

MODUL V

Implementasi Rangkaian Digital dengan FPGA

1 Tujuan

Tujuan dari praktikum ini adalah sebagai berikut,

1. mengenal teknik pemrograman FPGA dengan menggunakan bahasa deskripsi hardware (Verilog) dan skematik
2. mengenal gerbang-gerbang logika dan membuat rangkaian digital sederhana dengan skematik dan bahasa deskripsi hardware

2 Alat dan Bahan

1. Komputer / laptop dengan aplikasi Quartus Prime
2. Board FPGA (Altera Cyclone IV E)

3 Dasar Teori

programmable logic devices dan FPGA

Entri desain

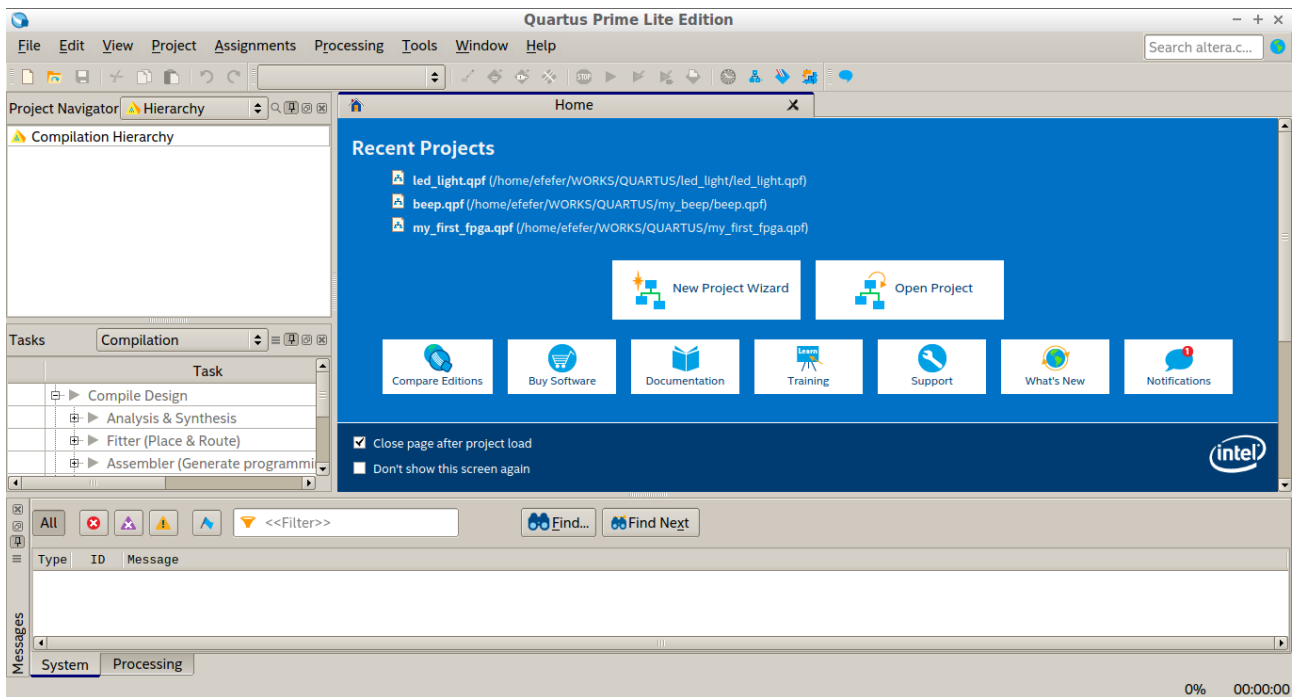
3.1 Pengenalan Quartus Prime

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 Altera¹. 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>



