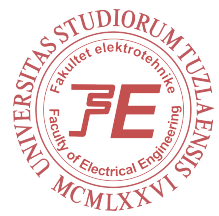
UNIVERZITET U TUZLI
FAKULTET ELEKTROTEHNIKE



VHDL - procesi

Dr. Sc. Asmir Gogić, vanr. prof.

Tuzla, 2020.

VHDL - Constraints

- ***Constraints*** - predstavljaju metod zadavanja vremenskih ***time constraints*** zahtjeva i zahtjeva za smještanje implementacije sistema na stvarni čip ***placement constraints*** kako bi se osiguralo uspješno kompajliranje ***compilation***.
- U osnovi sa ***constraints*** dajemo instrukcije kako će kompajler dizajnirati i analizirati HDL kod u cilju kreiranja validnog i funkcionalnog bitstream-a.
- U većini slučajeva ograničenja se mogu kategorizirati na:
  - ***timing assertions*** - predstavljaju niz instrukcija kojima se osigurava pravilna konfiguracija parametara ***izvora takt impulsa***,
  - ***timing exceptions*** - definiraju ***vrijeme propagacije signala*** kroz jedan ili više CLB-ova za vrijeme jednog takt impulsa i maksimalno kašnjenja signala.
  - ***physical locations*** - definicija ***funkcija IO pinova*** na stvarno čipu kao i asociranje sa signalima u HDL dizajnu
  - ***synthesis constraints*** - definira kako će se sintetizirati FSM, multiplekseri, množači i dr.
- Priliko dizajna komunikacijskih interfejsa sa izuzetno visokim bitskim brzinama (*High Speed Design*) glavni fokus će biti na modifikaciji ***timing assertions and exception constraints*** sa ciljem izvođenja ispravne sinteze bitstream-a.

VHDL - Constraints

- Dodavanjem ***ograničena se omogućuje kompajleru*** da razumije gdje će biti dodatno ***pažljiv*** prilikom ***rutiranja signala***[1] ili gdje će se vremenska ograničenja zanemariti.
- Jedan te isti HDL kod može biti sintetizovan za jednu verziju FPGA čip a za drugi ne zbog specifičnih hardverskih karateristika čipa.
- Za Xilinx FPGA ograničenja definiramo kroz tekstualni file sa ekstenzijom ****.ucf*** - ISE IDE UG625 i UG612
- Priliko definiranja ograničena neophodno je poštovati sljedeća pravila:
  - iskazi ***moraju biti terminirani*** sa karakterom ***";"***
  - ograničenja su ***osjetljiva na velike/male karaktere***,
  - jednolinijski komentari počinju sa karaterom ***#***
  - redoslije iskaza u ****.ucf*** fajlu nije bitan.
- ***Preporuka:*** odvojiti ***timing constraints*** od ***placement constraints*** u istom fajlu u cilju postizanja kompatnog zapisa.

[1] routiranje - proces povezivanja CLB-ova

VHDL - TNM

- ***TNM*** - Timing NaMe kreira grupu signala sa istim karakteristikama.
- Svi elementi označeni sa istim TNM smatraju se grupom pri čemu se novi elementi mogu dodavati koristeći višestruke iskaze.
- Osnovna sintaksa je:
  - ***INST ime_objekta TNM = identifikator***;
  - ***NET ime_objekta TNM = identifikator***;
  - ***PIN ime_objekta TNM = identifikator***;

  gdje je ***ime_objekta*** stvarno ime elementa, net-a ili pina u dizajnu a ***identifikator*** je ime time (vremenske) grupe koju kreiramo ili joj dodajemo element.
- TNM ograničenje neće proći kroz IBUF komponente. IBUF komponente su IO interejs tj ulazni i izlazni signali u entity.
- Da bi smo prevazišli prethodni problem potrebno je koristiti ***TNM_NET***.
  - ***NET net_name TNM_NET = identifikator***;
- Primjer za FPGA clock

```
NET "fclk" TNM_NET = "fclk_id";
```

VHDL - Global Timing Constraint

- ***PERIOD*** - osigurava definiranje specifikacije takt impulsa tj. perioda i popunjenosti impulsa izvora takt impulsa.

  NET clk_net_name TNM_NET = clk_group;
  TIMESPECS TS_identifikator = PERIOD clk_group value [INPUT_JITTER value];

- Primjer za FPGA clock

```
NET "fclk" TNM_NET = "fclk_id";
TIMESPEC "TS_CLK" = PERIOD "fclk_id" 10 ns HIGH 50 %;
```

VHDL - Global Timing Constraint

- ***OFFSET IN*** - osigurava definiranje kašnjenja na ulazu.

```
TIMEGRP pad_groupname OFFSET = IN  offset_name [VALID datavalid_time]
BEFORE|AFTER clk_name [TIMEGRP reg_groupname] [RISING|FALLING];
```

- ***OFFSET OUT*** - osigurava definiranje kašnjenja na izlazu.

```
TIMEGRP pad_groupname OFFSET = OUT  [offset_name] BEFORE|AFTER clk_name
TIMEGRP reg_groupname [REFERENCE_PIN ref_pin] [RISING|FALLING];
```

