1 D4 - TEKKOM B

Conditional dan Selected Signal Assignment Statement

Nama : Septian Bagus Jumantoro

Kelas : 1 – D4 Teknik Komputer B

NRP : 3221600039

Dosen : Reni Soelistijorini B.Eng., MT.

Mata Kuliah : Praktikum Rangkaian Logika 2

Hari/Tgl. Praktikum : Jumat, 25 Maret 2022

1. Tujuan

- Mahasiswa dapat membuat program VHDL dengan *Conditional dan Selected Signal AssignmentStatement*
- Mahasiswa dapat membuat dan mendisain dalam rangkaian priority encoder dan decodermendisain rangkaian dari persamaan boolean
- Mahasiswa dapat membuat simulasi testbench dari rangkaian logika.

2. Teori

Concurrent Signal Assignment Statements

Sebuah *concurrent signal assignment statemen*t adalah proses yang berisi hanya *statement* tersebut.Sebuah *statement* dieksekusi secara pararel dengan *concurrent statement* yang lain atau proses lainnya. Terdapat 3 macam *concurrent signal statements* : *signal assignment* sederhana, kondisional *signal statement*, dan *signal statement* terpilih.

Statement ini adalah versi *concurrent* dari sekuensial *signal assignment statement* dan mempunyai bentuk yang sama dengan ini. Sebagai versi sekuensial, *concurrent assignment* mendefinisikan *driver*baru untuk sinyal yang ditetapkan. Sebuah *concurrent assignment statement* muncul diluar sebuah proses, di dalam sebuah arsitektur. Sebuah *concurrent assignment statement* menyajikan bentuk yang disederhanakan dari proses penulisan dan itu setara dengan proses yang berisi *sequential assignment statement* tunggal.Pendeskripsian dari *full adder* bisa jadi dapat disederhanakan dengan menggunakan *concurrent assignment statements*, seperti yang ditunjukan Script 3.1

```
entity add_1 is
      port( a, b, cin: in bit
      ;
            s, cout : out bit
            );
end add_1;

architecture concurrent of
 add_1 issignal s1, s2, s3, s
```

```
begin

        s1 <=  b xor cin;s
        2 <= a and b;

        s3 <=  a and cin;s
        4 <=  b and cin;s
        <= a xor s1;

        cout <= s2 or s3 or s4; end
concurrent ;
```

Script 3. 1. Full Adder

Seperti yang dapat diamati dari contoh sebelumnya, *concurrent assignment* statements muncul secara langsung di dalam arsitektur, bukan di dalam proses. Urutan dimana *statements* ditulis adalah tidak relevan. Pada simulasi, semua *statements* dieksekusi pada siklus simulasi yang sama. Pada kasus proses, aktivitas mereka ditentukan dari perubahan sinyal pada list sensitivitas mereka atau dengan menemui sebuah *wait* statement. Pada kasus *concurrent assignment statements*, perubahan dari berbagai sinyal yang muncul pada sisi kanan dari simbol assignment mengaktivasi eksekusi *statement*, tanpa secara gambling menspesifikasikan list sensitivitas. Aktivasi dari *assignment statement* merupakan aktivasi *independent* dari *concurrent statements* lainnya didalam arsitektur. Sebuah *concurrent assignment statement* digunakan untuk pendeskripsian *dataflow.*

Dengan mensintesis *stetements* tersebut, rangkaian kombinasi telah ditentukan.

Catatan

- Jika ada beberapa *concurrent signal assignment* untuk sinyal yang sama di dalam satu arsitektur, multi *driver* akan dibuat untuk sinyal tersebut. Pada kasus ini, fungsi resolusi juga telah ditentukan sebelumnya oleh *user* untuk tipe sinyal. Sebagai oposisi untuk *concurrent assignment,* jika suatu proses berisi beberapa sekuensial *assignments* untuk sinyal yang sama, maka hanya *assignment* terakhir yang akan efektif.

a. Conditional Signal Assigment Statement.

Kondisional *assignment statement* secara fungsional sama dengan kondisioanal *if statement* danmemiliki sintak sebagai berikut:

signal <= [expression ***when*** *condition* ***else*** *...]expression;*

Nilai dari salah satu ekspresi sumber ditetapkan untuk sinyal target. Ekspresi yang ditetapkan akan menjadi yang pertama yang memiliki kondisi *Boolean* benar (*true*). Ketika mengeksekusi sebuah kondisional *assignment statement*, kondisinya diuji sesuai urutan dimana mereke ditulis. Ketika kondisi sinyal pertama yang dievaluasi bernilai bexnliaiir (*true),* maka ekspresi yang berkaitan ditetapkan pada sinyal target. Jika tidak ada satupun kondisi yang bernilai benar (*true)*, maka ekspresi yang berkaitan dengan *else clause* lah yang ditetapkan pada sinyal target. Perbedaan antara kondisional *assignement statement* dan kondisional *if-statement (*pernyataanpengandaian) adalah sebagai berikut :

