Praktikum Rangkaian Logika 2

Disain Rangkaian Digital dengan VHDL

VC 042107

Oleh:

Fernando Ardilla S.ST, M.T

NIP. 198202032008121001

Program Studi Teknik Komputer

Departemen Teknik Informatika dan Komputer

Politeknik Elektronika Negeri Surabaya

2018

SURAT PERSETUJUAN PROPOSAL PEMBUATAN BUKU AJAR

Nama Mata Kuliah : Rangkaian Logika 2
Kode Mata Kuliah : VC 042107
Waktu Perkuliahan : 2 jam/minggu
Nama Penulis : Fernando ardilla, S.ST., MT.
Golongan/Pangkat : Penata Muda Tk 1 (III/b)
Jabatan Fungsional : Asisten Ahli

Menyetujui,
Ketua Program Studi
Teknik Komputer

Surabaya, 26 Juni 2018
Ketua Tim Penyusun,

Riyanto Sigit, ST., M.Kom, Ph.D.
NIP. 197008111995121001

Fernando Ardilla,S.ST MT.
NIP. 198202032008121001

Mengetahui,

Ketua P3AI

Ketua Departemen
Teknik Informatikan dan Komputer

Dr. Rusminto Tjatur Widodo
NIP. 196604231989031001

Tri Harsono, S.Si, M.Kom, Ph.D
NIP. 196901071994031001

Table of Contents

Percobaan 1 - Pengenalan FPGA dan ISE Design Tool Xilinx

1. Tujuan

Dapat membuat *new project* menggunakan software ISE Xilinx.

Dapat membuat program rangkaian gerbang logika dasar dan memasukan program VHDL ke dalam *Board Xilinx Spartan-3*.

2. Teori

a. FPGA dan VHDL

Field Programmable Gate Arrays (FPGA) adalah programmable device besar yang tersusun atas modul-modul logika independen yang dapat di konfigurasi oleh pemakai yang di hubungkan melalui kanal-kanal routing yang dapat di program.

FPGA mempunyai kelebihan sebagai berikut :

1.Dikonfigurasi oleh *End User*

2.Tidak memerlukan proses Fabrikasi

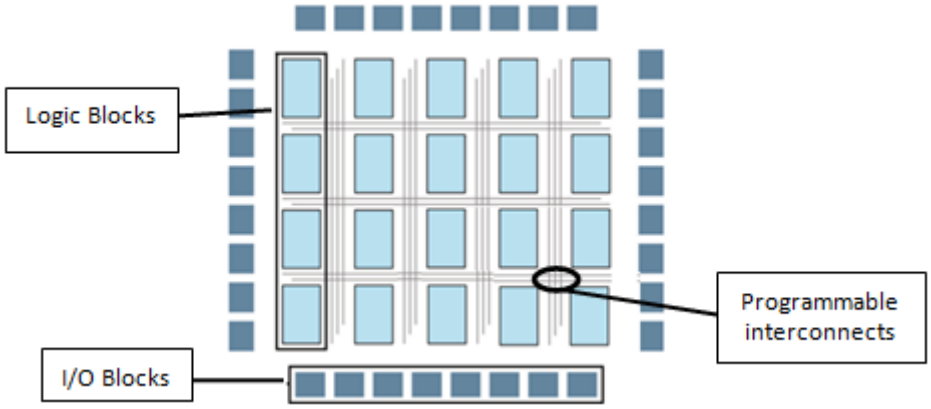
3.Tersedia solusi yang mendukung chip customized VLSI :

Mampu menimplementasikan logic circuit, instant manufacturring, very-low cost prototype

Pemrograman yang singkat untuk fungsi dan kemampuan yang setara dengan ASIC

FPGA ini sendiri juga merupakan IC tipe HDL yang mana pemakai dapat mendesain hardware sesuai yang diinginkan di dalam IC FPGA. Perkembangan FPGA pada saat ini berlangsung dengan cepat. Terdapat bermacam-macam keluarga FPGA dengan kebutuhan perancangan dan perangkat perancangan (design tools) yang berbeda. Xilinx merupakan salah satu perusahaan yang memproduksi FPGA disamping perusahaan lain yang juga memproduksi FPGA seperti ACTEL dan PLESSEY Semiconductor. Xilinx sendiri memproduksi beberapa jenis FPGA yaitu VIRTEX, SPARTAN, XC3000, XC4000 dan XC5000.

Secara umum arsitektur bagian dalam dari IC FPGA terdiri atas tiga elemen utama yaitu : *Input/Output Blok (IOB), Configurable Logic Block (CLB)* dan*Programmable Interconnect.* Fungsi logika dan interkoneksi FPGA ditentukan oleh data yang tersimpan pada sel memori statik internal. Ada beberapa cara untuk membuat data konfigurasi ke dalam sel memori ini, baik yang dilakukan secara otomatis pada waktu catu daya diberikan maupun dengan membaca konfigurasi data dari eksternal Serial atau byte Paralel PROM (mode master) atau Data dapat dituliskan pada FPGA dari eksternal device (mode slave dan peripheral).



Gambar 1. 1 *Struktur Xilinx FPGA*

Proses implementasi pada FPGA :

1. FPGAs diimplementasikan dengan *efficient CAD systems*
2. Design Entry dapat dilakukan dengan cara
   a. Schematic capture program,
   b. VHDL / Verilog
3. Logic Optimization
   a. Memodifikasi Boolean Expression untuk mengooptimalkan penggunaan
   b. Logic berbasis area dan kecepatan atau kombinasi keduanya
4. Technology Mapping
   a. Transformasi dari Boolean Expression atau VHDL kedalam bentuk circuit padaFPGA logic blocks
   b. Area optimization – meminimalkan penggunaan block
   c. Delay optimization – meminimalkan jumlag stage pada critical path
5. Placement
   a. Setelah memetakan rangkaian ke dalam blok logika, selanjutnya algoritma penempatan digunakan untuk meletakan masing-masing blok ke dalam FPGA array.
   b. Meminimalkan total panjang interkoneksi yang diperlukan untuk penempatan blok logika.
6. Routing software
   a. Mengatur FPGA's wire segments dan menentukan programmable switches untuk menghubungkan antar FPGA blocks
   b. Memastikan 100% koneksi telah terbentuk
   c. Meminimalkan propagation delay pada *time-critical connections.*
7. Programming unit
   a. Mengkonfigurasi chip setelah placement & routing step tuntas.

b. Keseluruhan proses memakan waktu antara beberapa menit sampai beberapa jam.

Secara umum proses kerja FPGA adalah sebagai berikut :

1. **Design Entry** dengan cara schematic, ABEL, VHDL, dan/atau Verilog.
2. **Implementation** yaitu Placement & Routing dan bitstream generation. Juga, analyze timing, view layout,
3. **Download** meng-upload bitstream ke Hardware (FPGA chip)

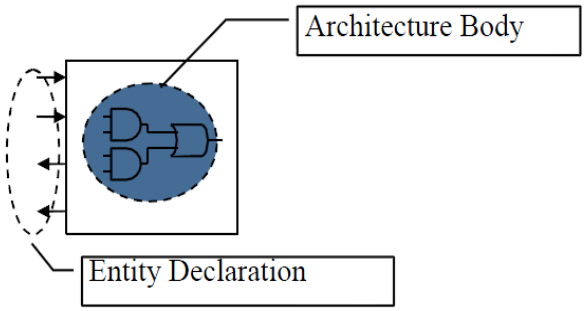
VHDL adalah singkatan ***V*** *HSIC (Very High Speed Integrated Circuits)* ***H****ardware* ***D****escription* ***L****anguage* . Pada pertengahan tahun 1980-an Departemen Pertahanan AS dan IEEE mensponsori perkembangan bahasa deskripsi perangkat keras dengan tujuan untuk mengembangkan sirkuit terintegrasi dengan kecepatan tinggi. Hal ini telah menjadi salah satu bahasa standar industri yang digunakan untuk menggambarkan sistem digital. Yang banyak digunakan hardware lainnya adalah bahasa deskripsi Verilog. Keduanya adalah bahasa yang kuat yang memungkinkan Anda untuk menggambarkan dan mensimulasikan sistem digital yang kompleks. Sebuah bahasa HDL ketiga adalah ABEL (*Advanced Boolean Equation Language*) yang khusus dirancang untuk Programmable Logic Devices (PLD). ABEL kurang kuat dibandingkan dengan dua bahasa lain dan kurang populer di industri. VHDL mengunakan standar IEEE 1.076-1.993.

Meskipun bahasa terlihat sama seperti bahasa pemrograman konvensional, ada beberapa perbedaan penting. Sebuah bahasa deskripsi hardware secara inheren paralel, yaitu perintah yang sesuai dengan gerbang logika, yang dieksekusi (computed) secara paralel, begitu masukan baru tiba. Sebuah program HDL meniru perilaku sistem fisik, biasanya digital. Hal ini juga memungkinkan penggabungan spesifikasi waktu (delay gate) serta untuk menggambarkan sistem sebagai interkoneksi dari komponen yang berbeda.

Saat ini para desainer menggunakan *hardware description languages (HDLs)* untuk mendesain system. Bahasa yang umum dipakai adalah HDL atau VHDL dan Verilog. Kesemua bahasa membolehkan user untuk mendesain system digital dengan menuliskan behavior dari rangkaian digital. Pemrogramn ini juga dapat digunakan untuk mensimulasikan operasi rangkaian dan men-*synthesize*-kan implementasi secara nyata dari rangkaian atau membuat aplikasi IC yang spesifik. Pada modul ini, pemrograman yang dipakai adalah pemrograman *VHDL*. *VHDL*mirip dengan bahasa pemrograman komputer biasa, seperti C ++. Sebagai contoh,memiliki konstruksi untuk tugas variabel, pernyataan bersyarat, loop, dan fungsi.Dalam bahasa pemrograman komputer, compiler digunakan untuk menerjemahkan kode sumber tingkat tinggi ke kode mesin. DiVHDL, bagaimanapun, synthesizer yang digunakan untuk menerjemahkan kode sumber untuk deskripsi dari sirkuit hardware sebenarnya yangmengimplementasikan kode. Dari uraian ini, yang kita sebut

netlist, perangkat digital fisik dapat dibuat secara otomatis.Keakurasian kode juga dapat di tes menggunakan simulasi.

VHDL adalah bahasa pemrograman yang digunakan untuk mendeskripsikan *logic circuit* yang dikehendaki.Secara umum struktur dari pemrograman *VHDL* terdiri atas dua bagian yaitu bagian ***ENTITY***dan bagian***ARCHITECTURE*.**

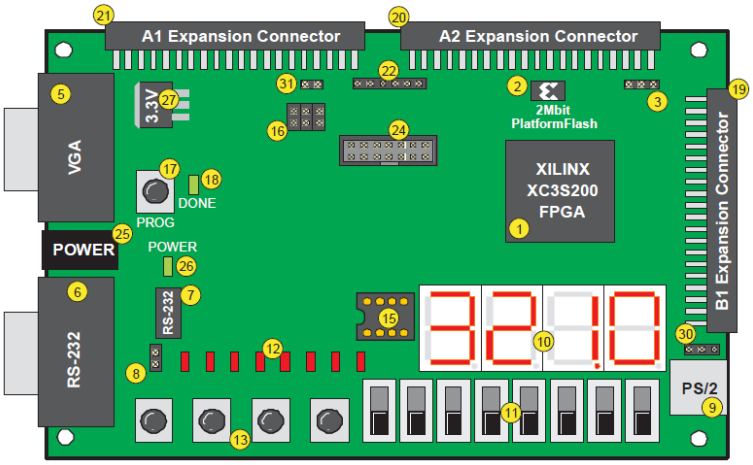


Gambar 1. 2 *Struktur Pemrograman VHDL*

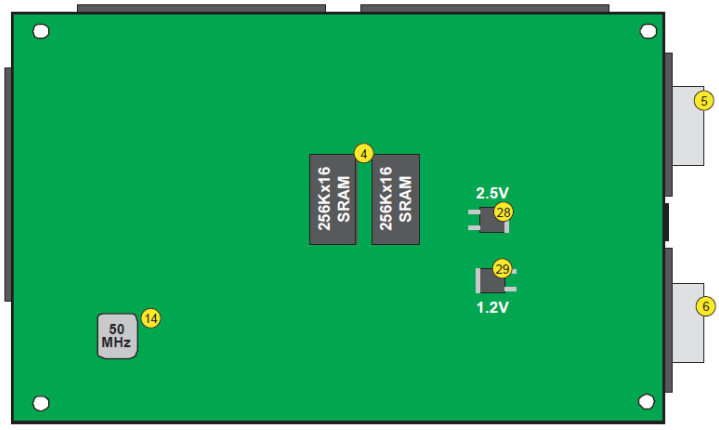
Bagian ***ENTITY*** menjelaskan spesifikasi pin-pin eksternal yang digunakan dari circuit atau rancangan yang akan dibuat.Bagian **ARCHITECTURE** menjelaskan atau mewakili fungsi sesungguhnya dari circuit atau rangkaian.

b. XILINX SPARTAN 3

Modul yang digunakan untuk mendesain chip adalah modul Xilinx Spartan 3. Fitur-fitur yang disediakan, seperti pada gambar 1.3 dan gambar 1.4.



