanja visokog logičkog nivoa i trajanja cele periode klok signala), nesigurnost periode (*jitter*), kao što je prikazano na slici 9.25.



Slika 9.25. Parametri klok signala

Vivado Tcl komanda *create_clock* koristi se u ovu svrhu. Ova komanda kreira klok objekat sa specificiranom periodom ili talasnim oblikom. Pomoću ove komande definišu se primarni klok objekti koje Vivado Design Suite alat zatim koristi prilikom procesa sinteze, implementacije i statičke vremenske analize. Pomoću *create_clock* komande može se kreirati i virtuelni klok objekat, koji nije povezan ni sa jednim klok signalom koji postoji u dizajnu. Virtuelni klok signal koristi se kao vremenska referenca prilikom definisanja kašnjenja ulaza i kašnjenja izlaza, o čemu će više reči biti u poglavlju 9.3.2.

Sintaksno pravilo *create_clock* Tcl komande ima sledeći oblik

create_clock *-period* <*arg*> [*-name* <*arg*>] [*-waveform* <*args*>] [*objects*]

Opcije *create_clock* komande imaju sledeće značenje:

- *period* <*arg*> – specificira periodu klok objekta koji se kreira. Specificirana vrednost periode, pomoću argumenta *arg*, mora biti strogo veća od nule. Perioda klok objekta mora biti specificirana u nanosekundama.
- *name* <*arg*> – specificira ime klok objekta koji se kreira. Ukoliko se ime klok objekta izostavi, Vivado će automatski generisati odgovarajuće ime bazirano na imenu fizičkog klok signala, definisanog preko *objects* polja, kojem se klok objekat koji se kreira asocira.

- *waveform <arg1 arg2 ...>* – pomoću ove opcije moguće je specificirati željene trenutke nailaska rastućih i opadajućih ivica unutar jednog perioda klok objekta koji se kreira. Moguće je koristiti veći broj rastućih i opadajućih ivica ali broj rastućih i opadajućih ivica mora biti isti. Pomoću argumenata specificiraju se vremenski trenuci nailaska rastućih i opadajućih ivica. Prvi argument specificira vremenski trenutak nailaska prve rastuće ivice, a drugi vremenski trenutak nailaska prve opadajuće ivice. Ukoliko je vremenski trenutak nailaska opadajuće ivice manji od vremenskog trenutka nailaska rastuće ivice, to znači da se opadajuća ivica pojavljuje pre rastuće.
- *objects* – specificira konkretni objekat unutar dizajna kome će biti asociran klok objekat koji se kreira. Objekti mogu biti portovi, pinovi ili mreže. Za specifikaciju objekta potrebno je koristiti *get_** Tcl komande.

NAPOMENA: Pomoću *create_clock* Tcl komande nije moguće specificirati vrednost džitera na klok objektu. Ukoliko je ovo potrebno mogu se koristiti sledeće dve Tcl komande

set_system_jitter <*vrednost*>
set_input_jitter <*ime_klok_objekta*> <*vrednost*>

Pomoću *set_system_jitter* Tcl komande može se specificirati vrednost sistemskog džitera, koji se primenjuje na sve klok objekte u istoj meri. Vrednost sistemskog džitera specificira se u nanosekundama.

Individualni džiter svakog klok objekta može se podesiti pomoću *set_input_jitter* Tcl komande. Ova komanda za specificirani klok objekat (pomoću polja *ime_klok_objekta*) postavlja vrednost njegovog džitera na željenu vrednost.

NAPOMENA: Ukoliko je potrebno specificirati koliko je kašnjenje duž spoljašnjih linija preko koji klok signal dolazi do FPGA komponente ovo se može uraditi korišćenjem *set_clock_latency* Tcl komande. Ovo može biti neophodno prilikom definisanja kašnjenja ulaza i kašnjenja izlaza, o kojima će biti reči u poglavlju 9.3.2.

NAPOMENA: Trenutna lista svih klok objekata koji su definisani u tekućem projektu može se dobiti izvršavanjem *report_clocks* Tcl komande.

Primer 9.23: Definisanje klok objekta pridruženog ulaznom klok portu HDL modela sa nultim faznim pomerajem i faktorom ispune od 50%

Pomoću sledeće *create_clock* Tcl komande definiše se klok objekat, koji je pridružen ulaznom klok portu *s00_axi_aclk* „*Naive*“ množača matrica.

```
create_clock -period 10.000 -name s00_axi_aclk -waveform {0.000 5.000} [get_ports s00_axi_aclk]
```

Na osnovu vrednosti parametara u prethodoj *create_clock* Tcl komandi možemo zaključiti sledeće:

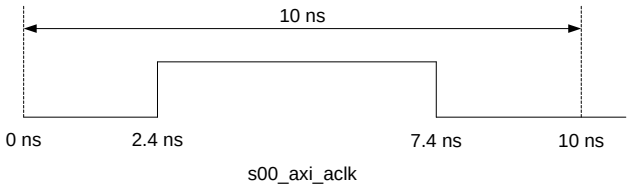
- Period kreiranog klok objekta postavljen je na 10 ns, jer je to vrednost argumenta *period* opcije.
- Ime klok objekta koji se kreira biće *s00_axi_aclk*, jer je to vrednost argumenta *name* opcije.
- Rastuća ivica nailazi u trenutku 0 ns unutar svake periode (jer je to vrednost prvog argumenta *waveform* opcije), a opadajuća nailazu u trenutku 5 ns (jer je to vrednost drugog argumenta *waveform* opcije). Ovako postavljene rastuća i opadajuća ivica impliciraju da je faktor ispune kreiranog klok objekta 50%. Takođe, pošto prva ivica (u ovom slučaju je to rastuća ivica) nailazi u trenutku 0 ns, fazni stav kreiranog klok objekta jednak je nuli.
- Kreirani objekat biće asociran *s00_axi_aclk* ulaznom portu unutar VHDL modela „*Naive*“ množača matrica, jer se objekat kome se asocira kreirani klok objekat dobija kao povratna vrednost poziva *get_ports* Tcl komande.

Primer 9.24: Definisanje klok objekta pridruženog ulaznom klok portu HDL modela sa faznim pomerajem i faktorom ispune od 50%

Pomoću sledeće *create_clock* Tcl komande definiše se klok objekat, koji je opet pridružen ulaznom klok portu *s00_axi_aclk* „*Naive*“ množača matrica, ali ovaj put poseduje fazni stav koji je različit od nule i faktor ispune koji je različit od 50%.

```
create_clock -period 10.000 -name s00_axi_aclk -waveform {2.400 7.400} [get_ports s00_axi_aclk]
```

Na osnovu vrednosti parametara *waveform* opcije u prethodoj *create_clock* Tcl komandi možemo zaključiti kreirani klok objekat ima fazni pomeraj od 2.4 ns (jer je to trenutak nailaska prve ivice u periodi klok objekta), a da je faktor ispune i dalje 50% (jer je trajanje visokog nivoa klok objekta jednako 7.4 - 2.4 = 5 ns, a perioda klok objekta je 10 ns), što se može videti i na slici 9.26.