- Pada kondisional *assignement statement* adalah sebuah *concurrent statement* (*statement* bersamaan), oleh sebab itu dapat digunakan di dalam sebuah arsitektur, sedangkan *if statement* adalah sekuensial *statement* dan dapat digunakan hanya di dalam proses.
- Kondisional *assignement statement* hanya dapat digunakan untuk menetapkan nilai untuksinyal, sedangkan *if statement* dapat digunakan untuk mengeksekusi sekuensial *statement* manapun.

```
entity xor2 is
 port(      a, b: in bit;
            x: out bit);
end xor2;
architecture arch1_xor2 of xor2 is
begin
      x <= '0' when a = b else
           '1';
end arch1_xor2;
```

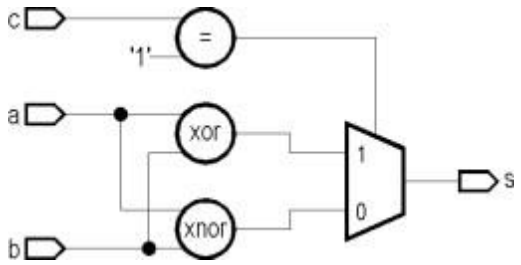
Script 3. 2 Mendefinisikan sebuah arsitektur menggunakan kondisional *assignment statement*.

Kondisional *signal assignment statement* di implementasikan oleh *multiplexer* yang memilih salah satu sumber ekspresi. Gambar 3.1 menyajikan rangkaian yang di *generate* untuk pernyataan berikut:

s <= a xor b ***when*** *c = '1'* ***else***

not (a xor b);

Rangakaian pada Gambar 3.1 awalnya di *generate* oleh sintetis *tool,* tetapi kesetaraan operator akandiminimalkan nanti untuk koneksi yang sederhana, jadi sinyal c tersebut akan mengontrol *multiplexer* secara langsung.

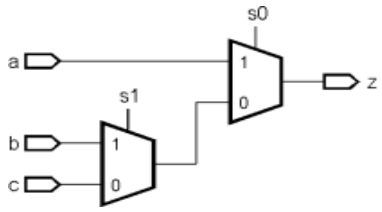


Gambar 3. 1. Conditional *signal assignment statement* sintetis.

Contoh sebelumnya adalah bentuk yang paling sederhana dari kondisional *signal assignment,* dengan hanya satu kondisi yang diuji. Contoh lain, dimana 2 kondisi diujikan, sebagai berikut

```
z <= a when s0 = '1' else
     b when s1 = '1' else
     c;
```

Kondisi tersebut dievaluasi secara berurutan, jadi ekspresi pertama yang memiliki kondisi benar (*true)* akan dipilih. Ini setara dengan kasus pada *hardware* untuk seri 2 *multiplexer,* dengan kondisi yang pertama mengontrol *multiplexer* yang terdekat pada *output.* Rangkaian untuk contoh ini diilustrasikan pada Gambar 2.1. Untuk rangkaian ini, kondisinya telah dioptimasi, sehingga sinyal s0 dan s1 mengontrol *multiplexer* secara langsung. Dari rangkaian ini, dapat dilihat bahwa ketika s0 adalah '1', lalu sinyal a dipilih terlepas dari nilai s1. Jika s0 adalah '0', lalu s1 memilih antara *input* b dan c.



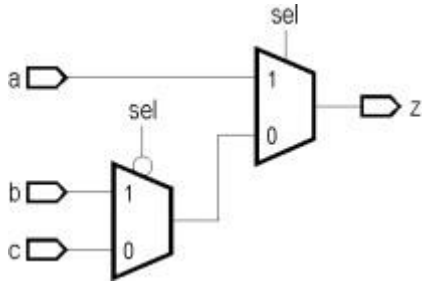
Gambar 3. 2. Sintetis dari 2 kondisional signal assignment statement.

Jika ada banyak cabang, rantai panjang dari *multiplexer* akan menghasilkan secara sintetis. Aspek iniharus dipertimbangkan didalam desain : semakin nanti ekspresi sumber muncul pada *selection list*, semakin banyak *multiplexer* sinyal dari ekspresi ini yang akan melewati rangkaian sintetis.

Setiap kondisi pada kondisional *signal assignment statement* diasumsikan menjadi *independent* dariyang lain nya ketika mensintesis *statement* ini. Ini berarti bahwa jika kondisi nya *dependent* (contohnya, mereka berdasarkan sinyal yang sama), hal itu memungkinkan bahwa tidak akan ada optimasi yang muncul. Contohnya:

```
z <= a when sel = '1' else
     b when sel = '0' else
     c;
```

Pada contoh ini, kondisi kedua adalah *dependent* terhadap kondisi pertama. Nyatanya, pada cabangkedua, sinyal *sel* hanya dapat bernilai '0'. Sehingga, kondisi kedua adalah sia – sia (redudansi) dan pada *else* cabang terakhir tidak dapat diraih. Kondisional *signal statement* ini akan tetap diimplementasikan oleh 2 *multiplexer*, seperti yang diilustrasikan pada Gambar 3.3.



