UNIVERZITET U TUZLI
FAKULTET ELEKTROTEHNIKE

VHDL: nizovi, generiranje sinusnog signala, petlje, generatori slučajnih vrijednosti

Dr. Sc. Asmir Gogić, vanr. prof.

Tuzla, 2020.

VHDL - nizovi

- Opšta sintaksa za ***definiranje niza*** je

```
type ime_niza is array (a to b) of tip_niza;
-- deklaracija niza tipa ime_niza
signal moj_niz: ime_niza;
```

Primjer:

```
type string is array (0 to 31) of character;
type bit_vector is array (12 downto 0) of bit;
-- ili
type uart_array is array (0 to 3) of unsigned (31 downto 0);
signal uart_data: uart_array :=  (0 => "00000000000000000000000001111111",
1 => "00000000000000000000000010100110",
2 => "00000000000000000000000011001010",
3 => "00000000000000000000000011100110");
--- ili
type t_matrix is array (0 to 3, 0 to 7) of integer; -- matrix 4 x 8
```

- Prilikom definiranja niza možemo se odlučiti da li će isti biti
  - ***Constrained*** - granice niza su određenje tokom deklaracije niza,
  - ***Unconstrained*** - granice niza se kreiranju tokom deklaracije niza.

VHDL - nizovi

- ***Pristupanje elementima*** niza i dodjeljivanje vrijednosti izvodi se operatorom par malih zagrada ( ) a indeksiranje kreće od vrijednosti ***nula***:

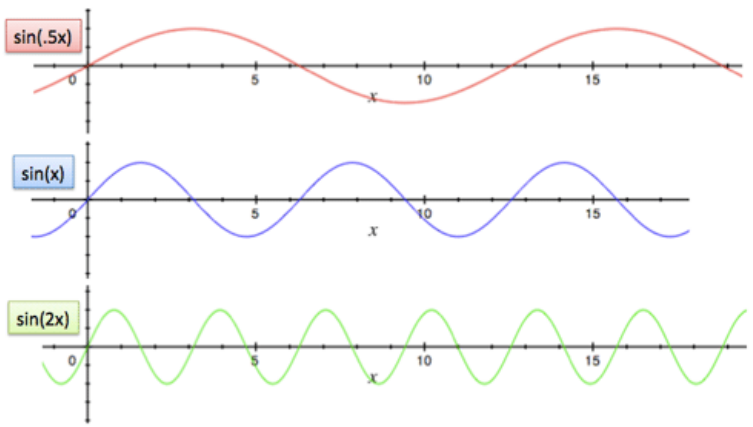
```
-- niz tipa integer od 4 element
type vec_t is array (0 to 7) of integer;
type mat_t is array (0 to 7, 0 to 15) of integer;
signal niz:vec_t;
signal matrica :mat_t;
-- dodjeljivanje vrijednosti
niz(0)  <= 1;
niz(3)  <= 5;
mat(1,2)  <= 5;
```

Primjer 1

Razviti VHDL model binarnog brojača modula 6 čija je sekvenca brojanja 3,7,1,4,6,2 koristeći nizove.

Sinusni signal - osnove

- ***Sinusni signal***[1] - fundamentalni signal pomoću kojeg možemo zapisati sve ostale periodične/aperiodične signale[2]
- ***Kako ga generirati?***
- Rješenje $\rightarrow$ ***LUT*** (Look Up Table), vektor/tabela/niz prethodno spremljenih vrijednosti amplituda sinusnog signala.
- Pogodno rješenje gdje imamo sinusni signal fiksne frekvencije **f** koji je uzorkovan frekvencijom f_s i varijabilne faze.
- FPGA čipovi rijetko imaju integriran ADC ili DAC.
- Upotreba eksternog ADC/DAC $\rightarrow$ neophodan komunikacijski modul koji će omogućiti prijenos podataka u formi pogodnoj za ADC/DAC.



