Radionica digitalnog dizajna I
Dan I

dr. sc. Jurica Kundrata

Centar za svemirsku i inovativnu tehnologiju

6. - 7. svibnja 2023.

Raspored

6. svibnja 2023., subota	
9.00 - 12.00	Uvod u Verilog jezik, testne postave i simulaciju digitalnih sklopova
12.00 - 12.45	Pauza za ručak
12.45 - 15.00	Dizajn kombinacijskih i sekvencijalnih logičkih sklopova
7. svibnja 2023., nedjelja	
9.00 -12.00	Radni tijek s FPGA platformom
12.00 - 12.45	Pauza za ručak
12.45 - 15.00	Implementacija projekta na FPGA razvojnoj ploči

Dan I

Sadržaj

- Osnovni koncepti
- Prvi primjer
- Testni postav i simulacija
- Zadaci
- Kombinacijski sklopovi
- Sekvencijalni sklopovi

Pregled

- Verilog HDL (Hardware Description Language) je programski jezik visoke razine koji se prvenstveno koristi za dizajn i provjeru digitalnih sklopova i sustava.

- Uveden je 1980-ih i od tada je postao industrijski standard za alate za automatizaciju elektroničkog dizajna (EDA).

- Verilog dizajnerima hardvera omogućuje da opišu strukturu i ponašanje digitalnih sustava na sažet i ljudima čitljiv način.

Moduli

- Verilog dizajni organizirani su u module, koji su osnovni gradivni blokovi jezika.

- Modul predstavlja digitalnu komponentu ili skupinu međusobno povezanih komponenti i može se instancirati unutar drugih modula za stvaranje složenih hijerarhijskih dizajna.

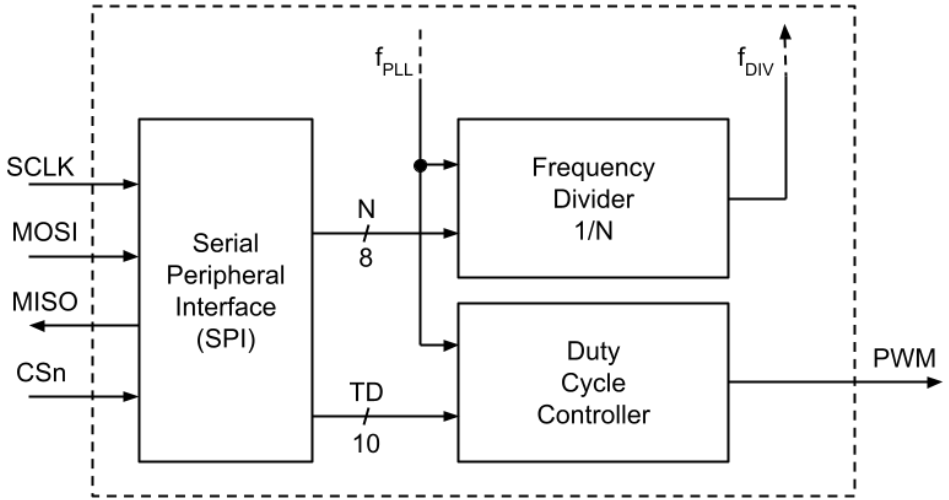
Vrste podataka

- Verilog podržava različite vrste podataka za predstavljanje signala, varijabli i konstanti u dizajnu. Najčešći tipovi podataka su:
  - **`wire`** - Predstavlja fizičku žicu ili vezu između komponenti.
  - **`reg`** - Predstavlja element za pohranu ili varijablu koja može sadržavati vrijednost.
  - **`integer`**, **`real`**, **`time`** - tipovi numeričkih podataka za predstavljanje brojeva i vremenskih vrijednosti.
  - Nizovi i memorije - strukture podataka za pohranu višestrukih vrijednosti.

Strukturni opis

- Verilog omogućuje dizajneru da opiše strukturu digitalnog sustava određivanjem komponenti i njihovih međusobnih veza.

- Ova metoda opisa je prikladna za dizajne na razini vrata ili razini prijenosa registra (RTL).



