LPRS2
Osnove VHDL-a

Miloš Subotić

24. februar 2016

1 Signali

VHDL jezikom se opisuje hardver. Hardver se sastoji od komponenti koje su međusobno povezane signalima tj. žicama. Signali imaju vrednost, **0** i **1**, koji u realnom hardveru prestavljaju 2 naponska nivoa. Jedan signal može da koristi više komponenti (engl. **sink**), ali mora obavezno da ima samo jedan izvor (engl. **source**) tj. samo jedna komponenta sme da definiše vrednost na signalu (engl. driving). Tip signala u VHDL-u je `std_logic` koji može imati pored `'0'` i `'1'` još neke vrednosti. Te dodatne vrednosti olakšavaju simulaciju. Kasnije ćemo se vratiti na njih.

Osim tipa `std_logic` koji je jednobitan postoji tip višebitni tip `std_logic_vector`. Kao što se da videti na Listingu 1 u okviru zagrada iza `std_logic_vector` daje se opseg od MSB to LSB bita. U ovom konkretnom slučaju opseg je od 7 do 0, uključujući iste, tako da je signal 8 bita širok.

```
1 ...
2
3     signal x    : std_logic;
4     signal data : std_logic_vector(7 downto 0);
5
6 ...
```

Listing 1: Signali

2 Moduli

Na listingu 2 je prikazan osnovni primer modula u VHDL-u. Na početku fajla se definišu biblioteke koje se koriste. `entity` blok definiše spregu modula prema spoljašnjem svetu. U `port` bloku se definišu ulazni i izlazni signali,

označeni sa `in` i `out`. U `architeture` bloku se definišu logika modula. Žice kojima se povezuju komponente označavaju se kao `signal`.

U okviru tela arhitekture stavljaju instanciraju se drugi moduli i pišu se konkurentni iskazi. Na listingu 2 se može videti par konkurentnih iskazi. Svaki od njih se karakteriše sa `<=` čime se povezuju komponente i signali. Konkurentni izraz se završava sa `;`. Kao što samo ime kaže svi iskazi se izvršavaju konkurentno, što modelira stvarnu situaciju u hardveru. Operator `&` služi za spajanje više signala u jedan širi (engl. concatenation). Neki od osnovnih logičkih operatora su `not`, `and`, `or`, `xor`.

```
1  -- Comments starts with "--".
2
3  -- Some libraries.
4  library ieee;
5  use ieee.std_logic_1164.all;
6
7  -- Entity is how module look like to rest of the world.
8  entity small_system is
9      -- Port are inputs and outputs to module.
10     port(
11         -- One bit. "i_" is convention for input.
12         i_a : in  std_logic;
13         i_b : in  std_logic;
14          -- Multiple bits. "o_" is convention for output.
15         o_ab  : out std_logic_vector(1 downto 0);
16         -- n in "on_" is convention that signal active on 0.
17         on_ab : out std_logic_vector(1 downto 0);
18         o_and : out std_logic -- There is no ";".
19     ); -- Watch for ;
20 end entity small_system;
21
22 -- Architecture is real implementation of module,
23 -- hidden to rest of the world.
24 architecture arch_small_system_v1 of small_system is
25     -- Signals are internal wires which connect components.
26     signal ab   : std_logic_vector(1 downto 0);
27     signal n_ab : std_logic_vector(1 downto 0);
28     -- Signals are before begin of architecture.
29
30 begin
31     -- Logic comes here, after begin.
32
33     ab <= i_a & i_b; -- Concatenation
34     n_ab <= not ab; -- 2-bit inverter.
35
36     o_ab <= ab;
37     on_ab <= n_ab;
38
```

```
39      o_and <= i_a and i_b; -- And gate.
40
41  end architecture arch_small_system_v1;
```