- ***TIMEGRP*** - grupa IO padova ili sinhronizovanih elemenata
- ***VALID*** - definira širinu prozora ulaznog podataka
- ***RISING, FALLING*** - specificira na koju ivicu se vrši prikupljanje podataka
- ***REFERENCE_PIN*** - pin u odnosu na kojeg se definira odstupanje (skew)
- Primjer:

```
NET "fclk" TNM_NET = "fclk_id";
TIMESPEC "TS_CLK" = PERIOD "fclk_id" 10 ns HIGH 50 %;
OFFSET = IN 6 ns BEFORE fclk;
OFFSET = OUT 6 ns AFTER fclk;
```

VHDL - IO atributi

- Osigurava asociranje IO signal iz entity bloka sa stvarnim IO pinovima FPGA/CPLD chip-a.

```
NET net_name LOC =  pin_number
```

```
NET "fclk" LOC = "P56";
```

- Na razvojnoj ploči doista je P56 pin na FPGA chipu XC6SLX9 koji je u TQG[2] kućištu spojen na P56 pin.
- Odabir IO pina nije slučajan jer P56 pin može pored IO funkcije da obnaša GCLK - global clock funkciju[3]
- Najčešće je situacija da na jednom FPGA čipu imamo veći broj IO pinova koji mogu obnašati ulogu GCLK-a. Na XC6SLX9 ima mogućnost zasebnih 32 izvora (IO pinova).
- IO pinovi[4] su podjeljeni u IO banke a za XC6SLX9 imamo 4 ukupno.
- Izvor tak impulsa je najčešće aktivni kristal i GPIO pin CPU/MCU-a. Na razvojoj ploči je to aktivni kristal frekvencije 25 MHz.

---

[2] TQG - package type QUAD FLAG PACK 22x22mm 102 IO pina pitch 0.5mm

[3] za više detalja o raspodjeli i funkciji IO pinova Spartan 6 FPGA familije pogledati UG385