[1] Sinusni kontinualni signal

[2] Kontinualni signali $\rightarrow$ Fourierova transformacija, Fourierovi redovi

Sinusni signal - primjer I - eksterni DAC sa MCU

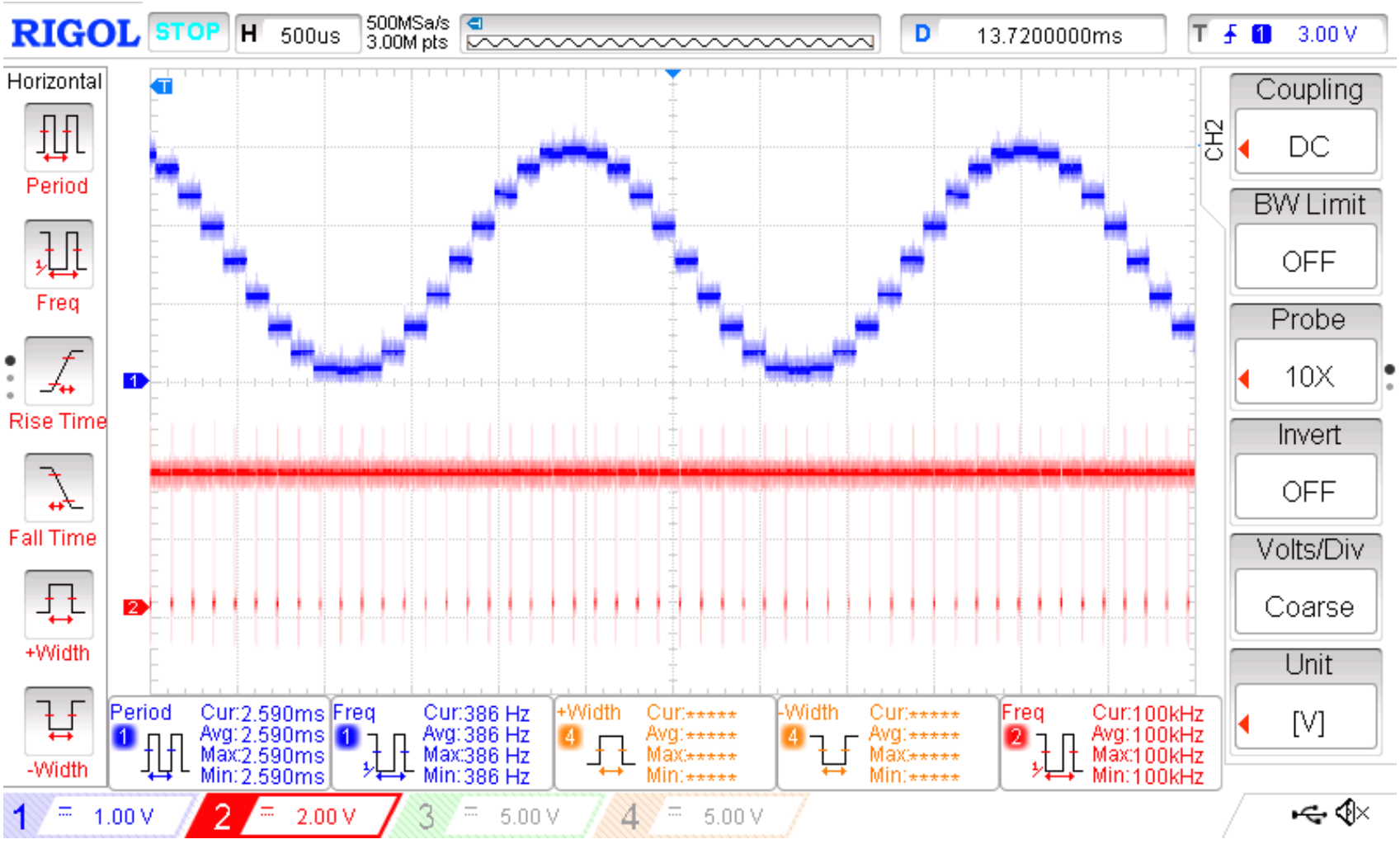
PRIMJER 2

Razviti VHDL model generatora sinusnog signala frekvencije f = 500 Hz koje je uzorkovan frekvencijom 10 kHz. Vrijednosti sinusnog signala emitovati putem UART komunikacijskog protokola prema DAC-u (implementiran sa STM32F407).