Listing 2: Primer VHDL modula

3 Literali

Signalima je moguće dodeliti konstante vrednosti kao što je prikazano na listingu 3 Jednobitni (**std_logic**) signali su pod jednostrukim navodnicima. Višebitni (**std_logic_vector**) signali su pod dvostrukim navodnicima. Ako je **x** ispred prvog navodnika, onda je u pitanju heksadecimalna prestava.

```
1
2   ...
3
4           o_en   : out std_logic;
5           o_data : out std_logic_vector(7 downto 0);
6           o_addr : out std_logic_vector(7 downto 0);
7
8   ...
9
10
11      o_en <= '0';
12      o_data <= "00000001";
13      o_addr <= x"1a";
14  ...
```

Listing 3: Primer VHDL modula

Redovnim brojevima su prestavljene celobrojne vrednosti. One se ne mogu direktno dodeliti signalima tima **std_logic** i **std_logic_vector**, ali se mogu vršiti operacije između njih.

4 Indeksiranje

Može se vršiti indeksiranje jednobitnog **std_logic** iz višebitnog **std_logic_vector** . To je kao kada se iz jednog višežilnog kabla izvlači jedna ili više žica za druge potrebe. Indeksiranje se vrži kao na listingu 4, gde se pozicijom selektuje potreban bit. Jedan ili više bita se uzima **slice** operatorom iz **std_logic_vector**.

```
1
2   ...
3
4       signal dbca : std_logic_vector(3 downto 0);
```

```
5
6       signal c_one_bit : std_logic_vector(0 downto 0);
7
8   begin
9
10      a <= dbca(0); -- Indexing.
11      d <= dbca(3); -- Indexing.
12
13      ab <= dbca(1 downto 0); -- Slicing.
14      dbc <= dbca(3 downto 1); -- Slicing.
15
16      c_one_bit <= dbca(2 downto 2); -- Slicing.
17
18  ...
```

Listing 4: Indeksiranje i Slicing

5 Osnovne kombinacione mreže

Listing 5 je prikazan komparator. Ovakav zapis je veoma blizak kao bi ljudi stvarno opisali komparator: `o_zero` je `'1'` kad je `i_x` jednako 0, u protivnom je `'0'`. Operator `<=` ne treba čitati `i_x(0)` se dodeljuje `o_y`.

```
1
2   ...
3       i_x    : in  std_logic_vector(7 downto 0);
4       o_zero : out std_logic;
5   ...
6
7       o_zero <= '0' when i_x = 0 else '1';
8
9   ...
```

Listing 5: Komparator

Listing 6 prikazuje 4 u 1 1-bitni multiplekser sa `with select` konstrukcijom.

```
1
2   ...
3       i_sel : in  std_logic_vector(1 downto 0);
4       i_x   : in  std_logic_vector(3 downto 0);
5       o_y   : out std_logic;
6   ...
7
8       with i_sel select o_y <=
9           i_x(0) when "00",
10          i_x(1) when "01",
```

```
        i_x(2) when "10",
        i_x(3) when others;

...
```

Listing 6: 4-to-1 1-bit mux

Listing 7 je 8-bitna verzija prethodnog multipleksera.

```

...
    i_sel : in  std_logic_vector(1 downto 0);
    i_a   : in  std_logic_vector(7 downto 0);
    i_b   : in  std_logic_vector(7 downto 0);
    i_c   : in  std_logic_vector(7 downto 0);
    i_d   : in  std_logic_vector(7 downto 0);
    o_y   : out std_logic_vector(7 downto 0);
...

    with i_sel select o_y <=
        i_a when "00",
        i_b when "01",
        i_c when "10",
        i_d when others;

...
```

Listing 7: 4-to-1 8-bit mux

Na sličan način se može izraditi i dekoder, kao što je 2 u 4 dekoder dat u listingu 8. Dekoder postavlja bit na 1 čiji je indeks na ulazu.

```

...
    i_x : in  std_logic_vector(1 downto 0);
    o_y : out std_logic_vector(3 downto 0);
...

    with i_x select o_y <=
        "0001" when "00",
        "0010" when "01",
        "0100" when "10",
        "1000" when others;

...
```