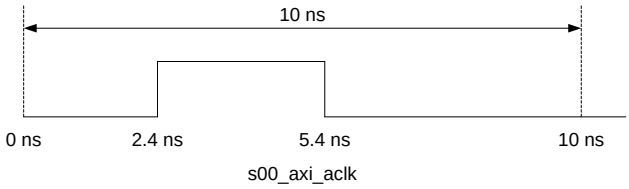
Slika 9.26. Parametri s00_axi_aclk klok objekta

Primer 9.25: Definisanje klok objekta pridruženog ulaznom klok portu HDL modela sa faznim pomerajem i faktorom ispune različitim od 50%

Pomoću sledeće *create_clock* Tcl komande definiše se klok objekat, koji je opet pridružen ulaznom klok portu *s00_axi_aclk* „*Naive*“ množača matrica, ali ovaj put poseduje fazni stav koji je različit od nule i faktor ispune koji je različit od 50%.

create_clock -period 10.000 -name s00_axi_aclk -waveform {2.400 5.400} [**get_ports** s00_axi_aclk]

Na osnovu vrednosti parametara *waveform* opcije u prethodoj *create_clock* Tcl komandi možemo zaključiti da kreirani klok objekat ima fazni pomeraj od 2.4 ns (jer je to trenutak nailaska prve ivice u periodi klok objekta), a da je faktor ispune i dalje 30% (jer je trajanje visokog nivoa klok objekta jednako 5.4 - 2.4 = 3 ns, a perioda klok objekta je 10 ns), što se može videti i na slici 9.27.



Slika 9.27. Parametri s00_axi_aclk klok objekta

Primer 9.26: Definisanje virtuelnog klok objekta

Pomoću sledeće *create_clock* Tcl komande definiše se virtuelni klok objekat, koji nije pridružen niti jednom portu, pinu ili mreži unutar tekućeg dizajna.

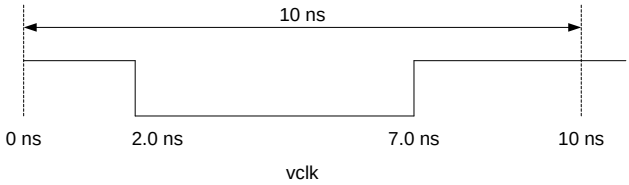
create_clock -period 10.000 -name vclk -waveform {7.000 2.000}

Na osnovu vrednosti parametara u prethodoj *create_clock* Tcl komandi možemo zaključiti sledeće:

- Period kreiranog klok objekta postavljen je na 10 ns, jer je to vrednost argumenta *period* opcije.
- Ime klok objekta koji se kreira biće *vclk*, jer je to vrednost argumenta *name* opcije.
- Rastuća ivica nailazi u trenutku 7 ns unutar svake periode (jer je to vrednost prvog argumenta *waveform* opcije), a opadajuća nailazu u trenutku 2 ns (jer je to vrednost drugog argumenta *waveform* opcije). U ovom slučaju prvo nailazi opadajuća pa onda rastuća ivica, što se može videti i na slici 9.28. Ovako postavljene rastuća i opadajuća ivica impliciraju da je faktor ispune kreiranog

klok objekta 50%. Takođe, pošto prva ivica (u ovom slučaju je to opadajuća ivica) nailazi u trenutku 2 ns, fazni stav kreiranog klok objekta jednak je 2 ns.

- Pošto nije specificiran objekat iz Vivado Design Suite baze podataka (port, pin ili mreža), kreirani klok objekat biće virtuelni klok objekat.



Slika 9.28. Parametri virtuelnog klok objekta vclk

Zadaci za vežbu

Zadatak 9.9.

Definisati klok objekat, pridružen klok portu *s00_axi_aclk* „*Naive*“ množača matrica, sa sledećim parametrima:

- perioda klok signala = 20 ns,
- fazni pomeraj = 3 ns,
- faktor ispune = 40%.

Zadatak 9.10.

Definisati klok objekat, pridružen klok portu *s00_axi_aclk* „*Naive*“ množača matrica, sa sledećim parametrima:

- perioda klok signala = 15 ns,
- fazni pomeraj = 3 ns,
- faktor ispune = 60%,
- prvo nailazi opadajuća ivica klok signala.

9.3.2. Definisanje vremenskih ograničenja na ulaznim i izlaznim portovima

Kao što je ranije rečeno, statička vremenska analiza proverava putanje signala koje započinju na izlazu nekog sekvencijalnog elementa i završavaju na ulazu nekog sekvencijalnog elementa. U slučaju unutrašnjih putanja signala FPGA komponente koje započinju i završavaju na unutrašnjim sekvencijalnim elementima svi potrebni podaci za sprovođenje analize uspostavljanja i držanja stoje na raspolaganju alatu za statičku vremensku analizu, nakon što se definišu parametri klok signala koji se dovode na klok ulaze dva sekvencijalna elementa, korišćenjem *create_clock* Tcl komande.

Međutim, putanje koje započinju ili završavaju na ulaznim odnosno izlaznim nožicama FPGA komponente ne zadovoljavaju ovaj uslov. Ovo je stoga što kod ovih putanja signala nisu poznate vremenske karakteristike dela putanje koja se nalazi izvan FPGA komponente, pa alat za statičku vremensku analizu ne može da proveri da li su vremena uspostavljanja i držanja zadovoljena. Očigledno je potrebno definisati vremenske karakteristike delova putanja koje se nalaze izvan FPGA komponente, kako bi se prilikom implementacije dizajna mogao generisati sistem koji ispravno funkcioniše. U ovu svrhu Vivado Design Suite nudi dve Tcl komande, *set_input_delay* i *set_output_delay*, za definisanje kašnjenja ulaza i kašnjenja izlaza, pomoću kojih se opisuju vremenske karakteristike delova putanja signala koje se nalaze izvan FPGA komponente.