Gambar 1. 3 *Modul Xilinx Spartan 3 tampak atas*



Gambar 1. 4 Modul Xilinx Spartan 3 tampak bawah

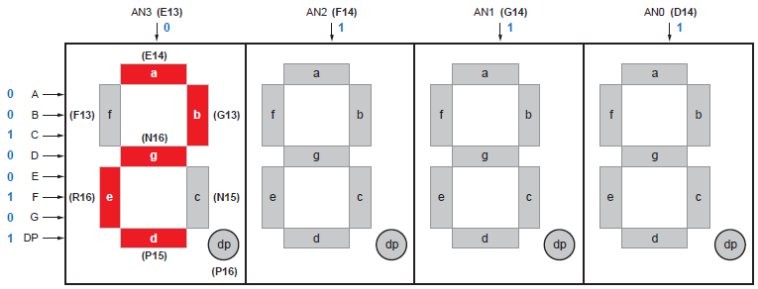
Secara ringkas fitur-fitur yang dimiliki oleh Xilinx Spartan 3 adalah :

1. 200,000-gerbang Xilinx Spartan-3XC3S200 FPGA in a 256-ball thin Ball Grid Array package (XC3S200FT256).
2. 2Mbit Xilinx XCF02S Platform Flash, in-system programmable configuration PROM.
3. Jumper options allow FPGA application to read PROM data or FPGA configuration from other sources
4. 1M-byte of Fast Asynchronous SRAM.
5. 3-bit, 8-color VGA display port.
6. 9-pin RS-232 Serial Port.
7. RS-232 transceiver/level translator
8. Second RS-232 transmit and receive channel available on board test points
9. PS/2-style mouse/keyboard port
10. 4 karakter, seven-segment LED display
11. 8 slide switches
12. 8 LED outputs
13. 4 push button
14. 50 MHz crystal oscillator clock source
15. Socket for an auxiliary crystal oscillator clock source
16. FPGA configuration mode selected via jumper settings
17. Push button switch to force FPGA reconfiguration (FPGA configuration happens automatically at power-on)
18. Indikator LED ketika FPGA sukses dikonfigurasi
19. 40-pin expansion connection ports to extend and enhance
20. 40-pin expansion connection ports to extend and enhance
21. 40-pin expansion connection ports to extend and enhance

22. JTAG port for low-cost download cable.
23. Digilent JTAG download/debugging cable connects to PC parallel port
24. JTAG download/debug port compatible with the Xilinx Parallel Cable IV and MultiPRO Desktop Tool
25. AC power adapter input for included international unregulated +5V power supply.
26. Power-on indicator LED
27. On-board 3.3V regulator
28. 2.5V regulator
29. 1.2V regulator.

FPGA koneksi ke Seven Segment.

Pada Seven Segment ada 8 jalur kontrol sinyal yang digunakan untuk menghidupkan led (untuk kombinasi angka tertentu). Dan setiap satu seven segment memiliki satu jalur control common anoda sebagai Enable. Detail dari jalur tersebut seperti pada gambar 1.5.



Gambar 1. 5 *Koneksi FPGA ke Seven Segment pada modul*

Untuk tabel pin FPGA dengan Seven segment ditunjukkan pada tabel 1.1

Tabel 1. 1 *FPGA Koneksi port ke Seven Segment (Aktif Low)*

Segment	FPGA Pin
A	E14
B	G13
C	N15
D	P15
E	R16
F	F13
G	N16
DP	P16

Tabel 1. 2 *Sinyal Common Enable Seven Segment (Kontrol Anoda)*

Anode Control	AN3	AN2	AN1	AN0
FPGA Pin	E13	F14	G14	D14

VI

FPGA koneksi dengan Switch

Pada board Spartan 3, tombol Switch dilambangkan dengan SW. Misal Switch ke-1 ditulis dengan SW1. Di modul ini switch yang dimiliki sebanyak 8 buah (SW7-SW0).

Tabel 1. 3 *Koneksi FPGA dengan Switch*

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA pin	K13	K14	J13	J14	H13	H14	G12	F12

Ketika SW pada posisi UP atau posisi ON, pin FPGA terhubung ke VCC/Logika High. Sedangkan jika SW pada posisi Down atau posisi OFF, pin FPGA terhubung ke Ground/ Logika Low.

FPGA koneksi dengan Push Button

Pada board Spartan 3 memiliki 4 tombol push button yang dituliskan pada board dengan BTN. (BTN3 -BTN0).

Tabel 1. 4 *Koneksi FPGA ke Push Button*

Push Button	BTN3	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

Ketika menekan push button, pin FPGA terhubung dengan VCC.

FPGA koneksi dengan LED

Pada board Spartan 3 memiliki 8 led yang dituliskan pada board dengan LD (LD7-LD0).

Tabel 1.5 *Koneksi Pin FPGA ke Led*

Led	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
FPGA Pin	P11	P12	N12	P13	N14	L12	P14	K12

Katoda dari Led terhubung ke Ground dengan Resistor 270Ω.

Catatan :

1. *Pada board Spartan 3 juga sudah tertuliskan alamat pin-pin dari FPGA beserta koneksinya, seperti pada Switch, Led dll.*
2. *Untuk datasheet lebih lengkap, baca* ***Spartan 3 Starter Kit Board user guide.***

c. Xilinx ISE Design Suite 13.1

Xilinx ISE (Integrated Synthesis Environment) adalah sebuah perangkat lunak yang diproduksi oleh Xilinx untuk sintesis dan analisa desain HDL, yang memungkinkan untuk *compile* desain, analisa waktu, RTL diagram, dan mengkonfigurasi perangkat target/*device* dengan programmer.
Xilinx ISE digunakan hanya untuk produk FPGA dari Xilinx. Xilinx ISE digunakan untuk mensintesa rangkaian dan desain sedangkan ModelSim logic Simulator digunakan untuk pengujian system level testing. Juga ada fitur tambahan termasuk program lain seperti Embedded Development Kit (EDK), Software Development Kit (SDK) dan ChipScope Pro.

Gambar 1. 6 *ISE Project Navigator*

3. Alat dan Bahan

Alat dan Bahan untuk praktikum ini adalah :

1. PC beserta software ISE Design Suite 13.1
2. Modul Xilinx Spartan 3
3. Downloader JTAG
4. Power Suplly DC 5V

4. Langkah percobaan

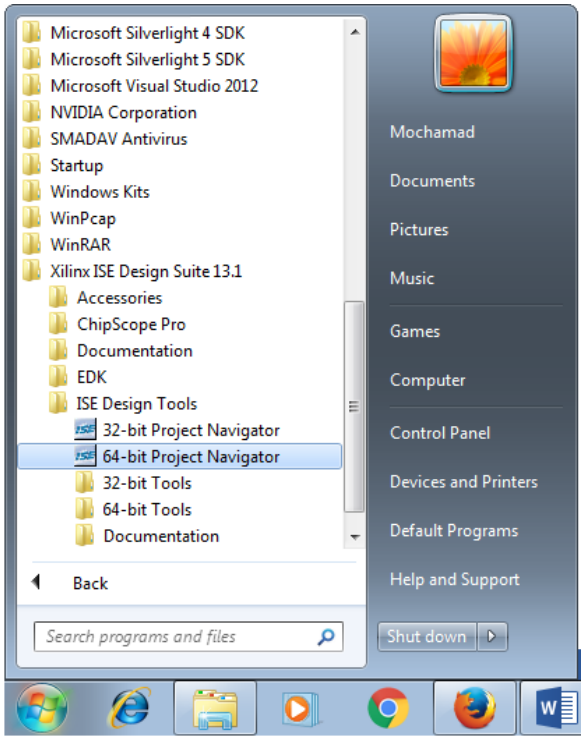
Rangkailah Modul Xilinx Spartan 3 dengan menghubungkan power Xilinx ke 5VDC. Kemudian pasanglah downloader USB untuk koneksi PC dengan chip pada Board. Posisi jumper pada posisi default (program tidak tersimpan dalam ROM).

Gambar 1. 7 *Modul Xilinx Spartan 3 dengan Downloader JTAG USB*

A. Membuat *New Project* pada ISE Design Suite 13.1

Pada contoh ini, akan dibuat sebuah program yang mengimplementasikan rangkaian AND.

1. Setelah tahap persiapan selesai, buka Start Window di PC Anda -> All Program -> ISE Design Suit 13.1 kemudian pilih Xilinx ISE Design Tools -> 32/64-bit Project Navigator.



Gambar 1. 8 *Start-All Program-Xilinx ISE 13-64bit Project Navigator*

2. Setelah dipilih, ISE Project Navigator akan menampilkan Window Project.

Pada tampilan ISE Project Navigator tersedia menu-menu seperti :

ix

a. Menu Bar

Menu Bar terdiri atas File, Edit, View, Project, Source, Process, Tool, Window, Layout dan Help.

b. Tool Bar

Tool Bar adalah komponen-komponen yang disediakan untuk mempercepat kerja user ketika sedang membuat desain. Seperti Tool Save, Open Project, New Project, Cut dll.

c. Window Log Proses, Error dan Warning

Adalah window yang digunakan untuk melihat proses synthesize desain “compile” dari program

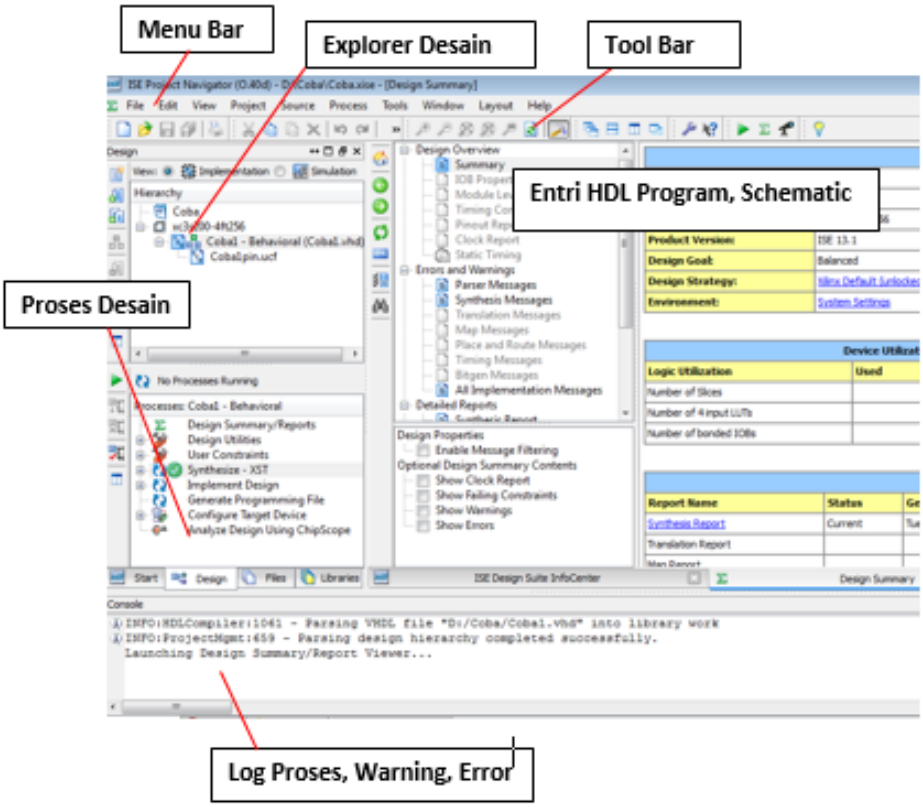
d. Window untuk tempat kita meng-entrykan program atau membuat skematik dari rangkaian.

e. Explorer Desain

Explorer Desain berisi tentang file-file apa saja yang sedang bekerja/sedang disunakan. Terdiri dari file.hdl, .ucf atau file top module.

f. Proses Desain

Adalah window dimana kita men-*synthesize*, mengimplementasikan, men-*generate* suatu file. File tersebut bisa berupa file program .hdl, .ucf atau file skematik.



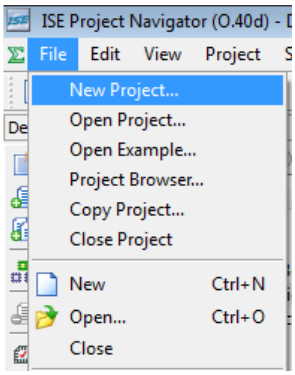
Gambar 1. 9 *Window Project ISE : tampilan awal*

x

Catatan : Pertama kali membuka biasanya tampilan ISE akan seperti pada gambar 1.9 yang memberitahukan tentang status project yang pernah dibuat sebelumnya (jika sebelumnya pernah membuat sebuah project). Untuk membuat project baru, pilih Menu ***File-Close Project*** *untuk menutup semua file. Sehingga project menjadi kosong.*

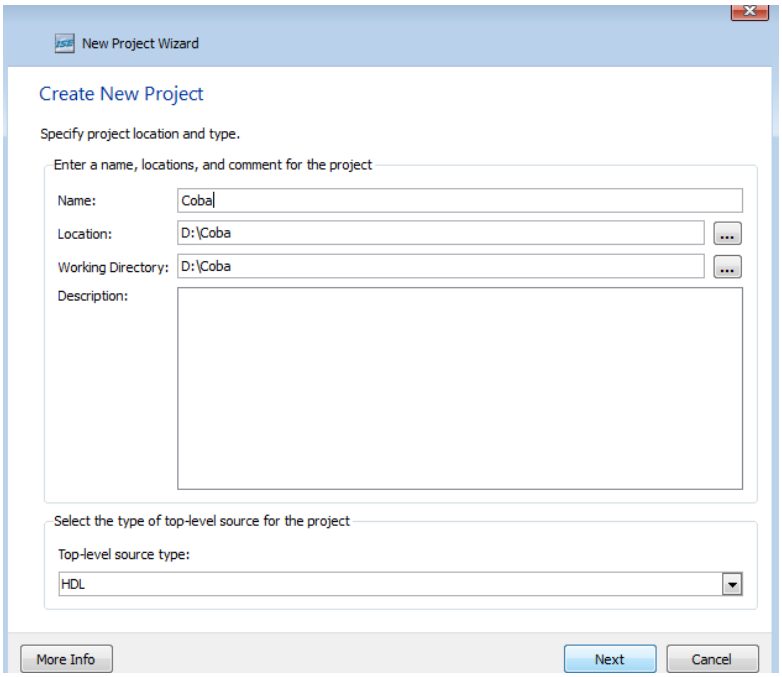
Gambar 1. 10 *Window Project :Close Project*

3. Buat project baru dengan klik ***File -> new project***



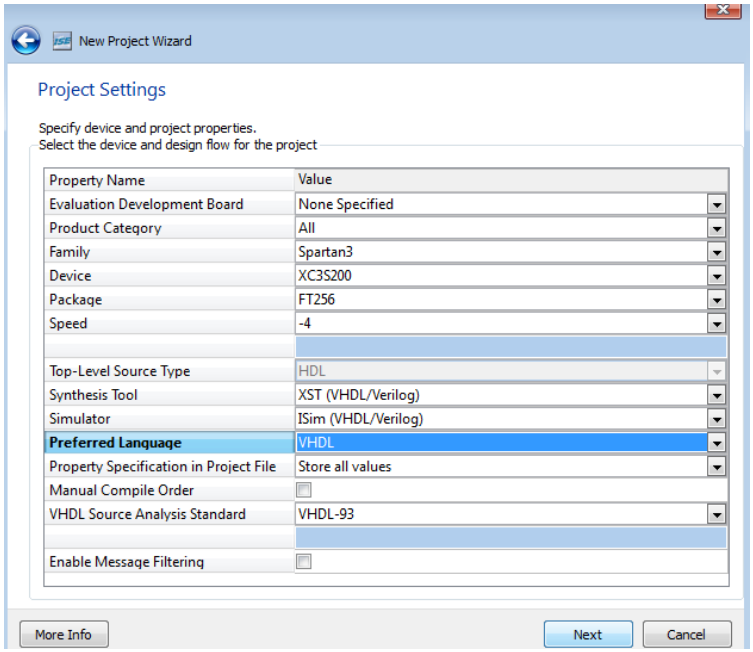
Gambar 1. 11 *Membuat Project Baru*

4. Isikan Nama Project, Lokasi dan Working Direktori anda. Kemudian pilih **HDL** pada Top-Level source type. Klik Next.



Gambar 1. 12 *Memberi nama project baru*

5. Pilih **Family= Spartan3, Device=XC3S200, Package = FT256, Speed =-4** (sesuai yang tertera di IC Spartan 3 pada Board). Pilih Preferred Language=VHDL. Kemudian klik Next.