Listing 8: 2-to-4 line decoder

Enkoder radi suprotnu stvar od dekodera. Enkoder daje indeks bita koji je na 1. Primer enkodera je dat na listingu 9. Nedostatak ovakvog enkodera je problem u detekciji kada su dva bita postavljena na 1. Rešenje tog problema je u korišćenju prioritetnog enkodera datog na listingu 10.

```
...
    i_x : in  std_logic_vector(3 downto 0);
    o_y : out std_logic_vector(1 downto 0);
...

    o_y <=
        "11" when i_x = "1000" else
        "10" when i_x = "0100" else
        "01" when i_x = "0010" else
        "00";

...
```

Listing 9: 4-to-2 encoder

```
...
    i_x : in  std_logic_vector(3 downto 0);
    o_y : out std_logic_vector(1 downto 0);
...

    o_y <=
        "11" when i_x(3) = '1' else
        "10" when i_x(2) = '1' else
        "01" when i_x(1) = '1' else
        "00";

...
```

Listing 10: 4-to-2 priority encoder

6 Instanciranje modula

Prvo je potrebno uraditi deklaraciju komponente. Komponenta izgleda indentično kao `entity` blok modula koji hoćete da uključite, samo je ključna reč `entity` zamenjena sa `component`. Komponenta se deklariše u sekciji pre `begin` arhitekture, što se da videti na listingu 11.

```
...

architecture arch_comb_logic_system of comb_logic_system is

    component bcd_to_7_segm_disp_decoder is
        port(
            i_bcd0       : in  std_logic;
            i_bcd1       : in  std_logic;
```

```
            i_bcd2       : in  std_logic;
            i_bcd3       : in  std_logic;
            o_7_segm_a   : out std_logic;
            o_7_segm_b   : out std_logic;
            o_7_segm_c   : out std_logic;
            o_7_segm_d   : out std_logic;
            o_7_segm_e   : out std_logic;
            o_7_segm_f   : out std_logic;
            o_7_segm_g   : out std_logic
        );
    end component bcd_to_7_segm_disp_decoder;

    signal bcd  : std_logic_vector(3 downto 0);
    signal segm : std_logic_vector(6 downto 0);

begin

...
```

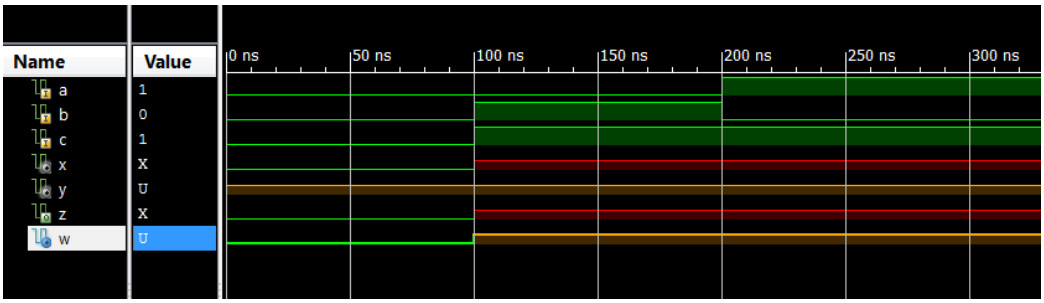
Listing 11: Uključivanje modula

Posle toga se modul instancira unutar arhitekture, posle **begin**, kao na listingu 12. Voditi računa da su portovi modula koji se instancira uvek sa leve strane, bez obzira da li je ulaz ili izlaz, i da je strelica **=>** uvek s leva na desno i ne označava pravac signala.

```

...

begin

    uut: bcd_to_7_segm_disp_decoder
    port map(
        -- Input and outputs of instancing module are
        -- on the left side and => is alway in that direction,
        -- no matter on signal direction.
        -- On right side are signals.
        i_bcd0     => bcd(0),
        i_bcd1     => bcd(1),
        i_bcd2     => bcd(2),
        i_bcd3     => bcd(3),
        o_7_segm_a => segm(0),
        o_7_segm_b => segm(1),
        o_7_segm_c => segm(2),
        o_7_segm_d => segm(3),
        o_7_segm_e => segm(4),
        o_7_segm_f => segm(5),
        o_7_segm_g => segm(6) -- No "," at the end.
    );