set_input_delay Tcl komanda

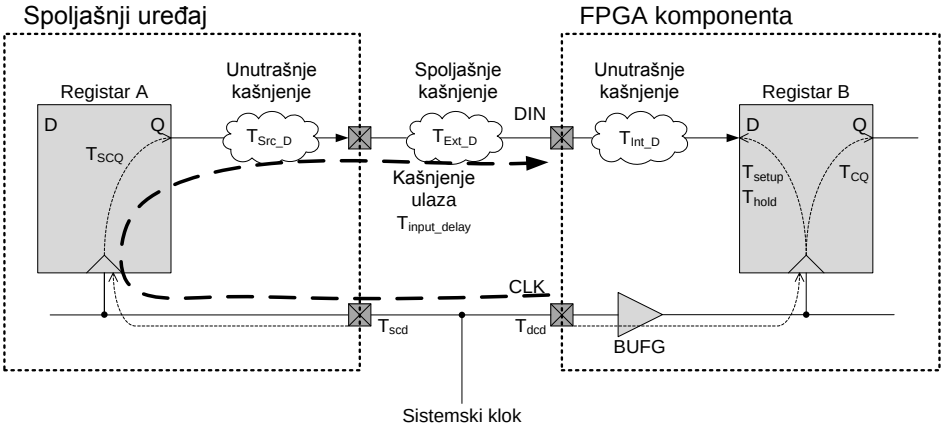
Da bi se mogla sprovesti statička vremenska analiza putanja signala koji započinju na izlazu nekog sekvencijalnog elementa koji se nalazi unutar spoljašnjeg uređaja, prolaze kroz ulazne nožice FPGA komponente, i završavaju na nekom sekvencijalnom elementu koji se nalazi unutar FPGA komponente, kao što je prikazano na slici 9.29, potrebno je definisati sledeće vremenske karakteristike dela putanje signala koja se nalazi izvan FPGA komponente:

- Koji klok signal koristi spoljašnji uređaj na čijem unutrašnjem sekvencijalnom elementu započinje putanja signala.

  Kako je moguće da spoljašnji uređaj koristi klok koji nije identičan sa klokom koji koristi FPGA komponenta, ili je reč o identičnom kloku ali postoji razlika u vremenima nailaska aktivnih ivica na spoljašnji uređaj i FPGA komponentu, usled različitih kašnjenja na linijama za razvođenje kloka (parametri T_{scd} i T_{dcd} na slici 9.29), ove karakteristike se moraju specificirati kako bi se korektno sprovela statička vremenska analiza. U ovom slučaju potrebno je koristiti virtuelni klok objekat, zajedno sa *set_clock_latency* Tcl komandom, za definisanje karakteristika klok signala koji koristi spoljašnji uređaj.

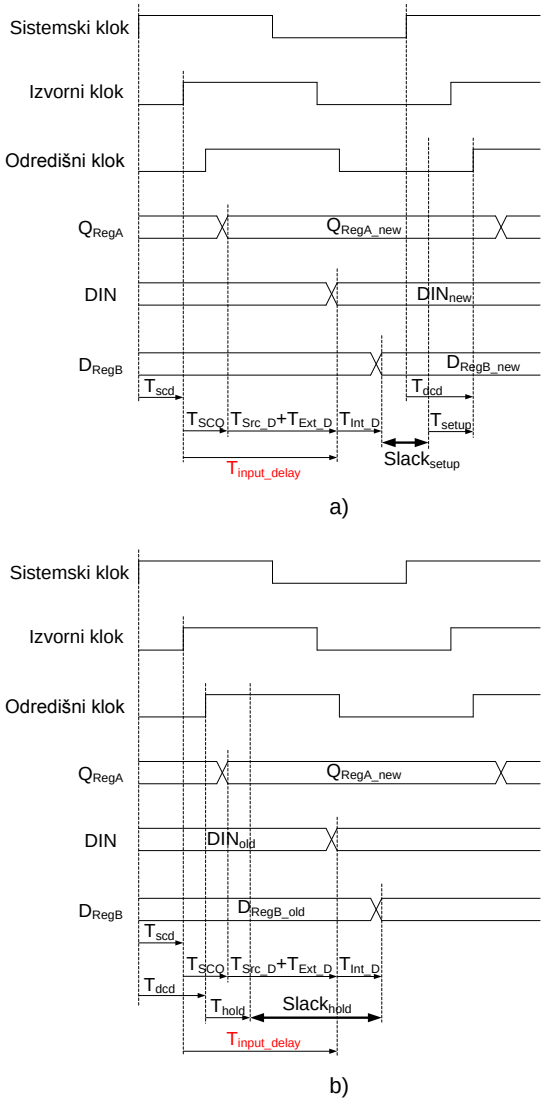
- Koliko je kašnjenje duž spoljašnje putanje signala, T_{input_delay}, koje se specificira korišćenjem *set_input_delay* Tcl komande, a koje treba da bude zbir sledećih kašnjenja:

- T_{SCQ} parametra sekvencijalnog elementa unutar spoljašnjeg uređaja na kojem započinje putanja signala,
- unutrašnjeg kašnjenja (koje uključuje propagaciju kroz logičke elemente, veze za rutiranje i izlaznu nožicu) dok signal ne napusti spoljašnji uređaj, T_{Src_D},
- kašnjenja koje je posledica prolaska signala kroz vezu koja spaja spoljašnji uređaj sa FPGA komponentom (ovo su najčešće odgovarajući vodovi na štampanoj ploči), T_{Ext_D}.



Slika 9.29. Parametri putanje signala koja ulazi u FPGA komponentu od interesa prilikom specifikacije kašnjenja ulaza

Na slici 9.30 prikazano je kako se nakon definisanja kašnjenja ulaza (parametar T_{input_delay}) sprovodi provera vremena uspostavljanja (vidi sliku 9.30a) i vremena držanja (vidi sliku 9.30b).



Slika 9.30. Korišćenje kašnjenja ulaza prilikom: a) provere vremena uspostavljanja b) provere vremena držanja, na ulaznim putanjama signala

U nastavku je prikazano uprošćeno sintaksno pravilo Vivado *set_input_delay* Tcl komande

set_input_delay [-*clock* <*args*>] [-*min*] [-*max*] <*delay*> <*objects*>

Opcije *set_input_delay* komande imaju sledeće značenje:

- *clock* <*args*> – specificira u odnosu na koji klok se definiše kašnjenje ulaza.
- *min* – specificira da se navedeno kašnjenje ulaza treba koristiti samo prilikom provere vremena držanja.
- *max* – specificira da se navedeno kašnjenje ulaza treba koristiti samo prilikom provere vremena uspostavljanja.
- <*delay*> – vrednost kašnjenja ulaza, specificirana u nanosekundama, koju je potrebno primeniti na navedene ulazne portove.
- <*objects*> – specificira listu ulaznih portova na koje će se primeniti specificirano kašnjenje ulaza. Za specifikaciju objekta potrebno je koristiti *get_ports* Tcl komande.

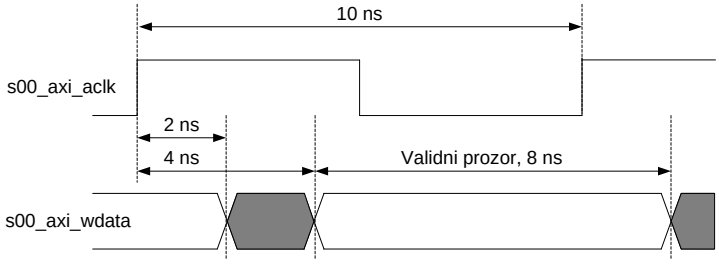
NAPOMENA: Ukoliko se prilikom definisanja kašnjenja ulaza ne koriste *–min* i *–max* opcije, specificirana vrednost kašnjenja ulaza koristi se i prilikom provere vremena uspostavljanja i vremena držanja.