Napomene za rješenje:

- Pošto je odnos frekvencije signala i frekvencije uzorkovanja jednak 1/20, dovoljno je samo spremiti 20 uzoraka amplitude sinusnog signala 500 Hz.
- Vrijednosti amplituda sinusnog signala preračunati u Matlab-u kao Integer vrijednosti u opsegu od 0 - $(2^8 - 1)$.
- Obratiti pažnju na vrijednost amplitude uzorka tokom perioda uzorkovanja. Kako riješiti primijećeni problem?

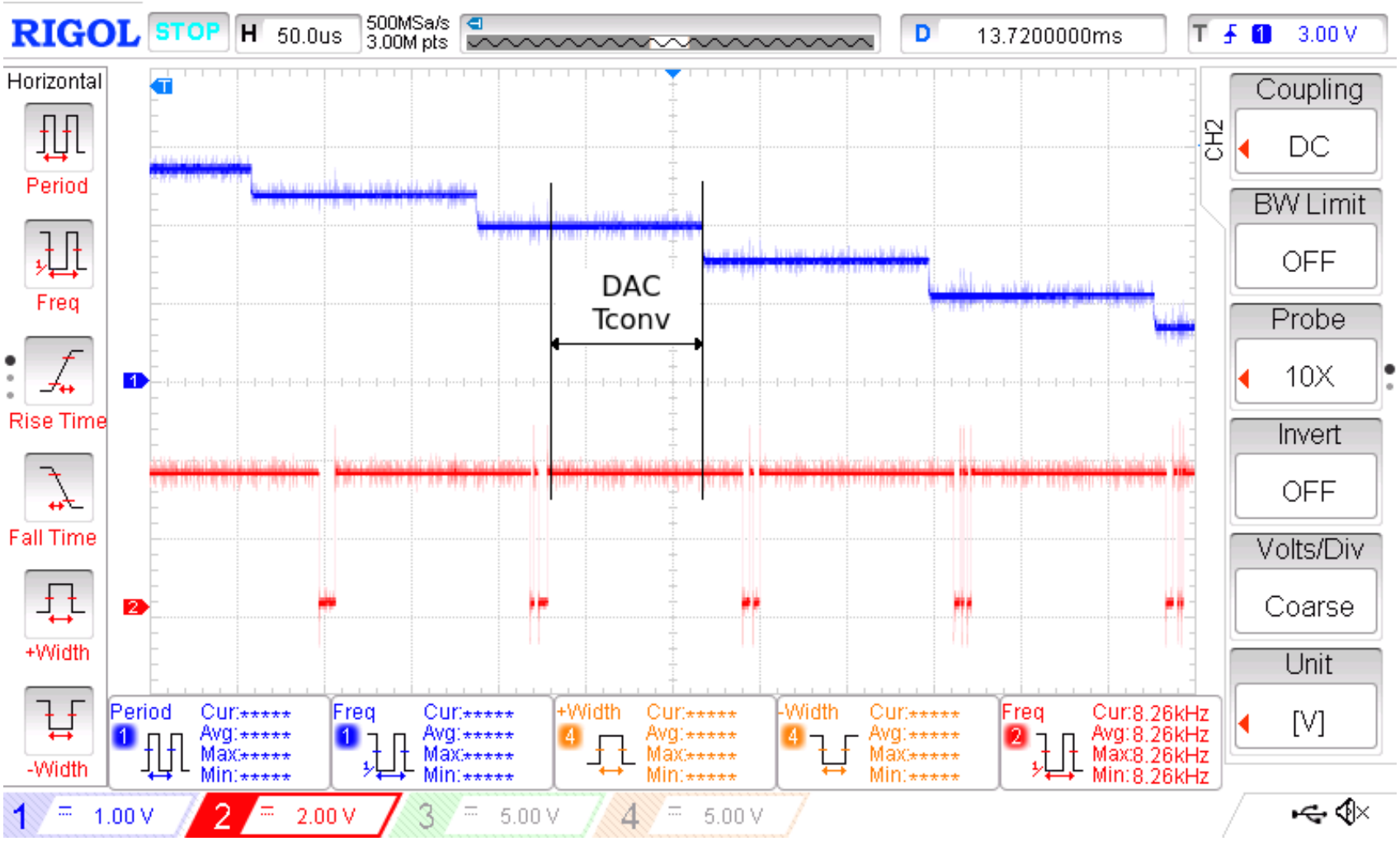
Sinusni signal - primjer I - LUT pristup