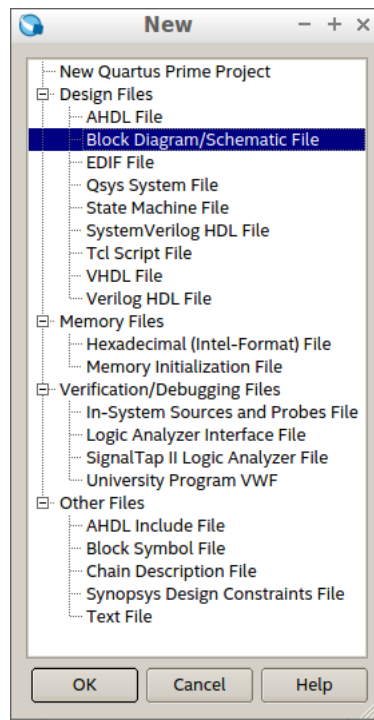
Gambar 1: Tampilan jendela utama Quartus Prime Lite

- 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.

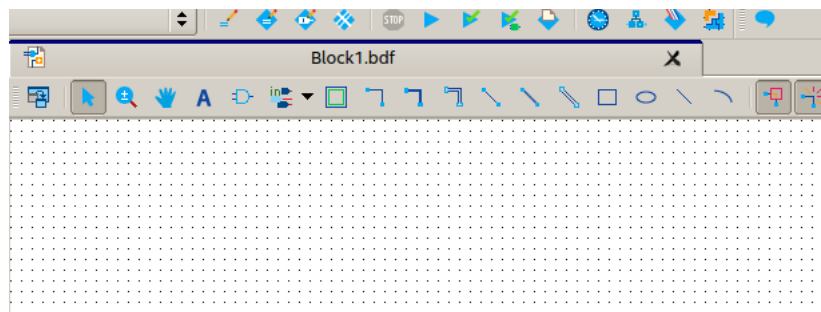
Kita akan mulai dengan menggunakan skematik.

3.2 Menambahkan skematik baru

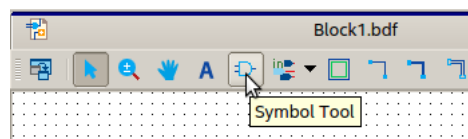
Skematik baru dapat ditambahkan ke dalam project dengan memilih menu **File** → **New**. Pilih **New Diagram/Schematic File**, kemudian **OK**.



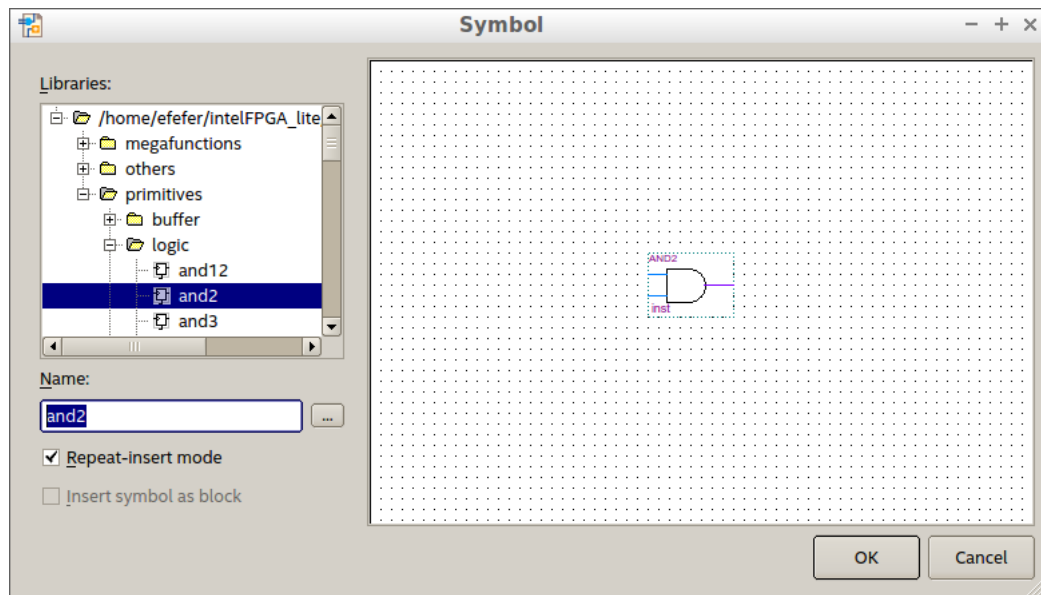
File skematik kosong akan terbuka pada tab baru dengan nama **Block1.bdf**. Kita dapat membuat skematik yang kita inginkan pada file ini.