[4] 102 single ended odnosno 51 differential

VHDL - IO atributi

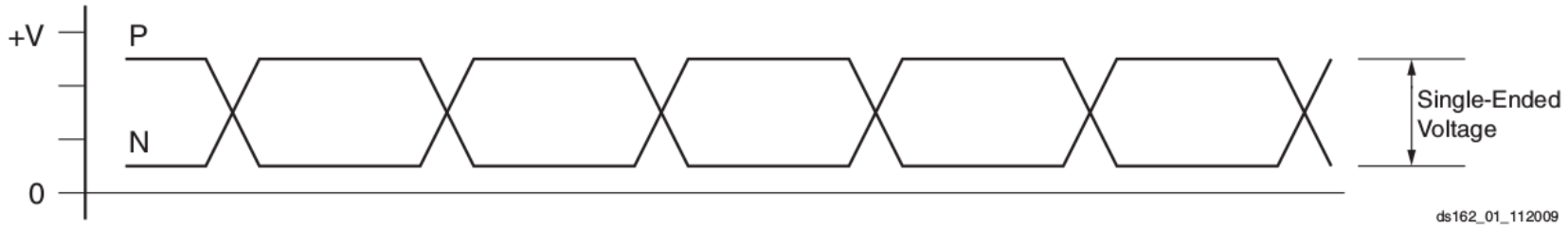


Figure 1: **Single-Ended Peak-to-Peak Voltage**

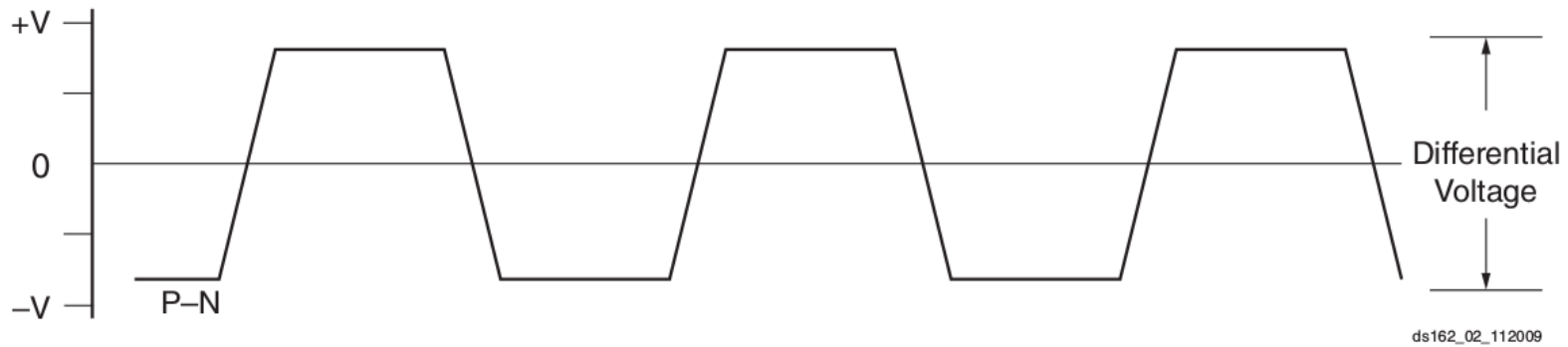


Figure 2: **Differential Peak-to-Peak Voltage**

VHDL - IO atributi

- Pored asocijacije pinova osigurana je mogućnost definiranja IO standarda[5]
  - ***LVTTL*** - single ended TTL izlaz/ulaz
  - ***LVCMOS*** - single ended CMOS izlaz/ulaz sa manjom snagom u odnosu na TTL.
  - ***LVDS*** - low voltage differential signaling - velike brzine prijenosa i zahtjeva dva IO pina.
  - ***SSTL*** - Stub Series Terminated Logic - koji je u osnovi TTL sa jedinom razlikom a to je terminiranje određenom impedansom.
  - ***HSTL*** - High-Speed Transceiver Logic -

```
NET net_name IOSTANDARD =  IO_standard
```

- Pored tipa standarda definiramo i radni napon korištenog IO standarda npr ***LVCMOS33*** ili ***LVCMOS18***[6]

[5] Karakteristika nekih standarada su djelimično opisane u Renesas AN-230

[6] za više detalja pogledati UG571 tabela 1-77

VHDL - IO atributi

- Maksimalna vrijednost struje IO pina IO standard-a definira se sa DRIVE atributom[7].

```
NET net_name DRIVE =  DRIVE_strength
```

Table 1-12: **Allowed Values for the DRIVE Attribute**

Standard	HR I/O Bank Current Drive (mA)		HP I/O Bank Current Drive (mA)	
	Allowed Values	Default	Allowed Values	Default
LVCMOS12	4, 8, or 12	12	2, 4, 6, or 8	12(1)
LVCMOS15	4, 8, 12, or 16	12	2, 4, 6, 8, or 12	12
LVCMOS18	4, 8, 12, or 16	12	2, 4, 6, 8, or 12	12
LVCMOS25	4, 8, 12, or 16	12	N/A	N/A
LVCMOS33	4, 8, 12, or 16	12	N/A	N/A
LVTTL	4, 8, 12, or 16	12	N/A	N/A

[7] Za više detalja o vrijednostima DRIVE-a pojedinih IO standarda pogledati UG571

VHDL - IO atributi

- Vrijednosti promjene napona i struje IO pinova deinirane su preko SLEW[8] atributa.
  NET net_name SLEW = SLOW|FAST
- Osnovne postavke IO pinova uključuju LVTTL IO standard 12mA struju i SLOW slew rate.

```
# TQG packaging
NET "fclk" LOC = "P56";
# BGA like packaging
NET data_in LOC = A5 | SLEW = FAST;

# example of POUT signal of type std_logic_vector association
NET "POUT(0)" LOC = "P121";
NET "POUT(1)" LOC = "P123";
NET "POUT(2)" LOC = "P124";
NET "POUT(3)" LOC = "P126";

NET "fclk" LOC = "P56";
NET "fclk" TNM_NET = "fclk";
TIMESPEC "TS_CLK" = PERIOD "fclk" 10 ns HIGH 50 %;
NET "en" LOC = "P92" | IOSTANDARD=LVCMOS33 | DRIVE=12 ;
NET "sclk" LOC = "P88" | IOSTANDARD=LVCMOS33 | DRIVE=12;
NET "dtx" LOC = "P26" | IOSTANDARD=LVCMOS33 | DRIVE=24 | SLEW=FAST;
```

[8] Za više detalja o vrijednostima DRIVE-a pojedinih IO standarda pogledati UG571

VHDL - process statement

- ***Process statement*** - predstavlja konkurentni iskaz koji sadrži grupu iskaza koji će biti izvršeni sekvencijalno (jedan za drugim).

```
my_label: process(sensitivity_list) is
  <item_declaration>
begin
  <sequential_statements>
end process my_label;
```

- ***my_label*** labela procesa i mora biti unikatna te služi u svrhu identificiranja processa
- ***sensitivity_list*** je lista signala čija promjena okida izvršenje proces-a. Tjelo procesa će biti izvršeno samo jednom na svaku promjenu signala.
- ***Vrijeme izvršenja*** jedne iteracije processa isključivo zavisi od kašnjenja logičkih blokova koji su iskorišteni za implementaciju procesa.
- Ukoliko ***lista osjetljivih*** signal ***ne postoji*** proces se izvrša neprekidno.
- Proces iskaz koristimo ako želimo implementirati funkcionalnost koja je bazirana na ***sukcesivnom izvršavanju manjih iskaza***.
- ***item_declaration*** predstavlja segment u kome se definiraju signali, varijable ili iskazi koji važe za dati process.
- ***Varijable*** možemo samo ***kreirati u procesu*** i imaju lokalni karakter $\Rightarrow$ vidljive samo u datom procesu.