Gambar 3. 3. Kondisional *signal statement* sintetis dengan cabang redudansi.

Pada kasus contoh sederhana sebelumnya, sangat menungkinkan bahwa *tool* sintetis akan mengeliminasi *multiplexer* redudansi, tetapi untuk contoh lebih kompleks tidak dapat dijamin. Alasan kenapa implementasi yang dioptimisasi tidak dapat ditentukan adalah bahwa pada kasusumum, mendeteksi kode VHDL yang tidak dapat dijangkau bukan lah tugas sederhana. Ketika suatu kondisi adalah

dependent terhadap lain nya, hal itu lebih menguntungkan untuk menggunakan *signal assignment* yang dipilih.

Contoh kondisional *signal assignment statement*.

1. Priority Encoder

   Pada contoh pertama adalah priority encoder. Priority encoder memiliki 4 *requests* r(4), r(3), r(2), dan r(1), yang dikelompokan sebagai r *input* 4-bit tunggal, dan r(4) memiliki prioritas tertinggi. *Output* nya adalah kode biner dari urutan *request* tertinggi. Tabel fungsi ditunjukan pada Tabel 3.1. Kode HDL ditnjukan pada script 3.3.

Tabel 3. 1 Fungsi dari 4-request priority encoder

input r	output pcode
1 – – –	100
0 1 – –	011
0 0 1 –	010
0 0 0 1	001
0 0 0 0	000

Pada kode, pertama, cek r(4) dan tandai “100” pada kode jika itu ditekan kan. Lalu dilanjutkan mengecek r(3) *request* jika r(4) tidak ditekan kan dan mengulang prosesnya sampai semua *request* di uji.

```
entity prio_encoder is
    Port ( r : in  STD_LOGIC_VECTOR (4 downto 1);
           pcode : out  STD_LOGIC_VECTOR (2 downto 0));
end prio_encoder;

architecture cond_arch of prio_encoder is
begin
   pcode <= "100" when (r(4)='1') else
            "011" when (r(3)='1') else
            "010" when (r(2)='1') else
            "001" when (r(1)='1') else
            "000";
end cond_arch;
```

Script 3. 3 Priority encoder using a conditional signal assignment statement

2. n-to-2^n binary decoder

   n-to-2^n binary decoder menekan kan 1 bit dari 2n-bit *output* tergantung dari kombinasi *input*. Tabel fungsi 2-4 decoder ditunjukan pada Tabel 3.5. rangkaian nya juga memiliki kontrol sinyal, **en**, yang meng-*enable* fungsi decoding ketika ditekankan. Kode HDL ditunjukan pada *script* 3.4. Kode pertama nya mengecek apakah **en** ditekan kana tau tidak. Jika kondisi **salah** (en = '1'), maka akan menngetes 4 kombinasi biner secara berurutan.

```
entity decoder_2_4 is
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           en : in  STD_LOGIC;
           y : out  STD_LOGIC_VECTOR (3 downto 0));
end decoder_2_4;

architecture cond_arch of decoder_2_4 is
begin
      y <=  "0000" when (en='0') else
            "0001" when (a="00") else
            "0010" when (a="01") else
            "0100" when (a="10") else
            "1000";   -- a="11"
end cond_arch;
```

Script 3. 4 Binary decoder menggunakan kondisional signal assignment statement

b. Selected Signal Assigment Statement.

Seperti kondisional *signal assignment statement, signal assignment statement* terpilih membolehkan untuk memilih sumber ekspresi berdasarkan kondisi. Perbedaan nya adalah bahwa *signal assignment statement* yang terpilih menggunakan kondisi tunggal unttuk memilih diantara beberapa opsi. *statement* ini secara fungsional setara dengan kasus sekuensial *statement.* Sintaknya sebagai berikut

```
with selection_expression select
        signal <= expression_1 when options_1,
                  ...
                  expression_n when options_n,
                  [expression when others];
```

Suatu sinyal target ditetapkan suatu nilai dari salah satu ekspresi. Ekspresi yang terpilih adalah yang pertama dari beberapa ekspresi yang memiliki

pilihan termasuk nilai dari ekspresi pilihan. Suatu sintak pilihan sama dengan *case statement*. Sehingga, setiap pilihan bisa jadi disajikan oleh nilai individual atau oleh satu set nilai. Jika suatu pilihan disajikan oleh satu set nilai, maka nilai individualdari set nilai tersebt juga di tentukan, terlepas dari symbol "|", atau nilai jangkauan, atau kombinasidari itu. Tipe dari pilihan ekspresi menentukan tipe dari setiap pilihan.