Sinusni signal - primjer I - LUT pristup

- ***Vrijeme konverzije DAC-a*** definira ***vrijeme emitovanja jednog simbola*** od izvora prema DAC-u. Za naš slučaj ono je duže nego što bi inače bilo jer MCU koji osigurava DAC ujedno vrši i ispis podataka na terminalu PC-a.
- Sa talasnog dijagrama (na sljedećem slajdu) možemo primijetiti da je vrijeme konverzije je cca 100us, što implicira da je i vrijeme za emitovanje 10 bita (1 UART simbol = START + INFORMACIJE + STOP) odnosno baudrate je cca 100kBaud-a.
- Dodatno, generirani sinusni signal nije glatka kriva zbog konačne rezolucije DAC-a (u našem slučaju 8 bita).
- ***Rješenje:*** izlaz DAC potrebno je interpolirati sa analognim NF filterom čija je granična frekvencija 5 kHz.
- Rješenje može biti i značajno povećanje frekvencije uzorkovanja x10 ili više puta.

Sinusni signal - primjer I - LUT pristup



Sinusni signal - proračun u realnom vremenu

- Drugi način generiranja sinusa jeste da se izvrši proračunavanje amplitude sinusa direktno na FPGA čipu.
- Navedeni pristup zahtijeva
  - računanje realne vrijednosti amplitude sinusnog signala (IEEE 754 support),
  - kvantiziranje vrijednosti amplitude,
  - emitovanje vrijednosti amplitude u binarnoj formi serijskim protokolom.
- ***Nedostatak:*** rad sa realnim brojevima (brojevi sa pomičnim zarezom) zahtjeva veliki broj Slice-ova.

Loop statement

- ***Loop*** iskaz omogućuje ponavljanje grupe iskaza neograničen ili ograničen broj puta, ili sve dok je zadovoljen neki uslov.
- VHDL podržava tri tipa petlji: ***loop***, ***while loop*** i ***for loop***.
- ***loop*** ili ***simple loop*** omogućava ponavljanje izvršavanja grupe iskaza neograničen broj puta. Opšta forma je:

```
[label:] loop
  statement_sequence
end loop [label];
```

- Sve petlje imaju opcionu labelu (nije mandatorna).
- Izvršenje petlje ***loop*** će biti prekinuto pozivom instrukcije ***exit***

```
runtx: loop
  cnt := cnt + 1;
  exit runtx when cnt = 5;
  end if;
end loop;
```

Loop statement

- ***while*** petlja omogućava implementiranje ponavljanja izvršavanja grupe iskaza ograničen broj puta tj. sve dok je neki uslov zadovoljen. Opšta forma je:

```
[label:] while condition loop
  statement_sequence
end loop [label];
```

- Tokom svake iteracije uslov u ***while*** petlji se testira i ukoliko je zadovoljen iskazi u tijelu petlje će sukcesivno biti izvršeni.

- ***BITNO*** je podvući da se hardware-ske petlje (loop, while i for) "odmotavaju"/unroll tokom sinteze te da će rezultirajuća logika se izvršavati paralelno.