VHDL - process statement

- ***Ažuriranje vrijednosti*** varijabli i signala se ne izvodi po istom pravilu,
  - vrijednosti ***varijabli*** se ***ažuriranju instant*** gdje se nalaze u procesu.,
  - vrijednosti ***signala*** se ***ažuriraju na kraju iteracije*** procesa.
- Deklaracija variable se izvodi na sljedeći način

```
variable variable_name : type := initial_value;
```

gdje ***type*** moze biti: ***integer, boolean, string, bit***

```
variable tmp : integer := 1;
variable flag : boolean := true;
variable tstr : string(1 to 32);
variable sflag : bit := '1';
variable cnt : integer range 0 to 3;
```

- U cilju identifikacije tipa događaja nad signalom koji je u osjetljivoj listi koristit ćemo funkcije ***rising_edge*** ili ***falling_edge***

Arithmetic Operators

- Sljedeća grupa ***aritmetičkih operatora*** je podržana nad varijablama:
  - **+** sabiranje,
  - **-** oduzimanje,
  - * množenje,
  - / dijeljenje, za cijele brojeve rezultat je vrijednost zaokrugljena na manji cijeli broj (bliži nula).
  - ***ABS*** apsulutna vrijednost,
  - ***MOD*** ostatak cijelobrojnog dijeljenja, ali rezultat (A MOD B) ima znak operanda B .
  - ***REM*** ostatak cijelobrojnog dijeljenja, ali rezultat (A REM B) ima znak operanda A .
  - ** stepenovanje,
- Aritmetičke operacije je dozvoljeno izvoditi samo na istom tipu (tj. sve varijable u izrazu moraju biti tipa integer ili real)

```
variable a, b, c: bit;
variable x, y: integer;
variable index range 1 to 10 := 1;
variable cycle_t: time range 10 ns to 50 ns := 10 ns;
variable mem: bit_vector (0 to 15);
variable bool_v: boolean := true;
```

Comparison Operators

- Podržani ***relacioni operatori*** su:
  - ***=*** jedanko,
  - ***/=*** različito,
  - ***<*** manje od,
  - ***>*** veće od,
  - ***<=*** manje ili jednako,
  - ***>=*** veće ili jednako.
- Podržani ***pomjeračkI/shift operatori*** su:
  - ***sll*** shift left logical,
  - ***srl*** shift right logical,
  - ***sla*** shift left arithmetic,
  - ***sra*** shift right arithmetic,
  - ***rol*** rotate left,
  - ***ror*** rotate right.

VHDL - process statement

PRIMJER 1

Napisati VHDL kod koji će implementirati jednostavnu funkcionalnost: mijenjati stanje IO pina P121 svaki 500 ms tvz. ***blinky***. Prilikom implementacije forsirati upotrebu varijabli u procesu.

PRIMJER 2

Ponoviti predthodni primjer tako što ćemo izbjeći upotrebu varijabli u procesu.

PRIMJER 3

Ponoviti predthodni primjer tako što ćemo kombinovati varijable i signale u cilju jednostavnije realizacije.

- U primjeru 2 i 3 ne možemo vidjeti efekte ažuriranja signala na kraju procesa jer je vrijeme između dvije iteracije procesa je 20 ns.

VHDL - process statement

PRIMJER 4

Modificirati na jednostavan način code primjera 2 tako da su efekti ažuriranja signala na kraju iteracije procesa jasno vidljivi u realnom vremenu.

PRIMJER 5

Modificirati primjer 1 u skladu sa logikom u primjeru 4 a zatim analizirati efekte instant ažuriranja varijabli.

VHDL - process statement - sinhronizacija

PRIMJER 6

Napisati VHDL kod koji će implementirati dva nezavisna processa. Prvi process je *free running blinky* sa LED1 a drugi process je uključivanje ili isključivanje LED-a LED2 na bazi stanja tastera.

- Broj procesa koji se može kreirati je ograničen resursima FPGA/CPLD čipa.
- Razumno je da se javlja potreba sinhronizacije dva ili više procesa ili djeljena informacija između procesa.
- Najjednostavni način sihronizacije procesa je na osnovi jednog zajedničkog signala koji u osnovi uzima informacije procesa tako što se isti implicitno mijenja $\Rightarrow$ XOR operacija signala procesa.

PRIMJER 7

Ponoviti prethodni primjer sa dva processa stime da se osigura sinhronizacija izmjene stanja LED2 ako je taster pritisnut i dešava se tranzicija sa LOW na HIGH stanja LED1. LED1 blink-a sa periodom od 2000 ms.

VHDL - process statement

- Karakterizaciju ***događaja*** možemo izvesti na dva načina koristeći:
  - ***rising_edge(signal_name)*** ili ***falling_edge(signal_name)*** funkcija,
  - ***signal_name'event and signal_name = '0'|'1'***
- Ukoliko ***signal_name*** može preći iz stanja '1' u '0' i obrnuto prirodnija je upotreba prve opcije.
- ... ***međutim*** ako to nije slučaj isti će generirati grešku.

```
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN IS
BEGIN
    RETURN (s'EVENT AND (To_X01(s) = '1') AND
                        (To_X01(s'LAST_VALUE) = '0'));
END;

FUNCTION To_X01  ( s : std_ulogic ) RETURN  X01 IS
BEGIN
    RETURN (cvt_to_x01(s));
END;

CONSTANT cvt_to_x01 : logic_x01_table := (
                      'X',  -- 'U'
                      'X',  -- 'X'
                      '0',  -- '0'
                      '1',  -- '1'
                      'X',  -- 'Z'
                      'X',  -- 'W'
                      '0',  -- 'L'
                      '1',  -- 'H'
                      'X'   -- '-'
                     );
```