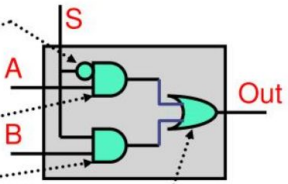
Opis ponašanja

- Verilog također podržava opis ponašanja, koji se fokusira na funkcionalnost i rad digitalnog sustava, a ne na njegovu strukturu.

- Ova se metoda koristi za opisivanje ponašanja komponenti na visokoj razini apstrakcije, često koristeći konstrukcije kao što su **always** blokovi, **if-else** naredbe i petlje.

Structural

```
module mux2to1(
  input S, A, B,
  output Out );
  wire S_, AnS_, BnS;
  not (S_, S);
  and (AnS_, A, S_);
  and (BnS, B, S);
  or (Out, AnS_, BnS);
endmodule
```



Behavioral

```
module mux2to1(
  input S, A, B,
  output Out );
  assign Out = (~S & A) | (S & B);
endmodule
```

Better:

```
assign Out = S? B:A;
```

Proceduralni blokovi

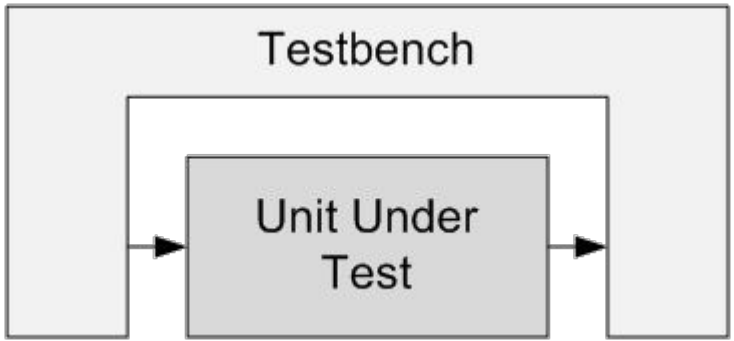
- Verilog sadrži konstrukcije za opisivanje ponašanja digitalnih sustava pomoću konstrukcija proceduralnog programiranja kao što su **always** blokovi, **initial** blokovi i zadaci.

- Ovi konstrukti omogućuju dizajnerima da opišu kako bi se sustav trebao ponašati u različitim uvjetima ili događajima.

Testni postav

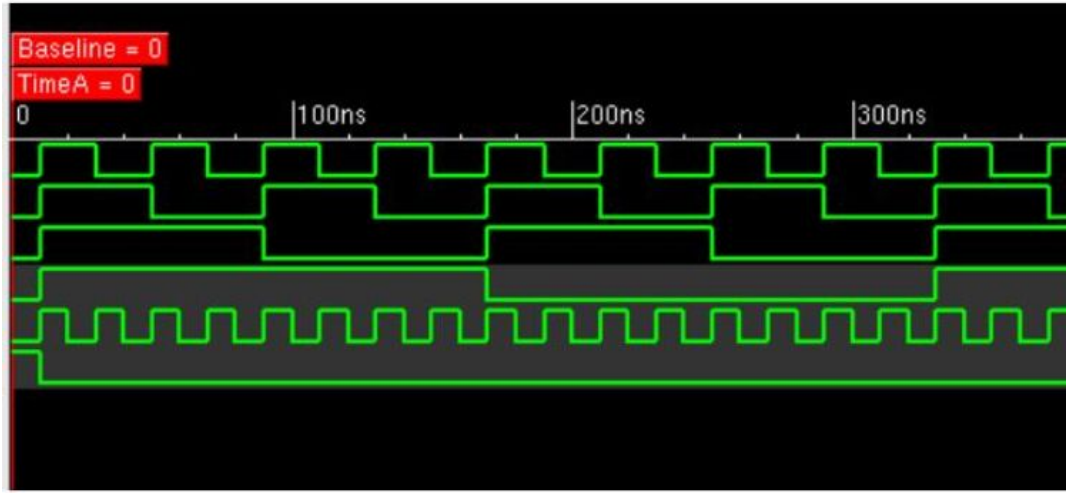
- U Verilogu, testni postav (*testbench*) je zasebni modul koji se koristi za simulaciju i provjeru funkcionalnosti dizajna.