```

```
...

    x <= a; -- Multisourcing.
    x <= b;

    -- y is sourceless.

    z <= x and c; -- Used 'X'.
    w <= y and c; -- Used 'U'.

...
```

Listing 13: Multisourcing, sourceless



Slika 1: Multisourcing

```
...
```

Listing 12: Instanciranje i povezivanje modula

Veoma česta greška je da kada se promeni `entity` blok, dodavanjem ili brisanjem ulaza tj. izlaza, ne izvrši se ista promena na komponenti i na instanci komponente.

7 Napredno o signalima

Ako bi bilo 2 izvora vezana na isti signal, jedan ima vrednost **0** a drugi **1**, koja će biti vrednost signala? Ako signal nema komponenti koja postavlja vrednost koja će biti vrednost signala? Prva situacija se zove**multisourcing** and kod druge je problem što je signal **sourceless**. Prilikom simulacije `'U'` označava da signal nema izvor **sourceless**, dok `'X'` označava da je signal **multisourced**. Takođe `'U'` i `'X'` se pojavljuku kao rezultat operacije nad `'U'` ili `'X'`.

Listing 13 prikazuje primer ova dva problema.

```
1
2  ...
3
4  entity mux_4_to_1 is
5      generic(
6          WIDTH : positive := 1 -- Default width.
7      );
8      port(
9          i_sel  : in  std_logic_vector(1 downto 0);
10         i_a    : in  std_logic_vector(WIDTH-1 downto 0);
11         i_b    : in  std_logic_vector(WIDTH-1 downto 0);
12         i_c    : in  std_logic_vector(WIDTH-1 downto 0);
13         i_d    : in  std_logic_vector(WIDTH-1 downto 0);
14         o_mux  : out std_logic_vector(WIDTH-1 downto 0)
15     );
16 end entity mux_4_to_1;
17
18 ...
```

Listing 14: Entity sa generic blokom

Slika 1 prikazuje kako bi izgledali problematični signali na waveform-u simulatora. Žutom bojom su označeni signal sa `'U'`, a sa crvenom signali sa `'X'`. Zeleni signali su u redu.

Kada se takvi signali pojave potrebno je ispratiti na waveform-u i u kodu odakle šta je uzročnik žutih ili crvenih signala, prateći unazad koja komponenta je izvor signala ili da li ga ima.

8 Generički dizajn

Ponekad je korisno neke pojedinosti dizajna parametrizovati. Najčešće je to širina signala. Na listingu 14 je prikazan `entity` sa `generic` blokom. U `generic` bloku postoji parametar `WIDTH` koji određuje širinu ulaznih i izlaznih signala za podatke.

Na listingu 15 je prikazano korišćenje ovakve generičke komponente. Na ovaj način moguće je podešavati komponente pre prevođenja, čime se postiže optimizacija korišćenja resursa i reusability komponenti.

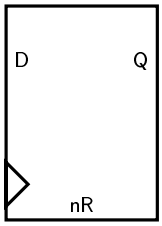
9 Sekvencijalne mreže

Seqvencijalne mreže su logičke mreže sa memorijom.

```
1
2  ...
3
4      data_mux: mux_4_to_1
5      generic map(
6          WIDTH => 8 -- Width.
7      )
8      port map(
9          i_sel  => addr,
10         i_a    => d0,
11         i_b    => d1,
12         i_c    => d2,
13         i_d    => d3,
14         o_mux  => data,
15     );
16
17 ...
```

Listing 15: Entity sa generic blokom

Flip-flop je osnovna memorijska jedinica koja se koristi za izgradnju sinhronih sekvencijalnih mreža. Jedan flip-flop može da bude u dva moguća stanja, 0 i 1. Postoje razne vrste flip-flopova i ovde će se koristiti jedna varijanta D flip-flop. Šematski simbol osnovnog D flip-flopa je data na Slici 2.