VHDL - process statement

- ***Signali čuvaju*** zadnju vrijednost koja im se dodjeljuje → isti se ne mogu koristiti za spremanje rezultata među koraka procesa.
- ***Varijable*** se mogu koristiti za:
  - spremanje među rezultata pojedinih izraza procesa,
  - samo unutar sekvencijalne domene VHDL jezika tj samo unutar procesa ili pod-programa.
  - ne mogu se deklarisati ili koristiti direktno u arhitekturi. To ima za posljedicu da su varijable lokalnog karaktera.
  - Na raspolaganju sljedeći tipovi varijabli: ***bit, integer, bit_vector, string, & bool***
- Izvođenje procesa ne znači izlazak iz procesa već samo ***suspendiranje do sljedeće promjene signala*** iz liste osjetljivih signala.
- Ako proces ne sadrži osjetljivu listu ***izvršavanje se izvodi bez suspendiranja*** (beskonačna petlja).

Process statement

- Suspendovanje izvođenja procesa izvodi se sa komandom ***wait*** koja ima sljedeće varijacije:
  - ***wait on sensitivity_list*** - izmjena bilo kojeg signala u osjetljivoj listi implicira ponovno izvršavanje procesa.
  - ***wait until conditional_expresion*** - izvršavanje procesa je uslovljeno određenim uslovom: izrazom, vrijednošću signala, ivici signala.
  - ***wait for time_expression*** - izvršavanje procesa se odlaže predefiniran iznos vremenski jedinica.

```
label: wait on signal_name until condition for time_expression;
```

- U jednom ***wait*** statementu može biti jedan, dva ili sva tri iskaza.
- ***NAPOMENA***: proces ***ne može*** imati istovremeno osjetljivu listu i ***wait*** iskaz!
- Najčešće se koriste u test aplikacijama/testbench.

PRIMJER 8

Ponoviti prethodni primjer sa dva processa stime ali da se koristi ***wait on*** i ***until*** statement.

Process statement - HW

PRIMJER 9

Razviti VHDL kod koji će implementirati 4-bitni binarni brojač (P127, P131, P132 i P133) na gore čije stanje će biti inkrementirano kada korisnik pritisne taster spojen na IO P120. Promjena stanja odvijat će se u sinhronizmu sa taktimpulsima frekvencije 1Hz. Nakon dostizanja stanja 1111 brojač se resetuje.

PRIMJER 10

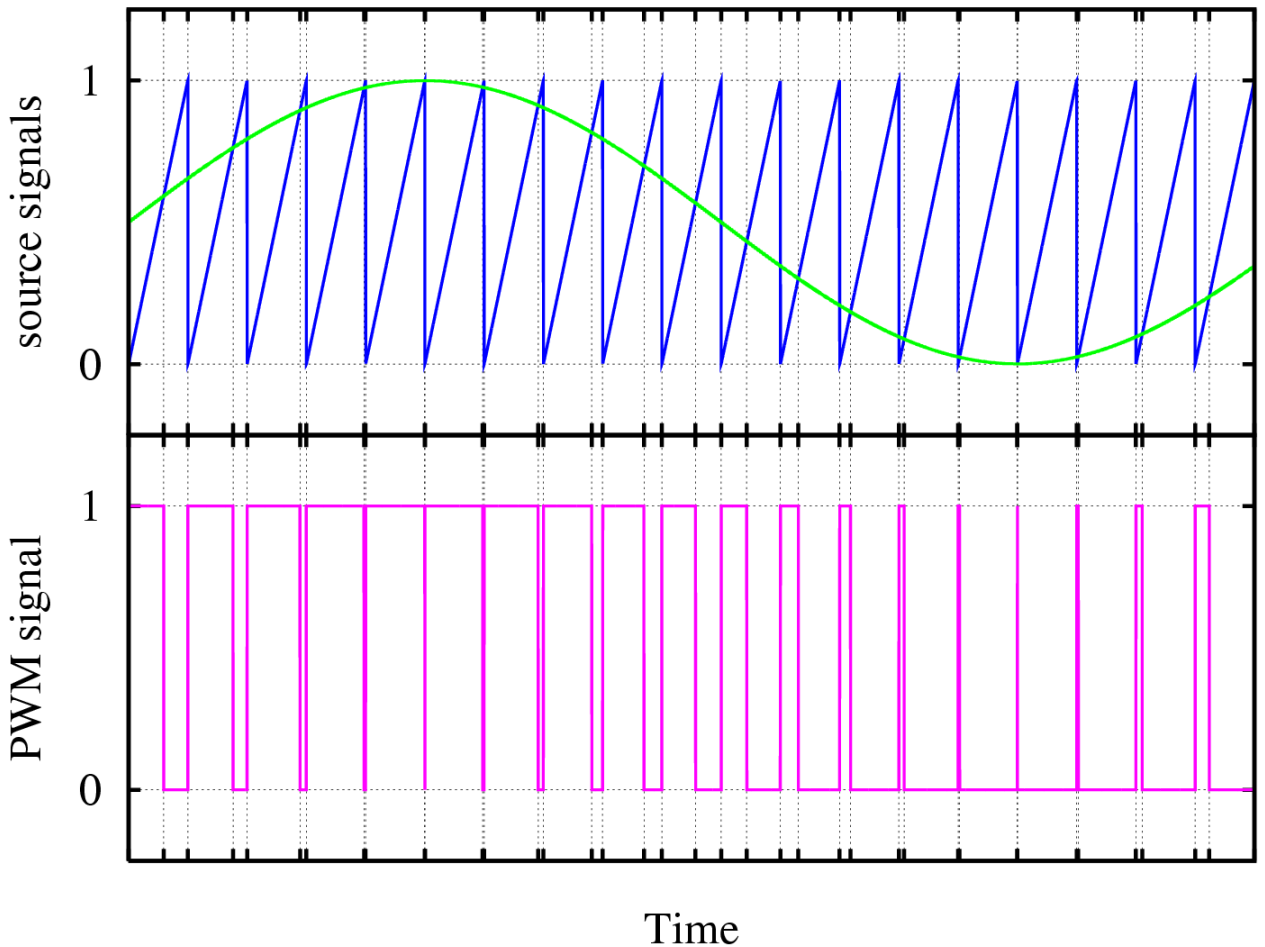
Ponoviti prethodni primjer tako da se implementira brojač na gore-dole tj kada brojač poprimi stanje 1111 svako novo okidanje tastera dekrementira stanje binarnog brojača.