- testni postav daju pobudu testiranom dizajnu (*Unit Under Test*, UUT) i prate njegove reakcije kako bi se osigurao ispravan rad.



Simulacija

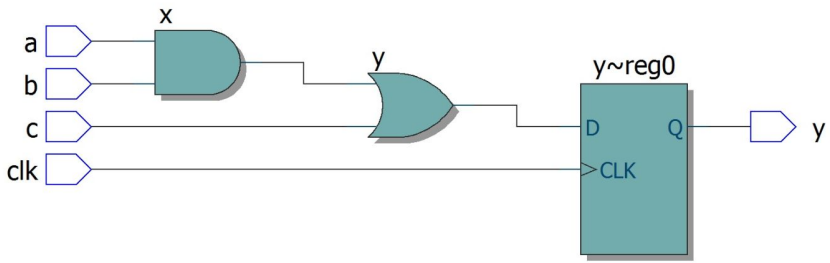
- Verilog dizajni obično se provjeravaju kroz simulaciju, koja dizajnerima omogućuje promatranje ponašanja sustava u različitim uvjetima i provjeru njegove funkcionalnosti prije nego što pređu na faze sinteze i implementacije.



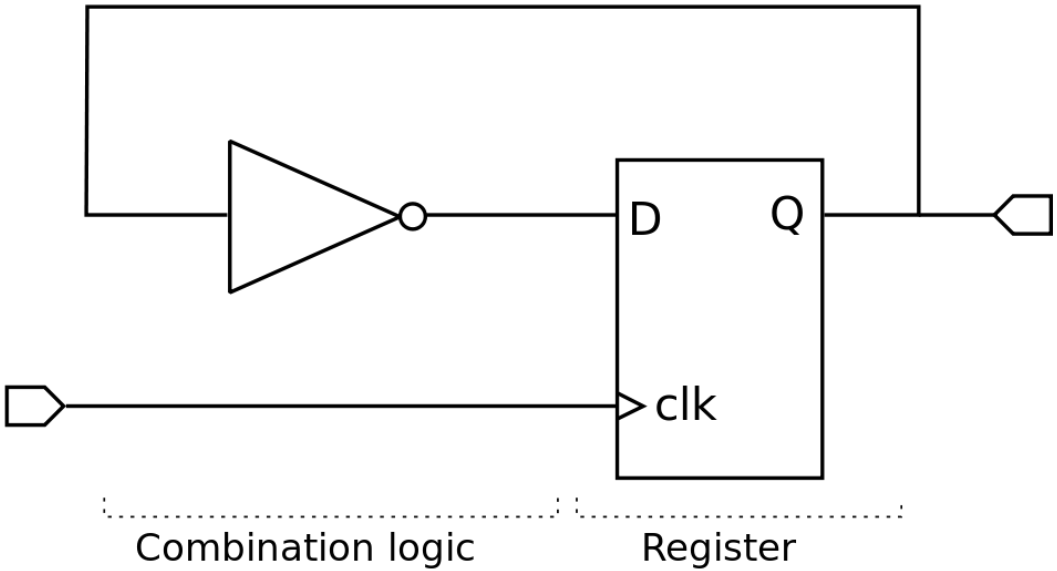
Sinteza

- Nakon provjere dizajna korištenjem simulacije, on se može sintetizirati u spojnu listu (*netlist*) na razini vrata, što je prikaz dizajna korištenjem logičkih vrata i flip-flopova.

- Tase spojna lista zatim može koristiti za daljnju obradu, kao što je analiza potrošnje i vremena, prije implementacije u FPGA ili ASIC.