Semua nilai dari *range* ekspresi pilihan harus di *cover* oleh satu pilihan. Pilihan terakir bisa jadi diindikasi oleh *keyword* lain nya, yang menspesifikasikan semua nilai dari *range* ekrpresi pilihan yang tidak di *cover* oleh opsi sebelumnya.

Berikut beberapa batasan *(constraints)* untuk pilihan yang bervariasi :

- Nilai dari pilihan (*options)* tidak dapat tumpang tindih satu sama lain.
- Jika pilihan (*option*) lain hilang, semua nilai dari ekpresi pilihan (*selection expression*) yangmemungkinkan harus di *cover* oleh suatu set pilihan.

Catatan

- Pilihan pada *signal assignment* yang dipilih adalah terpisah dari koma.

```
entity  xor 2 is
 port(       a, b: in bit;
             x: out bit);
end xor 2;
architecture  arch_xor 2 of  xor 2  is
      signal  temp: bit_vector (1 downto  0);
begin
 temp <= a & b;
 with  temp select
             x <= '0' when  "00",
             x <= '1' when  "01",
```

Script 3. 5, Dua *input* gerbang *XOR* dimodifikasi untuk menggunakan *signal assignment* terpilih

Contoh Selected signal assignment statement.

Priority Encoder

Kode untuk priority encoder ditunjukan script 3.6. isi deklarasi adalah identik dengan yang ada padascript 3.4 dan dihilangkan.

```
entity prio_encoder is
    Port ( r : in  STD_LOGIC_VECTOR (4 downto 1);
           pcode : out  STD_LOGIC_VECTOR (2 downto 0));
end prio_encoder;

architecture sel_arch of prio_encoder is
begin
   with r select
      pcode <= "100" when "1000"|"1001"|"1010"|"1011"|"1100"|"1101"|"1110"|"1111",
               "011" when "0100"|"0101"|"0110"|"0111",
               "010" when "0010"|"0011",
               "001" when "0001",
               "000" when others;   -- r="0000"
end sel_arch;
```

Script 3. 6 ***Priority encoder using a selected signal assignment statement***

3. n-to-2^n binary decoder

   Kita menggabungkan **en** dan a untuk membentuk sinyal 3-bit, **s**, dan menggunakan itu sebagai sinyalpilihan. Kode yang tersisa, mendaftar kemungkinan kombinasi secara menyeluruh dan nilai *output* yang terkait.

```
entity decoder_2_4 is
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           en : in  STD_LOGIC;
           y : out  STD_LOGIC_VECTOR (3 downto 0));
end decoder_2_4;

architecture sel_arch of decoder_2_4 is
   signal s: std_logic_vector(2 downto 0);
begin
   s <= en & a;
   with s select
      y <= "0000" when "000"|"001"|"010"|"011",
           "0001" when "100",
           "0010" when "101",
           "0100" when "110",
           "1000" when others;   -- s="111"
end sel_arch;
```

Script 3. 7 Binary decoder using a selected signal assignment statement

3. Peralatan

1. PC yang sudah terinstal ISE 13.1
2. Xilinx Spartan 3
3. Downloader JTAG USB
4. Power Suplly 5volt

4. Langkah Percobaan

- Tuliskan program VHDL dari script 3.3, 3.4, 3.6 dan 3.7.
- Tampilkan RTL Schematics dari masing-masing program
- Buatlah simulasi dari masing-masing program.
- Pada simulasi tampilkan semua kondisi *input*-nya.