PRIMJER 11

Ponoviti primjer 10 ali se za brojanje na gore koristi taster spojen na IO P120 a za brojanje na dole taster spoje na IO P134.

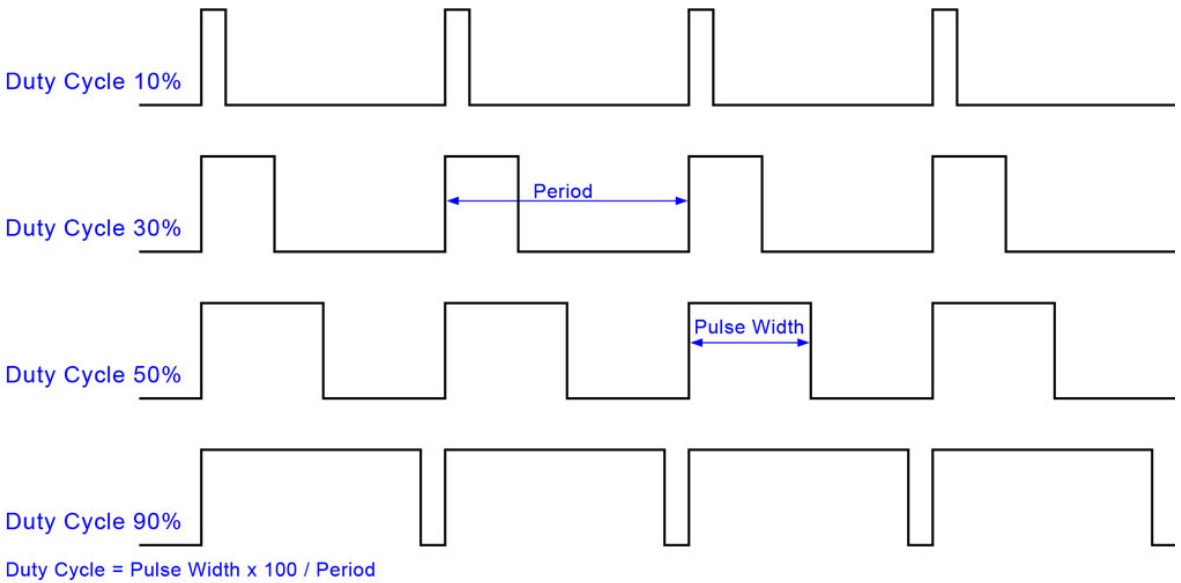
PWM

- ***PWM*** - ***P***ulse ***W***idth ***M***odulation
- Predstavlja modulacijsku tehniku u osnovno opsegu gdje se period popunjenosti povorke pravougaonih impulsa mijenja u skladu sa amplitudom signala poruke.



PWM

- U osnovi dva parametra karakteriziraju PWM modulaciju:
  - ***period*** - vremenski period koji je jednak za sve impulse i predstavlja zbir vremenskog intervala koji predstavlja stanje HIGH i stanje LOW pravougaonih impulsa.
  - ***duty cycle*** - procenat vremenskog perioda impulsa u kojem je impuls u stanju HIGH u odnosu na ukupno trajanje impulsa.