Slika 2: D flip-flop

U osnovi D flip-flop ima 3 ulaza, CLK, nR i D, i jedan izlaz Q. Na ulaz D dolaze podatak koji treba da se snimi, 0 ili 1. Kada u jednom trenutku ulaz CLK pređe iz logičke 0 u logičku 1 tj. kada se desi rastuća ivica ono što se nalazilo na ulazu D će biti zapamćeno u flip-flopu. Izlaz Q je trenutno stanje memorije.

Spajanjem većeg broja flip-flopova dobija se registar. Flip-flopovi u registru imaju onoliko bita širok izaz Q koliko je bita širok ulaz D, dok su CLK i nR signali zajednički.

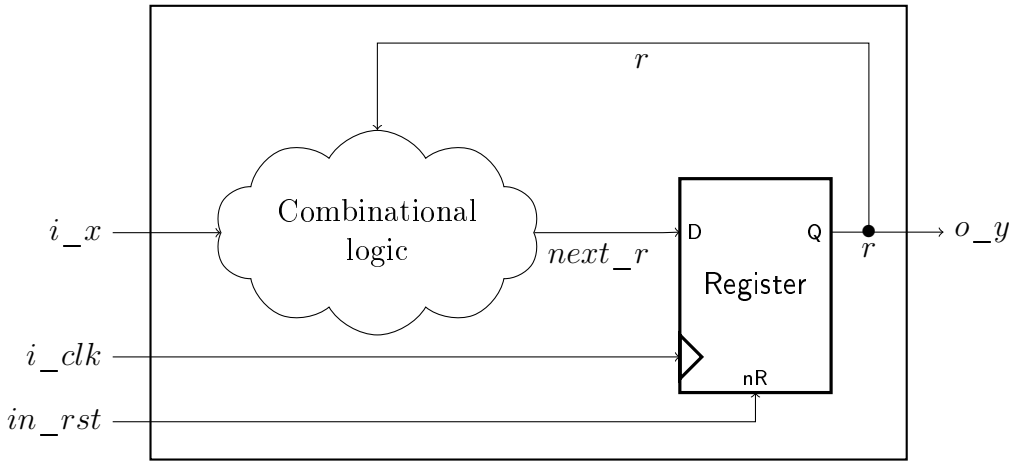
Ulaz CLK je takt (engl. Clock). Takt signal služi za sinhronizaciju u sekvencijalnim mrežama. Takt je signal koji neizmenično menja svoje logičko

stanje. Vreme između dve iste ivice takta je definisano kao perioda takta. Perioda se meri u sekundama. Perioda je obrnuto proporcionalna frekvenciji. Tako da ako uređaj radi na $1GHz$ to znači da perioda njegovog takta je $1ns$. Svrha takt signala je da određuje u kom trenutku će se izvršiti upis u D flip-flop. U trenutku kada je rastuća ivica na CLK signalu tj. kada se CLK signal menja sa logičke 0 na logičku 1 dešava se da podatak koji se nalazi na D ulazu sačuva u memoriju flip-flopu.

Reset signal nR služi da se flip-flop postavi u početno stanje. Kada je na nR signal 0 flip-flop se postavlja na početno stanje, najčešće 0. Sve vreme dok je nR na 0 izlaz će biti 0, nez obzira na takt i na podatke koji se dovode na ulaz flip-flopa. Kada se nR podigne na 1, flip-flop će početi da pamti podatke na sledeću ivicu takta.