Loop statement

- ***for*** petlja ima brojač i opseg vrijednosti koje brojač može da poprimi.
- Brojač se može inkrementirati koristeći ***to*** ili dekrementirati koristeći ***downto***.

```
for i in 1 to 10 loop
  i_square (i) <= i * i;
end loop;
```

- Za razliku od drugih programskih jezika gdje se vrijednosti brojača for petlje moglo pristupiti, u VHDL-u to nije moguće. Navedeno pravilo zabranjuje: inicijalizaciju varijabli brojačem petlje, izmjenu vrijednosti brojača unutar tijela petlje i izvođenje operacija sa brojačem.
- Brojačka varijabla se ***može koristi za indeksiranje vektora***!
- Brojačka varijabla je u osnovi (by default) tipa ***integer*** a može se eksplicitno i zadati.

```
for i in integer range 1 to 10 loop
 iskazi;
end loop;
```

Loop - primjer I

PRIMJER 3

Implementirati VHDL model koji će demonstrairati izlaz iz loop petlje nakon 5 iteracija.

PRIMJER 4

Ispraviti kod iz prethodnog primjera tako se izlazak iz ***loop*** petlje desava nakon 100 iteracija. Sintetizirati dati VHDL kod a za FPGA chip.

- Problem ***Non-static loop limit exceeded*** u osnovi govori da je nemoguće orediti količinu logičkih elemenata za datu petlju dok se data petlja ne pokrene tvz ***"CATCH 22"***.
- Zašto je tako pa zato što je početna vrijednost podložna promjeni stim u vezi i broj iteracija.
- ***Zaključak:*** loop i ***while*** petlje pogodne za simulacije.

Loop - primjer II

Primjer 5

Implementirati VHDL model koji će demonstrairati iteriranje for petlje sa 5 iteracija.

Primjer 6

Ispraviti kod iz prethodnog primjera tako se izlazak iz ***loop*** petlje desava nakon 100 iteracija. Sintetizirati dati VHDL kod a za FPGA chip.

Next Statement

- U situacijama kada se treba određena grupa iskaza u okviru petlje preskočiti uslijed činjenice da je zadovoljen određeni uslov koristimo ***next statement*** (C analogija sa continue and break).
- Opšta forma je:

```
next [label] [when condition];
```

- ***next statement*** se može samo pojaviti u petlji a preskakanje ostatka instrukcija se odvija samo za datu iteraciju.
- Osnovna uloga labele je da poveća čitljivost koda.
- Može se koristiti kao zamjena za if za uslovno izvođenje određene grupe iskaza.
- Sa hardverske perspektive ***next i if statement*** zahtijevaju isti hardware te je do dizajnera koju će sintaksu koristiti.
- Za primjer analizirajmo sistem koji broji ukupan broj jedinica na sabirnici.

Next Statement

```
library ieee;
use ieee.numeric_bit.all;
entity count_ones is
  port (v: in bit_vector (15 downto 0);
        count: out signed (3 downto 0));
end count_ones;

architecture functional of count_ones is
begin
  process (v)
  variable result: signed (3 downto 0);
  begin
    result := (others => '0');
    for i in v'range loop
      next when v(i) = '0';
      result := result + 1;
    end loop;
    count <= result;
  end process;
end functional;
```

- Ako je neophodno iterirati kroz elemente niza (sabirnica u našem slučaju) sa lijeva na desno ne ovisno o činjenici da li se radi o rastućem ili opadajućem slijedu, neophodno je koristi ključnu riječ ***'range***,

```
for i in v'range loop
```

Next Statement

- Ako je neophodno iterirati kroz elemente niza sa desna na lijevo neophodno je koristi ključnu riječ ***'reverse_range***,

```
for i in v'reverse_range loop
```

- Ako je neophodno iterirati kroz elemente niza od najmanjeg do najvećeg ne ovisno o činjenici da li niz ima rastući ili opadajući slijed neophodno je koristi atribute ***'low*** i ***'high*** zajedno sa ključnom riječi ***to***

```
for i in v'low to v'high loop
```