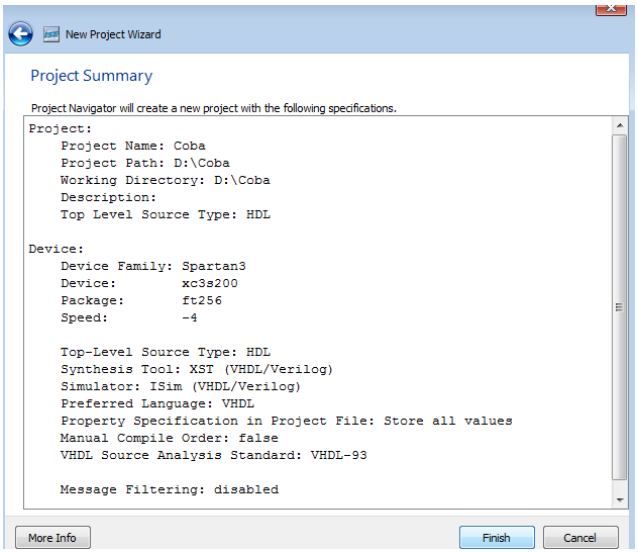


Gambar 1. 13 *Setting Project*

6. Setelah setting project, pilih Finish

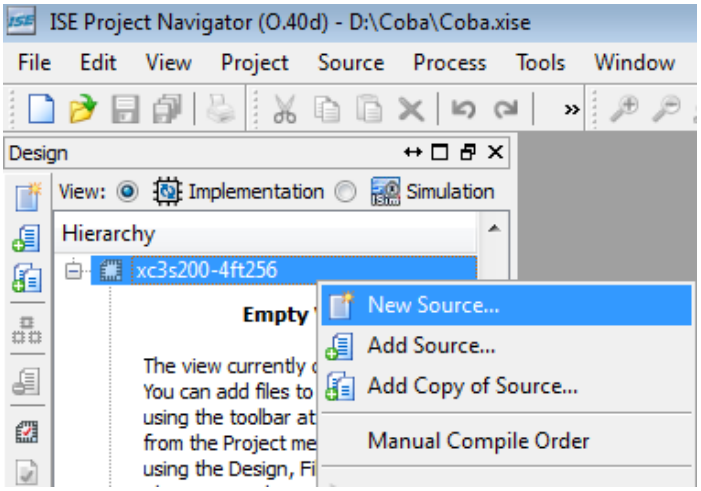
   Pada bagian ini diberitahukan tentang konfigurasi-konfigurasi project yang telah kita buat

yang berisi tentng penyimpanan file dan folder file juga tentang model chip IC xilinx yang dipakai. Jika sudah sesuai, kita bisa melanjutkan ke pemrograman dengan memilih **finish.** Jika konfigurasi project belum sesuai dengan apa yang kita inginkan, misalnya salah dalam memilih type IC, kita bisa membatalkan project ini dengan memilih **cancel.**



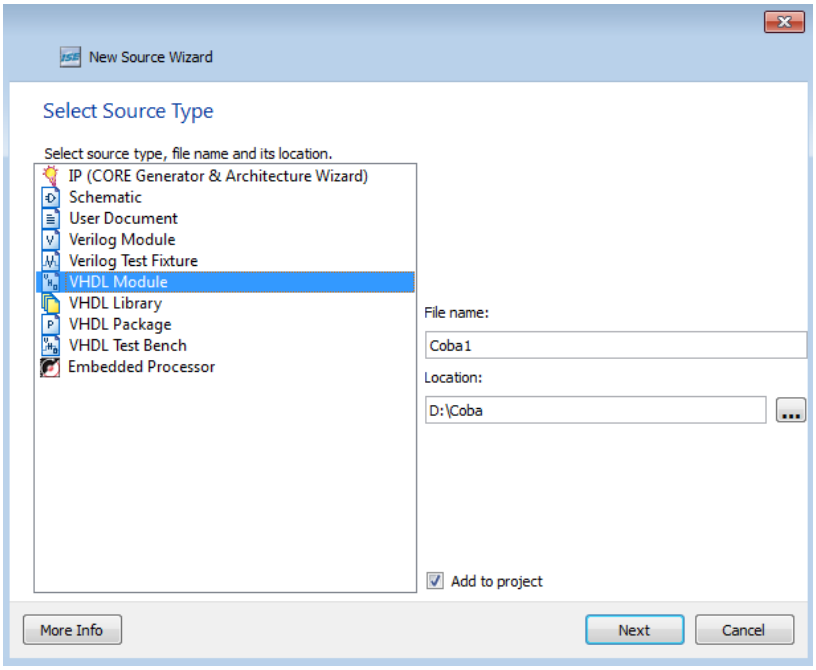
Gambar 1. 14 ***Selesai membuat project baru***

7. Pada kolom hierarchy, klik kanan pada nama project anda kemudian pilih new source untuk membuat file.hdl.



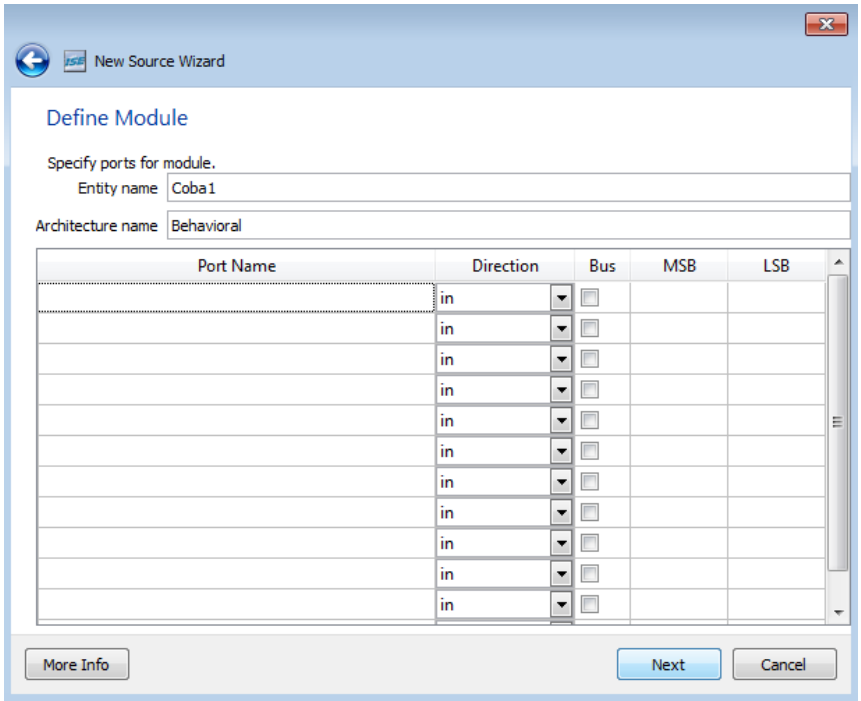
Gambar 1. 15 ***New Source untuk menambahkan file .hdl***

8. Pada New source wizard, pilih VHDL module pada source type, isikan nama file anda, misalkan Coba1. Location adalah lokasi tempat menyimpan file.vhdl tersebut (sudah dalm folder). Kemudian klik next.



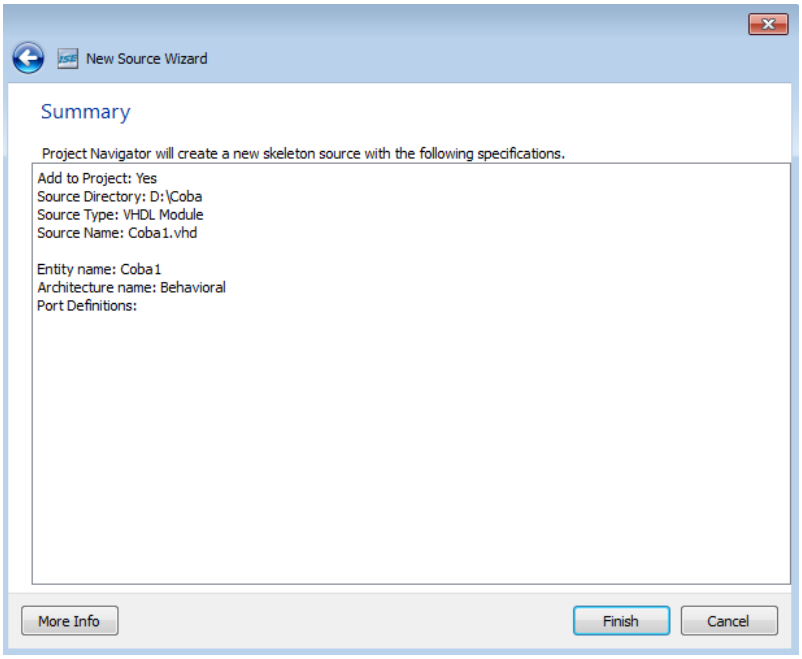
Gambar 1. 16 ***Membuat file program .vhdl***

9. Define module untuk mendefinisikan pin-pin yang digunakan, pada bagian ini langsung dilewati saja. Karena define pin akan kita konfigurasikan di akhir.Pilih **next**.



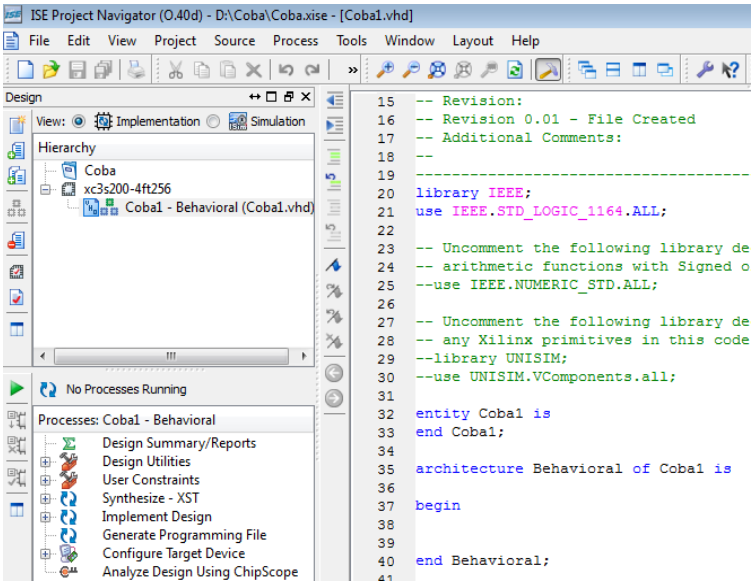
Gambar 1. 17 ***Define pin***

10. Jika konfigurasi sudah selesai, pilih **finish**



Gambar 1. 18 ***Selesai membuat source wizard***

11. Kemudian akan ditampilkan window template coding (Kode program kita akan diletakkan disini).



Gambar 1. 19 ***Tampilan Template Coding***

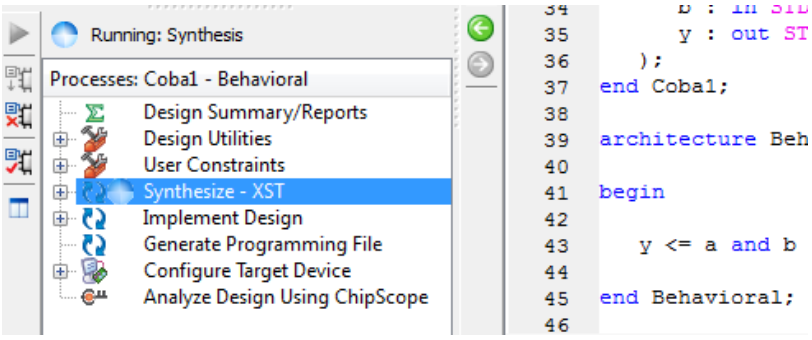
12. Lengkapi kode program pada gambar 1.17 seperti *script* program 1.1.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Coba1 is
port(
    a :in STD_LOGIC;
    b :in STD_LOGIC;
    y :out STD_LOGIC
```