Na Slici 3 je prikazana tipična primena D flip-flopa. Postoji D flip-flop na koji se spolja dovode takt i_clk i reset in_rst. Njegov izlaz je signal r. Postoji kombinaciona mreža u koju spolja dolazi ulaz i_x i takođe signal r koji je trenutno stanje flip-flopa. Izlaz iz kombinacione mreže je $next\_r$. $next\_r$ se dovodi na ulaz flip-flopa D.

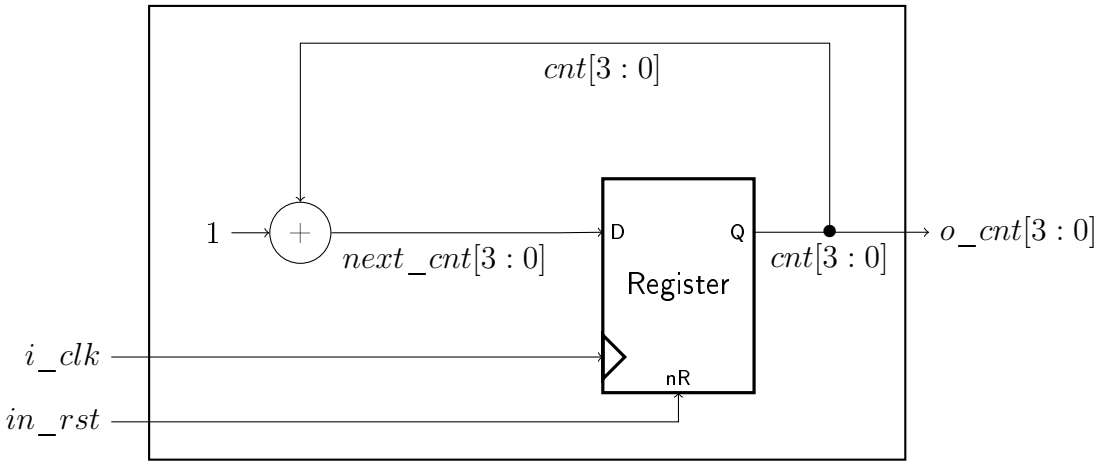
Kombinaciona mreža računa na osnovu ulaza i_x i r svoj izlaz $next\_r$. U trenutku iduće ivice takta vrednost na signalu $next\_r$ se sačuvava u flip-flop ili registar i izlazi na r. U prevodu kombinaciona mreža na osnovu ulaza u sekvencijalnu mrežu i_x i trenutnog stanja registra r računa sledeće stanje registra $next\_r$, koje se sačuvava na sledeću rastuću ivicu takta.



Slika 3: Korišćenje D flip-flopa

Na Slici 4 je prikazan brojač. Njegova svrha je da broji taktove tj. da poveća svoju vrednost na svaku rastuću ivicu takta. To je sekvencijalna mreža bez ulaza (osim i_clk i in_rst koji su specijalni). Izlaz brojača $o\_cnt[3:0]$

je njegovo trenutno stanje.



Slika 4: Brojač

Na Listingu 16 je prikazan VHDL kod za opis ovog brojača.