Primer 9.27: Definisanje minimalnih i maksimalnih kašnjenja ulaza na ulaznom portu

Pomoću sledeće dve *set_input_delay* Tcl komande definišu se minimalna i maksimalna kašnjenja ulaza (2 ns i 4 ns respektivno) na ulaznom portu *s00_axi_wdata* HDL modela „*Naive*“ množača matrica. Pretpostavka je da je ranije definisan klok objekat *s00_axi_aclk*, kao što je pokazano u primeru 9.23.

```
set_input_delay -clock s00_axi_aclk 2 -min [get_ports s00_axi_wdata*]
set_input_delay -clock s00_axi_aclk 4 [get_ports s00_axi_wdata*]
```

Na osnovu prethodne dve Tcl komande, definisan je validni prozor na *s00_axi_wdata* ulaznom portu, kao što se može videti na slici 9.31.



Slika 9.31. Minimalne i maksimalne vrednosti kašnjenja ulaza na ulaznom portu

set_output_delay Tcl komanda

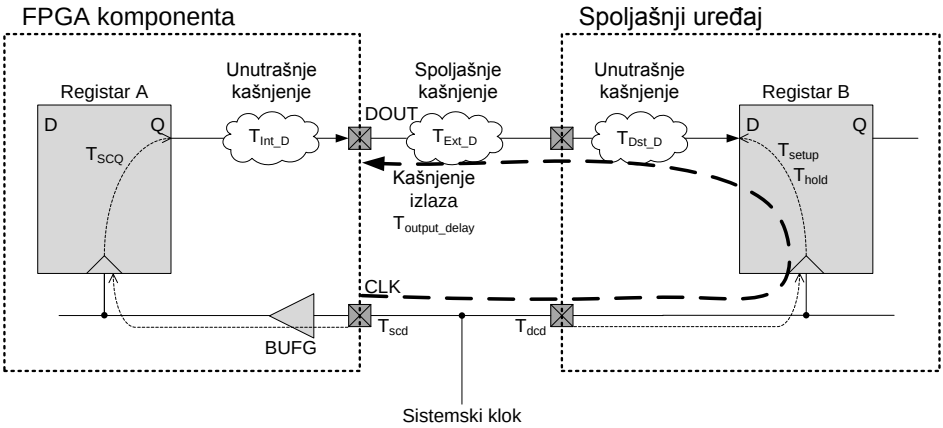
Da bi se mogla sprovesti statička vremenska analiza putanja signala koji započinju na izlazu nekog sekvencijalnog elementa koji se nalazi unutar FPGA komponente, prolaze kroz izlazne nožice FPGA komponente i završavaju na nekom sekvencijalnom elementu koji se nalazi unutar nekog spoljašnjeg uređaja, kao što je prikazano na slici 9.32, potrebno je definisati sledeće vremenske karakteristike dela putanje signala koja se nalazi izvan FPGA komponente:

- Koji klok signal koristi spoljašnji uređaj na čijem unutrašnjem sekvencijalnom elementu završava putanja signala.

  Kako je moguće da spoljašnji uređaj koristi klok koji nije identičan sa klokom koji koristi FPGA komponenta ili je reč o identičnom kloku ali postoji razlika u vremenima nailaska aktivnih ivica na spoljašnji uređaj i FPGA komponentu, usled različitih kašnjenja na linijama za razvođenje kloka (parametri T_{scd} i T_{dcd} na slici 9.32), ove karakteristike se moraju specificirati kako bi se korektno sprovela statička vremenska analiza. U ovom slučaju potrebno je koristiti virtuelni klok objekat, zajedno sa *set_clock_latency* Tcl komandom, za definisanje karakteristika klok signala koji koristi spoljašnji uređaj.

- Koliko je kašnjenje duž spoljašnje putanje signala, T_{output_delay}, koje se specificira korišćenjem *set_output_delay* Tcl komande, a koje treba da bude zbir sledećih kašnjenja:

  - T_{setup} ili -T_{hold} parametra sekvencijalnog elementa unutar spoljašnjeg uređaja na kojem završava putanja signala, u zavisnosti od provere koja se sprovodi (vremena uspostavljanja ili vremena držanja),
  - unutrašnjeg kašnjenjeg (koje uključuje propagaciju kroz ulaznu nožicu, logičke elemente i veze za rutiranje) dok signal ne dođe do untrašnjeg sekvencijalnog elementa spoljašnjeg uređaja, T_{Dst_D},
  - kašnjenja koje je posledica prolaska signala kroz vezu koja spaja spoljašnji uređaj sa FPGA komponentom (ovo su najčešće odgovarajući vodovi na štampanoj ploči), T_{Ext_D}.



Slika 9.32. Parametri putanje signala koja izlazi iz FPGA komponente od interesa prilikom specifikacije kašnjenja izlaza

U nastavku je prikazano uprošćeno sintaksno pravilo Vivado *set_output_delay* Tcl komande

set_output_delay [-*clock* <*args*>] [-*min*] [-*max*] <*delay*> <*objects*>

Opcije *set_output_delay* komande imaju sledeće značenje:

- *clock* <*args*> – specificira u odnosu na koji klok se definiše kašnjenje izlaza.
- *min* – specificira da se navedeno kašnjenje izlaza treba koristiti samo prilikom provere vremena držanja.
- *max* – specificira da se navedeno kašnjenje izlaza treba koristiti samo prilikom provere vremena uspostavljanja.
- <*delay*> – vrednost kašnjenja izlaza, specificirana u nanosekundama, koje je potrebno primeniti na navedene izlazne portove.
- <*objects*> – specificira listu izlaznih portova na koje će se primeniti specificirano kašnjenje izlaza. Za specifikaciju objekta potrebno je koristiti *get_ports* Tcl komande.

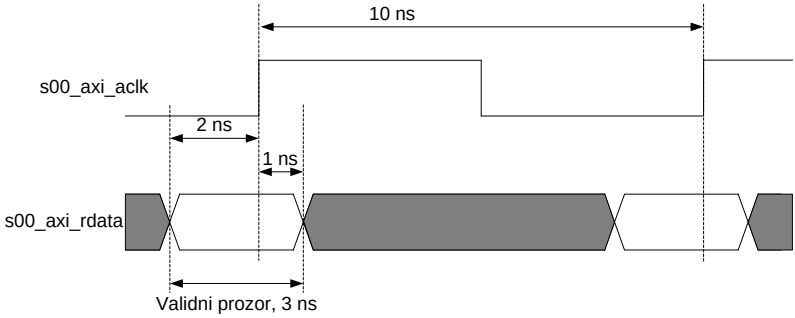
NAPOMENA: Ukoliko se prilikom definisanja kašnjenja izlaza ne koriste –*min* i –*max* opcije, specificirana vrednost kašnjenja izlaza koristi se i prilikom provere vremena uspostavljanja i vremena držanja.