```
);
end Coba1;

architecture Behavioral of Coba1 is
begin

y <= a and b

end Behavioral;
```

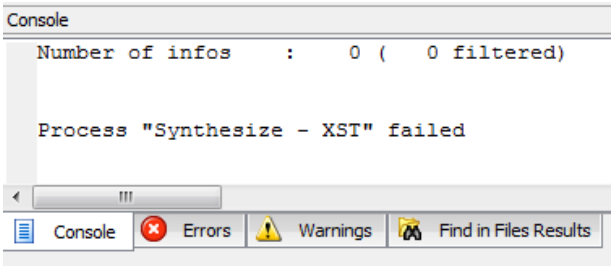
Script 1. 1 ***Kode program gerbang Logika AND***

13. Save project anda kemudian **double klik Synthesize-XST** untuk mensintesis program (compile program). Jika program anda benar maka akan muncul File Succed jika terdapat kesalahan akan mucul File failed atau Error.



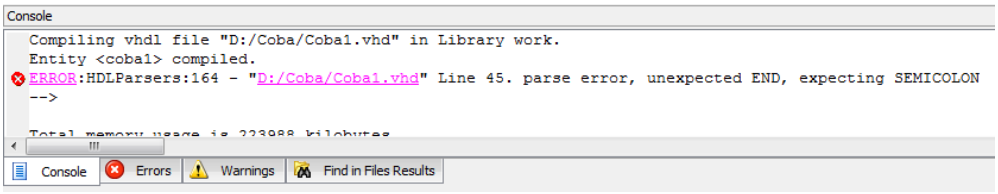
Gambar 1. 20 ***Proses Synthesize-XST***

Ketika selesai mensynthesize program, lihat pada window Log Proses. Jika terdapat Error maka akan muncul pesan Synthesize –XST seperti pada gambar 1.21.



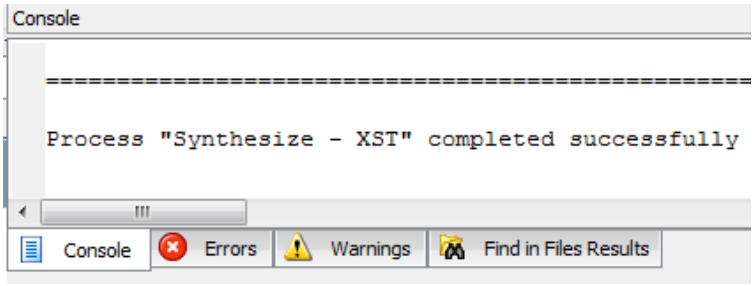
Gambar 1. 21 ***Proses Synthesize Error***

Untuk melihat error, scroll window log process keatas. Kemudian cari bagian keterangan Error. Error bisa terjdi karena salah penulisan kode program. Dalam keterangan tersebut, juga disebutkan letak baris kode program yang dinyatakan sebagai error. Dari keterangan ini, kita bisa membenarkan program yang salah dengan cepat. xvi



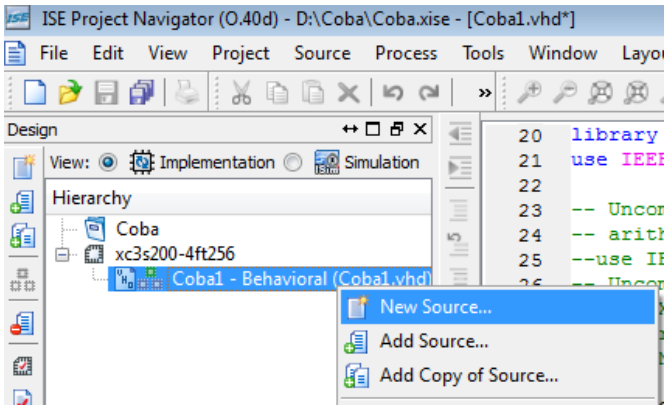
Gambar 1. 22 ***Pesan Error***

Jika program sudah dibenarkan, kemudian lakukan proses Synthesize lagi. Setelh proses synthesize selesai, lihat window Log Prosess. Apabila program tidak terdapat kesalahan, maka program akan menampilkan pesan ***Synthesize –XST completed successfully.***



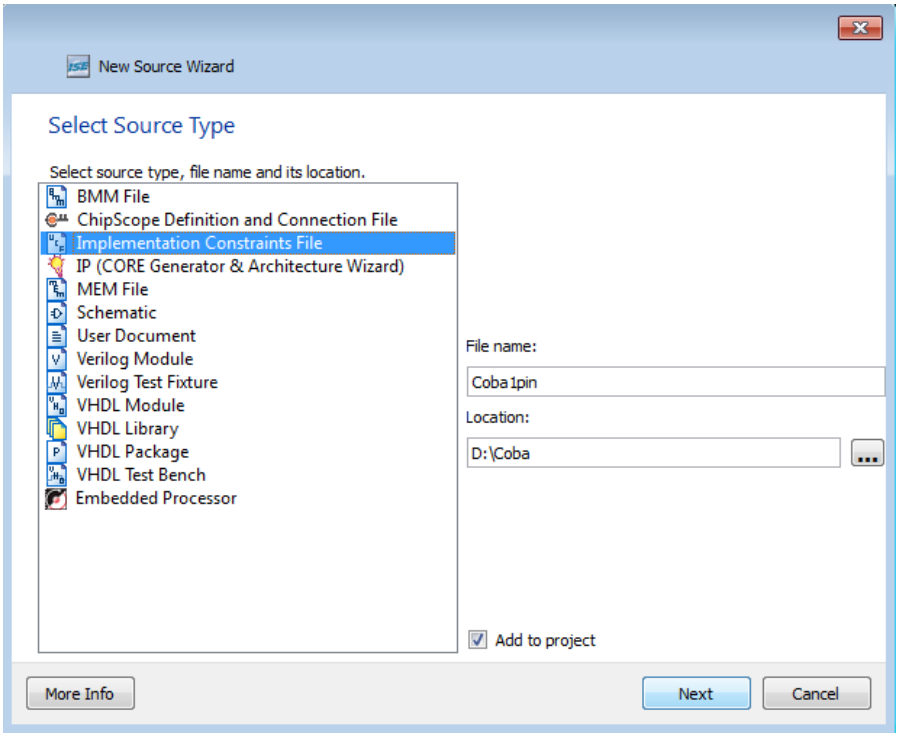
Gambar 1. 23 ***Synthesize successfully***

14. Kemudian klik kanan pada nama file project anda di kolom hierarchy, pilih new source.



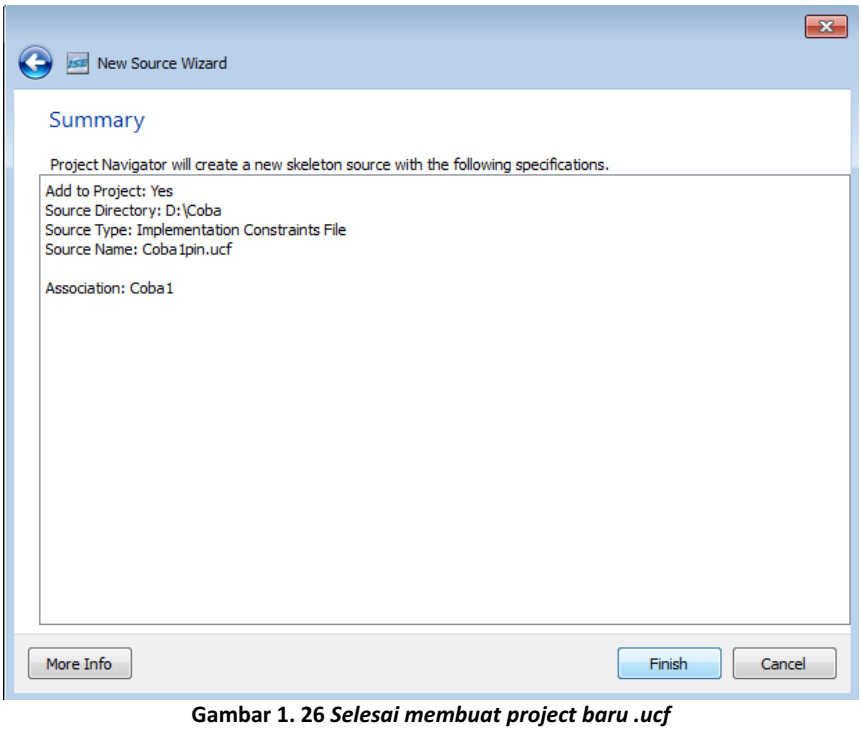
Gambar 1. 24 ***Menambahkan file baru***

15. Pilih Implementasi Constrain File untuk mendefinisikan pin-pin dari program pada Spartan 3. Kemudian isi nama file. (File ini akan berekstensi .ucf)



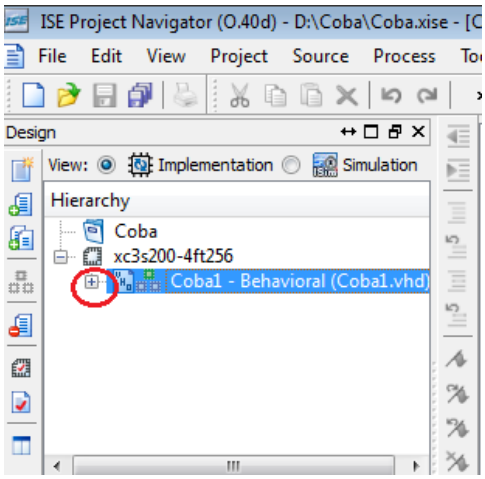
Gambar 1. 25 ***Membuat source file baru .ucf***

16. Setelah selesai, pilih **finish**



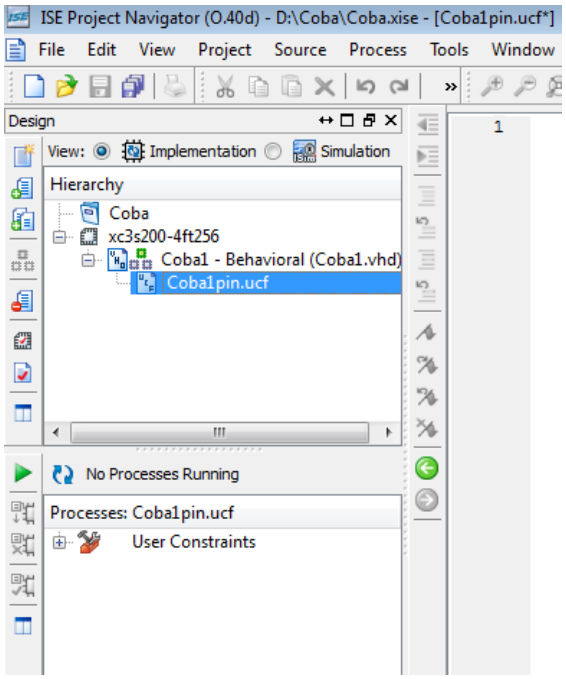
Gambar 1. 26 ***Selesai membuat project baru .ucf***

17. File .ucf yang telah kita buat berada pada sub program .vhdl. Untuk membuka klik tanda panah yang berbentuk tanda plus di project vhdl.



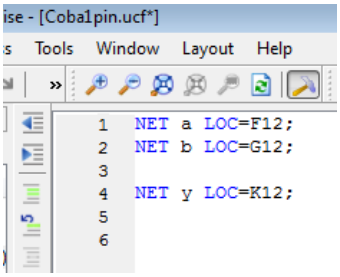
Gambar 1. 27 ***Letak file .ucf***

18. Klik file .ucf yang sudah kita buat, akan muncul template kosong di window sebelah kanan.



Gambar 1. 28 ***Template untuk mendefiniskan pin***

19. Definisikan pin-pin sesuai dengan board Spartan 3. Lihat datasheet pada modul Xilinx Spartan3.

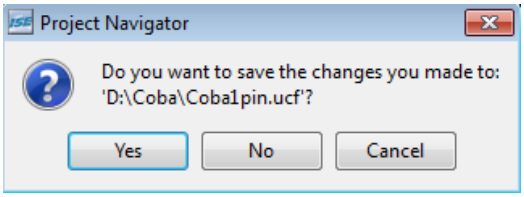


Gambar 1. 29 ***Penulisan Alamat pin***

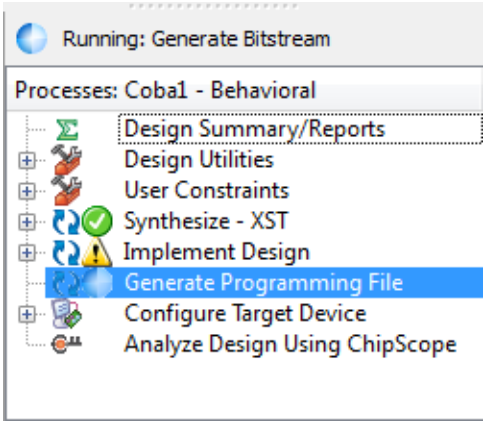
```
NET a LOC=F12;
NET b LOC=G12;
NET y LOC=K12;
```

Script 1. 2 ***Alamat pin***

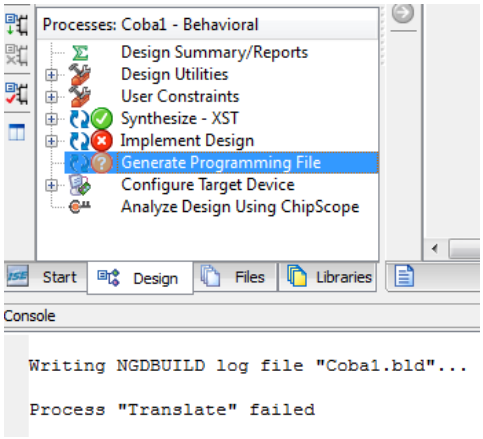
20. Setelah selesai, save project anda. Kemudian klik kembali file project .vhdl untuk menampilkan window process desain. Setelah selesai klik **Generate Programming File** pada kolom Processes. Jika centang hijau maka success dan program siap di downloadkan. Jika Ada tanda silang merah berarti program/konfigurasi pin ada yang salah.



Gambar 1. 30 ***Save Project***

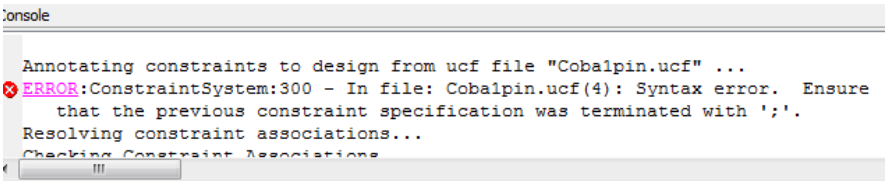


Gambar 1. 31 Process Running Generate Programming File



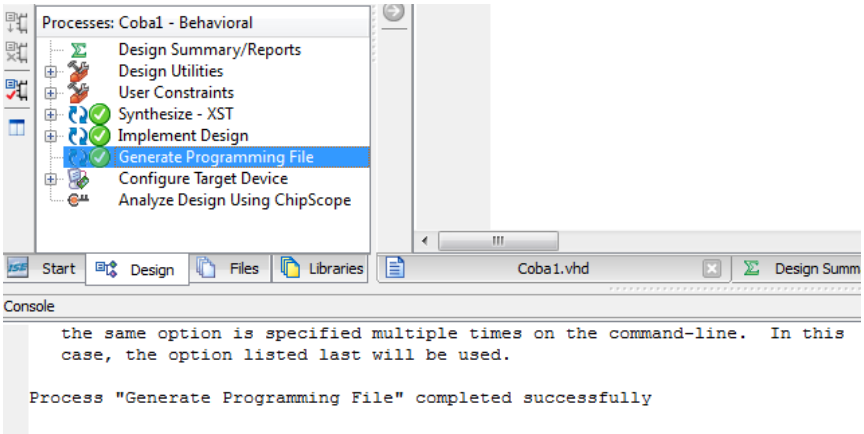
Gambar 1. 32 *Program failed*

Jika pada process generate file terdapat error, scroll keatas pada window log process. Kemudian lihat error yang terjadi. Error bisa terjadi karena kesalahan sintak dari program.



Gambar 1. 33 Pesan Error

Jika pada process generate file tidak terdapat pesan error, maka program success.

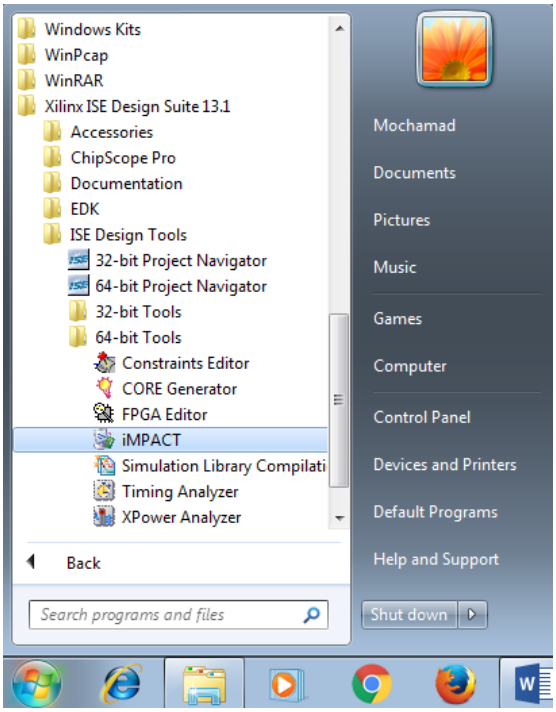


Gambar 1. 34 *Process Generate Success*

Sampai step ini pembuatan project telah selesai, kemudian file project akan di downloadkan ke IC FPGA Xilinx Spartan 3.

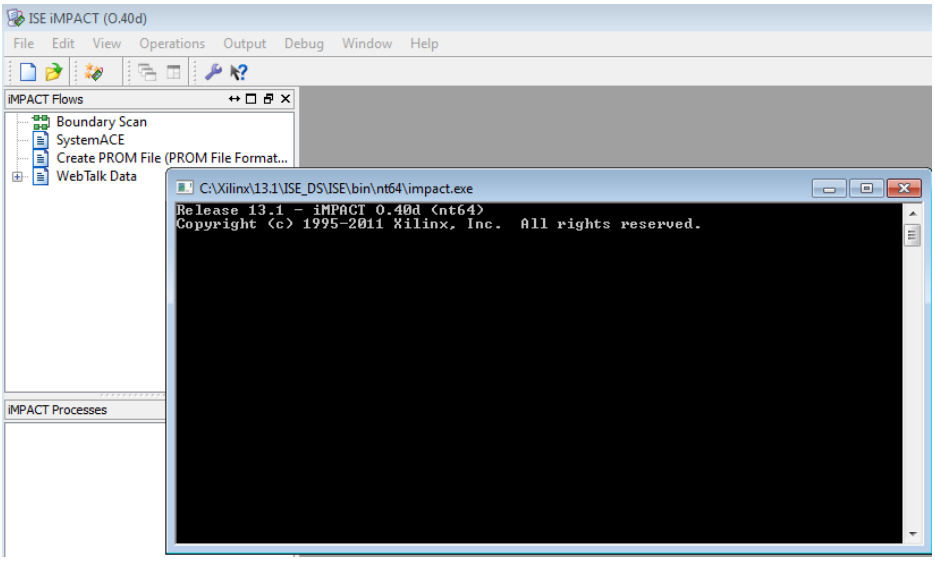