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  -- Library for arithmetic.
5  use ieee.std_logic_unsigned.all;
6
7  -- Register component.
8  use work.reg;
9
10 entity counter is
11     port(
12         i_clk  : in  std_logic; -- Clock.
13         in_rst : in  std_logic; -- Reset.
14         -- Counter output.
15         o_cnt  : out std_logic_vector(3 downto 0)
16     );
17 end entity counter;
18
19 architecture arch_counter of counter is
20
21     -- This wire is output from register.
22     signal cnt      : std_logic_vector(3 downto 0);
23     -- Wire connect output from adder and input to register.
24     signal next_cnt : std_logic_vector(3 downto 0);
25
26     -- Component declaration.
27     component reg is
28         generic(
```

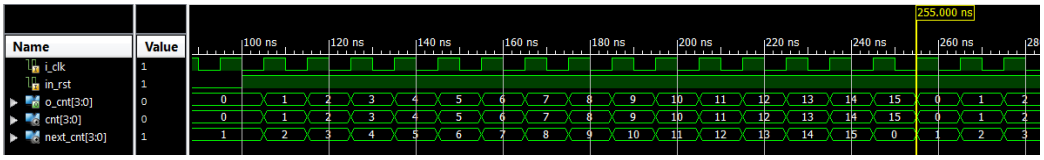
```
29              WIDTH : positive := 1
30          );
31          port(
32              i_clk  : in  std_logic;
33              in_rst : in  std_logic;
34              i_d    : in  std_logic_vector(WIDTH-1 downto 0);
35              o_q    : out std_logic_vector(WIDTH-1 downto 0)
36          );
37      end component reg;
38
39  begin
40
41      cnt_reg: reg
42      generic map(
43          WIDTH => 4 -- Register width in bits.
44      )
45      port map(
46          i_clk  => i_clk,
47          in_rst => in_rst,
48          i_d    => next_cnt,
49          o_q    => cnt
50      );
51
52      -- Adder.
53      next_cnt <= cnt + 1;
54
55      -- Output from register to output from counter.
56      o_cnt <= cnt;
57
58  end architecture arch_counter;
```

Listing 16: 4-bitni brojač

Neka je u početku trenutno stanje registra 0. $cnt[3:0]$ će sadržati trenutno stanje registra tj. 0. Sledeće stranje $next\_cnt[3:0]$ će biti $cnt[3:0]$ uvećano za 1 tj. 1. Na iduću rastuću ivicu takta trenutno stanje registra će postati $next\_cnt[3:0]$ tj. 1, a i $cnt[3:0]$ će sada biti 1. Sada će $next\_cnt[3:0]$ biti 2 i na iduću rastuću ivicu takta trenutno stanje će biti 2. Proces se povalja i u jednom trenutku će stanje biti 15 ili binarno 0b1111. Kada se 15 poveća za jedan dobije se broj 16 tj. 0b10000. Međutim kako je sabirač 4-bitni, i njegov izlaz je samo 4 bita, $next\_cnt[3:0]$ će biti 0b0000 tj. 0. Dakle nakon 15 ide 0, jer je brojač 4-bitan. U prevodu brojač će kada dostigne vrednost $2^n - 1$ preći samo po sebi na 0, gde je n širina brojača u bitima.

Na slici 5 se može videti kako izgleda **waveform** ovog brojača, pri periodi takta od $10ns$ (t_{clk}).

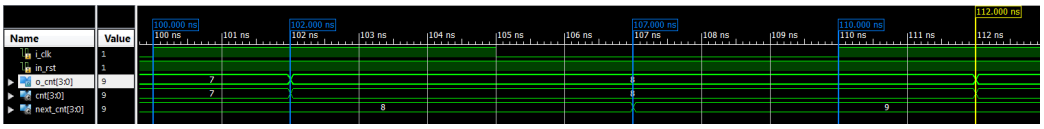
Ovakva simulacija je previše savršena pošto se sve dešava trenutno, kom-



Slika 5: Simulacija brojača

binacione mreže nemaju kašnjenje tj. daju rezultat oma.

Recimo da hoćemo da simuliramo realističan signal. Neka signal na ulazu mora da se zadrži oko $3ns$ pre rastuće ivice takta da bi valjano bio sačuvan u flip-flopu (t_{setup}). Neka signal na ulazu mora da se zadrži oko $1ns$ posle rastuće ivice takta da bi valjano bio sačuvan u flip-flopu (t_{hold}) Neka signal na izlazu kasni $2ns$ posle rastuće ivice takta (t_{delay}) Neka je kašnjenje kombinacione mreže realističnog brojača $5ns$ (t_{comb}). Da bi resultat se dobro setup-ovo mora biti ispunjen uslov $t_{clk} > t_{delay} + t_{comb} + t_{setup}$. U