Primer 9.28: Definisanje minimalnih i maksimalnih kašnjenja izlaza na ulaznom portu

Pomoću sledeće dve *set_output_delay* Tcl komande definišu se maksimalna i minimalna kašnjenja izlaza (2 ns i -1 ns respektivno) na ulaznom portu *s00_axi_rdata* HDL modela „*Naive*“ množača matrica. Pretpostavka je da je ranije definisan klok objekat *s00_axi_aclk*, kao što je pokazano u primeru 9.23.

```
set_output_delay -clock s00_axi_aclk 2 [get_ports s00_axi_rdata*]
set_output_delay -clock s00_axi_aclk -1 -min [get_ports s00_axi_rdata*]
```

Na osnovu prethodne dve Tcl komande, definisan je validni prozor na *s00_axi_rdata* ulaznom portu, kao što se može videti na slici 9.33.



Slika 9.33. Minimalne i maksimalne vrednosti kašnjenja izlaza na izlaznom portu

Zadaci za vežbu

Zadatak 9.11.

Pretpostavimo da kašnjenja pojedinih sekcija dela putanje koja se nalazi izvan FPGA komponente sa slike 9.32 imaju sledeće vrednosti:

- T_{SCQ}_min = 0.3 ns, T_{SCQ}_max = 0.7 ns
- T_{Src_D}_min = 3.1 ns, T_{Src_D}_max = 3.9 ns
- T_{Ext_D}_min = 1.5 ns, T_{Ext_D}_max = 1.7 ns

Pretpostavimo takođe da su kašnjenja klok signala CLK od izvora do FPGA komponente i spoljašnjeg uređaja usklađena, odnosno da važi $T_{scd} = T_{dcd}$. Definisati potrebna ograničenja u vezi sa kašnjenjem ulaza DIN.

Zadatak 9.12.

Pretpostavimo da kašnjenja pojedinih sekcija dela putanje koja se nalazi izvan FPGA komponente sa slike 9.29 imaju sledeće vrednosti:

- T_{Setup} = 0.2 ns, T_{Hold} = 0.5 ns
- $T_{Dst_D_}$min = 2.1 ns, $T_{Dst_D_}$max = 2.3 ns
- $T_{Ext_D_}$min = 0.8 ns, $T_{Ext_D_}$max = 1.1 ns

Pretpostavimo takođe da su kašnjenja klok signala CLK od izvora do FPGA komponente i spoljašnjeg uređaja usklađena, odnosno da važi $T_{scd} = T_{dcd}$. Definisati potrebna ograničenja u vezi sa kašnjenjem izlaza DOUT koja su neophodna zadovoljila vremena uspostavljanja i držanja na D ulazu registra *B* lociranom unutar spoljašnjeg uređaja.

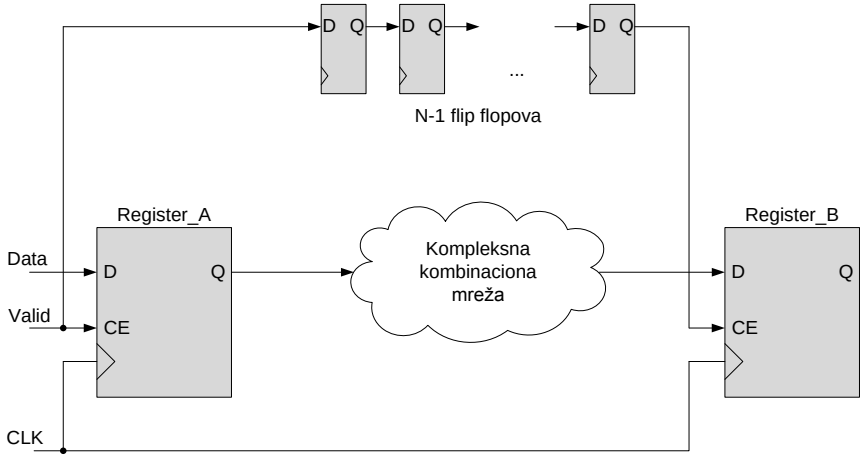
9.3.3. Definisanje izuzetaka

Vivado Design Suite alat pruža nekoliko načina da se definišu izuzeci od već definisanih ograničenja, koja su najčešće tipa *create_clock*. Pomoću izuzetaka moguće je opisati ponašanje logike koja se ne ponaša standardno. Na ovaj način alati za sintezu i implementaciju u stanju su da korektno razumeju željeno ponašanje dizajna, što uglavnom rezultuje kraćim vremenima sinteze i implementacije, i boljim krajnjim rezultatima. Unutar Vivado Design Suite alata postoji mogućnost da se definišu sledeći izuzeci:

- specifikacija putanja signala koje traju više od jednog ciklusa klok signala (*Multicycle Paths*),
- specifikacija lažnih putanja (*False Paths*),
- specifikacija minimalnih i maksimalnih kašnjenja duž odabrane putanje (*Min Delay, Max Delay*).

Specifikacija putanja signala koje traju više od jednog ciklusa klok signala (*Multicycle Paths*)

Često se u dizajnu javlja situacija da neke putanje imaju toliko veliko kašnjenje da se signali ne mogu propagirati unutar jedne periode klok signala. Jedan primer takve putanje prikazan je na slici 9.34.



Slika 9.34. Primer sistema sa putanjom čije je kašnjenje veće od jedne periode klok signala

Kompleksna kombinaciona mreža sa slike 9.34 zahteva više od jedne periode da se ulazni signali propagiraju do izlaza mreže (u primeru sa slike 9.34 potrebno je *N* perioda klok signala za propagaciju signala kroz kompleksnu kombinacionu mrežu). Zbog toga se upis novog rezultata u registar *B* vrši jednom u svakih *N* taktova. Kako su svi flip flopovi sa slike 9.34 okidani istim klok signalom, *CLK*, njegova perioda morala bi da bude dovoljno velika da se stabilizuju signali na *D* ulazu registra *B* pre upisa novih podataka u njega. Međutim, kako perioda *CLK* klok signala može biti znatno kraća, jer sve ostale putanje signala iz ovog klok domena (koje nisu prikazane na slici 9.34 ali pretpostavimo da takve putanje postoje u ostatku dizajna koji se ne vidi na slici 9.34) imaju znatno manje kašnjenje (recimo *N* puta), onda bi ovakva specifikacija periode *CLK* klok signala rezultovala u *N* puta sporijem sistemu.