B. Download Program dari ISE ke Chip IC Xilinx Spartan 3

1. Pastikan bahwa downloader sudah terhubung antara PC- dengan board Spartan 3.
2. Pilih IMPACT di menu Start- All Program – ISE 13 - ISE DESIGN TOOLS – 64-bit Tools-iMPACT.



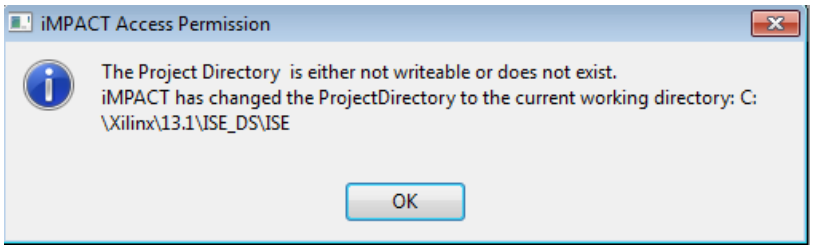
Gambar 1.33 iMPACT

3. Akan muncul beberapa window ketika proses membuka iMPACT, diikuti saja.



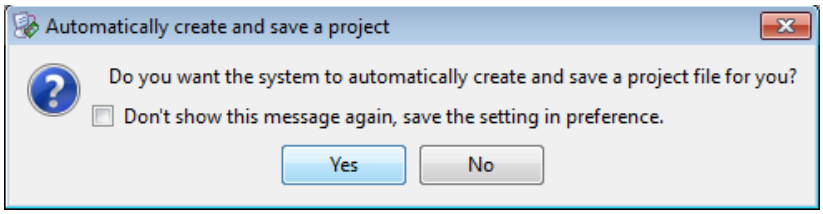
Gambar 1. 35 ***Proses membuka ISE iMPACT***

4. Jika iMPACT meminta ijin akses, pilih OK.



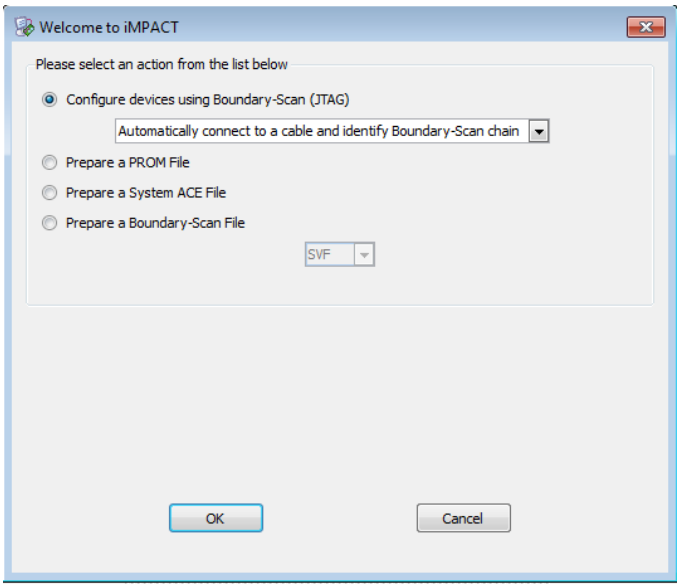
Gambar 1. 36 *iMPACT Access Permisssion*

5. Ketika ISE iMPACT sudah terbuka, iMPACT akan memberitahukan untuk secara otomatis membuat project. Pilih **Yes.**



Gambar 1. 37 *Automatically create and save a project*

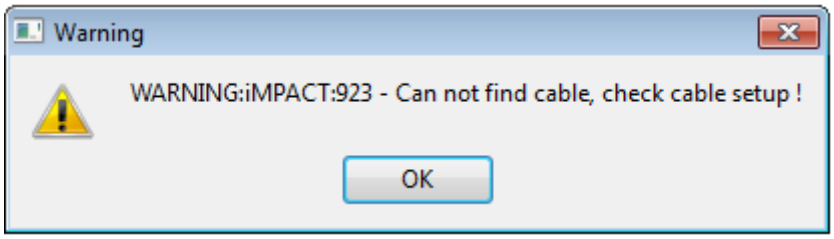
6. Setelah selesai, akan muncul window Welcome to iMPACT, pilih **Configure devices using Boundary-Scan (JTAG)** kemudian pilih **OK**



Gambar 1. 38 *Configure JTAG*

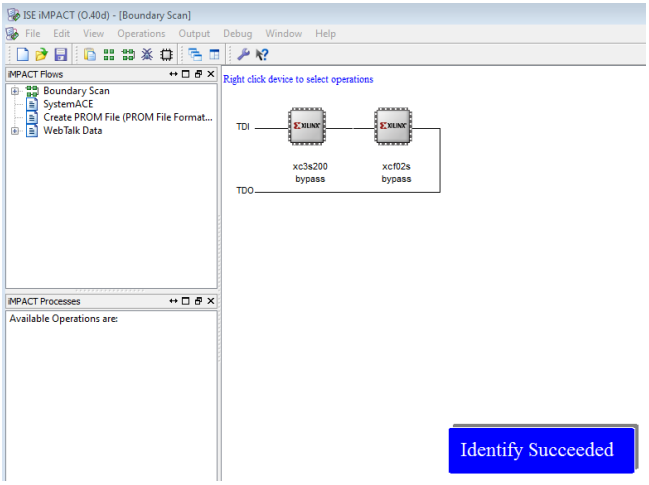
7. Setelah selesai, iMPACT akan mencek sudah tersambung/tidak downloader JTAG dengan PC. Jika downloader belum disambungkan dengan PC, maka akan keluar peringatan seperti

gambar 1.39.



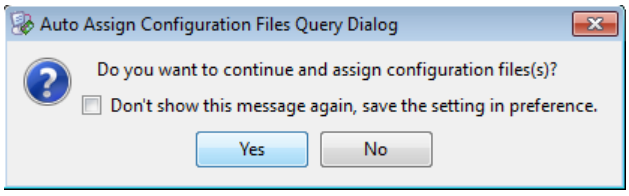
Gambar 1. 39 *Warning Cable Downloader belum terpasang*

8. Apabila downloader sudah tersambung, maka dalam window iMPACT akan muncul **IC Xilinx** dan terdapat pesan **Identify Succeeded**



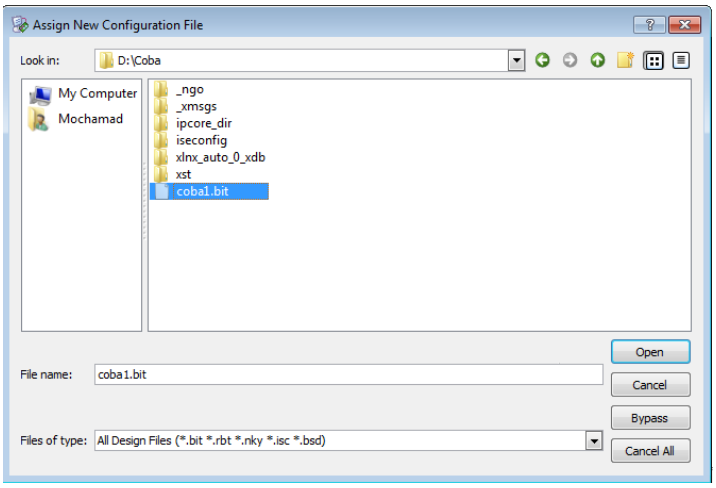
Gambar 1. 40 *PC sudah tersambung dengan Board Xilinx Spartan 3*

9. Setelah itu akan muncul window*Auto Assign Configuration Files Query Dialog*, pilih **Yes**



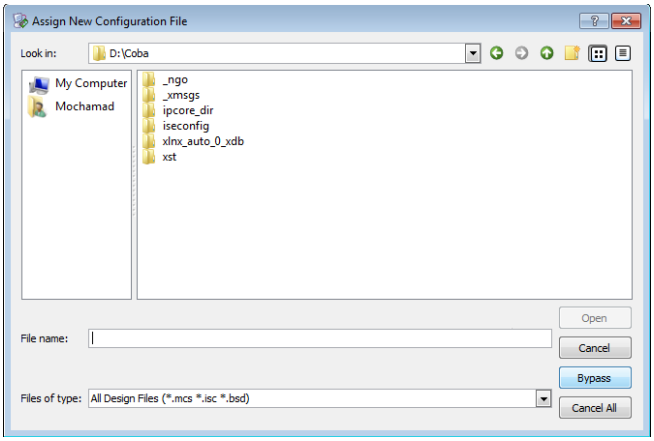
Gambar 1. 41 *Auto Assign Configuration Files Query Dialog*

10. Langkah selanjutnya adalah **Assign New Configuration File** yang meminta anda untuk memasukkan file yang akan di downloadkan, pilih file .bit di foleder anda dari program yang sudah anda buat. File yang didownloadkan ke IC XILINX SPARTAN 3 adalah file .bit. (catatan : *Ketika anda membuat project dari step 1 di dalam file project anda sudah tergenerate secara otomatis file .bit*). xxiv



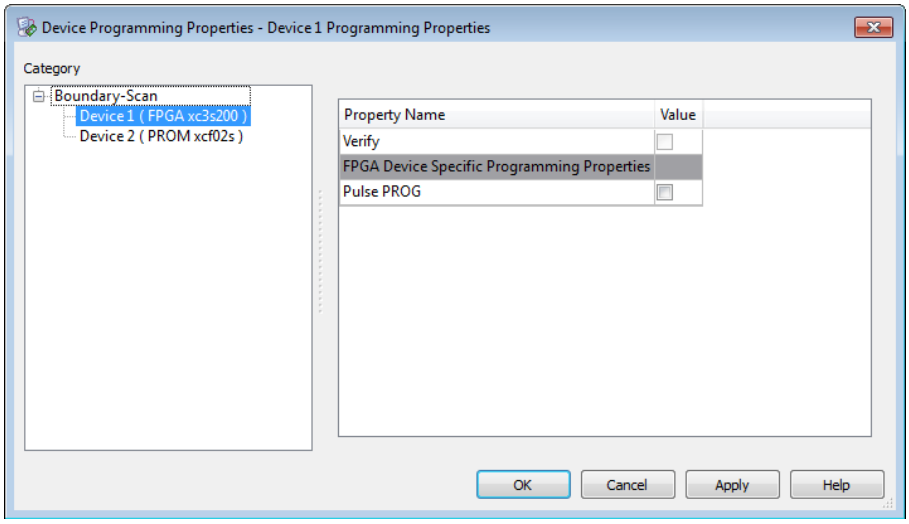
Gambar 1. 42 *Memasukkan file .bit*

11. Kemudian akan muncul perintah untuk memasukkan file .msc. Kita lewati saja dengan **Bypass.** (File .msc digunakan untuk memprogram menggunakan Flash/ disimpan dalam memory)



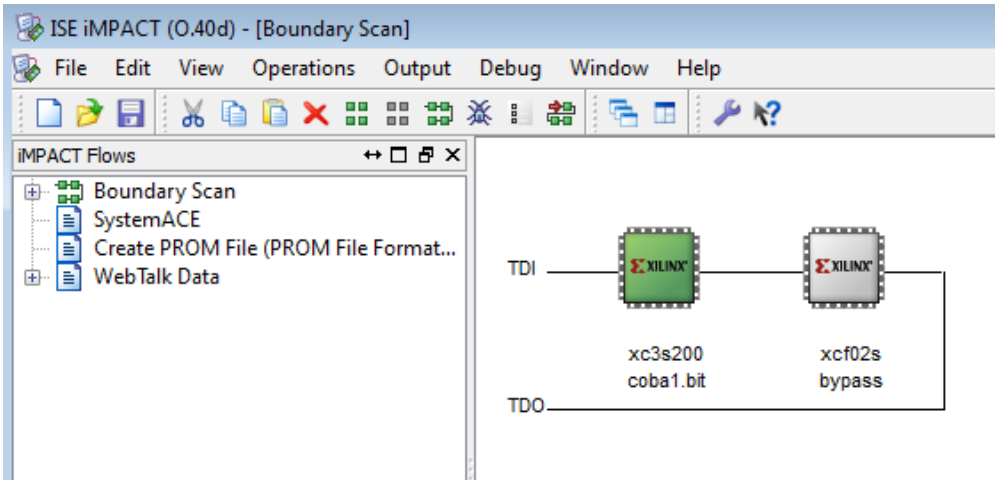
Gambar 1. 43 *Memasukkan file .msc (sementara dilewati saja)*

12. Setelah itu akan muncul window Device Programming Properties –Device 1 Programming Properties. Pilih **Device 1 (FPGA xc3s200)** pada kolom **Category**. Dan **Property Name : FPGA Device Specific Programming Properties.** Kemudian pilih **OK.**



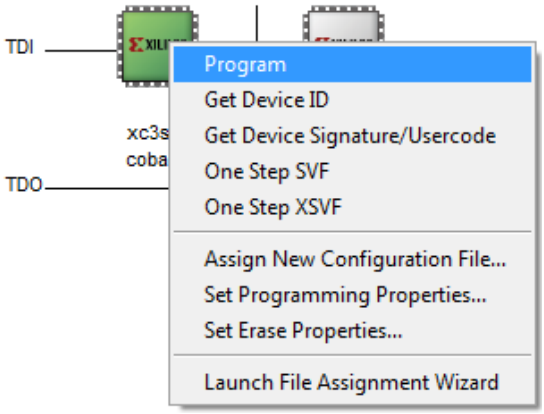
Gambar 1. 44 *Memilih device FPGA*

13. Kemudian akan tampil iMPACT dengan IC XIlinx dengan type xc3s200 yang sudah kita masukkan program file.bit.



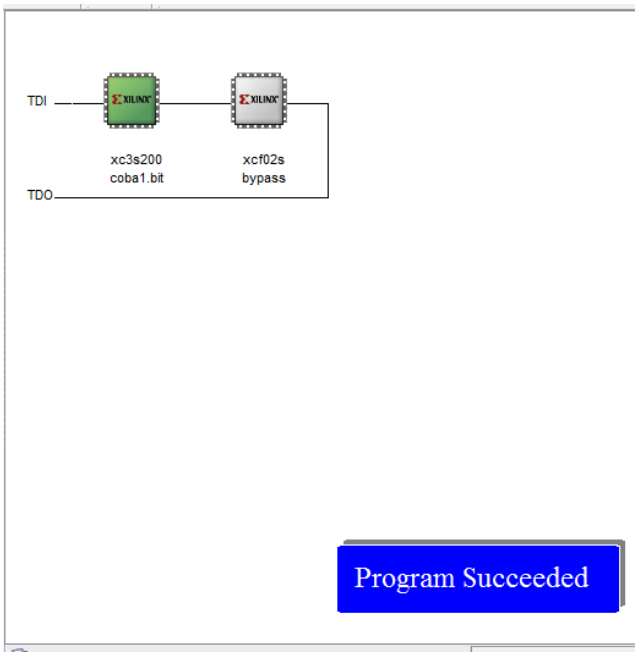
Gambar 1. 45 *IC FPGA pada iMPACT*

14. Untuk mendownloadkan program ke IC FPGA, **klik kanan** pada IC FPGA xc3s200 kemudian pilih **program**



Gambar 1. 46 *Mendownloadkan ke IC FPGA*

15. Setelah itu akan terjadi proses download, kemudian jika sudah selesai. Pada window iMPACT akan muncul pesan Program Succeeded



Gambar 1. 47 *Proses Download berhasil*

16. Setelah proses download berhasil, amati board Xilinx Spartan 3 anda. Apakah hasilnya sesuai dengan program anda?

5. Latihan

1. Buatlah sebuah program gerbang logika dasar dimana *input*-nya adalah tombol SW0 dan SW1

sedangkan *output*-nya adalah Led (LD0-LD4) dimana LD0 menampilkan hasil gerbang logika AND dari input SW0 dan SW1, LD1 untuk gerbang logika OR dari input SW0 dan SW1, LD2 untuk gerbang logika NOT dari input SW0, LD3 untuk gerbang logika NAND dari input SW0 dan SW1, LD4 untuk gerbang logika NORdari input SW0 dan SW1.