prevodu signal posle padajuće ivice kroz flip-flop i kombinacionu mrežu mora da se stabiliše t_{setup} pre sledeće ivice. To znači da ako bi smanjili priodu takta ispod ukupnog kašnjenja i vremena za setup naš digitalni sistem bi brljavio. Takođe ako bi stavili više logike između flip-flopova kombinaciono kašnjenje bi bilo veće i morali bi da povećamo periodu takta tj. oborimo radnu frekvenciju. Na ovaj način se definiše radna frekvencija nekog digitalnog sistema, procesora ili grafičke. U slučaju ovog našen primera minimana perioda takta t_{clk} morala bi biti minimum $10ns$. Slika 6 prikazuje waveform sa označenim kašnjenjima.



Slika 6: Simulacija realističnog brojača

Recimo da je potrebno zaustaviti uvećavanje brojača na određeno vreme, putem nekog ulaza dozvole i_en. Kada je ovaj signal dozvole na logičkoj 1 brojač treba da se uvećava, a kada je na 0 uvećanje treba da stagnira. Ovo se može izvesti korišćenjem jednog 2-u-1 multipleksera. Selekcioni ulaz multipleksera će biti vezan za signal dozvole. Ulaz multipleksera selektovan sa 1 (dozvola aktivna) će biti izalaz iz sabirača, dok drugi ulaz multipleksera selektovan sa 0 će biti izlaz iz registra. Izlaz multipleksera je vezan na ulaz u registar. Sada kada je signal dozvole na 0 ista vrednost koja je trenutno stanje registra će biti i sledeće stanje registra, čime će registar efektivno ostati na istoj vrednosti.

Recimo da je potrebno napraviti brojač koji broji od 0 do 9, ukupno 10

stanja. Za ovakav brojač se kaže da je modula 10. Da bi se ovo postiglo potrebno je modifikovati kombinacionu mrežu tako da kada je trenutna vrednost brojača 9 on pređe u 0 umesto u 10. Ovo se može postići upotrebnom komparatora koji poredi trenutnu vrednost brojača sa 9 i daje na izlazu logičku 1 kada jeste. Ovaj signal koji govori da li je brojač na 9 se dalje može izvući na selekcioni ulaz 2-u-1 multipleksera sa izlazom koji vodi na ulaz registra, a na čijem ulazu selektovanom 0 stoji izlaz sabirača, i na ulazu selektovanim 1 stoji 0. Tako da kada je trenutno stanje brojača nije 9 izlaz iz komparatora je 0, multiplekser propušta izlaz sabirača i vrednost se povećava. Kada je trenutna vrednost brojača 9 izlaz iz komparatora je 1, multiplekser propušta 0 na ulaz registra i time se brojač vraća na 0.

Signal dozvole može biti korišćen za ulančavanje brojača. Neka imamo samo brojače modula 10. Neka je takt $100ns$. Mi želimo da se jedan brojač uvećava na svaku $1\mu s$ tj. na svakih 10 taktova. Ovo se može uraditi tako što prvi brojač broji taktove direktno tj. uvećava se na svakih $100ns$. Na svaku $1\mu s$ ovaj brojač će da uvek dođe u isto stanje, tj. na svaku $1\mu s$ će brojač da ima vrednost 9. Komparator koji poredi stanje prvog brojača sa 9 će da bude na 1 u dužini od 1 takta na svakih 10 taktova tj. bi će 1 za $100ns$ na svakih $1\mu s$. Ovu vrednost izlaza komparatora možemo koristiti kao signal dozvole drugog brojača. Kada vrednost prvog brojača nije 9 izlaz komparatora će biti 0 i time će drugi brojač biti zaustavljen. Tek kada je vrednost prvog brojača 9 izlaz komparatora će biti 1 i u tom i samo tom taktu će drugi brojač biti uvećan. Time će se postići da se brojač uvećava na svakih 10 taktova.