Register Transfer Level (RTL) dizajn




```
module rtl_example (clk, Q);

    input   wire   clk;
    output  reg    Q;

  wire D;

  // combinatorial
  assign D = ~Q;

  // sequential
  always @ (posedge clk)
    Q <= D;

endmodule
```

Pregled Verilog jezika

Komentari, blokovi i konstante

Comments

```
// One-liner
/* Multiple
lines */
```

Blocks

```
begin
 // block content
end
```

Numeric Constants

```
// The 8-bit decimal number 106:
8'b_0110_1010 // Binary
8'o_152 // Octal
8'd_106 // Decimal
8'h_6A // Hexadecimal
"j" // ASCII

78'bZ // 78-bit high-impedance
```

Mrežne točke, varijable, parametri, moduli…

Nets and Variables

```
// Assign wires outside always blocks
wire [3:0] w;
// Assign regs inside always blocks
reg  [1:7] r;
reg  [7:0] mem [31:0];
integer j; // Compile-time var.
```

Parameters

```
parameter N = 8;
localparam State = 2'd3;
```

Module

```
module MyModule
  #(parameter N = 8) // Optional
parameter
   (input Reset, Clk,
    output [N-1:0] Output);
  // Module implementation
endmodule
```

Operatori

Operators

```
// Select
A[N] A[N:M]
// Reduction AND, NAND, OR, NOR,
XOR, XNOR
&A ~&A |A ~|A ^A ~^A
// Compliment LOGICAL, BITWISE
!A ~A
// Unary
+A -A
// Concatenate
{A, ..., B}
// Replicate
{N{A}}
```

```
// Arithmetic
A*B A/B A%B
A+B A-B
// Shift
A<<B A>>B
// Relational
A>B A<B A>=B A<=B
A==B A!=B
// Bit-wise AND, XOR, XNOR, OR
A&B
A^B A~^B
A|B
// Logical AND, OR
A&&B
A||B
// Conditional IF THEN ELSE
A ? B : C
```

Instanciranje modula i dodjeljivanje vrijednosti

Module Instantiation

```
// Override default parameter:
// setting N = 13
MyModule #(13) MyModule1
          (Reset, Clk, Result);
```

Assignments

```
assign Output = A * B;
assign {C, D} = {D[5:2], C[1:9], E};
// blocking assignment in always block
a = b; // use in combinational logic
// nonblocking assignment in always block
a <= b; // use in sequential logic
```

Naredba case, sinkroni blok i petlja

Case

```
always @(*) begin
  case(Mux)
    2'd0: A = 8'd9;
    2'd1,
    2'd3: A = 8'd103;
    2'd2: A = 8'd2;
    default:;
  endcase
end
```

Synchronous

```
always @(posedge Clk) begin
  if(Reset) B <= 0;
  else      B <= B + 1'b1;
end
```

Loop

```
always @(*) begin
  Count = 0;
  for(j = 0; j < 8; j = j+1)
    Count = Count + Input[j];
end
```

Osnovni primjer

Primjer modula - Brojilo

```
1  module counter (
2    input clk,
3    input reset,
4    output reg [3:0] count
5  );
6
7    // Always block triggered on positive edge of the clock
8    always @(posedge clk) begin
9      if (reset) begin
10       count <= 4'b0000; // Reset count to 0 if reset is high
11     end
12     else begin
13       count <= count + 4'b0001; // Increment count by 1 on every clock cyc
14     end
15   end
16
17 endmodule
```

```
1 module counter (
```

- Ovaj redak deklarira novi modul pod nazivom "counter".

- Moduli su osnovni građevni blokovi u Verilogu, koji predstavljaju digitalne komponente ili grupe međusobno povezanih komponenti.

```
2   input clk,
3   input reset,
```

- Ovi redovi deklariraju ulazne portove za modul.

- U ovom primjeru, ulazi su signal takta (clk) i signal resetiranja (reset).

```
4    output reg [3:0] count
```

- Ovaj redak deklarira izlazni port pod nazivom "count" s 4-bitnim reg tipom podataka.

- Vrsta podataka "reg" koristi se za predstavljanje elementa za pohranu ili varijable koja može sadržavati vrijednost.

```
8    always @(posedge clk) begin
```

- always blok je proceduralna konstrukcija koja se koristi za opisivanje ponašanja modula.

- U ovom slučaju, aktivira se na pozitivnom rubu taktnog signala (posedge clk).

```
9       if (reset) begin
10        count <= 4'b0000;
11      end
```

- Izjava if-else provjerava je li signal resetiranja visok.

- Ako je tako, vraća broj na 0.