```
y[ 0] <= a and b;
y[ 1] <= a or b;
y[ 2] <= not a;
y[ 3] <= a nand b;
y[ 4] <= a nor b;
```

6. Bentuk dan Format Luaran

a. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.
b. Anatomi laporan : Judul, Nama, NRP, Kelas, *Screen capture* program VHDL, Foto tampilan LED dan Switch pada board Spartan, Analisa program.
c. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas1-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkan.

Referensi

1. 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.
2. *Spartan 3 Starter Kit Board User Guide*, Mei 2005. Xilinx.

Percobaan 2 – Isim Simulator

1. Tujuan

Mahasiswa dapat mendisain rangkaian dari persamaan Boolean.

Mahasiswa dapat membuat RTL dari rangkaian logika.

Mahasiswa dapat membuat simulasi testbench dari rangkaian logika.

2. Teori

Ketika sebuah rancangan rangkaian diimplementasikan ke dalam FPGA, baik menggunakan metode schematic atau metode HDL, hendaknya perlu diuji agar kita dapat mengetahui hasil/output dari rangkaian tersebut. Proses pengujian rancangan rangkaian ini sering disebut proses simulasi. Melalui proses simulasi, kita dapat mengetahui apakah hasil rancangan rangkaian yang sudah dibuat sudah sesuai dan memenuhi tujuan yang diinginkan atau belum. Proses ini biasanya dilakukan sebelum rancangan rangkaian diimplementasikan ke dalam FPGA. Jadi urutan sebagai berikut:

1. Pembuatan rancangan rangkaian (Metode *schematic* atau metode HDL).
2. Proses simulasi rancangan rangkaian.
3. Implementasi ke dalam FPGA.

Nantinya, proses simulasi berfungsi untuk mengetahui sekaligus menguji apakah rancangan rangkaian yang telah dibuat mampu berjalan dengan baik atau tidak. Selain itu, lewat proses simulasi dapat diketahui bagaimana output dari rancangan rangkaian tadi. Selanjutnya, proses simulasi membutuhkan suatu bentuk stimulus/pemicu. Stimulus ini akan bertindak sebagai input awal bagi rancangan rangkaian yang hendak diuji. Kemudian, setelah diberikan stimulus maka rancangan rangkaian tersebut dapat diketahui bagaimana hasil outputnya. Keseluruhan proses simulasi ini dilakukan dengan bantuan perangkat lunak (software) yang ada.

Pada umumnya, proses simulasi terbagi atas 2 bentuk yakni:

Bentuk *testbench*.

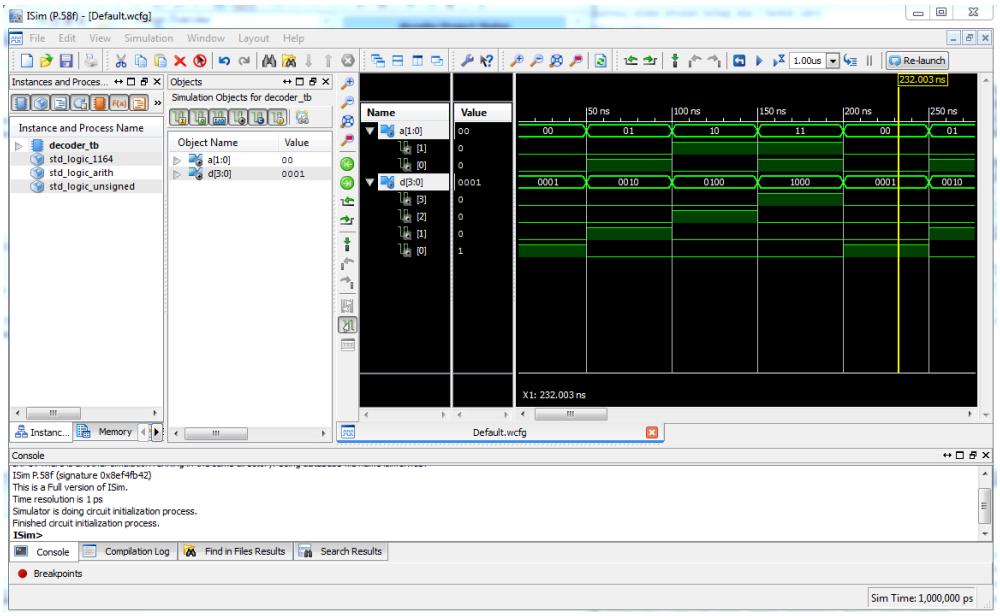
Bentuk *timing diagram*.

Timing Diagram

Proses simulasi dapat diketahui dengan membuat gambar timing diagram. Dengan menggunakan gambar timing diagram, dapat dilihat mengenai kondisi input, output serta hal-hal lain yang terkait. Bentuk *timing diagram* ini dibuat berdasarkan satuan waktu. Pada mulanya, untuk membuat timing diagram ini diperlukan stimulus untuk menentukan kondisi input awal. Selanjutnya, software

simulator akan melakukan simulai guna menghasilkan kondisi output sesuai dengan kondisi input tadi.

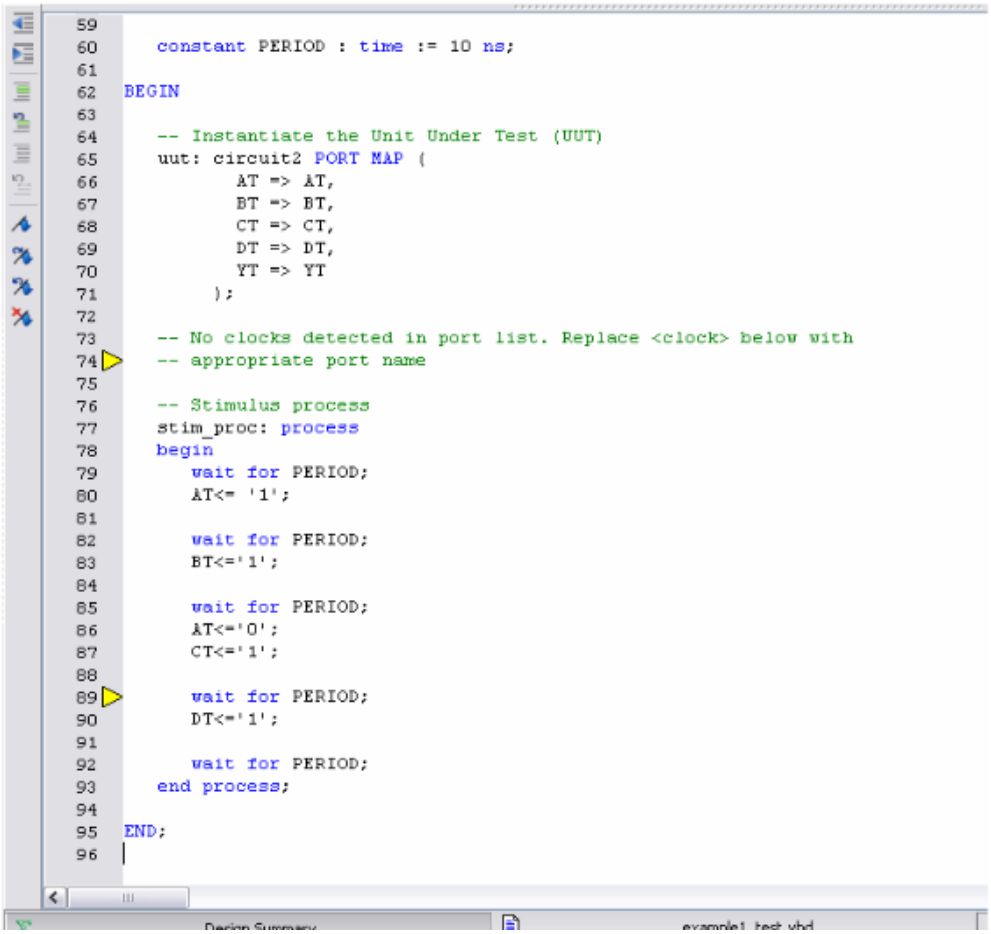
Cara ini merupakan cara yang sederhana dan mudah dilakukan, apalagi bagi para pegguna awal FPGA, bila dibandingkan dengan bentuk testbench. Hal ini disebabkan karena tampilan timing diagram yang berupa gambar sehingga memudahkan siapa saja untuk mengamati, menganalisa, dan menjelaskan proses simulasi.



Gambar 2. 1 Isim Simulator

Testbench

Selain cara pertama, dikenal pula bentuk simulasi yang lain yaitu *testbench*. **Testbench** adalah proses pengujian suatu rancangan rangkaian. Dalam proses perancangan rangkaian, testbench akan menguji design rangkaian apakah sudah sesuai atau belum. Testbench ini dilakukan dengan menggunaan file HDL (berbentuk kode, baik VHDL maupun Verilog). Sehingga berbeda dengan bentuk diagram yang menggunakan gambar sebagai tampilannya, testbench menggunakan kode-kode tulisan sebagai tampilannya. Berikut contoh testbench :