- Ako je neophodno iterirati kroz elemente niza od najvećeg do najmanjeg ne ovisno o činjenici da li niz ima rastući ili opadajući slijed neophodno je koristi atribute ***'low*** i ***'high*** zajedno sa ključnom riječi ***downto***

```
for i in v'high downto v'low loop
```

- Za primjer, ako želimo pristupiti bit-ima sabrinice ***nbus*** kroz brojač ***i*** od najznačajnijeg bita do najmanje značajnog bita koristiti ćemo princip

```
for i in nbus'range loop
```

u suprotnom

```
for i in nbus'reverse_range loop
```

Exit Statement

- ***exit*** iskaz omoguće prekidanje daljnjeg izvođenja petlje (for ili while).
- Može se primjenjivati u ugnježdenim petljama.
- Nakon izvršenja ***exit*** iskaza nastavlja se sa izvršenjem koda iza ***end loop*** dijela.

```
variable my_sum : integer := 0;

for cnt_val in 0 to 50 loop
   if (my_sum = 20) then
      exit;
   end if;
   my_sum := my_sum + 1;
end loop;
```

```
variable my_sum : integer := 0;

while (my_sum < 300) loop
   exit when (my_sum = 20);
   my_sum := my_sum + 1;
end loop;
--
--
```

Sequential assert Statement

- ***assert*** iskaz je pogodan za prikaz/ispis upozorenja (unutar konzole ISim-a) u tok izvođenja simulacije modela.
- Iskaz provjerava logički test i na bazi rezultata (ako je rezultat false) ispusuje definiranu poruku, u suprotnom ispis se preskače.
- Opšti oblik sintakse je:

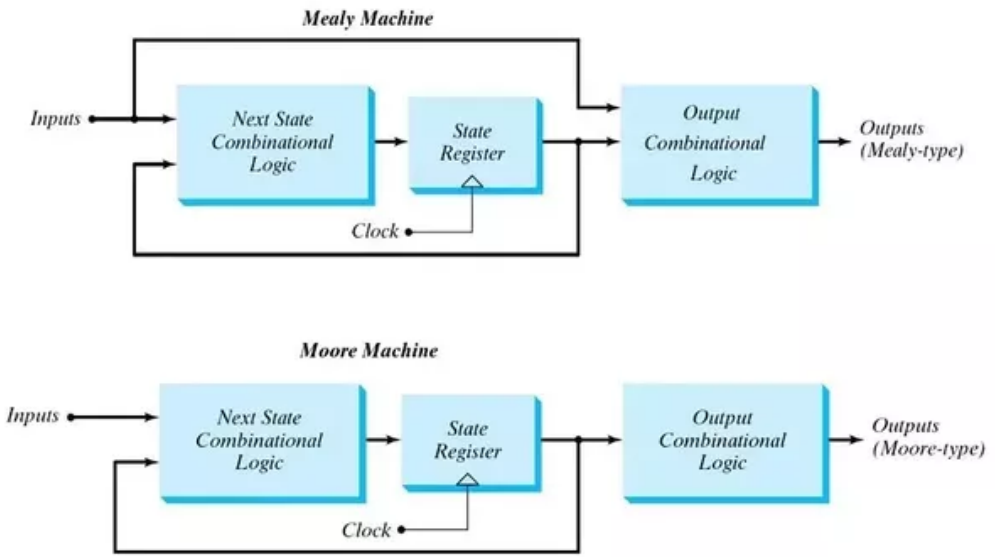
```
assert condition
[report character_string]
[severity severity_level];
```

- Ako se ***report*** segment izostavi, ispis će biti: ***Assertion violation***.
- ***severity*** predstavlja nivo značaja upozorenja i može biti: *note*, *warning*, *error* i *failure*. Ako se izostavi smatra se tipom *error*.

```
-- ispis ako su oba signal '1'
assert not (R = '1' and S = '1')
  report "Both signals R and S have value '1'"
  severity error;
-- forsiranje ispisa
assert (FALSE) report "Start simulation";
```

FSM

- ***FSM*** *Final State Machine* (Mašina sa konačnim broja stanja) - predstavlja matematičku apstrakciju koja se koristi za rješavanja velikog broja problema a jedan od njih je dizajn automata, komunikacijskih protokola, algoritama parsiranja, i dr.
- Generalno, postoje dva tipa FSM-ova:
  - ***Moore FSM*** - izlaz je funkcija trenutnih stanja sistema,
  - ***Melay FSM*** - izlaz je funkcija ulaza i trenutnih stanja sistema.