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

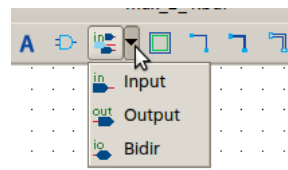


Komponen yang ingin ditambahkan pada skematik dapat diperoleh dengan ekspansi node **Libraries**, mencari komponen tersebut, dan memilihnya. Misalkan kita ingin menambahkan gerbang AND dengan dua input, maka dapat dipilih pada **primitives** → **logic** → **and2**. Klik **OK** setelah komponen yang diinginkan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

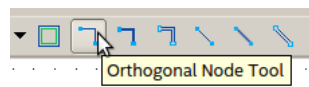


Pemilihan komponen juga dapat dilakukan dengan mengetikkan nama komponen yang diinginkan pada isian **Name**, misalnya **jkff** untuk J-K flip-flop.

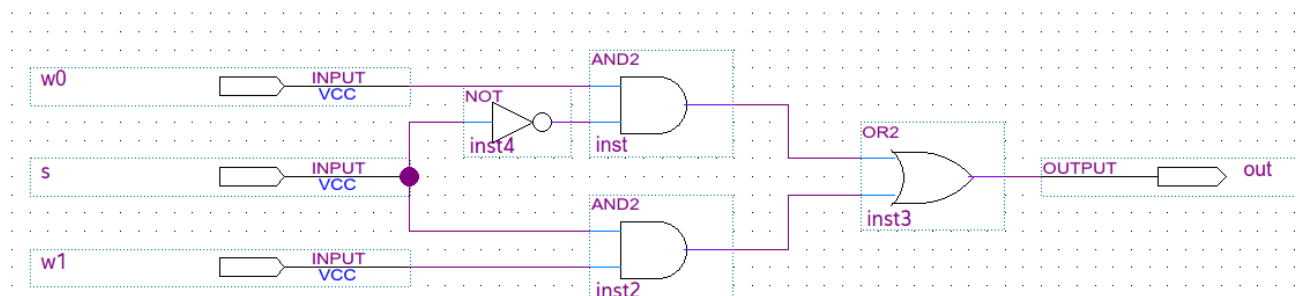
Khusus untuk menambahkan komponen input dan output, dapat juga digunakan toolbar **Pin Tool**.



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

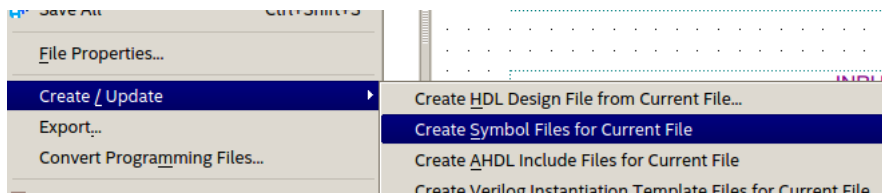


Berikut ini adalah contoh skematik untuk multiplexer 2-to-1:

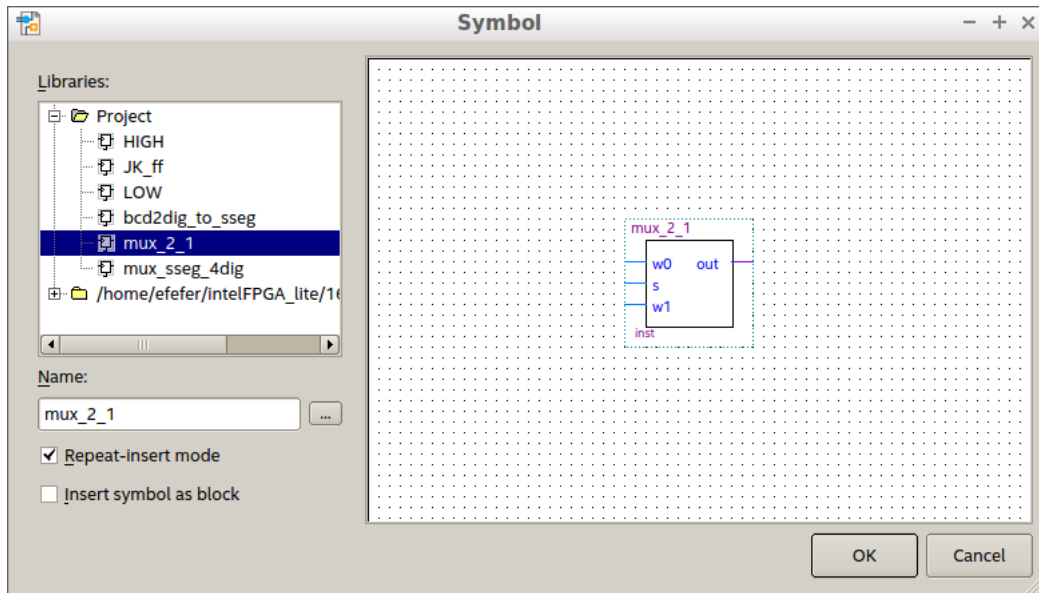


Skematik ini kemudian dapat digunakan untuk proses lebih lanjut seperti simulasi dan download ke hardware FPGA.

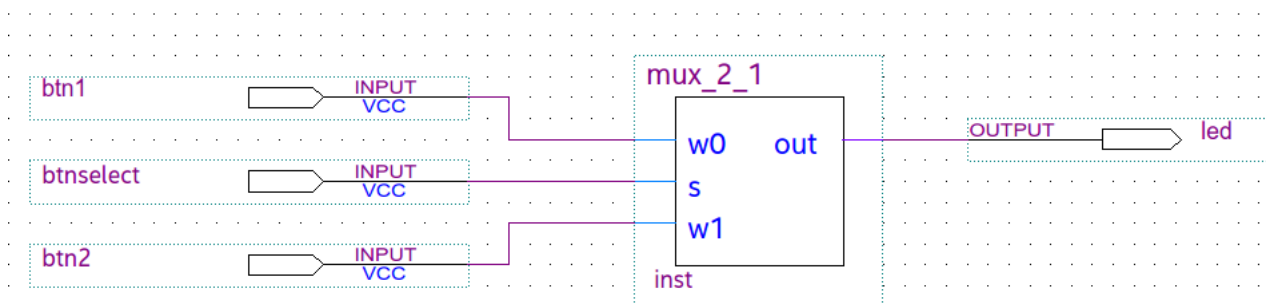
Menkonversi ke skematik ke simbol/blok sehingga dapat digunakan sebagai bagian dari skematik lainnya: gunakan menu **File** → **Create/Update** → **Create Symbol Files for Current File**.



Simbol ini dapat kita gunakan untuk pada skematik lain dengan cara yang sama ketika kita menambahkan gerbang logika dasar, yaitu dengan mengklik icon **Symbol Tool**.



Contoh penggunaan (yang paling sederhana) dapat dilihat pada gambar berikut.



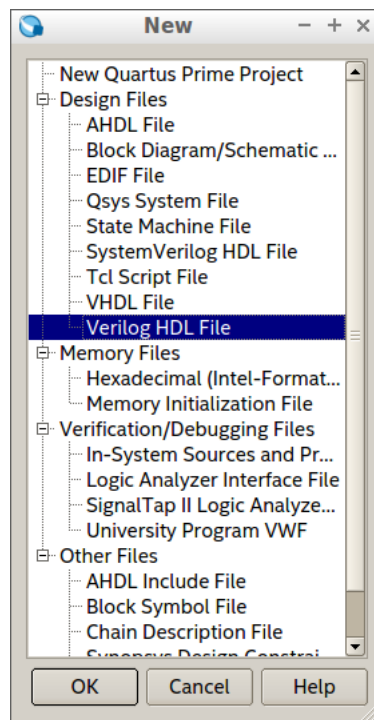
3.3 Bahasa Deskripsi Hardware: Verilog

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.

