Pengenalan Quartus Prime Lite dan Verilog

ffr

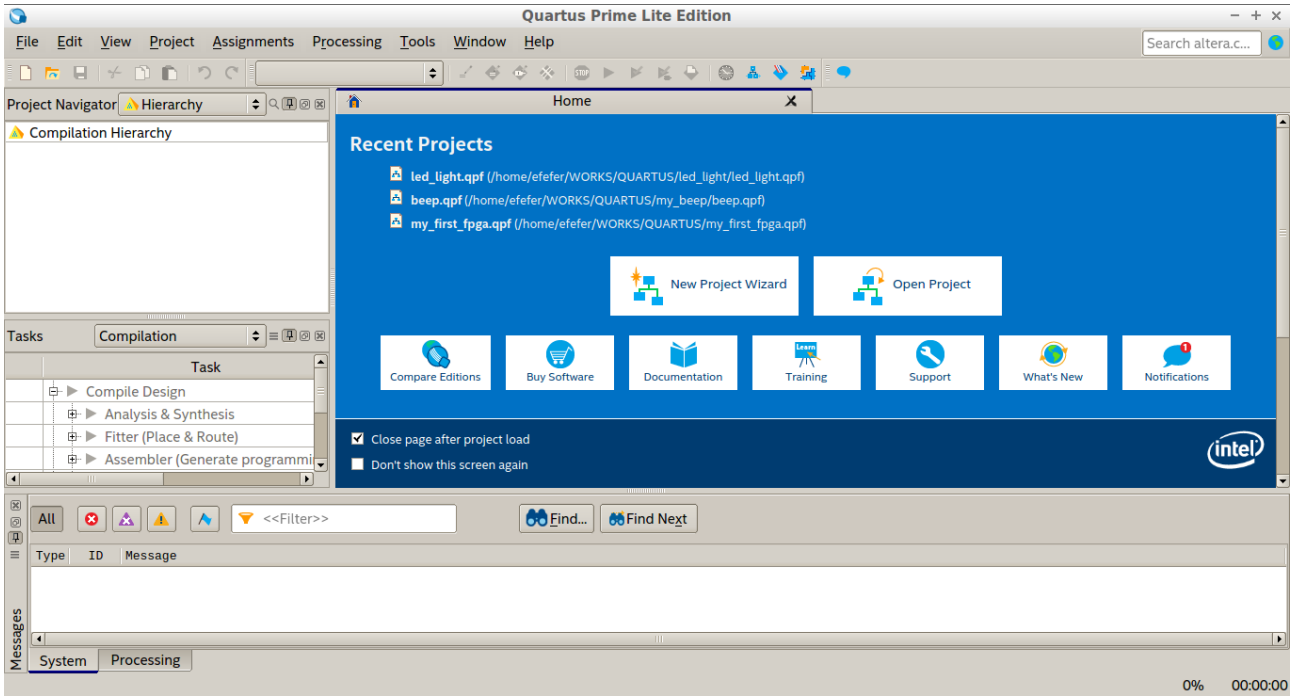
Daftar Isi

1 Pengenalan Quartus Prime

Quartus Prime merupakan perangkat lunak CAD (computer-assisted design) yang digunakan untuk desain rangkaian digital. Quartus Prime dikembangkan oleh Altera. Versi Lite dari Quartus Prime dapat diunduh secara gratis pada laman Altera[1]. Quartus Prime dapat dijalankan pada platform Windows dan Linux. Jendela utama dari Quartus Prime Lite dapat dilihat pada Gambar 1.
Dengan menggunakan CAD, setidaknya ada dua cara untuk mendesain rangkaian digital:

- *schematic capture*, dengan membuat skematik dari rangkaian yang diinginkan.

[1] `http://dl.altera.com/?edition=lite`



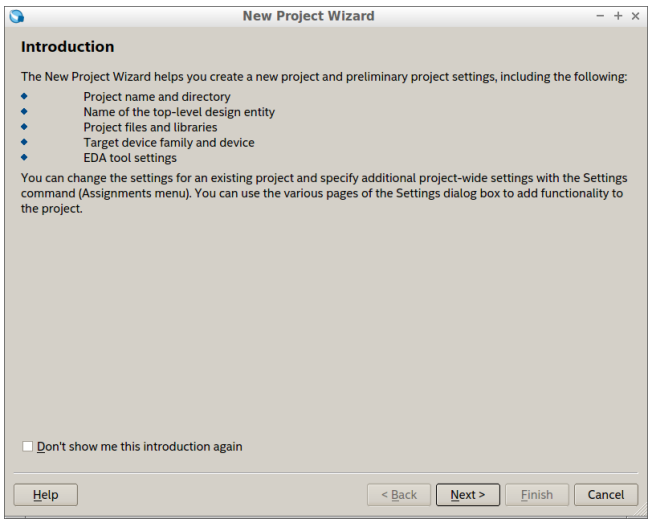
Gambar 1: Tampilan jendela utama Quartus Prime Lite

- menggunakan Hardware Description Language (HDL). Dua jenis HDL yang paling populer adalah Verilog dan VHDL. Kedua bahasa tersebut telah diadopsi sebagai IEEE Standard. Pada tulisan ini akan digunakan Verilog.

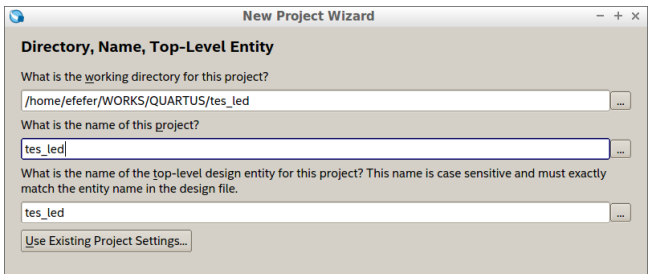
1.1 Membuat project baru

Berikut ini adalah langkah-langkah yang digunakan untuk membuat New Project pada Quartus Prime Lite.