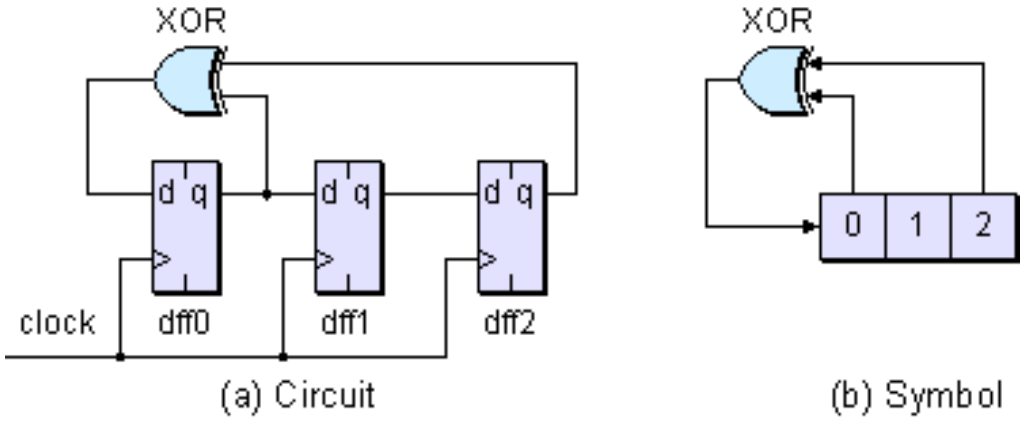


Generatori slučajnih vrijednosti - osnove

- Generiranje slučajnih vrijednosti se izvodi sa:
  - ***Softverski*** - na bazi softverski algoritama nad binarnim podacima:
    - ***LFSR*** - *Linear Feedback Shift Register*,
    - ***MSM*** - *Middle Square Method* John von Neumann,
    - ***LCG*** - *Linear Congruential Generator*,
    - ***Mersenne Twister 1997*** - period $2^{19937} - 1$,
    - ***Well 2006*** - *Equidistributed Long-period Linear* - izveden na bazi Mersenne Twister algoritma.
  - ***Hardverski*** - na bazi fizičkih procesa: termički šum, fotoelektrični efekat, kvantni procesi, itd.
- Značajni za ***simulaciju, validaciju i verifikaciju*** modela fizičkih sistema.
- Testiranje robusnosti sistema na vanjske smetnje različitog karaktera $\rightarrow$ uobličavanje slučajnih vrijednosti tako da podliježu ciljanoj funkciji raspodjele.
- Omogućuju modelovanje širokog spektra "nepredvidivih" okolnosti (šuma) unutar i izvan sistema.

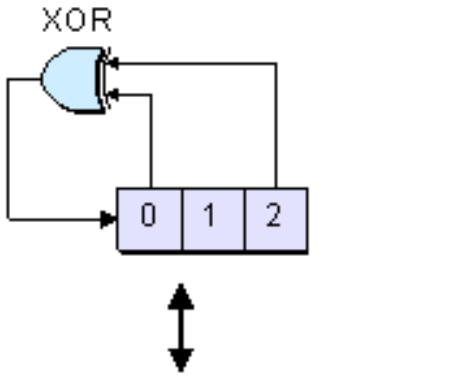
Generatori slučajnih vrijednosti - primjer I

- Posmatrajmo primjer LFSR-a od 3 bita na slici ispod koji je prikazan u izvornoj formi i kao simbol.