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 apa saja untuk mengedit file ini.

3.3.1 Representasi rangkaian digital dengan Verilog

Terdapat dua pendekatan yang akan kita gunakan untuk mendesain rangkaian digital dengan Verilog pada praktikum ini

- struktural: rangkaian digital dideskripsikan dengan struktur rangkaian tersebut, membangunnya dengan elemen-elemen rangkaian seperti gerbang logika, dan mendefinisikan 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.

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 );
```

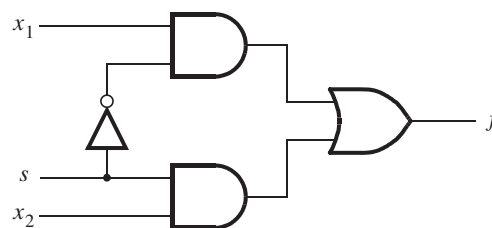
Beberapa gerbang primitif pada Verilog dapat diberikan pada tabel berikut ini.

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;

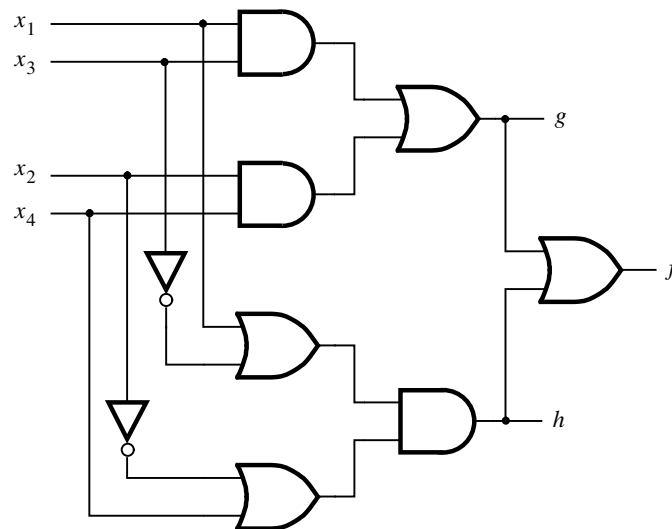
  not(k,s);
  and(g,k,x1);
  and(h,s,x2);
  or(f,g,h);
endmodule
```

```
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 *x1*, *x2*, *x3*, dan *x4*, 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

```

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 abstraksi yang lebih tinggi serta konstruksi 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