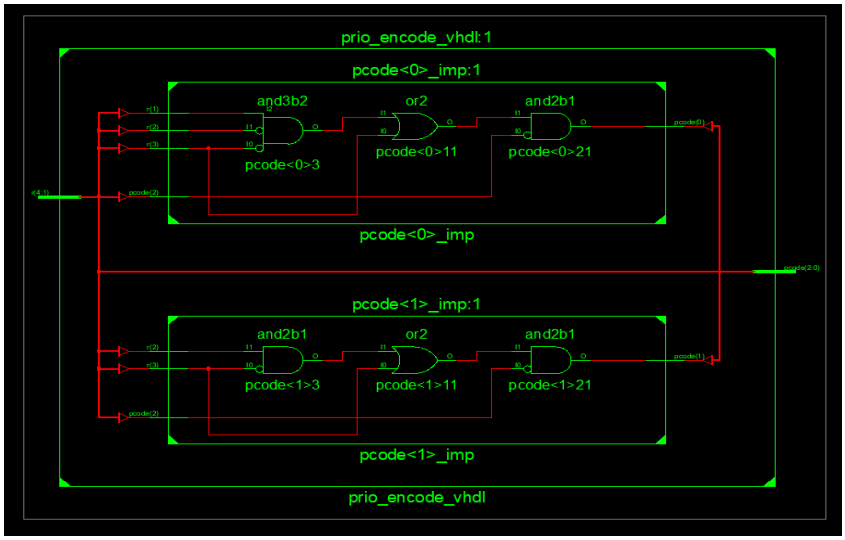
5. Hasil Percobaan

➢ Script 3.3

Kode program VHDL

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  entity prio_encode_vhdl is
24     port (
25     r : in STD_LOGIC_VECTOR (4 downto 1);
26     pcode : out STD_LOGIC_VECTOR (2 downto 0)
27     );
28
29  end prio_encode_vhdl;
30
31  architecture cond_arch of prio_encode_vhdl is
32  begin
33     pcode <= "100" when (r(4) = '1') else
34              "011" when (r(3) = '1') else
35              "010" when (r(2) = '1') else
36              "001" when (r(1) = '1') else
37              "000";
38
39  end cond_arch;
40
```

RTL Schematik



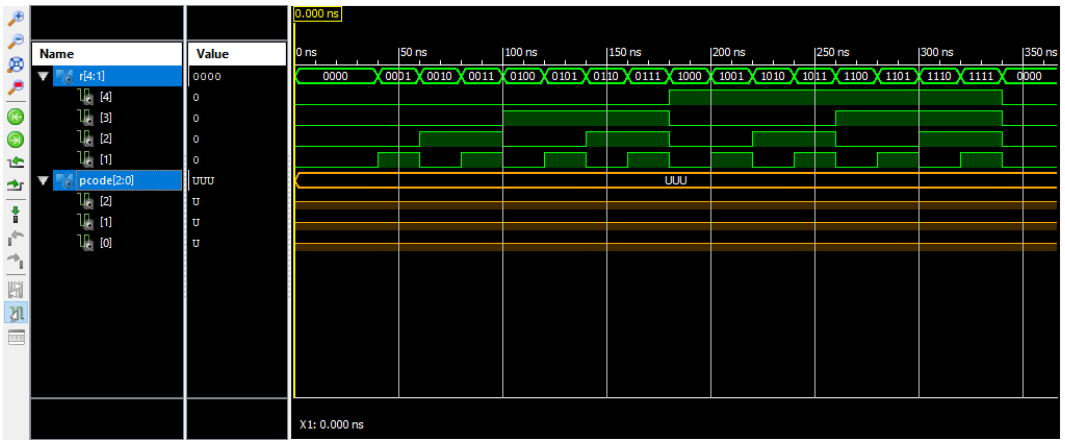
Kode Program TestBench

```
27  --------------------------------------------------------------------------------
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY Tugas3_tb IS
36  END Tugas3_tb;
37
38  ARCHITECTURE behavior OF Tugas3_tb IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT prio_encoder_1
43      PORT(
44           r : IN std_logic_vector(4 downto 1);
45           pcode : OUT std_logic_vector(2 downto 0)
46          );
47      END COMPONENT;
48
49
50     --Inputs
51     signal r : std_logic_vector(4 downto 1) := (others => '0');
52
53     --Outputs
54     signal pcode : std_logic_vector(2 downto 0);
```

```
60
61  BEGIN
62
63     -- Instantiate the Unit Under Test (UUT)
64     uut: prio_encoder_1 PORT MAP (
65            r => r,
66            pcode => pcode
67          );
68
69     -- Stimulus process
70       stim_proc: process
71     begin
72        wait for 20 ns;
73        r <= "0000";
74        wait for 20 ns;
75        r <= "0001";
76        wait for 20 ns;
77        r <= "0010";
78        wait for 20 ns;
79        r <= "0011";
80        wait for 20 ns;
81        r <= "0100";
82        wait for 20 ns;
83        r <= "0101";
84        wait for 20 ns;
85        r <= "0110";
86        wait for 20 ns;
87        r <= "0111";
```

```
88        wait for 20 ns;
89        r <= "1000";
90        wait for 20 ns;
91        r <= "1001";
92        wait for 20 ns;
93        r <= "1010";
94        wait for 20 ns;
95        r <= "1011";
96        wait for 20 ns;
97        r <= "1100";
98        wait for 20 ns;
99        r <= "1101";
100       wait for 20 ns;
101       r <= "1110";
102       wait for 20 ns;
103       r <= "1111";
104       wait for 20 ns;
105       r <= "0000";
106    end process;
107
108 END;
109
```

Timing Diagram

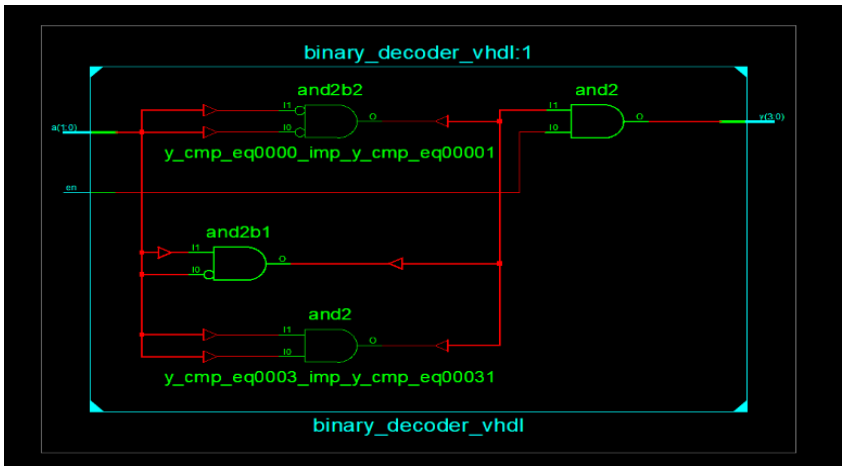


- Script 3.4

Kode program VHDL

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 entity binary_decoder_vhdl is
24    port (
25       a : in STD_LOGIC_VECTOR (1 downto 0);
26       en : in STD_LOGIC;
27       y : out STD_LOGIC_VECTOR (3 downto 0)
28       );
29 end binary_decoder_vhdl;
30
31 architecture Behavioral of binary_decoder_vhdl is
32 begin
33    y <= "0000" when (en = '0') else
34         "0001" when (a = "00") else
35         "0010" when (a = "01") else
36         "0100" when (a = "10") else
37         "1000"; --a = "11"
38 end Behavioral;
```