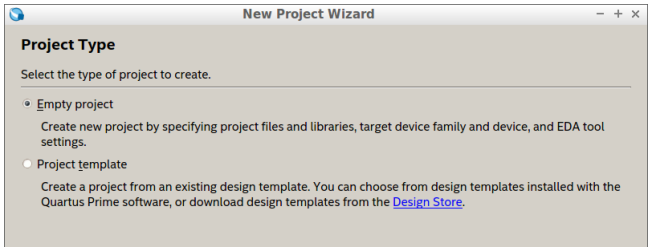
Langkah 1 Pilih menu: File → New Project Wizard. Jendela baru seperti pada gambar berikut akan muncul. Centang Don't show me this introduction again jika perlu. Klik Next.



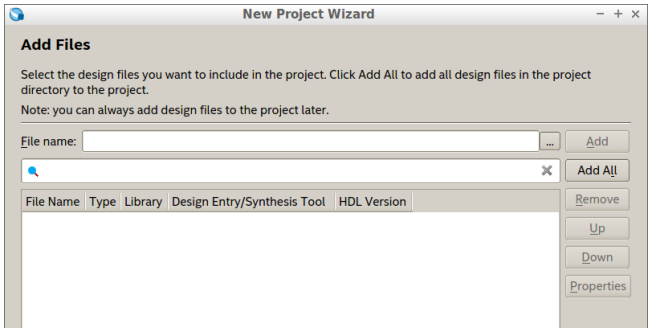
Langkah 2 Tentukan nama Project yang akan dibuat dan direktori di mana file-file yang terkait dengan Project ini akan disimpan. Contoh dapat dilihat pada gambar berikut. Setelah itu, klik Next setelah semua isian diberikan.



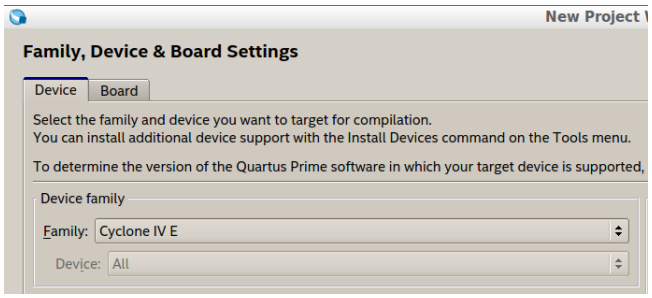
Langkah 3 Kita diminta untuk memilih jenis Project. Pilih Empty Project. Setelah itu, klik Next.



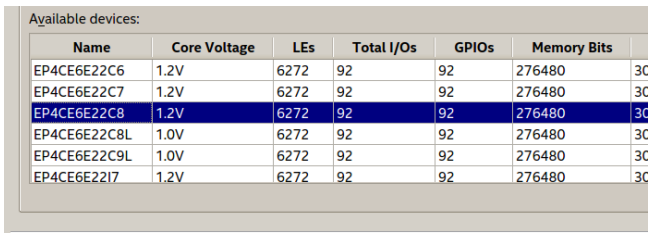
Langkah 4 Pada langkah ini kita dapat menambahkan file yang sudah ada ke Project yang akan dibuat. Jika tidak ada langkah ini dapat dilewati. Klik Next.



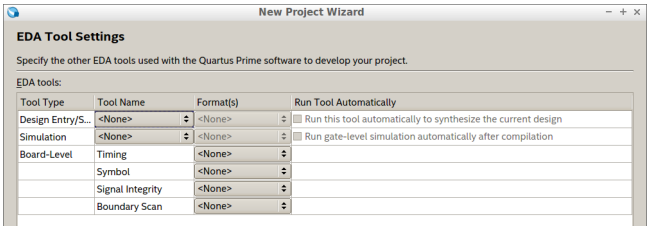
Langkah 5 Pada langkah ini, kita harus memilih Device Family dan Name. Untuk Device Family pilih Cyclone IV E.



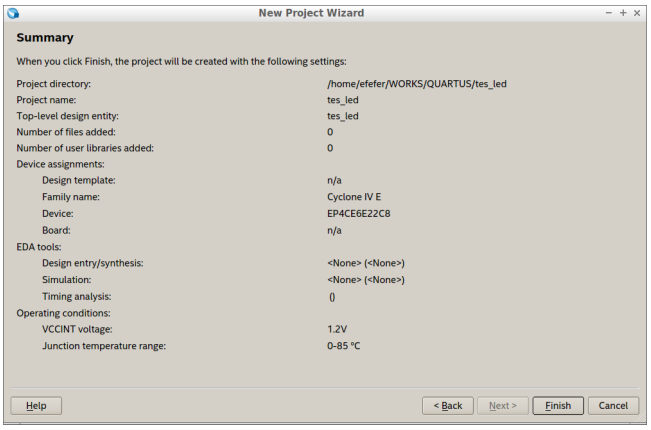
Pada Available Device pilih EP4CE6E22C8. Setelah itu, klik Next.



Langkah 6 Quartus akan meminta kita untuk memilih setting beberapa tools yang mungkin digunakan. Untuk sementara pilih None untuk semua tools. Setelah itu, klik Next.

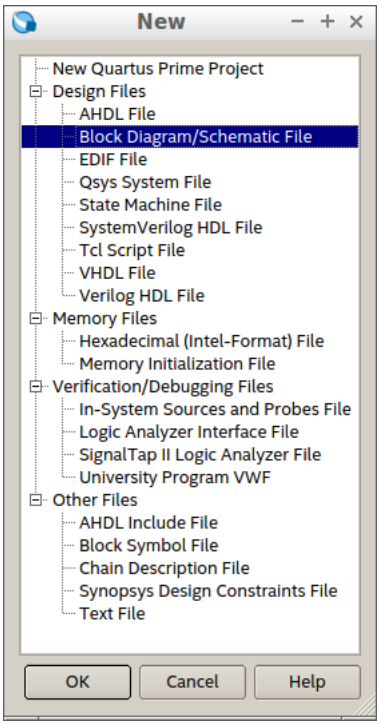


Langkah 7 Pada bagian akhir, Quartus akan memberikan Summary dari Project yang akan dibuat. Setelah itu, klik Finish.