Bolji način bio bi da periodu *CLK* klok signala definišemo da bude *N* puta kraća, što bi zadovoljile sve ostale putanje signala, osim onih koje prolaze kroz kompleksnu kombinacionu mrežu sa slike 9.34, a Vivado Design Suite alatu pomoću neke direktive saopštimo da putanje signala koje prolaze kroz kompleksnu kombinacionu mrežu zapravo imaju na raspolaganju *N* perioda klok signala za njihovu potpunu stabilizaciju. Na ovaj način Vivado Design Suite alat bi prilikom provere vremena uspostavljanja i držanja na ovim putanjama koristio *N* puta duže periode klok signala i ne bi prijavljivao narušavanja. Takođe bi alati za sintezu, mapiranje, raspoređivanje i povezivanje, mogli posvetiti znatno manje pažnje putanjama signala koje prolaze kroz složenu kombinacionu mrežu, jer kašnjenja duž njih mogu biti *N* puta duža, što je znatno lakši zahtev koji treba zadovoljiti. Ostatak vremena mogao bi se posvetiti optimizaciji stvarno kritičnih sekcija u dizajnu.

Vivado Design Suite alat obezbeđuje *set_multicycle_path* Tcl komandu, pomoću koje je moguće specificirati koliki broj perioda klok signala traje propagacija signala duž

odabrane putanje. Uprošćeno sintaksno pravilo *set_multicycle_path* Tcl komande ima sledeći oblik

set_multicycle_path [-*from* <*args*>] [-*to* <*args*>] [-*through* <*args*>] <*mult*>

Opcije *set_output_delay* komande imaju sledeće značenje:

- *from* <*args*> – specificira listu izvorišnih tačaka.
- *to* <*args*> – specificira odredišnih tačaka.
- *through* <*args*> – specificira listu pinova, ćelija ili mreža kroz koje putanja mora da prolazi.
- <*mult*> – kašnjenje signala duž specificirane putanje, izraženo u celobrojnom multiplu klok signala.

Primer 9.29: Definisanje višeciklusne putanje za kritične putanje sa slike 9.34

Pomoću sledeće *set_multicycle_path* Tcl komande možemo definisati da propagacija signala duž putanja koje prolaze kroz složenu kombinacionu mrežu sa slike 9.34 traju N perioda klok signala. U donjem primeru pretpostavljeno je da je $N = 10$.

set_multicycle_path 10 -from [**get_pins** Register_A/Q*] -to [**get_pins** Register_B/D*]

Specifikacija lažnih putanja (*False Paths*)

Pod lažnom putanjom podrazumeva se putanja koja postoji unutar dizajna ali nije fukcionalna, ili ne zahteva proveru vremena uspostavljanja i držanja. Primeri lažnih putanja uključuju:

- Putanje koje prelaze iz jednog klok domena u drugi, ali na kojima postoje pravilno implementirana sinhronizaciona kola.
- Registre u koje se podaci upisuju samo jednom, prilikom uključivanja napona napajanja.
- Reset ili test logiku.

Vivado Design Suite alat obezbeđuje *set_false_path* Tcl komandu pomoću koje je moguće definisati putanje signala koje ne treba da učestvuju u procesu statičke vremenske analize. Uprošćeno sintaksno pravilo *set_false_path* Tcl komande ima sledeći oblik

set_false_path [-*from* <*args*>] [-*to* <*args*>] [-*through* <*args*>]

Opcije *set_output_delay* komande imaju sledeće značenje:

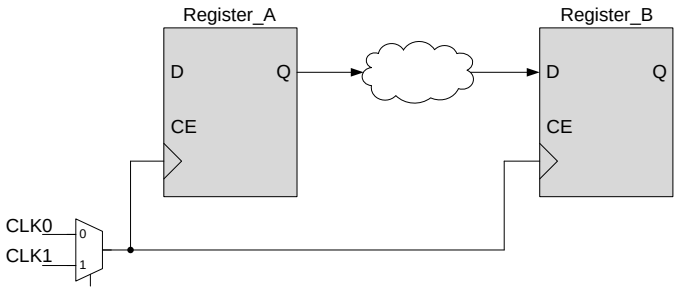
- *from* <*args*> – specificira listu izvorišnih tačaka.
- *to* <*args*> – specificira odredišnih tačaka.
- *through* <*args*> – specificira listu pinova, ćelija ili mreža kroz koje lažna putanja mora da prolazi.

Primer 9.30: Definisanje lažnih putanja u sistemu sa više klok signala

Na slici 9.35 prikazan je blok dijagram sistema koji koristi dva klok signala koji su povezani na klok multiplekser, tako da se u svakom trenutku samo jedan od njih dovodi na klok ulaze sekvencijalnih elemenata u sistemu. Usled toga ne mogu postojati putanje koje polaze iz jednog klok domena a završavaju u drugom. Vivado Design Suite alat inicijalno smatra da su sve putanje između različitih klok domena dozvoljene i smislene, tako da će, ukoliko se drugačije ne specificira, u procesu statičke vremenske analize biti analizirane i *CLK0→CLK1* i *CLK1→CLK0* putanje iako one zapravo ne mogu da postoje.

Korišćenjem sledećih *set_false_path* Tcl komandi Vivado Design Suite alatu se saopštava da ove putanje signala ne treba da se proveravaju u procesu statičke vremenske analize.

set_false_path -from [**get_clocks** CLK0] -to [**get_clocks** CLK1]
set_false_path -from [**get_clocks** CLK1] -to [**get_clocks** CLK0]



Slika 9.35. Sistem sa dva klok signala između kojih ne može biti interakcije

Specifikacija minimalnih i maksimalnih kašnjenja duž odabrane putanje (*Min Delay, Max Delay*)

U nekim slučajevima kašnjenje neke putanje signala, estimirano na način prikazan u poglavlju 9.2, može biti pogrešno. Ovo je na primer slučaj u sledećim situacijama:

- Putanja signala koji prelazi iz jednog klok domena u drugi.
- Putanje signala koje se nalaze između flip flopova koji čine kolo za sinhronizaciju.
- Čisto kombinacione putanje koje počinju na ulaznoj nožici FPGA komponente i završavaju na izlaznoj nožici FPGA komponente (putanja broj 4 na slici 9.6).