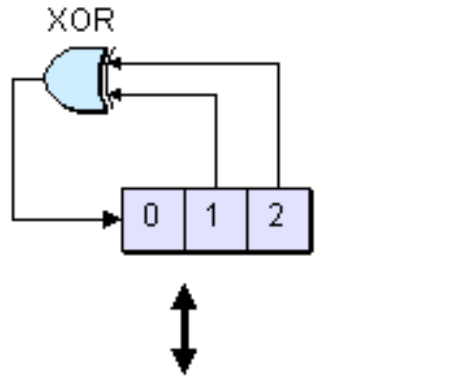


- Trenutna vrijednost registra LFSR-a se nakon svakog takt impulsa pomjera u desno ja jedno mjesto a nova vrijednost se upisuje na bit 2.
- Redoslijed stanja registra LFSR-a zavisi od početne vrijednosti koja se naziva ***seed***.
- Period ponavljanja dato LFSR-a je 7. Međutim, iako je broj bita registra LFSR-a tri, to ne znači da je period limitiran na osam već isti može bit mnogo veći.

Generatori slučajnih vrijednosti - primjer I



Clock	q0	q1	q2	
--	0	0	1	← Initial value
1	1	0	0	
2	1	1	0	
3	1	1	1	
4	0	1	1	
5	1	0	1	
6	0	1	0	
7	0	0	1	
8	1	0	0	
9	1	1	0	
:	:	:	:	



Clock	q0	q1	q2	
--	0	0	1	← Initial value
1	1	0	0	
2	0	1	0	
3	1	0	1	
4	1	1	0	
5	1	1	1	
6	0	1	1	
7	0	0	1	
8	1	0	0	
9	0	1	0	
:	:	:	:	

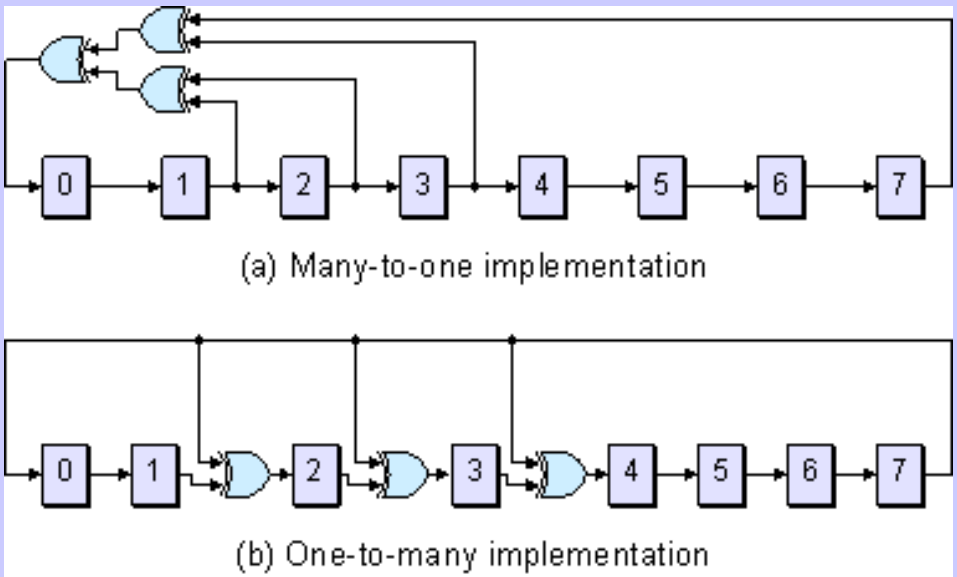
Generatori slučajnih vrijednosti - primjer I

PRIMJER 7

Implementirati VHDL model LFSR-a koji je prikazan na prethodnom slajdu i prikazati sve vrijednosti LFSR-a.

PRIMJER 8

Implementirati VHDL model LFSR-a koji je prikazan na sljedećoj slici. generirati 32 vrijednosti LFSR-a i odrediti period. Početne vrijednosti su 0x00.



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.