- Blokovi koda se nalaze između begin-end ključnih riječi.

```
12      else begin
13        count <= count + 4'b0001;
14      end
```

- Izjava if-else provjerava je li signal resetiranja visok.

- Ako nije tako, ovaj redak povećava brojač za 1.

```
17 endmodule
```

- Ova linija označava kraj definicije modula.

Primjer testnog postava - Brojilo (I)

```
1  `timescale 1ns/1ps
2
3  module counter_tb;
4
5    // Declare signals
6    reg clk;
7    reg reset;
8    wire [3:0] count;
9
10   // Instantiate the counter module
11   counter my_counter (
12     .clk(clk),
13     .reset(reset),
14     .count(count)
15   );
16
17   // Clock generation
18   always begin
19     #5 clk = ~clk;
20   end
```

Primjer testnog postava - Brojilo (II)

```
22    // Stimulus generation and checking
23    initial begin
24      // Initialize signals
25      clk = 0;
26      reset = 0;
27
28      // Apply reset
29      #10 reset = 1;
30      #10 reset = 0;
31
32      // Monitor count and VCD
33      $monitor("At time %t, count = %b", $time, count);
34      $dumpfile("dump.vcd"); $dumpvars;
35
36      // Run simulation for a specified time
37      #100 $finish;
38    end
39
40  endmodule
```

```
1 `timescale 1ns/1ps
```

- Ova direktiva postavlja vremensku rezoluciju i preciznost za simulaciju.

- U ovom slučaju, rezolucija je 1 nanosekunda, a preciznost je 1 pikosekunda.

```
3 module counter_tb;
```

- Ovaj redak deklarira novi testni modul pod nazivom "counter_tb".

```
5   // Declare signals
6   reg clk;
7   reg reset;
8   wire [3:0] count;
```

- Redovi 6 i 7 deklariraju signale takta i resetiranja kao reg tipove podataka, koji mogu sadržavati vrijednosti tijekom simulacije.

- Redak 8 deklarira 4-bitnu žicu za izlaz brojača iz modula brojača.

```
10  // Instantiate the counter module
11  counter my_counter (
12    .clk(clk),
13    .reset(reset),
14    .count(count)
15  );
```

- Redak 11 instancira modul "counter" s imenom instance "my_counter".

- Linije 12-14 povezuju signale testnog postava s odgovarajućim priključcima modula "counter".

```
17  // Clock generation
18  always begin
19    #5 clk = ~clk;
20  end
```

- Ovaj blok generira periodički signal sata s periodom od 10 vremenskih jedinica (5 vremenskih jedinica visoko, 5 vremenskih jedinica nisko).

```
22  // Stimulus generation and checking
23  initial begin
24    // Initialize signals
25    clk = 0;
26    reset = 0;
27
28    // Apply reset
29    #10 reset = 1;
30    #10 reset = 0;
```

- Ovaj blok primjenjuje pobudu na dizajn koji se testira (UUT).

- Inicijalizira signale takta i resetiranja, primjenjuje impuls resetiranja, a zatim nadzire izlaz brojača.

```
32      // Monitor count
33      $monitor("At time %t, count = %b", $time, count);
34      $dumpfile("dump.vcd"); $dumpvars;
```

- $monitor se koristi za praćenje i ispis vrijednosti signala tijekom simulacije
- $dumpfile i $dumpvars rade zajedno za pohranu promjena vrijednosti signala u izlaznu datoteku za analizu nakon simulacije i otklanjanje pogrešaka.
- Ovi su zadaci korisni za razumijevanje ponašanja vašeg dizajna tijekom simulacije i identificiranje problema koji se mogu pojaviti.

```
35
36      // Run simulation for a specified time
37      #100 $finish;
```

- Ovaj zadatak sustava koristi se za završetak simulacije nakon određenog vremena.