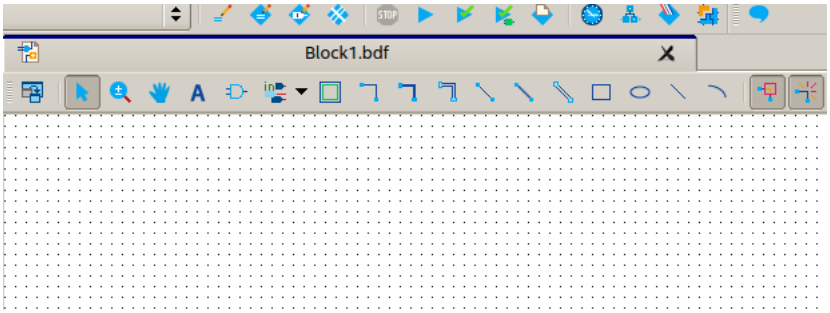


1.2 Menambahkan skematik baru

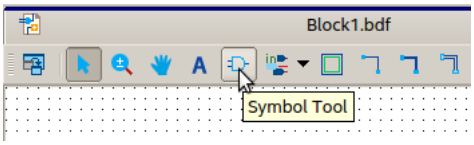
Skematik baru dapat ditambahkan ke dalam project dengan memilih menu File → New. Pilih New Diagram/Schematic File, kemudian OK.



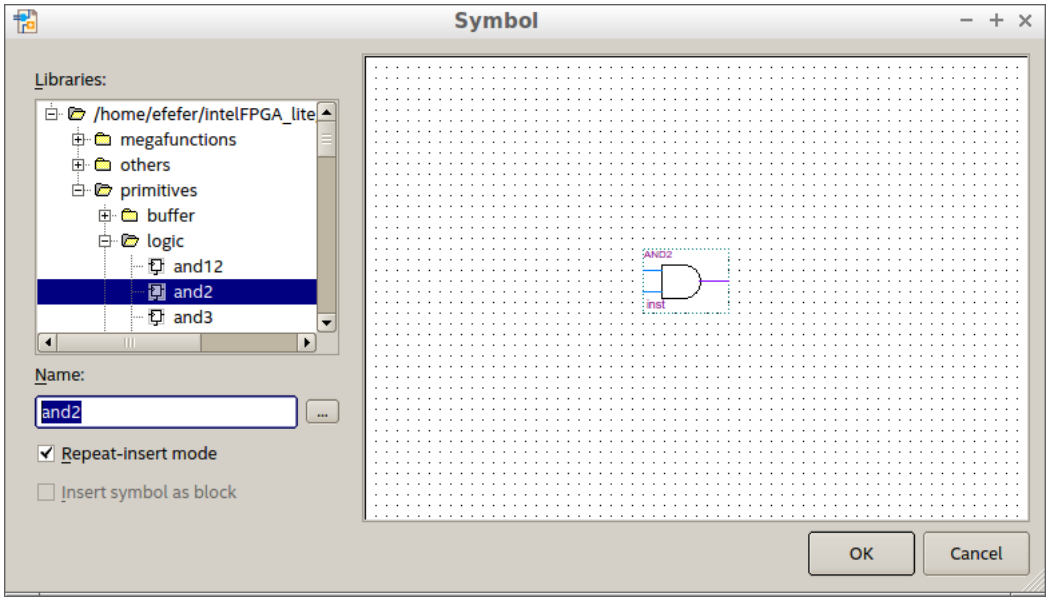
File skematik kosong akan terbuka pada tab baru dengan nama Block1.bdf. Kita dapat membuat skematik yang kita inginkan pada file ini.



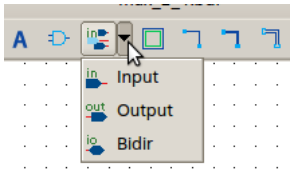
Untuk menambahkan komponen, dapat dilakukan dengan cara mengklik toolbar Symbol Tool.



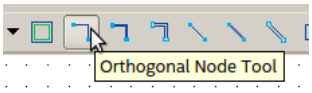
Komponen yang ingin ditambahkan pada skematik dapat diperoleh dengan ekspasi node Libraries, mencari komponen tersebut, dan memilihnya. Misalkan kita ingin menambahkan gerbang AND dengan dua input, maka dapat dipilih pada primitives → logic → and2. Klik OK setelah komponen yang diinginkan telah dipilih.



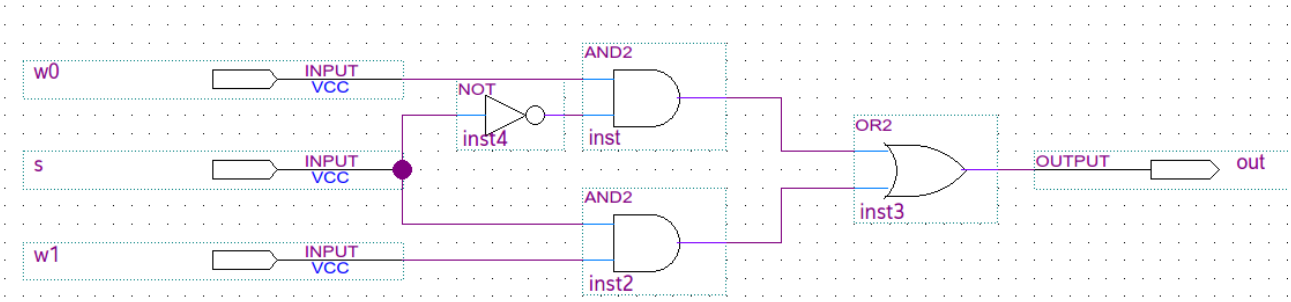
Pemilihan komponen juga dapat dilakukan dengan mengetikkan nama komponen yang diinginakan pada isian Name, misalnya jkff untuk J-K flip-flop.
Khusus untuk menambahkan komponen input dan output, dapat juga digunakan toolbar Pin Tool.



Untuk menghubungkan antara satu komponen dengan komponen yang lain, dapat digunakan Orthogonal Node Tool.



Berikut ini adalah contoh skematik untuk multiplexer 2-to-1:



Skematik ini kemudian dapat digunakan untuk proses lebih lanjut seperti simulasi dan download ke hardware FPGA.