U ovim slučajevima provera vremena uspostavljanja i držanja treba da se modifikuje. Ovo je moguće korišćenjem *set_max_delay* i *set_min_delay* Tcl komandi respektivno. Uprošćena sintaksna pravila za *set_max_delay* i *set_min_delay* Tcl komande prikazana su u nastavku

set_max_delay [-*from* <*args*>] [-*to* <*args*>] [-*through* <*args*>] <*delay*>

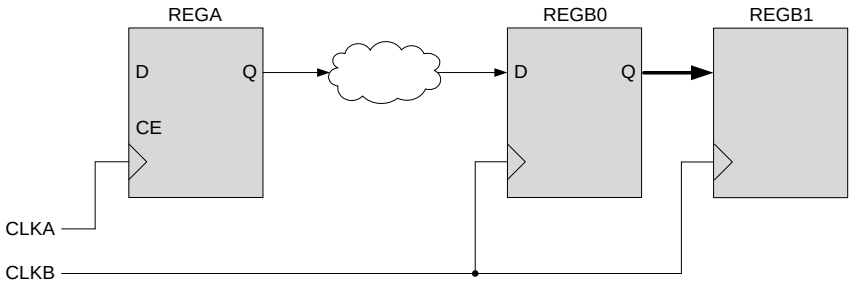
set_min_delay [-*from* <*args*>] [-*to* <*args*>] [-*through* <*args*>] <*delay*>

Opcije *set_max_delay* i *set_min_delay* komande imaju sledeće značenje:

- *from* <*args*> – specificira listu izvorišnih tačaka.
- *to* <*args*> – specificira odredišnih tačaka.
- *through* <*args*> – specificira listu pinova, ćelija ili mreža kroz koje putanja mora da prolazi.
- <*delay*> – potrebna vrednost kašnjenja, specificirana u nanosekundama.

Primer 9.31: Definisanje propagacionog kašnjenja na putanji unutar kola za sinhronizaciju

Na slici 9.36 prikazano je poznato kolo za sinhronizaciju, bazirano na dva redno povezana flip flopa (*REGB0* i *REGB1*). Vivado Design Suite alat putanju između flip flopova *REGB0* i *REGB1* (označenu debljom linijom na slici 9.36) vidi kao i svaku drugu putanju signala, pa će prilikom sinteze i impementacije nastojati da kašnjenje duž nje ne bude veće od jedne periode *CLKB* klok signala. Međutim, iz teorije sinhronizacionih kola poznato je da je za što bolji rad sinhronizatora sa slike 9.36 potrebno što je moguće više skratiti kašnjenje duž putanje koja spaja flip flopove *REGB0* i *REGB1*. Na ovaj način se pruža maksimalno vreme flip flopu *REGB0* da izađe iz metastabilnog stanja.



Slika 9.36. Sinhronizator sačinjen od dva redno vezana flip flopa

Korišćenjem *set_max_delay* Tcl komande možemo postaviti gornju granicu na kašnjenje signala između flip flopova *REGB0* i *REGB1* na vrlo malu vrednost, recimo 1 ns.

set_max_delay -from [**get_cells** REGB0] -to [**get_cells** REGB1] 1

Vivado Design Suite alat će prilikom raspoređivanja i povezivanja pokušati da zadovolji nametnutno ograničenje, smeštajući flip flopove *REGB0* i *REGB1* direktno jedan pored drugog, što predstavlja najbolju moguću implementaciju sinhronizacionog kola.

Primer 9.32: Definisanje propagacionog kašnjenja duž čisto kombinacione putanje koje prolaze kroz FPGA komponentu

Pomoću sledeće *set_max_delay* Tcl komande možemo definisati maksimalno propagaciono kašnjenje duž čisto kombinacione putanje koja prolazi kroz FPGA komponentu (putanja 4 sa slike 9.6). U donjoj komandi maksimalno propagaciono kašnjenje postavljeno je na vrednost od 8 ns.

set_max_delay -from [**get_ports** Comb_In] -to [**get_ports** Comb_Out] 8

9.3.4. Definisanje fizičkih ograničenja

Fizička ograničenja koriste se da bi se definisale fizičke karakteristika dizajna, kao što su raspored ulaznih i izlaznih portova po nožicama FPGA komponente, ulazno/izlazni (I/O) standard koji je potrebno koristiti na svakoj od nožica, pozicija elementarnih hardverskih blokova na FPGA komponenti, konfiguracioni mod koji će biti korišćen prilikom programiranja FPGA komponente, itd. Fizička ograničenja dalje se dele na:

- **Ulazno/izlazna ograničenja (*I/O Constraints*)** – pomoću kojih se definiše željeni raspored portova dizajna po raspoloživim nožicama FPGA komponente i definišu karakteristike ulazno/izlaznih blokova FPGA komponente.

- **Ograničenja u rasporedu (*Placement Constraints*)** – pomoću kojih se ograničava raspored ćelija koje čine dizajn po FPGA komponenti.
- **Ograničenja u povezivanju (*Routing Constraints*)** – pomoću kojih se fiksira način povezivanja ćelija koje čine dizajn.
- **Konfiguraciona ograničenja (*Configuration Constraints*)** – pomoću kojih se podešavaju parametri konfiguracionog fajla.

Većina fizičkih ograničenja definiše se podešavanjem vrednosti odgovarajućih osobina koje su asocirane odgovarajućem objektu, pomoću *set_property* Tcl komande

set_property <*osobina*> <*vrednost*> <*objekat*>

Ulazno/izlazna ograničenja (*I/O Constraints*)

Pomoću ulazno/izlaznih ograničenja definišu se potrebne karakteristike ulaznih i izlaznih portova, kao i ćelija koje su povezane sa ovim portovima. Ova ograničenja uključuju:

- definisanje karakteristika ulazno/izlaznog standarda koji je potrebno koristiti na odabranom portu,
- definisanje lokacije FPGA nožice na koju je potrebno mapirati odabrani port.

Kao što je rečeno, sva ova ograničenja definišu se postavljanjem odgovarajućih osobina portova na potrebne vrednosti. Najvažnije osobine portova koje se koriste u ovu svrhu su:

- **DRIVE** – specificira željenu jačinu strujnog izvora izlaznog bafera, izraženu u mA, ukoliko I/O standard bafera podržava ovu opciju. Moguće vrednosti su: 2, 4, 6, 8, 12, 16, 24 mA.
- **IOSTANDARD** – specificira koji je ulazno/izlazni standard potrebno koristiti na ulaznim, izlaznim ili bidirekcionim nožicama FPGA komponente. Podržani I/O standardi zavise od FPGA komponente koja se koristi. Za seriju 7 Xilinx FPGA komponenti neki od najčešće korišćenih I/O standarda su:
    - LVTTL – 3.3V TTL standard,
    - LVCMOS12, LVCMOS15, LVCMOS18, LVCMOS25, LVCMOS33 – 1.2V, 1.5V, 1.8V, 2.5V, 3.3V CMOS standardi,
    - LVDCI_15, LVDCI_18 – 1.5V i 1.8V I/O standardi sa mogućnošću digitalne kontrole izlazne impedanse,

- HSTL – veliki broj varijanti *High-Speed Transceiver Logic* I/O standarda (1.8V, 1.5V, 1.2V) koji se koristi za povezivanje sa memorijama,
- SSTL – veliki broj varijanti *Stub-Series Terminated Logic* I/O standarda (1.8V, 1.5V i 1.35V) koji se koristi za povezivanje sa memorijama,
- LVDS, LVDS_25 – 1.8V i 2.5V *Low Voltage Differential Signaling* I/O standard za diferencijalne veze.