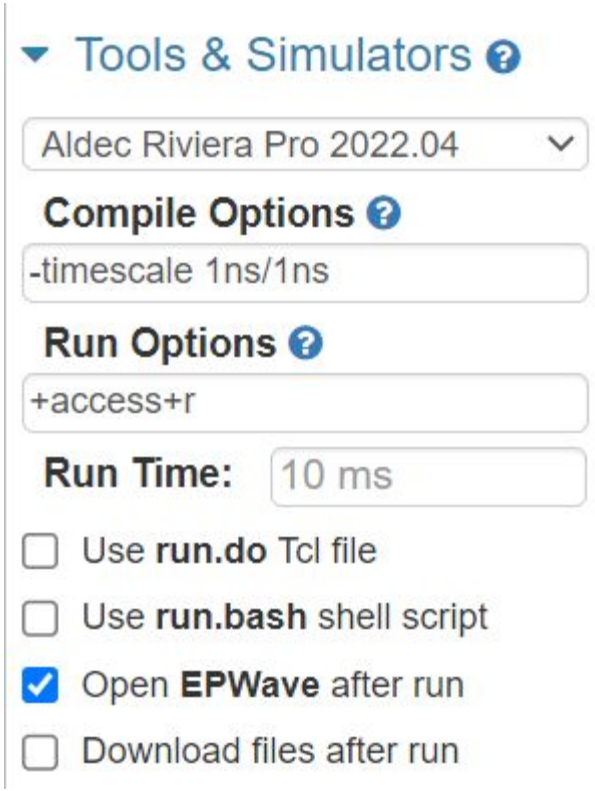
EDA playground (I)

- Online simulator za HDL jezike
- Obavezna izrada računa
- Okviri za unos modula i ispitnog postava (testbench)
- Podržava niz simulatora i HDL knjižnica

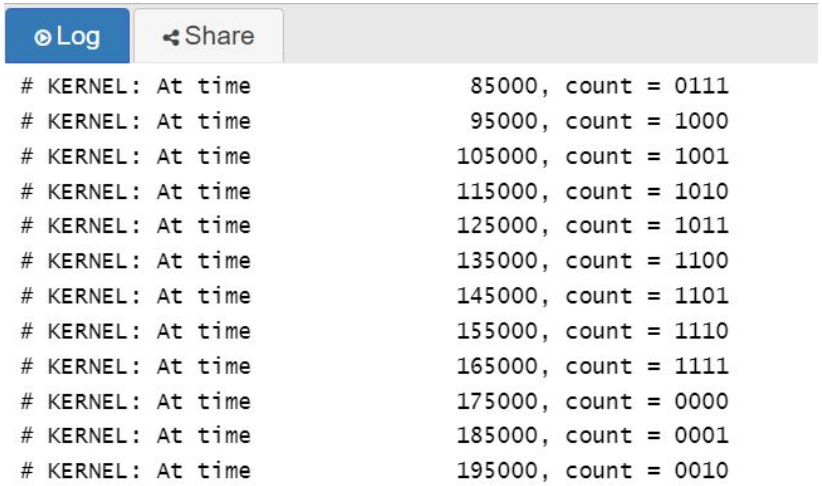
EDA playground (II)