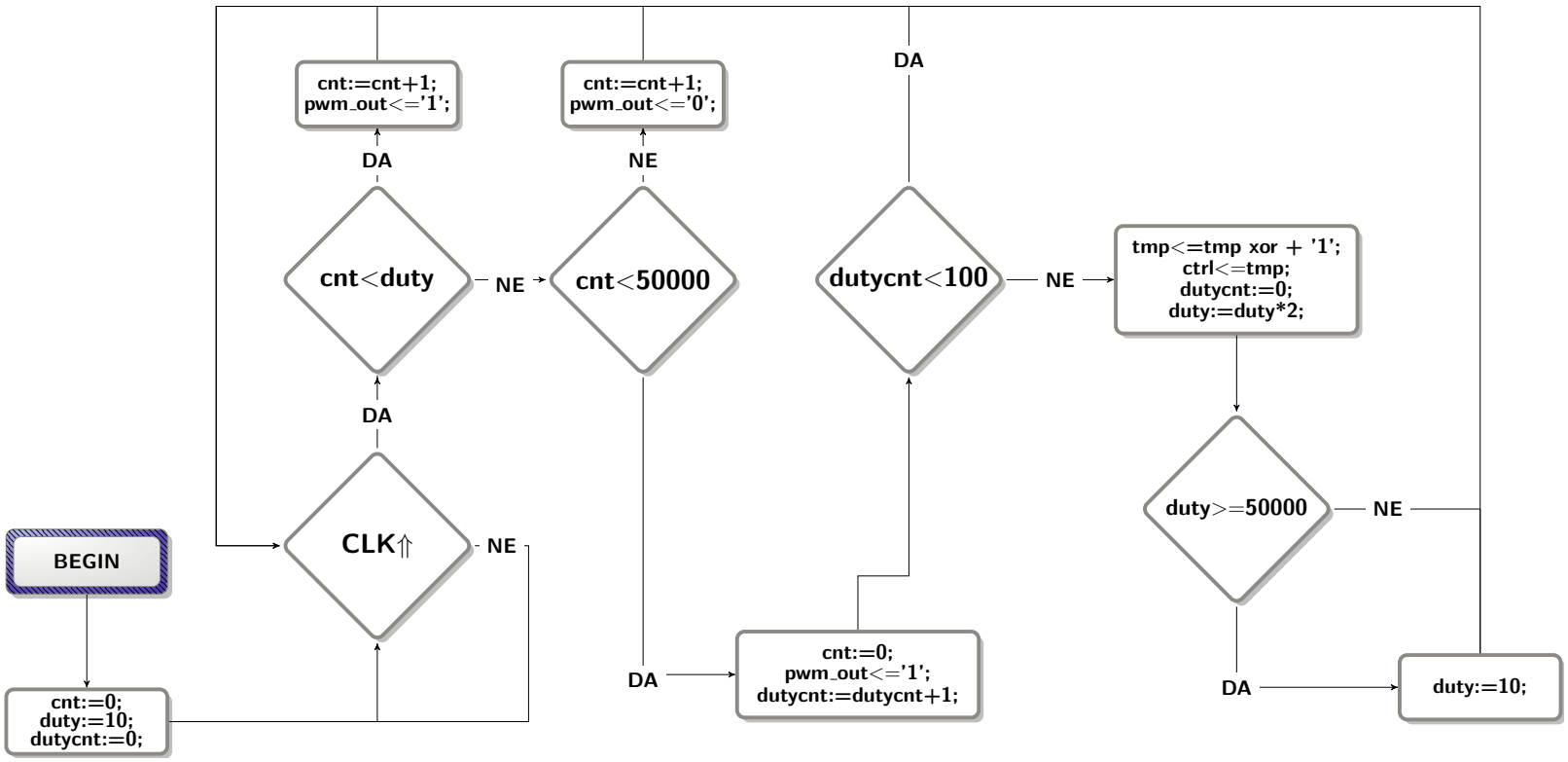
PWM - primjer

PRIMJER 12

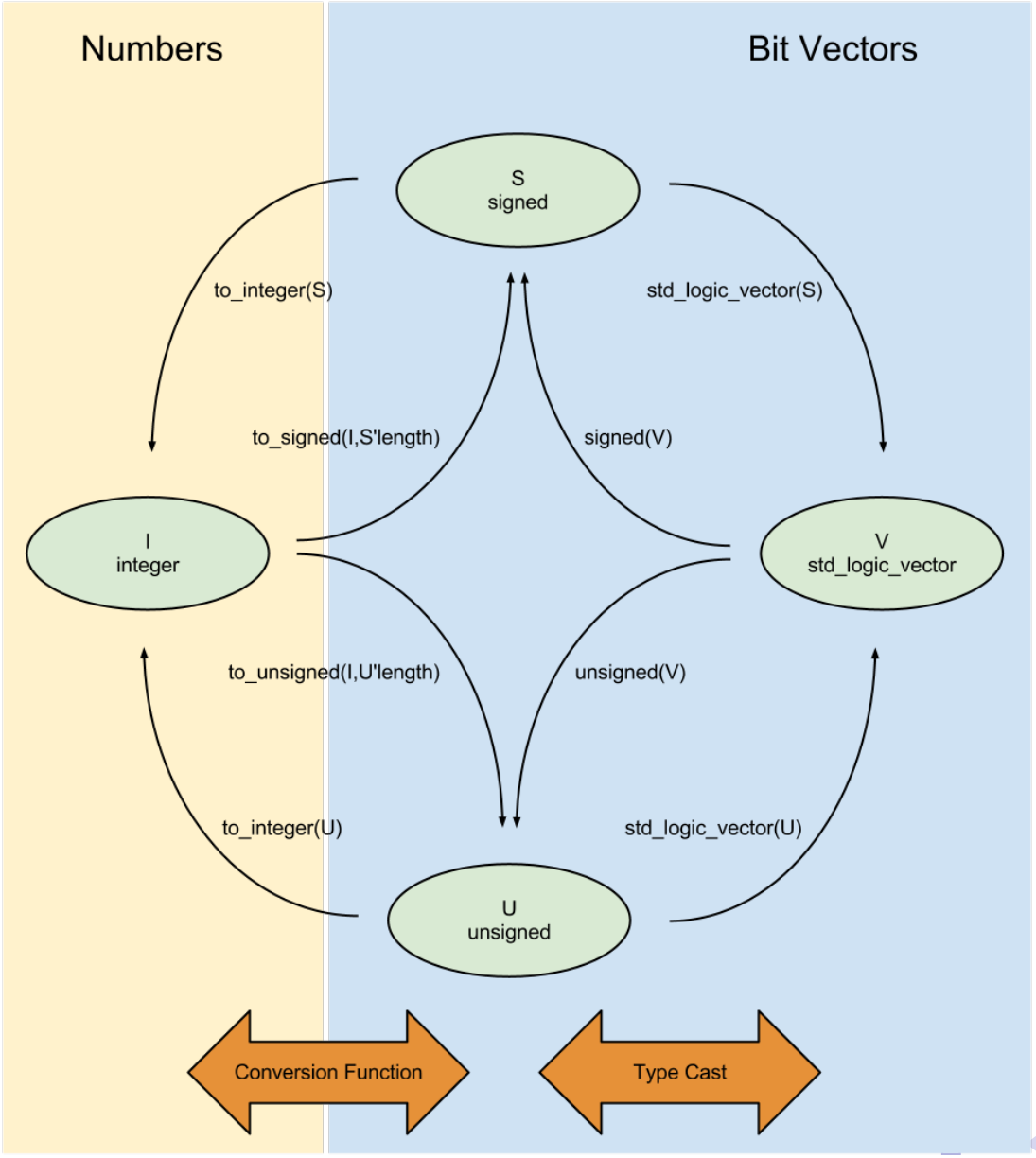
Napisati VHDL kod koji će implementirati funkcionalnost PWM modulatora kod koga je period impulsa 1 ms. Funkcionalnost testirati na način da se popunjenost impulsa mijenja u opsegu od 0 do 100%. Vrijednosti popunjenosti impulsa mijenjati svakih 100 ms. Nakon što PWM modulator dosegne popunjenost impulsa od 100%, postaviti popunjenost na 0 i ponavljati proces. Kao izlaz PWM modulatora konfigurisati IO pin P121.

NAPOMENA: Analizirati šta se dešava sa nivoom svjetline LED diode kako se povećava popunjenost diode? Da li imamo linearni inkrement nivoa svjetline? Zbog čega je prisutan navedeni problem? Kako ga riješiti?

PWM - primjer


BEGIN
cnt:=0;
duty:=10;
dutycnt:=0;
CLK⇑
NE
DA
cnt<duty
NE
DA
cnt:=cnt+1;
pwm_out<='1';
cnt<50000
NE
DA
cnt:=cnt+1;
pwm_out<='0';
dutycnt<100
NE
DA
cnt:=0;
pwm_out<='1';
dutycnt:=dutycnt+1;
tmp<=tmp xor + '1';
ctrl<=tmp;
dutycnt:=0;
duty:=duty*2;
duty>=50000
NE
DA
duty:=10;

funkcije i operatori za konverziju tipova podataka



PWM - primjer

PRIMJER 13

Unaprijediti prethodni primjer tako da se popunjenost impulsa mijenja samo dok je taster koji je spojen na IO pin P120 pritisnut.

- Modifikacija je jednostavna i sastoji se od sljedećih koraka
  - deklarisati ulazni signala ***btn*** tipa ***std_logic***
  - u tijelu procesa kreirati if iskaz koji će provjeriti stanje ulaza ***btn*** te ukoliko je isti '0'[9] izvršiti kod iz primjera 2.

PRIMJER 14

Dodatno unaprijediti prethodni primjer tako da se popunjenost impulsa mijenja na svaki "klik" tastera koji je spojen na IO pin P120.

[9] zbog činjenice da se na razvojnoj ploči nalazi pull-up na tasteru

Literatura

- RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability, 1st Editon by Pong P. Chu, 2006.
- Digital Systems Design Using VHDL 2 nd Edition, by Charles H. Roth, Jr. and Lizy Hurian John, Thomson, 2007.
- The Designer's Guide to VHDL, Third Edition, Peter J. Ashenden, 2008.