RTL Schematik

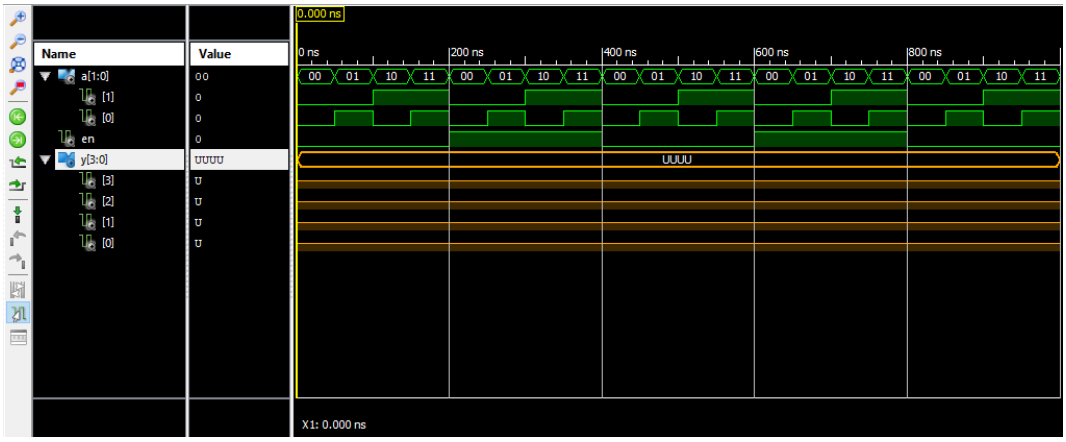


Kode Program TestBench

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY Tugas3_tb IS
5  END Tugas3_tb;
6
7  ARCHITECTURE behavior OF Tugas3_tb IS
8
9      COMPONENT tiga_empat
10     PORT(
11          a : IN std_logic_vector(1 downto 0);
12          en : IN std_logic;
13          y : OUT std_logic_vector(3 downto 0)
14         );
15     END COMPONENT;
16
17
18    --Inputs
19    signal a : std_logic_vector(1 downto 0) := (others => '0');
20    signal en : std_logic := '0';
21
22    --Outputs
23    signal y : std_logic_vector(3 downto 0);
24
25 BEGIN
26
27    -- Instantiate the Unit Under Test (UUT)
28    uut: tiga_empat PORT MAP (
```

```
29            a => a,
30            en => en,
31            y => y
32          );
33
34     -- Stimulus process
35     stim_proc1: process
36     begin
37        wait for 50 ns;
38        a(0) <= not a(0);
39     end process;
40
41     stim_proc2: process
42     begin
43        wait for 100 ns;
44        a(1) <= not a(1);
45     end process;
46
47     en_proc: process
48     begin
49        wait for 200 ns;
50        en <= not en;
51     end process;
52  END;
53
```

Timing Diagram

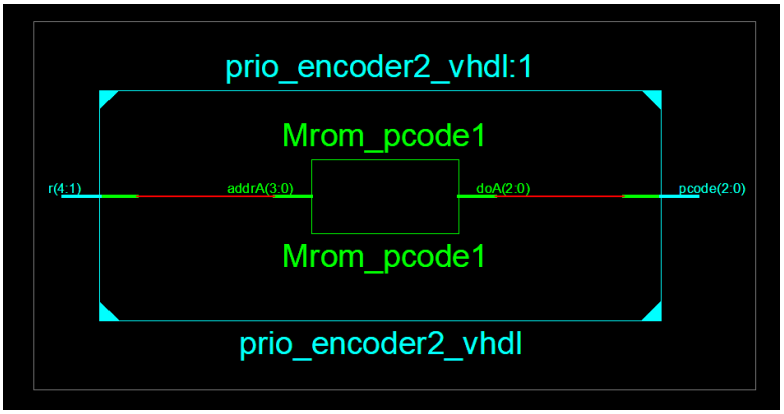


- Script 3.6

  Kode program VHDL

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 entity prio_encoder2_vhdl is
24    port (
25       r : in STD_LOGIC_VECTOR (4 downto 1);
26       pcode : out STD_LOGIC_VECTOR(2 downto 0)
27    );
28 end prio_encoder2_vhdl;
29
30 architecture Behavioral of prio_encoder2_vhdl is
31 begin
32    with r select
33       pcode <= "100" when "1000"|"1001"|"1010"|"1011"|"1100"|"1101"|"1110"|"1111",
34                "011" when "0100"|"0101"|"0110"|"0111",
35                "010" when "0010"|"0011",
36                "001" when "0001",
37                "000" when others; --r = "0000"
38 end Behavioral;
39
```