- Postavke simulatora
  - Aldec Riviera Pro 2022.04

- Odabrati Open EPWave after run

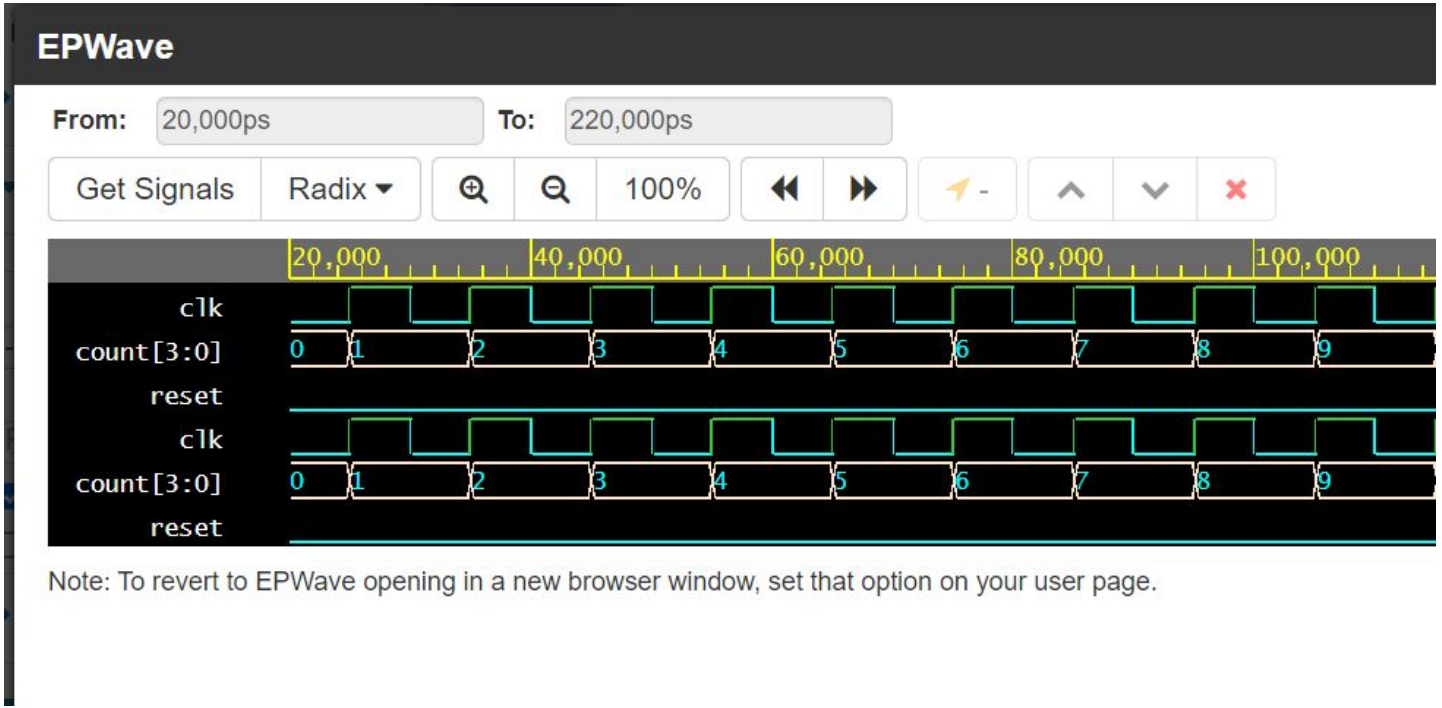


EDA playground (II)



- Okvir Log za prikaz rezultata simulacije

Simulacija brojila



Zadaci (I)

1. Promijeniti naziv modula iz **counter** u **brojilo**.

2. Promijeniti naziv ulaznog priključka **reset** u **reset_n**.

3. Promijeniti aktivnu razinu priključka **reset_n** iz visoke u nisku.

Zadaci (II)

4. Promijeniti širinu brojila s **4 bita** na **8 bitova**.

5. Produljiti vrijeme simulacije sa **~100** na **~1000** vremenskih jedinica.

6. Prilagoditi ispis simulacije iz **At time 105000, count = 1001**
   u **Vrijeme: 105000, broj = 1001**

7. Promijeniti period signala takta iz **10** u **8** vremenskih jedinica.

Zadaci (III)

8. Dodati ulazni priključak **zadano** širine 8 bitova.

9. Implementirati postavljanje registra **count** na vrijednost **zadano** tokom resetiranja.

10. Dodati ulazni priključak **gore_dolje**.

11. Implementirati upravljanje brojanjem prema gore i dolje u ovisnosti o priključku **gore_dolje.**

Zadaci (IV)

12. Dodati izlazni priključak **gotov**.

13. Implementirati postavljanje izlaznog priključka **gotov** kada registar **count** je jednak nuli.

Kombinacijski sklopovi

- Kombinacijski moduli su digitalni sklopovi koji nemaju memorijske elemente niti povratne sprege.

- Njihovi izlazi ovise isključivo o trenutnim vrijednostima njihovih ulaza.

Primjer kombinacijskog sklopa - Multiplekser

- Multiplekser je uređaj koji odabire jedan od nekoliko ulaznih signala na temelju signala odabira i usmjerava odabrani ulaz na izlaz.