2 Pengenalan Verilog

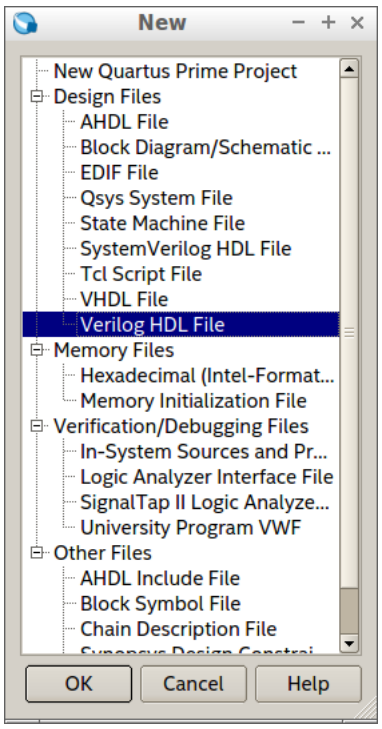
2.1 Sejarah singkat

Selain menggunakan skematik, desain rangkaian digital juga dapat dilakukan dengan menggunakan HDL (hardware description language) seperti Verilog dan VHDL. Pada kesempatan kali ini kita akan menggunakan Verilog.

Verilog pada awalnya dikembangkan sebagai alat untuk simulasi serta verifikasi dari rangkaian digital. Verilog dikembangkan oleh Gateway Design Automation, yang sekarang menjadi bagian dari Cadence Design Systems. Pada awal perkembangannya Verilog merupakan bahasa proprietary, akan tetapi pada tahun 1990 Verilog dilepaskan ke domain publik. Sejak saat itu Verilog menjadi salah satu bahasa yang populer untuk mendeskripsikan rangkaian digital. Pada tahun 1995 Verilog diadopsi sebagai IEEE Standard, yaitu standard 1364-1995. Pada tahun 2001, versi terbaru dari Verilog yang dikenal dengan Verilog 2001 diadopsi menjadi IEEE Standard 1364-2001.

2.2 File Verilog

Pada Quartus Prime, file Verilog dapat ditambahkan ke Project dengan memilih menu File → New dan pilih Verilog HDL File.



Gambar 2: New File Verilog HDL

File Verilog merupakan file teks sehingga kita dapat menggunakan teks editor mana saja untuk meng-edit file ini.

2.3 Representasi rangkaian digital dengan Verilog

Terdapat dua pendekatan untuk mendesain rangkaian digital dengan Verilog:

- struktural: rangkaian digital dideskripsikan dengan struktur rangkaian tersebut, membangunnya dengan elemen-elemen rangkaian seperti gerbang logika, dan mendefisinikan bagaimana elemen-elemen tersebut saling terhubung satu sama lainnya.
- perilaku (behavioral): rangkaian digital dideskripsikan dengan perilaku rangkaian tersebut, bukan dengan struktur rangkaiannya secara langsung. Pendekatan ini dilakukan dengan menggunakan konstruksi pemrograman Verilog seperti konstruksi **if**, **switch**, dan **for** yang juga biasa ditemukan pada bahasa pemrograman komputer. Kompiler Verilog akan melakukan translasi dari konstruksi pemrograman tersebut menjadi rangkaian digital yang sesuai.z

2.3.1 Representasi struktural

Verilog memiliki gerbang logika primitif yang merepresentasikan gerbang logika yang biasa dipakai pada rangkaian.
Gerbang AND dengan dua input, x_1 dan x_2, dan output y, misalnya dapat dinyatakan dengan kode Verilog sebagai berikut.

```
and( y, x1, x2 );
```

Gerbang OR dengan 4 input

```
or( y, x1, x2, x3 x4 );
```

Inverter $y = \bar{x}$

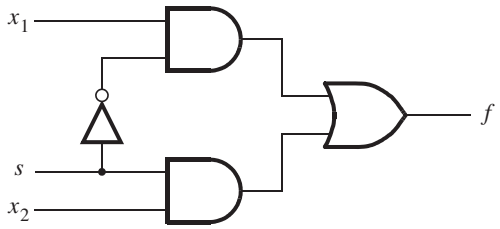
```
not( y, x );
```

Beberapa gerbang primitif pada Verilog dapat diberikan pada tabel berikut ini.

Nama	Deskripsi	Penggunaan
and	$f = (a \cdot b \cdots)$	`and(f, a, b, ...)`
nand	$f = \overline{(a \cdot b \cdots)}$	`nand(f,a,b,...)`
or	$f = (a + b + \cdots)$	`or(a,b,...)`
nor	$f = \overline{(a + b + \cdots)}$	`nor(a,b,...)`
xor	$f = a \oplus b \oplus \cdots$	`xor(a,b,...)`
xnor	$f = a \odot b \odot \cdots$	`xnor(a,b,...)`
not	$f = \bar{a}$	`not(f,a)`

Tabel 1: Beberapa gerbang primitif pada Verilog

Rangkaian digital yang lebih kompleks dapat direpresentasikan dengan menggunakan gerbang primitif tersebut.
Dalam tulisan ini, penjelasan mengenai Verilog akan diberikan melalui contoh.



Gambar 3: Multiplexer

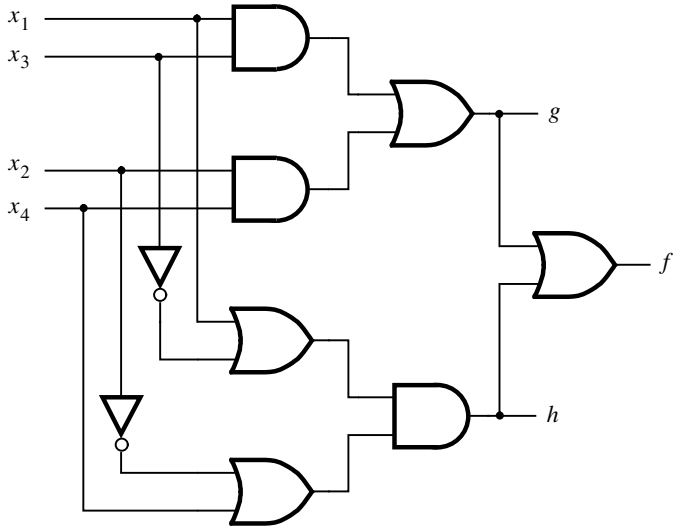
Contoh 1 Multiplexer pada Gambar 3 dapat dijelaskan dengan kode Verilog berikut.

```
module mux2( x1, x2, s, f );
  input x1, x2, s;
  output f;

  not(k,s);
  and(g,k,x1);
  and(h,s,x2);
  or(f,g,h);
endmodule
```