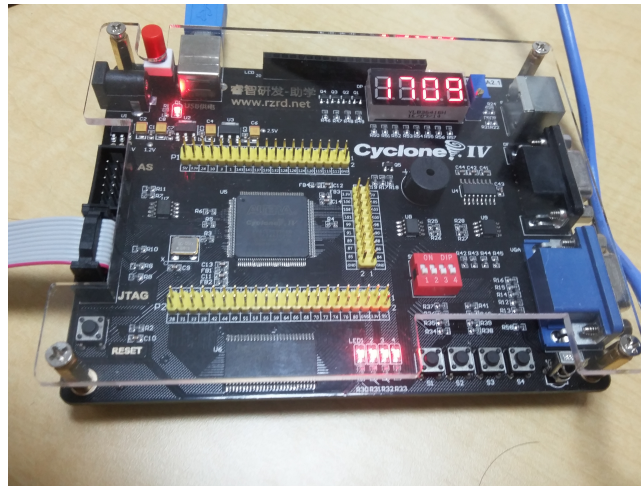
```

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**.

4 Praktikum

Berikut ini adalah board FPGA yang akan kita gunakan dalam praktikum ini.



Gambar 5: FPGA Board Versi A

Perlu gambar FPGA yang kedua dan informasi PIN untuk board kedua.

Berikut ini adalah pedoman umum yang akan kita lakukan pada praktikum ini.

- Buat proyek Quartus Prime baru, misalkan dengan nama **Modul5_Kelompok_XX**. di dalam direktori dengan nama yang sama. *Hindari penggunaan spasi dalam nama proyek.*
- Langkah untuk menambahkan file skematik baru dapat dilakukan sesuai dengan langkah yang dijelaskan pada bagian 3.2. Penambahan file Verilog juga dapat dilakukan dengan cara yang mirip.
- Sebelum melakukan kompilasi desain (skematik atau Verilog) pastikan bahwa desain tersebut telah dibuat menjadi top-level entity ² melalui menu **Project** → **Set as Top-Level Entity** atau keyboard shortcut **Ctrl+Shift+J**.
- Lakukan kompilasi setelah desain selesai dibuat.
- Lakukan PIN assignment. Pastikan bahwa semua port yang akan digunakan telah diberikan PIN yang sesuai.
- Lakukan kompilasi sekali lagi.
- Download desain yang telah dibuat ke FPGA dan verifikasi apakah telah berjalan sesuai dengan yang diharapkan.

Langkah-langkah tersebut akan diilustrasikan pada eksperimen dengan LED.

4.1 Pengenalan IO: LED, push button dan seven segment

4.1.1 LED

LED merupakan output yang paling sederhana pada board FPGA yang akan kita gunakan. Salah satu tujuan utama dari eksperimen ini adalah untuk mengetahui keadaan LED apabila diberikan nilai logika

²Top-level entity memiliki peran yang sama dengan fungsi `main` pada bahasa C/C++

0 dan 1. Hal ini lebih mudah dilakukan dengan menggunakan Verilog. Kita juga akan mempelajari sintaks Verilog untuk

Buat file baru, misalnya dengan nama **led_light.v**. *Hindari nama file dengan menggunakan spasi.* Kemudian ketik kode Verilog berikut pada file tersebut. 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**.

Berikut ini adalah data PIN yang diberikan.

Tabel 2: Data PIN untuk LED

LED	
LED1	PIN_87
LED2	PIN_86
LED3	PIN_85
LED4	PIN_84

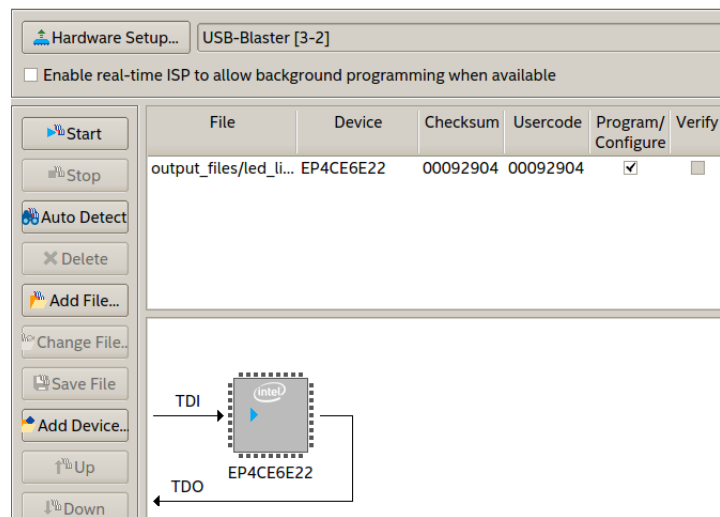
Atur PIN assignment sesuai dengan keperluan. Jendela PIN assignment ini dapat dibiarkan terbuka.

Node Name	Direction	Location	I/O Bank	VREF Group	Filter Location
 led[3]	Output	PIN_87	5	B5_NO	PIN_87
 led[2]	Output	PIN_86	5	B5_NO	PIN_86
 led[1]	Output	PIN_85	5	B5_NO	PIN_85
 led[0]	Output	PIN_84	5	B5_NO	PIN_84
<<new node>>					

Gambar 6: Contoh PIN Assignment untuk 4 LED, mohon disesuaikan dengan kode Verilog dan board yang digunakan

Compile lagi file tersebut setelah PIN assignment dilakukan.

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 proyek. Pastikan juga hardware telah terdeteksi. Jika belum terdeteksi, tambahkan melalui dengan mengklik button **Hardware Setup**.



Gambar 7: Tampilan tool **Programmer**

Misalkan 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 assignment 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	

Untuk penggunaan berikutnya, isikan juga PIN untuk LED, sesuai dengan letaknya di board FPGA. Dari kiri ke kanan:

PIN:	PIN:	PIN:	PIN:
------	------	------	------

4.1.2 Push buttons

Setelah mengetahui keadaan LED jika diberikan nilai logika 0 dan 1, kita dapat mengetahui keadaan logika push button apabila dilepas dan ditekan.

Buat file Verilog baru dengan mendefinisikan 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 menggunakan operator Verilog seperti

Nilai logika	Keadaan PB
0	
1	

Untuk penggunaan berikutnya, isikan juga PIN untuk push button, sesuai dengan letaknya di board FPGA. Dari kiri ke kanan:

PIN:	PIN:	PIN:	PIN:
------	------	------	------

4.1.3 Seven segments

Lihat juga Modul 1.

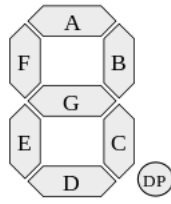
Berikut ini adalah tabel nomor PIN terkait dengan seven-segment display yang diberikan oleh distributor FPGA.

Tabel 3: Port Seven-segment LED

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

Seperti pada percobaan sebelumnya, buat file Verilog dengan konten sebagai berikut. Lengkapi kode tersebut dan modifikasi bagian yang Anda perlukan.