- Zadatak: Zamijeniti izbor izlaza.

```
module mux_2to1 (
  input a,
  input b,
  input sel,
  output y
);
  assign y = sel ? b : a;
endmodule
```

Primjer kombinacijskog sklopa - Dekoder

- Dekoder je sklop koji uzima N-bitni binarni ulaz i aktivira jednu od 2^N izlaznih linija ovisno o ulaznoj vrijednosti.

```
module decoder_2to4 (
  input [1:0] in,
  output [3:0] out
);
  assign out = 4'b0001 << in;
endmodule
```

- Zadatak: Proširiti dekoder na **3to8**.

Primjer kombinacijskog sklopa - Enkoder

- Enkoder je sklop koji pretvara 2^N-u-1 ulaz u N-bitni binarni kod koji predstavlja aktivnu ulaznu liniju.

- Zadatak: Proširiti enkoder na **8to3**.

```
module encoder_4to2 (
  input [3:0] in,
  output reg [1:0] out
);
  always @* begin
    case (in)
      4'b0001: out = 2'b00;
      4'b0010: out = 2'b01;
      4'b0100: out = 2'b10;
      4'b1000: out = 2'b11;
      default:  out = 2'b00;
    endcase
  end
endmodule
```

Primjer kombinacijskog sklopa - Komparator

- Komparator je sklop koji uspoređuje dva binarna broja i utvrđuje jesu li jednaki ili je jedan veći ili manji od drugog.

- Zadatak: Proširiti komparator na **8 bita**.

```
module comparator_4bit (
  input [3:0] a,
  input [3:0] b,
  output eq,
  output a_gt_b,
  output a_lt_b
);
  assign eq = (a == b);
  assign a_gt_b = (a > b);
  assign a_lt_b = (a < b);
endmodule
```

Sekvencijalni sklopovi

- Sekvencijalni moduli su digitalni sklopovi koji sadrže memorijske elemente, kao što su bistabili.

- Izlaz sekvencijalnog modula ne ovisi samo o njegovim trenutnim ulazima, već i o njegovom prethodnom stanju.

Primjer sekvencijalnog sklopa - D bistabil

- D bistabil je memorijski element koji hvata i pohranjuje vrijednost D ulaza na uzlaznom rubu takta.

- Zadatak: Proširiti u registar širine **8**.

```
module d_flip_flop (
  input clk,
  input d,
  output reg q
);
  always @(posedge clk) begin
    q <= d;
  end
endmodule
```

Primjer sekvencijalnog sklopa - Posmačni registar

- Posmačni registar je sekvencijalni sklop koji pomiče svoje pohranjene podatke za jednu poziciju u svakom taktu.

- Zadatak: Promijeniti ubacivanje nove vrijednosti s **LSB** na **MSB.**

```
module shift_register_4bit (
  input clk,
  input d_in,
  output [3:0] d_out
);
  reg [3:0] data;

  always @(posedge clk) begin
    data <= {data[2:0], d_in};
  end

  assign d_out = data;
endmodule
```

Primjer sekvencijalnog sklopa - Datoteka registra (I)

```
module register_file_4x4 (
  input clk,
  input [1:0] read_addr,
  input [1:0] write_addr,
  input [3:0] data_in,
  input write_enable,
  output [3:0] data_out
);
  reg [3:0] registers[3:0];
  always @(posedge clk) begin
    if (write_enable) begin
      registers[write_addr] <= data_in;
    end
  end

  assign data_out =
registers[read_addr];
endmodule
```

Primjer sekvencijalnog sklopa - Datoteka registra (II)

- Datoteka registra je skup registara koji se mogu čitati i pisati pomoću kontrolnih signala.

- Zadatak: Proširiti na **128x8.**

Primjer sekvencijalnog sklopa - Automat

- Automat je sekvencijalni sklop koji prelazi između unaprijed definiranog skupa stanja na temelju svojih ulaza i trenutnog stanja.

- Zadatak: Proširiti na **4 stanja** s prijelazima:
  - 00 → 11
  - 01 → 10
  - 10 → 00
  - 11 → 01

```
module simple_fsm (
  input clk,
  input reset,
  output reg state
);
  always @(posedge clk) begin
    if (reset) begin
      state <= 1'b0;
    end else begin
      state <= ~state;
    end
  end
endmodule
```