Dalam Verilog, rangkaian logika diberikan dalam bentuk modul, didefinisikan dengan kata kunci **`module`**. Suatu modul dapat terdiri dari input dan output yang disebut sebagai *port*. Pada contoh di atas, modul dengan nama **`mux2`** didefinisikan. Modul **`mux2`** memiliki 4 port yang bernama `x1`, `x2`, `s`, dan `f`. Pernyataan modul ini diakhiri dengan titik koma. Pada pernyataan berikutnya, `x1`, `x2` dan `s` dinyatakan sebagai sinyal input, sedangkan f sebagai sinyal output. Empat baris kode berikutnya menyatakan struktur dari rangkaian.
Variabel k, g, dan h tidak dideklarasikan pada kode di atas. Secara default tipe dari variabel tersebut adalah `wire`.

Contoh 2 Perhatikan rangkaian berikut.



Gambar 4: Rangkaian untuk Contoh 2

Rangkaian ini memiliki 4 input, yaitu x_1, x_2, x_3, dan x_4, serta 3 output, yaitu f, g, dan h. Rangkaian ini mengimplementasikan fungsi logika berikut.

$$g = x_1x_3 + x_2x_4$$
$$h = (x_1 + \bar{x_3})(\bar{x_2} + x_4)$$
$$f = g + h$$

Rangkaian ini dapat diimplementasikan dengan menggunakan kode berikut. Pada kode ini operator digunakan sebagai pengganti gerbang NOT.

```
module Contoh2( x1, x2, x3, x4, f, g, h );
  input x1, x2, x3, x4;
  output f, g, h;

  and( z1, x1, x3 );
  and( z2, x2, x4 );
  or( g, z1, z2 );
  or( z3, x1, ~x3 );
  or( z4, ~x2, x4 );
  and( h, z3, z4 );
  or( f, g, h );
endmodule
```

2.3.2 Representasi behavioral

Menggunakan gerbang logika primitif pada Verilog untuk rangkaian yang kompleks dapat menjadi tugas yang sulit. Sebagai alternatif, kita dapat menggunakan ekspresi logika dan level abtraksi yang lebih tinggi serta kontruksi pemrograman Verilog untuk mendeskripsikan rangkaian berdasarkan perilaku dari suatu rangkaian.
Sebagai contoh, output dari multiplexer pada Gambar 3 dapat dijelaskan dengan persamaan logika berikut.

$$f = \bar{s}x_1 + sx_2$$

Dalam Verilog, persamaan ini dapat dinyatakan dengan kode berikut.

```
module mux2_logic_expr( x1, x3, s, f );
  input x1, x2, s;
  output f;

  assign f = ( ~s & x1 ) | ( s & x2 );
endmodule
```

Pada kode tersebut, operasi AND dan OR dilakukan dengan menggunakan operator "`&`" dan "`|`". Kata kunci **`assign`** berarti *continuous assignment* untuk sinyal f. Ketika ada sinyal pada ruas kanan yang berubah, maka f akan dievaluasi ulang. Jika tidak, maka f akan tetap pada nilai sebelumnya. Dengan menggunakan ekspresi logika, rangkaian pada Gambar 4 dapat dituliskan sebagai berikut.

```
module Contoh2_logic_expr( x1, x2, x3, x4, f, g, h );
  input x1, x2, x3, x4;
  output f, g, h;

  assign g = ( x1 &  x3 ) | (  x2 & x4 );
  assign h = ( x1 | ~x3 ) & ( ~x2 & x4 );
  assign f = g | h;
endmodule
```

Penggunaan ekspresi logika dapat mempermudah penulisan kode Verilog. Akan tetapi level abstraksi yang lebih tinggi juga dapat digunakan dengan cara memberikan spesifikasi mengenai perilaku dari rangkaian. Rangkaian ini juga dapat dideskripsikan dengan menyatakan perilakunya sebagai berikut.

- $f = x_1$ jika $s = 0$, dan
- $f = x_2$ jika $s = 1$

Dalam Verilog, perilaku ini dapat dijelaskan dengan menggunakan pernyataan **if-else** seperti pada kode berikut.

```
module mux2_behavioral( x1, x2, f );
  input x1;
  input x2;
  output f;

  reg f;

  always @(x1 or x2 or s)
    if( s == 0 )
      f = x1;
    else
      f = x2;
endmodule
```

Beberapa hal yang perlu diperhatikan terkait kode di atas.

- Pernyataan **`if-else`** pada Verilog termasuk dalam *pernyataan prosedural*. Verilog mengharuskan pernyataan prosedural diletakkan pada blok **`always`**, seperti pada kode di atas. Setiap blok **`always`** dapat terdiri dari satu pernyataan, seperti pada contoh di atas, atau beberapa pernyataan prosedural. Suatu modul Verilog dapat memiliki beberapa blok **`always`** yang masing-masing blok tersebut menyatakan suatu bagian dari rangkaian yang sedang dimodelkan.

- Pernyataan dalam blok `always` akan dievaluasi sesuai dengan urutan yang diberikan dalam kode. Hal ini kontras dengan pernyataan *continuous assignment* yang dievaluasi secara paralel dan tidak bergantung pada urutan yang diberikan di kode.
- Pada blok `always`, setelah simbol , dalam tanda kurung, disebut dengan *sensitivity list*. Pernyataan pada blok `always` akan dieksekusi jika satu atau lebih dari sinyal yang ada pada *sensitivity list* berubah. Hal ini berguna pada waktu simulasi, di mana simulator tidak perlu mengeksekusi pernyataan setiap waktu. Untuk keperluan sintesis, *sensitivity list* memberitahu *compiler* sinyal apa saja yang secara langsung mempengaruhi keluaran yang diberikan oleh blok `always`.
- Jika suatu sinyal diberikan suatu nilai dengan menggunakan pernyataan prosedural, Verilog mengharuskan sinyal tersebut dideklarasikan sebagai *variabel*. Hal ini dilakukan dengan cara menggunakan kata kunci `reg`.