```
59
60      constant PERIOD : time := 10 ns;
61
62  BEGIN
63
64      -- Instantiate the Unit Under Test (UUT)
65      uut: circuit2 PORT MAP (
66             AT => AT,
67             BT => BT,
68             CT => CT,
69             DT => DT,
70             YT => YT
71           );
72
73      -- No clocks detected in port list. Replace <clock> below with
74      -- appropriate port name
75
76      -- Stimulus process
77      stim_proc: process
78      begin
79         wait for PERIOD;
80         AT<= '1';
81
82         wait for PERIOD;
83         BT<='1';
84
85         wait for PERIOD;
86         AT<='0';
87         CT<='1';
88
89         wait for PERIOD;
90         DT<='1';
91
92         wait for PERIOD;
93      end process;
94
95  END;
96
```

Gambar 2. 2 Test Bench

3. Alat dan Bahan

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

4. Langkah Percobaan

A. RTL dan Simulasi Test Bench

1. Sebelum melakukan test bench, buatlah sebuah program .vhdl misalnya membuat rangkaian logika AND.

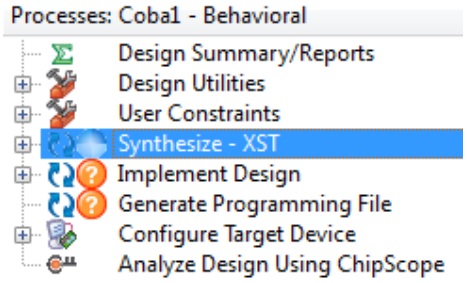
```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Coba1 is
       port(
              a :in STD_LOGIC;
              b :in STD_LOGIC;
              y :out STD_LOGIC
);
```

xxxi

```
end Coba1;

architecture Behavioral of Coba1 is
begin
        y <= a and b;
end Behavioral ;
```

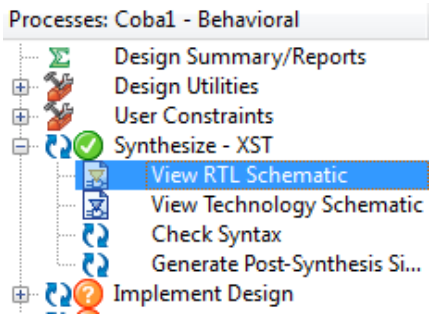
Script 2. 1 Rangkaian Logika AND

2. Kemudian lakukan Synthesize-XST.



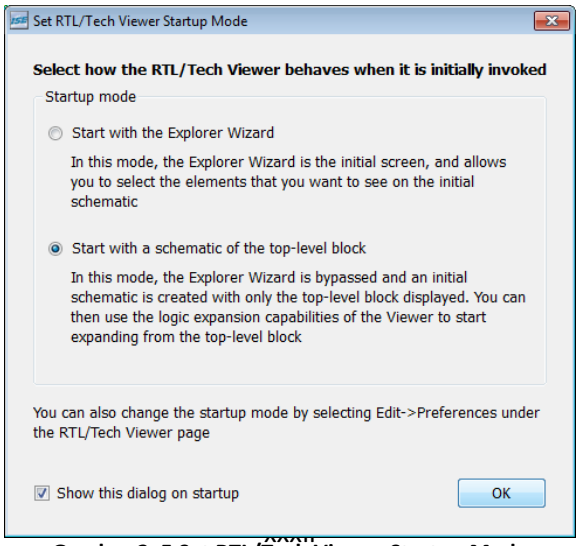
Gambar 2. 3 Proses Synthesize-XST

3. Klik 2x pada View RTL Simulation,



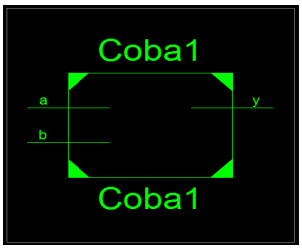
Gambar 2. 4 View RTL Simulation

4. kemudian pilih *Start with a schematic of the top-level block* pada window Set RTL/Tech Viewer Startup Mode kemudian OK.



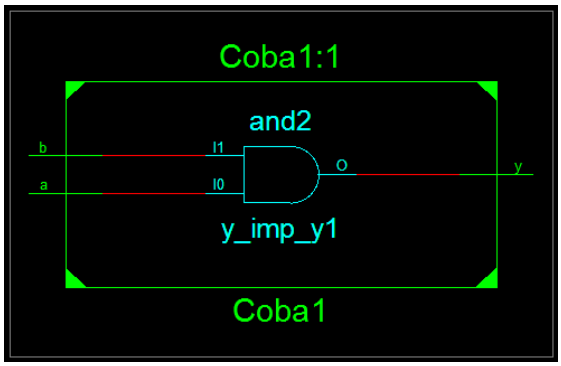
Gambar 2. 5 Set RTL/Tech Viewer Startup Mode

5. Akan muncul RTL skematik yaitu rangkaian yang sudah kita buat.



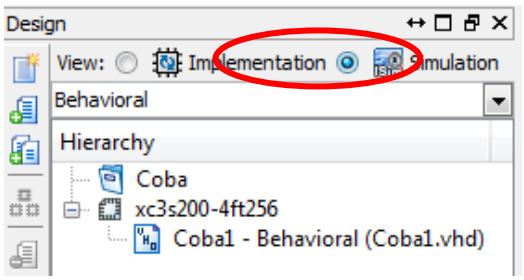
Gambar 2. 6 2.6 RTL Skematik

6. Klik 2x di dalam diagram block, maka akan muncul rangkaian logikanya



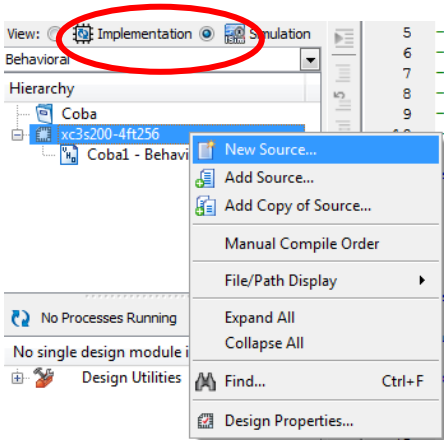
Gambar 2. 7 Rangkaian Logika AND

7. Kembali ke kode program, jika tidak terdapat error, pilih **Simulation** pada **View** di window kolom **Design**



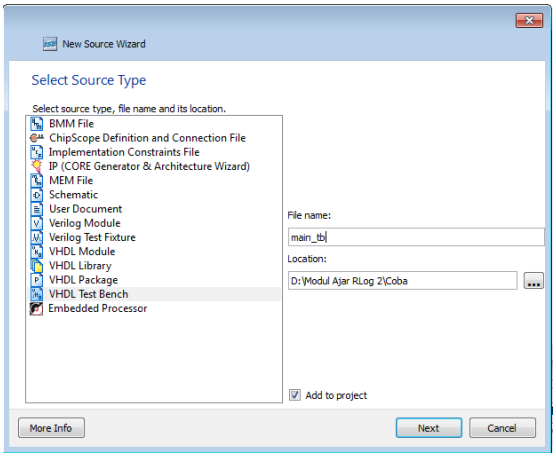
Gambar 2. 8 Simulation View

8. Kemudian buat project baru dengan memilih- New Source



Gambar 2. 9 Membuat New Source

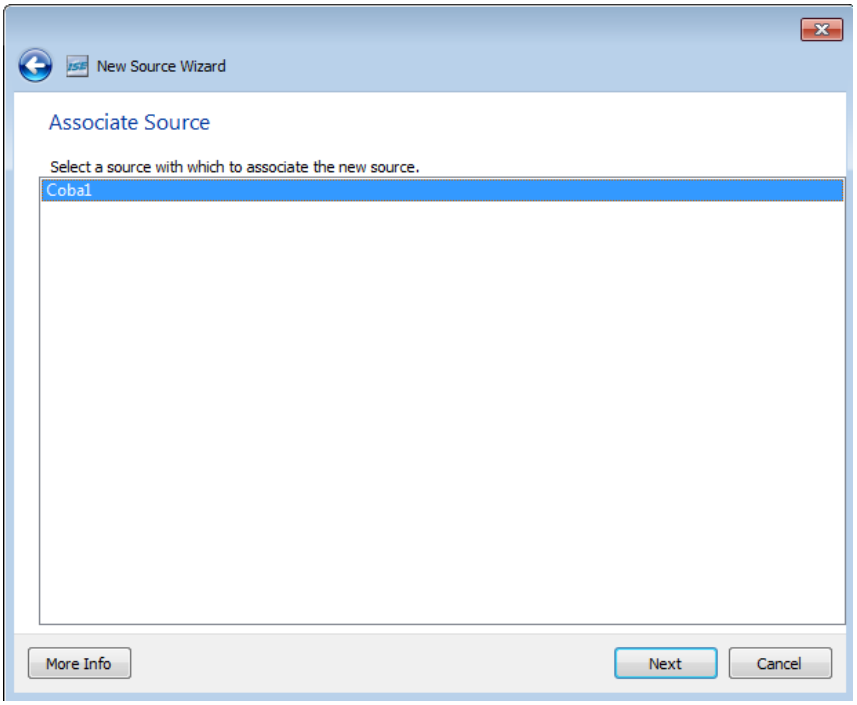
9. Pilih **VHDL Test bench** dan isi **file name-**nya dengan nama **main_tb**, kemudian klik Next.



Gambar 2. 10 VHDL Test Bench

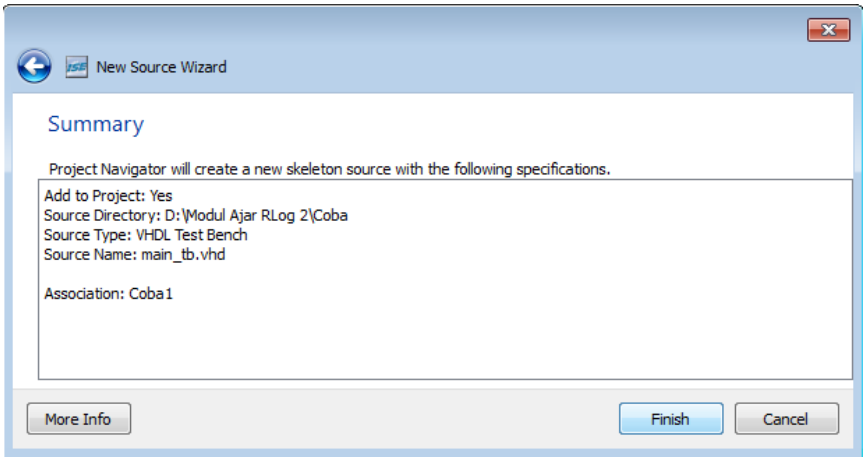
10. Bagian ini menunjukkan tentang file yang akan disimulasikan. Kemudian pilih Next.

    (Pada gambar 2.10 file yang akan disimulasikan adalah program untuk membuat rangkaian logika "**AND"** dan nama filenya adalah Coba1)



Gambar 2. 11 VHDL Test Bench

11. Kemudian pilih finish.



Gambar 2. 12 Finish Wizard

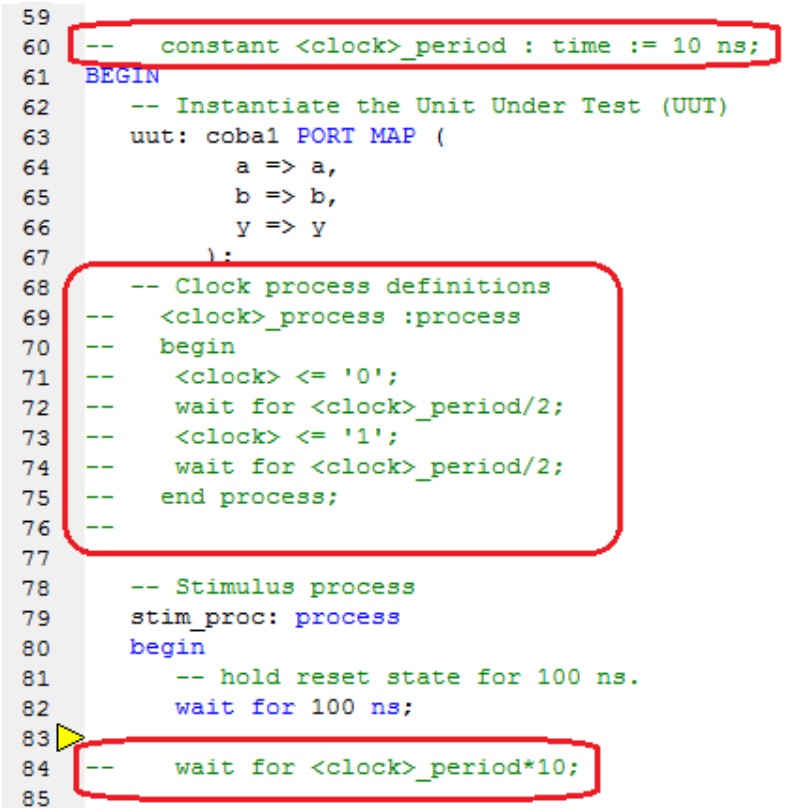
12. Setelah itu akan muncul kode dari main_tb.