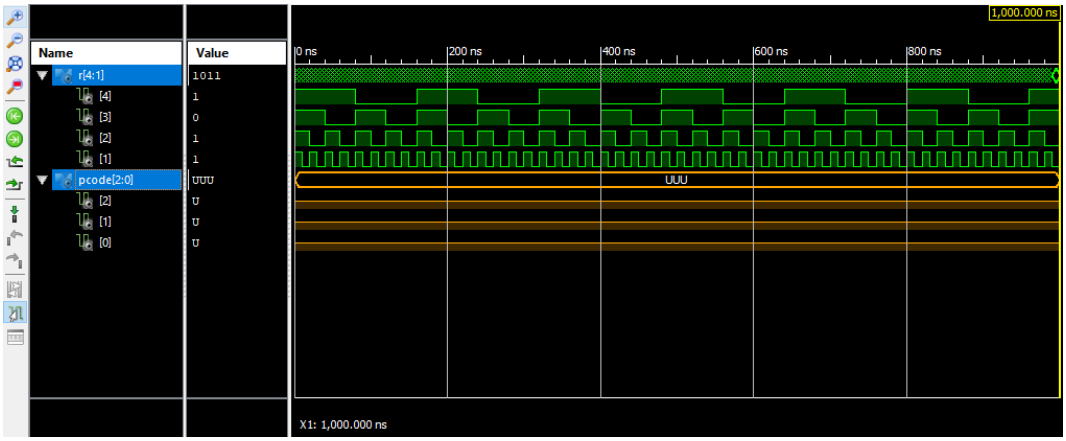
RTL Schematik



Kode Program TestBench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Tugas3_tb IS
END Tugas3_tb;

ARCHITECTURE behavior OF Tugas3_tb IS

    COMPONENT prio_encoder2
    PORT(
         r: IN std_logic_vector(4 downto 1);
         pcode : OUT std_logic_vector(2 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal r : std_logic_vector(4 downto 1) := (others => '0');

   --Outputs
   signal pcode : std_logic_vector(2 downto 0);

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: prio_encoder2 PORT MAP (
          r => r,
          pcode => pcode
        );

   -- Stimulus process
   stim_proc1: process
   begin
      r(1) <= not r(1);
      wait for 10 ns;
   end process;

   stim_proc2: process
   begin
      r(2) <= not r(2);
      wait for 20 ns;
   end process;

   stim_proc3: process
   begin
      r(3) <= not r(3);
      wait for 40 ns;
   end process;

   stim_proc4: process
   begin
      r(4) <= not r(4);
      wait for 80 ns;
   end process;
END;
```

Timing Diagram

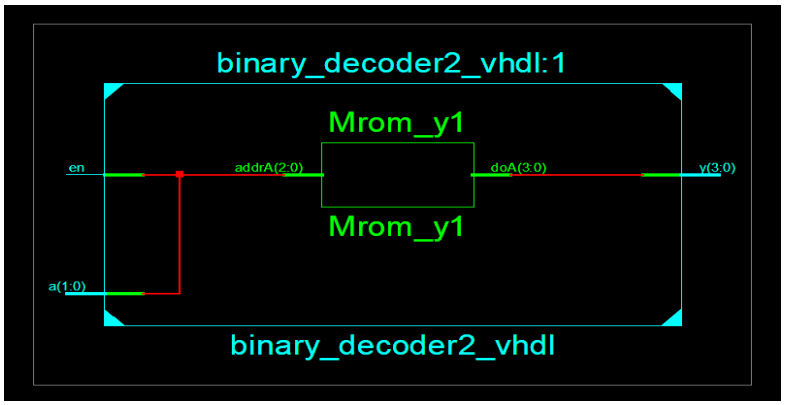


- Script 3.7

Kode program VHDL

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  
23  entity binary_decoder2_vhdl is
24      Port (
25          a  : in STD_LOGIC_VECTOR (1 downto 0);
26          en : in STD_LOGIC;
27          y  : out STD_LOGIC_VECTOR (3 downto 0)
28          );
29  end binary_decoder2_vhdl;
30  
31  architecture sel_arch of binary_decoder2_vhdl is
32      signal s : STD_LOGIC_VECTOR (2 downto 0 );
33  begin
34      s <= en & a;
35      with s select
36          y <= "0000" when "000"|"001"|"010"|"011",
37               "0001" when "100",
38               "0010" when "101",
39               "0100" when "110",
40               "1000" when others; -- s = "111"
41  end sel_arch;
```