3 Simulasi rangkaian digital dengan Verilog

Selain menggunakan Quartus Prime, kita juga dapat menggunakan tools lain untuk mempelajari Verilog. Beberapa tools yang sering digunakan adalah **Icarus Verilog** dan **GTKWave**. Icarus Verilog adalah compiler Verilog gratis yang dapat digunakan untuk simulasi rangkaian digital dengan Verilog. Simulasi ini biasanya menghasilkan output waveform yang dapat divisualisasi dengan menggunakan GTKWave.
Pada Ubuntu, Icarus Verilog dan GTKWave dapat diinstal dengan menggunakan perintah

```
sudo apt install iverilog gtkwave
```

Icarus Verilog dan GTKWave juga dapat digunakan pada sistem operasi Windows.
Kode Verilog yang ditulis untuk simulasi suatu rangkaian digital sering disebut dengan *textbench*. Modul yang diuji pada simulasi tersebut disebut sebagai *unit under testing*.
Sebagai contoh pertama, kita akan membuat kode Verilog untuk simulasi modul `mux2`. Buat file baru dengan nama `test_mux2.v` yang isinya sebagai berikut.

```
`timescale 1ns / 1ps
`include "mux2.v"

module test_mux2;
  reg i1, i2, s;
  wire o;

  mux2 uut( .x1(i1), .x2(i2), .s(s), .f(o) );

  initial begin
    $dumpfile("test_mux2.vcd");
    $dumpvars(0, test_mux2);

    i1 = 1'b0;
    i2 = 1'b1;
    s  = 1'b1;

    #1 i1 = 1'b1; i2 = 1'b0;
    #2 i2 = 1'b1;
    #4 s  = 1'b0; i1 = 1'b0;
```

```
    #1 i1 = 1'b1;

    #1 $finish;
  end

  initial begin
    $display("  t   i1   i2   s    o");
    $monitor("%1dns   %b    %b    %b    %b", $time, i1, i2, s, o);
  end

endmodule
```

Compile file ini dengan Icarus Verilog:

```
iverilog test_mux2.v -o test_mux2
```

Perintah ini akan menghasilkan file baru dengan nama `test_mux2`. File ini merupakan file teks yang dapat dieksekusi dengan perintah `vvp`. Pada sistem Linux, file ini sudah bersifat *executable* dan dapat dieksekusi langsung di terminal

```
> ./test_mux2
VCD info: dumpfile test_mux2.vcd opened for output.
  t   i1   i2   s    o
0ns   0    1    1    1
1ns   1    0    1    0
3ns   1    1    1    1
7ns   0    1    0    0
8ns   1    1    0    1
```

Berikut ini adalah beberapa penjelasan kode di atas.

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

Baris ini menyatakan bahwa satuan waktu dalam simulasi adalah 1 ns dan resolusinya adalah 1 ps (0.001 ns).

```
`include "mux2.v"
```

Baris ini hanya menyisipkan file `mux2.v` yang berisi definisi dari modul `mux2`. Baris ini tidak diperlukan apabila *testbench* diberikan pada file yang sama. Beberapa tools simulasi dapat mencari definisi modul yang diperlukan sehingga baris ini biasanya juga tidak diperlukan pada kasus tersebut.

```
module test_mux2;
  reg i1, i2, s;
  wire o;
```

Baris pertama mendefinisikan suatu modul bernama `test_mux2`. Kode ini tidak memiliki definisi port karena hanya dimaksudkan sebagai *testbench* untuk modul **`mux2`**. Pada baris selanjutnya, tiga sinyal dengan nama **`i1`**, **`i2`**, dan **`s`** dideklarasikan. Tiga sinyal ini nantinya akan menjadi input untuk modul **`mux2`** yang nilainya dapat kita atur nanti. Sebagai output dari modul **`mux`**, satu sinyal dengan tipe **`wire`** didefinisikan pada baris selanjutnya.

```
  mux2 uut( .x1(i1), .x2(i2), .s(s), .f(o) );
```

Pada baris ini, satu *instance* dari modul **mux2** didefinisikan dengan menggunakan nama uut (*unit under test*).
Kode yang dieksekusi pada awal simulasi diberikan di dalam blok **initial**.

```
  $dumpfile("test_mux2.vcd");
  $dumpvars(0, test_mux2);
```

Dua baris di atas menyatakan bahwa hasil simulasi akan akan ditulis ke dalam *waveform* yang disimpan pada file **test_mux2.vcd**. File ini dapat divisualisasikan dengan menggunakan GTKWave.

```
i1 = 1'b0;
i2 = 1'b1;
s  = 1'b1;
```

Karena potongan kode di atas diberikan di dalam blok **initial** maka kode di atas memberikan nilai input i1, i2, dan s pada saat awal simulasi.
Untuk mengubah nilai dari input pada saat simulasi, dapat digunakan delay pada blok **initial**, seperti pada baris berikutnya. Delay dispesifikasikan dengan menggunakan tanda # diikuti dengan lama delay dalam satuan yang telah dispesifikasikan sebelumnya (ns).

```
#1 i1 = 1'b1; i2 = 1'b0;
#2 i2 = 1'b1;
#4 s  = 1'b0; i1 = 1'b0;
#1 i1 = 1'b1;
#1 $finish;
```

Baris terakhir, simulasi akan dihentikan 1ns setelah eksekusi baris sebelumnya.
Kode berikut ini juga berada dalam blok **initial**, akan tetapi pada blok yang berbeda. Bagian ini juga dapat diletakkan pada blok **initial**. Kode ini akan menampilkan hasil simulasi ke terminal.

```
$display("  t   i1   i2   s    o");
$monitor("%1dns   %b    %b    %b    %b", $time, i1, i2, s, o);
```