```
module tes_sseg(  
    output [7:0] sseg,  
    output [3:0] en_dig  
);  
  
    assign sseg[7] = 0;  
    // lengkapi jika diperlukan  
    assign sseg[0] = 1;  
  
    assign en_dig[3] = 1;  
endmodule
```



Gambar 8: Seven segment

Untuk referensi lebih lanjut isilah tabel berikut. dengan memperhatikan Gambar 8.

Segmen	PIN
A	
B	
C	
D	
E	
F	
G	
DP	

4.2 Rangkaian kombinasional

4.2.1 Rangkaian XOR

TUGAS

Buatlah rangkaian digital dengan spesifikasi sebagai berikut.

- Input dua buah push button, output 1 LED
- LED akan menyala jika dan hanya jika 1 push button ditekan.

Implementasikan dengan menggunakan gerbang logika AND, OR, dan NOT (jika diperlukan). (Verilog + skematik)

4.2.2 Pendeteksi ganjil

Membuat rangkaian logika digital sederhana dengan spesifikasi sebagai berikut.

- input push button sebanyak 3 buah, misalkan bernama A, B dan C
- output berupa satu LED dengan nama led
- LED akan menyala jika jumlah push button yang ditekan ganjil

Misalkan ketika push button ditekan nilai logikanya adalah 1 dan LED akan menyala jika nilai logika adalah 1, maka dapat diperoleh tabel sebagai berikut.

A	B	C	LED
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$LED = ABC + \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

Kode Verilog yang mengimplementasikan persamaan logika di atas adalah sebagai berikut.

```

module deteksi_ganjil_3(
    input A, input B, input C,
    output led );

    assign led = ( A & B & C ) | ( ~A & ~B & C ) |
                ( ~A & B & ~C ) | ( A & ~B & ~C );

endmodule

```

TUGAS Buatlah desain (skematik atau Verilog) untuk mendeteksi apakah jumlah push button yang ditekan adalah ganjil untuk **empat** buah push button. Sederhanakan persamaan logika yang diperoleh jika diperlukan dengan metode Karnaugh map.

4.2.3 Converter BCD ke seven-segment

Input BCD (dari) ke output seven segment. Buat tabel kebenaran dan rangkaian (dalam skematik atau Verilog struktural). Minimisasi persamaan logika yang diperoleh dengan menggunakan metode Karnaugh mapping.

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								

5 Tugas Pendahuluan

1. Carilah informasi umum mengenai FPGA:
 - penggunaan FPGA
 - perbandingan FPGA dengan mikrokontroler, kelebihan dan kekurangan
2. Carilah informasi mengenai bahasa deksripsi hardware, terutama Verilog.
3. Lengkapi tabel kebenaran untuk konversi dari BCD ke seven segment.