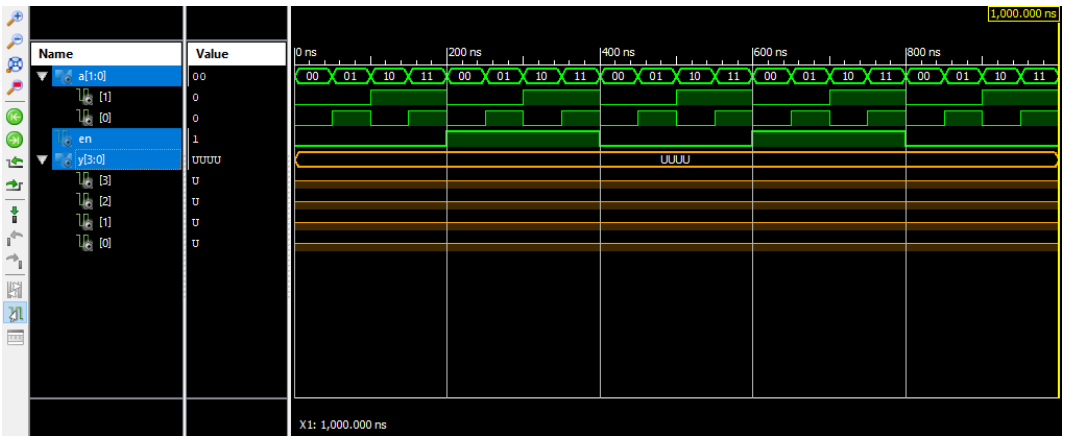
RTL Schematik



Kode Program TestBench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Tugas3_tb IS
END Tugas3_tb;

ARCHITECTURE behavior OF Tugas3_tb IS

    COMPONENT binary_decoder2
    PORT(
         a : IN std_logic_vector(1 downto 0);
         en : IN std_logic;
         y : OUT std_logic_vector(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal a : std_logic_vector(1 downto 0) := (others => '0');
   signal en : std_logic := '0';
   --Outputs
   signal y : std_logic_vector(3 downto 0);

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: binary_decoder2 PORT MAP (
          a => a,
          en => en,
          y => y
        );

   -- Stimulus process
   stim_proc1: process
   begin
      wait for 50 ns;
      a(0) <= not a(0);
   end process;

   stim_proc2: process
   begin
      wait for 100 ns;
      a(1) <= not a(1);
   end process;

   en_proc: process
   begin
      wait for 200 ns;
      en <= not en;
   end process;

END;
```

Timing Diagram



6. Analisa

No.

Date:

Berdasarkan percobaan 3.3 diketahui bahwa program tersebut berfungsi sebagai priority encoder yang memiliki 4 sinyal dan 6 gerbang logika (2 AND, 2 OR, dan 2 AND utk output). Untuk outputnya sebagai berikut:

- => 1xx => output akan keluar jika inputnya 1100, 1101, 1010, 1011, 1xx, 1xx1, 1010, 1111
- => 01x => output akan keluar jika inputnya 0101, 0111, 0110, 0100
- => 001 => output akan keluar jika inputnya 0001
- => 000 => Output akan keluar jika inputnya tidak sesuai

Pada percobaan 3.4 diketahui bahwa program tersebut berfungsi sebagai binary decoder yang terdapat sinyal enable. Untuk inputnya berasal dari biner dan outputnya berupa desimal. Pada saat menggunakan testbench output yang muncul :

INPUT			OUTPUT			
en	a(1)	a(0)	y(3)	y(2)	y(1)	y(0)
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Pada percobaan 3.6 diketahui bahwa program tersebut berfungsi sebagai priority encoder yang hampir sama dengan percobaan 3.3. Namun berbeda pada ketentuan isi dari input r nya.

Pada percobaan 3.7 diketahui bahwa program tersebut berfungsi sebagai binary decoder yang hampir sama dengan percobaan 3.4. Namun perbedaan terdapat pada ketentuan isi dari en dan a(n) yang digabungkan menjadi signal s.

Referensi

1. FPGA Prototyping by VHDL Example. Xilinx Spartan-3. Pong P. Chu
2. Haskell, Richard and Darrin, *Learning By Example Using VHDL Advanced Digital Design With a NEXYS 2TM FPGA Board,* 2009,Oakland university Rochester, Michigan, ISBN 978-0-9801337-4-5.