Perintah $monitor dan $display memiliki kemiripan dengan printf pada bahasa C.
TODO: Contoh simulasi dengan ModelSim

4 Eksperimen dengan hardware (FPGA)

4.1 Pengenalan IO

Perlu gambar board FPGA yang akan digunakan
Beberapa daftar port dari board FPGA yang digunakan:

Tabel 2: Beberapa port yang digunakan pada board FPGA

Clock	PIN_23
Reset Push Button	PIN_25
Buzzer	PIN_110
IR	PIN_100
Push Button	
key1	PIN_88
key2	PIN_89
key3	PIN_90
key4	PIN_91
LED	
LED1	PIN_87
LED2	PIN_86
LED3	PIN_85
LED4	PIN_84
Seven-segment LED	
DIG1	PIN_133
DIG2	PIN_135
DIG3	PIN_136
DIG4	PIN_137
SEG0	PIN_128
SEG1	PIN_121
SEG2	PIN_125
SEG3	PIN_129
SEG4	PIN_132
SEG5	PIN_126
SEG6	PIN_124
SEG7	PIN_127

Untuk PIN assignment, buat skematik atau file Verilog terlebih dahulu, kemudian compile. Buka PIN assignment, set pin yang diperlukan.
Prosedur ini diberikan pada contoh menyalakan LED.

4.1.1 LED

Buat project baru dengan nama led_light, misalnya. Kemudian tambahkan file Verilog berikut ke project. Kode Verilog ini memberikan nilai logika ke tiap LED (hardwired, tanpa ada input).

```
module led_light(led);
  output[3:0] led;
  assign led = 4'b0000; // coba ubah-ubah nilai ini untuk tiap LED
endmodule
```

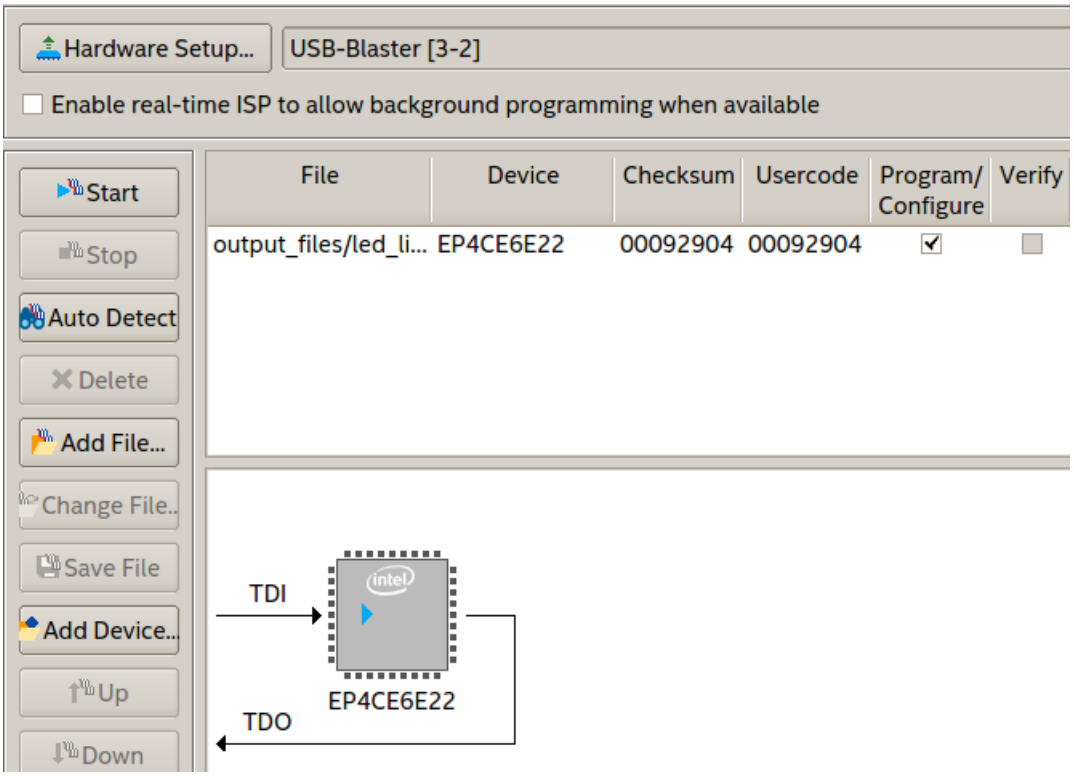
Compile file ini dengan cara klik icon Compile atau dengan menu Processing -> Start Compilation atau menggunakan shortcut Ctrl + L.

Jika tidak ada kesalahan pada saat proses kompilasi, maka langkah selanjutnya adalah melakukan PIN assignment, yang dapat dilakukan dengan memilih menu Assignment -> Pin Planner atau menggunakan shortcut Ctrl + N. Atur PIN assigment sesuai dengan Tabel 2.

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location
led[3]	Output	PIN_87	5	B5_N0	PIN_87
led[2]	Output	PIN_86	5	B5_N0	PIN_86
led[1]	Output	PIN_85	5	B5_N0	PIN_85
led[0]	Output	PIN_84	5	B5_N0	PIN_84
<<new node>>					

Gambar 5: PIN Assignment untuk 4 LED

Compile lagi file tersebut.
Jika tidak ada pesan error, langkah selanjutnya adalah mendownload program ini ke FPGA. Proses ini dapat dilakukan dengan cara memilih menu Tools -> Programmer. Klik button Add File untuk menambahkan file led_light.sof. File ini biasanya ada di dalam subdirektori output dari direktori project. Pastikan juga hardware terdeteksi. Jika belum terdeteksi, tambahkan melalui dengan mengklik button Hardware Setup.



Gambar 6: Tampilan tool Programmer

Catatan

Pada board FPGA yang digunakan urutan LED dari kiri ke kanan adalah LED1, LED2, LED3, dan LED4. Misalkan memberikan assignment sebagai berikut.