- **SLEW** – specificira *slew rate* izlaznog bafera kada se koriste I/O standardi koji podržavaju ovu opciju. Moguće vrednosti su SLOW i FAST.
- **IN_TERM** – specificira nekalibrisanu vrednost ulazne impedanse za ulazne portove. Moguće vrednosti su NONE, UNTUNED_SPLIT_40 (terminacija od 40 Ω), UNTUNED_SPLIT_50 (terminacija od 50 Ω), UNTUNED_SPLIT_60 (terminacija od 60 Ω).
- **DIFF_TERM** – specificira da li je potrebno koristiti 100 Ω ulaznu diferencijalnu terminaciju, u slučaju kada se koriste diferencijalni I/O standardi. Moguće vrednosti su TRUE i FALSE.
- **PULLTYPE** – specificira tip slabog drajvera koji treba da se koristi na izlaznim portovima sa stanjem visoke impedanse i bidirekcionim portovima. Moguće vrednosti su PULLUP, PULLDOWN i KEEPER.
- **IOB** – nalaže Vivado Design Suite alatu da registar koji je povezan sa specificarnim portom mapira unutar ulazno/izlaznog bloka (*I/O Block* ili *IOB*), radi poboljšanih vremenskih karakteristika. Moguće vrednosti su TRUE, FALSE, AUTO.
- **PACKAGE_PIN** – specificira fizičku lokaciju (nožicu FPGA komponente) na koju je potrebno mapirati odabrani port.

Primer 9.33: Definisanje pozicije nožice na FPGA komponenti na koju je potrebno mapirati odabrani interfejs port dizajna

Pomoću sledeće *set_property* Tcl komande možemo definisati na koju FPGA nožicu želimo da povežemo odabrani port dizajna. U donjoj komandi nulti bit *s00_axi_wdata* ulaznog porta „*Naive*“ množača matrica mapiran je na nožicu FPGA komponente koja se nalazi na lokaciji F22.

```
set_property PACKAGE_PIN  F22 [get_ports s00_axi_wdata[0]]
```

Primer 9.34: Definisanje I/O standarda koji je potrebno koristiti na odabranom portu

Vivado Design Suite alat za svaki interfejs port koji postoji u dizajnu zahteva da se eksplicitno specificira I/O standard koji je potrebno koristiti. Pomoću sledeće

set_property Tcl komande specificira se LVCMOS25 I/O standard kao standard koji želimo da se koristi na nultom bitu ulaznog porta *s00_axi_wdata*.

set_property IOSTANDARD LVCMOS25 [**get_ports** s00_axi_wdata[0]]

Primer 9.35: Definisanje jačine strujnog izvora na izlaznom portu dizajna

Pomoću sledeće *set_property* Tcl komande možemo definisati jačinu strujnog izvora izlaznog bafera koju želimo koristiti na odabranom izlaznom portu. U donjoj komandi spcificirano je da strujni izvor na nultom bitu izlaznog porta *s00_axi_rdata* bude u stanju da generiše struju jačine 16 mA.

set_ property DRIVE 16 [**get_ports** s00_axi_rdata[0]]

Ograničenja u rasporedu (*Placement Constraints*)

Pomoću ograničenja u rasporedu definišu se željene lokacije odabranih ćelija po FPGA komponenti, postavljanjem odgovarajućih osobina ćelija na potrebne vrednosti. Najvažnije osobine ćelija koje se koriste u ovu svrhu su:

- **LUTNM** – nalaže Vivado Design Suite alatu da smesti dve LUT5, SRL16 ili LUTRAM komponente sa kompatibilnim ulazima unutar jedne LUT6 komponente. Dve komponente koje se mapiraju u istu LUT6 komponentu mogu se nalaziti bilo gde unutar dizajna.
- **HLUTNM** – nalaže Vivado Design Suite alatu da smesti dve LUT5, SRL16 ili LUTRAM komponente sa kompatibilnim ulazima unutar jedne LUT6 komponente. Dve komponente koje se mapiraju u istu LUT6 komponentu moraju se nalaziti unutar istog modula, na istom nivou hijerarhije.
- **PROHIBIT** – specificira koje SITE ili BEL komponente se ne mogu koristiti u fazi raspoređivanja. Moguće vrednosti su TRUE.
- **LOC** – specificira lokaciju unutar programabilne zone FPGA komponente na koju je potrebno mapirati odabranu ćeliju dizajna.
- **BEL** – specificira unutar kojeg SLICE/CLB bloka je potrebno mapirati odabranu ćeliju. Ćelije koje se mogu mapirati untar SLICE/CLB bloka su: flip flopovi (FD, FDCE, FDPE, FDRE, FDSE), LUT (LUT1, LUT2, LUT3, LUT4, LUT5, LUT6, LUT6_2), SRL (SRL16E, SRLC32E), LUTRAM (RAM32X1S, RAM64X1S) i konfiguracione komponente (BSCAN, ICAP, itd.).

Primer 9.36: Definisanje propagacionog kašnjenja duž čisto kombinacione putanje koje prolaze kroz FPGA komponentu

Pomoću sledeće dve *set_property* Tcl komande mapiraju se dve LUT5 ćelije (*U1/state0_inst* i *U2/state1_inst*) unutar iste LUT6 ćelije.

set_property LUTNM LUT_group1 [**get_cells** U1/state0_inst]
set_property LUTNM LUT_group1 [**get_cells** U2/state1_inst]

Primer 9.37: Definisanje nožice FPGA komponente koja se ne sme koristiti prilikom mapiranja interfejs portova dizajna

Pomoću sledeće *set_property* Tcl komande možemo zabraniti korišćenje FPGA nožice sa lokacijom Y32 u fazi mapiranja portova dizajna po nožicama FPGA komponente.

set_property PROHIBIT TRUE [**get_sites** Y32]

Primer 9.38: Definisanje lokacije na koju je potrebno mapirati registar

Pomoću sledeće *set_property* Tcl komande možemo definisati tačnu lokaciju na FPGA komponenti na koju želimo da mapiramo odabrani hardverski resurs. Sledeća komanda mapira *placed_reg* registar unutar SLICE bloka na lokaciji SLICE_X0Y0.

set_property LOC SLICE_X0Y0 [**get_cells** placed_reg]

Zadaci za vežbu

Zadatak 9.13.

Podesiti *slew rate* izlaznih bafera koji se nalaze na nožicama na koje su mapirani signali izlazne magistrale podataka *s00_axi_rdata* na vrednost FAST.

Zadatak 9.14.

Podesiti vrednost ulazne impedanse nožica na koje su mapirani signali ulazne magistrale podataka *s00_axi_wdata* na vrednost od 50 Ω.