- LED1 diwakili dengan `led[0]`
- LED2 diwakili dengan `led[1]`

- LED3 diwakili dengan `led[2]`
- LED4 diwakili dengan `led[3]`

Misalkan juga kita memberikan nilai logika pada `led` dengan kode Verilog berikut.

```
  led = 4'b1010;
```

Maka nilai 0 (nilai bit paling kanan atau LSB) diberikan pada `led[0]` atau LED1. Nilai pada bit kedua dari kanan diberikan untuk `led[1]`, bit ketiga untuk `led[2]`, dan bit keempat (paling kiri atau MSB) untuk `led[3]`.
Bagian `output [3:0] led` pada kode di atas dapat diganti dengan `output [1:4] led` untuk memudahkan assignment nilai logika sesuai dengan urutan LED di board yang digunakan. Sehingga kita dapat melakukan assigment sebagai berikut.

- LED1 diwakili dengan `led[1]`
- LED2 diwakili dengan `led[2]`
- LED3 diwakili dengan `led[3]`
- LED4 diwakili dengan `led[4]`

Cobalah bereksprimen dengan cara mengganti-ganti nilai logika dari `led`, kemudian isilah tabel berikut.

Nilai logika	Keadaan LED (on/off)
0	
1	

4.1.2 Push buttons

Buat project baru, dan buat file Verilog dengan mendefiniskan satu modul dengan input dari push button dan output ke LED.

```
module test_buttons( buttons, led );
  input  [3:0] buttons;
  output [3:0] led;

  assign led = buttons;
endmodule
```

Bisa juga menggunakan potongan kode berikut.

```
  assign led[0] = buttons[0];
  assign led[1] = buttons[1];
  assign led[2] = buttons[2];
  assign led[3] = buttons[3];
```

Cobalah bereksperimen dengan kode Verilog yang ada dan juga menggukan operator Verilog seperti

Nilai logika	Keadaan PB
0	
1	

4.1.3 Seven segments

Lihat Modul 1.

4.1.4 Clock

Berapa frekuensi clock yang digunakan ? 50 MHz ?
LED blinking (sudah menggunakan counter, implementasinya mudah pada Verilog)

4.1.5 IR

Menggunakan protokol NEC.
Ubah kode Verilog yang sudah ada menjadi blok yang menampilkan

4.2 Rangkaian kombinasional

- Implementasi XOR
- half adder dan full adder
- Menyalakan satu atau beberapa LED dengan kombinasi input 4 push button yang diberikan.

  - LED1 menyala jika button1 dan button3 ditekan atau button1 dan button 4 ditekan
  - LED2 menyala jika button2 dan button4 ditekan atau button1 ditekan

- Input BCD (dari) ke output seven segment. Buat tabel kebenaran dan rangkaian (dalam skematik atau Verilog struktural).

Input				Output							
PB1	PB2	PB3	PB4	a	b	c	d	e	f	g	dp
0	0	0	0								
0	0	0	1								
0	0	1	0								
...	...	...	...								
1	1	1	1								

4.3 Rangkaian sekuensial

- Implementasi D flip-flop
- Implementasi J-K flip-flop dan
- Implementasi T flip-flip. Toggle operation: buzzer dan LED
- register, counter
- Kalkulator sederhana, input dari remote IR ?

- Rangkaian multiplexing LED
- Menampilkan LED perdigit

A Persiapan USB-Blaster pada Linux

Diperlukan untuk hardware.
Berikut ini adalah referensi yang dapat digunakan:

- `https://www.altera.com/support/support-resources/download/drivers/dri-usb_b-lnx.html`
- `http://www.fpga-dev.com/altera-usb-blaster-with-ubuntu/`

USB Blaster diperlukan untuk mendownload program ke board.

Tidak perlu instalasi driver khusus untuk Linux.

Quartus Prime menggunakan driver `usb_device` pada Linux untuk mengakses USB-Blaster.

Secara default, yang dapat mengakses USB Blaster adalah root. Agar dapat digunakan untuk user lain selain root, perlu setting tambahan untuk `udev`.

Hubungkan USB-Blaster ke komputer.
Cek output dari `dmesg` ketika USB-Blaster terhubung ke komputer.

```
dmesg | tail
[29293.151196] usb 3-2: new full-speed USB device number 2 using xhci_hcd
[29293.282045] usb 3-2: New USB device found, idVendor=09fb, idProduct=6001
[29293.282056] usb 3-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[29293.282061] usb 3-2: Product: USB-Blaster
[29293.282066] usb 3-2: Manufacturer: Altera
[29293.282070] usb 3-2: SerialNumber: 00000000
```

Cek juga output dari `lsusb`:

```
lsusb
Bus 003 Device 002: ID 09fb:6001 Altera Blaster
```

Berdasarkan output dari `lsusb` diperoleh informasi sebagai berikut:

```
Vendor  ID: 09fb
Product ID: 6001
```

Buat file baru: `/etc/udev/rules.d/51-altera-usb-blaster.rules` dengan konten sebagai berikut. (perlu akses root)

```
SUBSYSTEM=="usb", ATTR{idVendor}=="09fb", ATTR{idProduct}=="6001", MODE="0666"
```

Kemudian reload `udev` (perlu akses root). Lepaskan koneksi USB Blaster dari komputer sebelum melakukan perintah ini.

```
udevadm control --reload
```

Jika tidak ada error, maka seharusnya USB-Blaster dan board yang digunakan sudah dapat diakses melalui komputer. Tes apakah sudah dapat FPGA sudah dapat diakses

```
<quartus_dir>/16.1/quartus/bin/jtagconfig
```

Sebaiknya langkah berikut ini juga dilakukan agar daemon `jtagd` dapat mengenali nama FPGA yang digunakan.

```
cp <quartus_dir>/16.1/quartus/linux64/pgm_parts.txt /etc/jtagd/jtagd.pgm_parts
```

Tes apakah FPGA sudah dapat diakses

```
<quartus_dir>/16.1/quartus/bin/jtagconfig
```

Perintah `jtagconfig` akan secara otomatis menjalankan daemon `jtagd`. Daemon ini dapat konflik dengan daemon `jtagd` yang akan dijalankan ketika menjalankan Quartus Prime Lite. Untuk amannya, pastikan bahwa tidak ada daemon `jtagd` yang berjalan sebelum menjalankan Quartus Prime Lite.