```
35  ENTITY main_tb IS
36  END main_tb;
37
38  ARCHITECTURE behavior OF main_tb IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT coba1
43      PORT(
44           a : IN  std_logic;
45           b : IN  std_logic;
46           y : OUT  std_logic
47          );
48      END COMPONENT;
49
50
51     --Inputs
52     signal a : std_logic := '0';
53     signal b : std_logic := '0';
54
55     --Outputs
56     signal y : std_logic;
57     -- No clocks detected in port list. Replace <clock> below with
58     -- appropriate port name
59
60     constant <clock>_period : time := 10 ns;
```

```
61
62  BEGIN
63
64      -- Instantiate the Unit Under Test (UUT)
65      uut: coba1 PORT MAP (
66             a => a,
67             b => b,
68             y => y
69           );
70
71      -- Clock process definitions
72      <clock>_process :process
73      begin
74         <clock> <= '0';
75         wait for <clock>_period/2;
76         <clock> <= '1';
77         wait for <clock>_period/2;
78      end process;
79
79
80
81      -- Stimulus process
82      stim_proc: process
83      begin
84         -- hold reset state for 100 ns.
85         wait for 100 ns;
86
87         wait for <clock>_period*10;
88
89         -- insert stimulus here
90
91         wait;
92      end process;
93
94  END;
```

Gambar 2. 13 Kode Program main_tb

13. Pada source code uut-main_src, **comment baris program yang dilingkari warna merah.**



Gambar 2. 12 Comment baris program

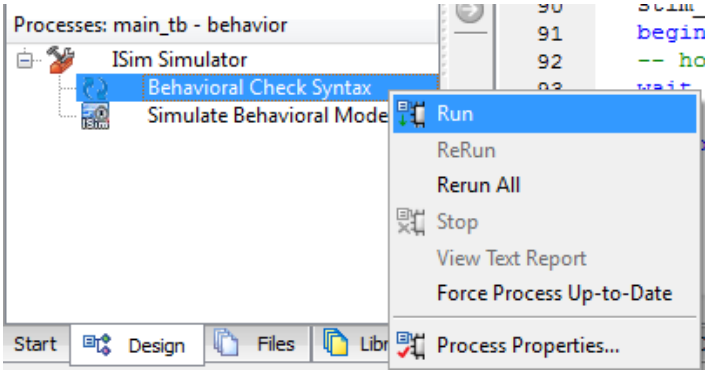
14. Pada bagian **stim_proc : process**, ubahlah baris kodenya menjadi seperti skrip 2.2.

```
-- Stimulus process
stim_proc:process
begin
-- hold reset state for 100 ns.
Wait for 100 ns;
a <=not a;
end process;

-- Stimulus process
stim_proc1:process
begin
-- hold reset state for 100 ns.
Wait for 50 ns;
b <=not b;
end process;
```

Script 2. 2 Membagi periode waktu

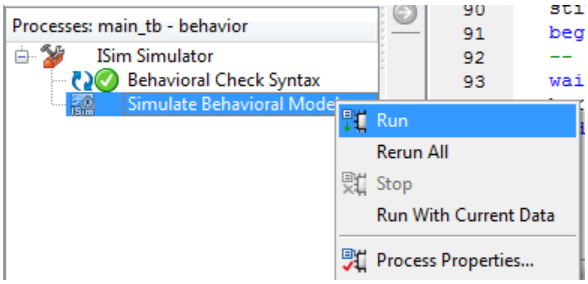
*"Catatan : Dalam kode diatas, maksudnya adalah waktu sinyal **a** akan high selama 100ns dan low selama 100ns, demikian juga pada signal **b** akan high selama 50ns dan low selama 50ns. "*

15. Kemudian Cek syntax, dengan klik kanan pada **Behavioral Check Syntax-Run.**



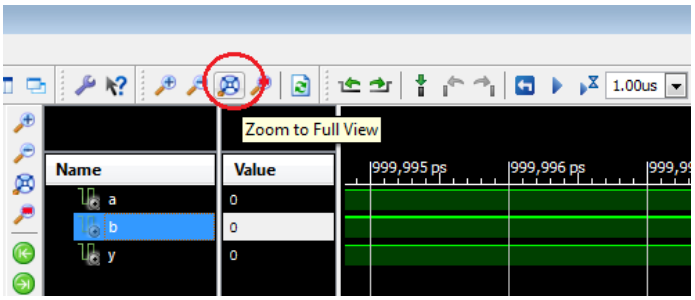
Gambar 2. 13 Cek syntax simulasi

16. Kemudian klik kanan **Simulate Behavioral Model-Run**untuk menampilkan simulasi.



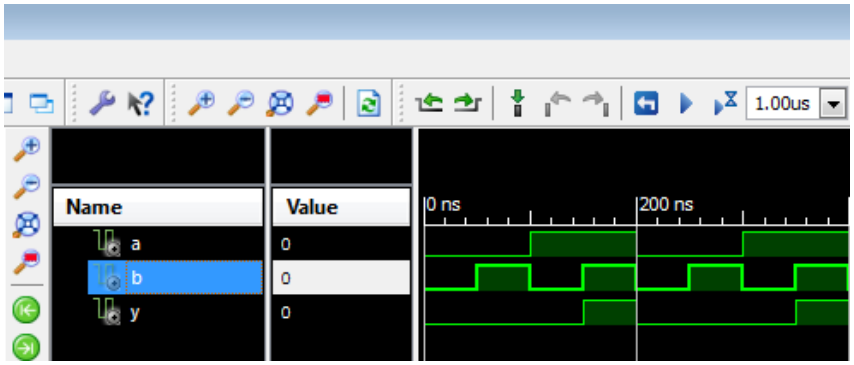
Gambar 2. 14 Menampilkan Simulasi Isim

17. Pilih Zoom to full view untuk melihat secara detail.



Gambar 2. 14 Zoom to full view

18. Kemudian akan tampak timing diagram seperti gambar 2.16



Gambar 2. 15 Timing Diagram rangkaian logika AND

"Pada timing diagram gambar 2.16 hanya ketika sinyal a dan b berlogika high, sinyal output

akan mengeluarkan logika high (C=A.B) "

B. Project 2

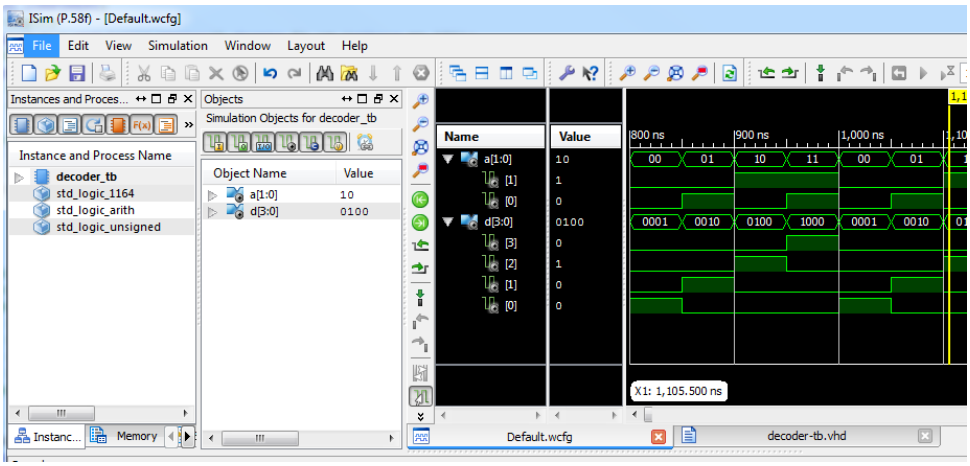
1. Buatlah project baru untuk disain decoder 2 to 4. Lakukan seperti langkah sebelumnya. Berikut program decoder 2 to 4.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder is
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           d : out  STD_LOGIC_VECTOR (3 downto 0)
         );
end decoder;

architecture Behavioral of decoder is

begin
   with a select
      d(3 downto 0) <=  "0001" when "00",
                        "0010" when "01",
                        "0100" when "10",
                        "1000" when "11",
                        "0000" when others;
end Behavioral;
```

2. Berikut program testbench.

```
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: decoder PORT MAP (
          a => a,
          d => d
        );

-- Clock process definitions
--   <clock>_process :process
--   begin
--    <clock> <= '0';
--    wait for <clock>_period/2;
--    <clock> <= '1';
--    wait for <clock>_period/2;
--   end process;

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

   stim_proc1: process
   begin
      wait for 100 ns;
      a(1) <= not a(1);
   end process;
END;
```

3. Dan hasil dari simulasinya ditampilkan seperti gambar dibawah ini.



5. Latihan

Dari sebuah rangkaian Biner to Gray 4 bit, buatlah

a. Buatlah Tabel Kebenaran rangkaian Biner to Grey 4 bit.

b. Gambarkan K-map untuk masing-masing keluaran.

c. Tuliskan persamaan Boolean

d. Buatlah program VHDL

e. Tampilkan RTL Schematics

f. Tampilkan Timing diagram

6. Bentuk dan Format Luaran

xl

a. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.
b. Anatomi laporan :

   Judul, Nama, NRP, Kelas.

   Hasil latihan.

   Analisa hasil latihan.
c. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas2-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkan.

Referensi

1. 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.

Percobaan 3 – Conditional dan Selected Signal Assigment Statement

1. Tujuan

Mahasiswa dapat membuat program VHDL dengan *Conditional dan Selected Signal Assignment Statement*

Mahasiswa dapat membuat dan mendisain dalam rangkaian priority encoder dan decoder mendisain 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 is
 signal s1, s2, s3, s4: bit;
```

xlii

```
begin
      s1 <= b xor cin;
      s2 <= a and b;
      s3 <= a and cin;
      s4 <= 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* dan memiliki 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 benar (*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 (*pernyataan pengandaian) 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 untuk sinyal, sedangkan *if statement* dapat digunakan untuk mengeksekusi sekuensial *statement* manapun.

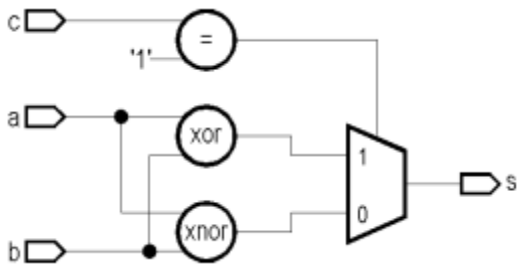
```
entity xor2 is
 port(       a, b: in bit;
             x: out bit);
end xor2;
architecture arch1_xor2 of xor2 is
begin
      x <= '0' when a = b else
           '1';
end arch1_xor2;
```

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 akan diminimalkan nanti untuk koneksi yang sederhana, jadi sinyal c tersebut akan mengontrol *multiplexer* secara langsung.

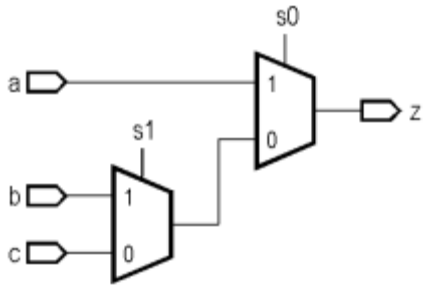


Gambar 3. 1. Conditional *signal assignment statement* sintetis.

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.

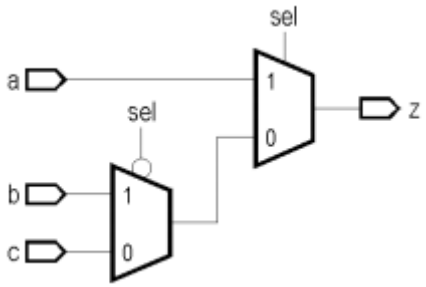
Jika ada banyak cabang, rantai panjang dari *multiplexer* akan menghasilkan secara sintetis. Aspek ini harus dipertimbangkan didalam desain : semakin nanti ekspresi sumber muncul pada *selection list*, semakin banyak *multiplexer* sinyal dari ekspresi ini yang akan melewati rangkaian sintetis.
Setiap kondisi pada kondisional *signal assignment statement* diasumsikan menjadi *independent* dari yang 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 cabang kedua, 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 kasus umum, 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 individual dari set nilai tersebt juga di tentukan, terlepas dari symbol "|", atau nilai jangkauan, atau kombinasi dari 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*) yang memungkinkan harus di *cover* oleh suatu set pilihan.

Catatan

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

```
entity xor2 is
 port(      a, b: in bit;
            x: out bit);
end xor2;
architecture arch_xor2 of xor2 is
      signal temp: bit_vector (1 downto 0);
begin
 temp <= a & b;
 with temp select
            x <= '0' when "00",
            x <= '1' when "01",
            x <= '1' when "10",
            x <= '0' when "11";
end arch_xor2;
```

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

xlviii

Contoh Selected signal assignment statement.

Priority Encoder

Kode untuk priority encoder ditunjukan script 3.6. isi deklarasi adalah identik dengan yang ada pada script 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 sinyal pilihan. 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

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

5. Bentuk dan Format Luaran

a. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.
b. Anatomi laporan :

    Judul, Nama, NRP, Kelas

    Screen capture program VHDL, program simulasi, RTL Schematics.

    Screen capture hasil simulasi dengan semua kombinasi input.
c. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas3-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkan

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.

Percobaan 4 – if and Case Statement

1. Tujuan

Mahasiswa dapat membuat progam dasar VHDL menggunakan *if dan case statement.*

Mahasiswa dapat membuat rangkaian logika sederhana menggunakan *if dan case statement.*

Mahasiswa dapat membuat RTL dari rangkaian logika.

Mahasiswa dapat membuat simulasi testbench dari rangkaian logika.

Mahasiswa dapat membuat *Top Module* dari rangkaian logika.

2. Teori

Pemrograman *VHDL* menggunakan *if statement*

Ekspresi *if* adalah salah satu ekspresi kondisional. Pada pemrograman VHDL, ekspresi ini digunakan untuk mendeskripsikan perilaku dari rangkaian logika. Bentuk umum dari statemen *if* adalah sebagai berikut :

```
[label:]                          -- optional label
If <Boolean expression> then
      <sequential statement>
elsif <Boolean expression>then
      <sequential statement>
else
      <sequential statement>
end if[label];                   --optional label
```

If statement adalah contoh sequential statement yang letaknya berada di dalam blok proses pada sebuah pemrograman VHDL.

```
[label:]                -- optional label
process    (<sensitivity list>)
--declarations
<variable declarations>
begin
      --ekspresi if statement dituliskan disini
      <sequential statements>
end process[label];
```

Sebuah blok proses diawali dengan :

Process (<sensitivity list>)

Dimana *sensitivity list* berisi semua sinyal yang akan di-*generate* di dalam proses blok.

li

VHDL* menggunakan *case statement

V*HDL case statement* digunakan untuk mendeskripsikan behavioral dari system digital. Case statement sering digunakan untuk memilih keadaan dengan banyak pilihan berdasarkan kondisi tertentu. Bentuk umum dari case statement adalah

```
[label:]     --optional label
case <expression>is
      when<choices>=><sequential statement>
      when<choices>=><sequential statement>
      ……..
end case[label];
```

Case statement adalah contoh sebuah *sequential statement* yang harus terjadi pada ***process.*** Pada case statement, semua *<choices>* dari *<expression>*harus dimasukkan. Sebuah ekspresi *<expression>* sering menggunakan tipe STD_LOGIC. Selanjutnya untuk pilihan case yang terakhir adalah

when others *=><sequential statement>*

Dimana *Null* juga dapat digunakan pada <sequential statement>.

Berikut contoh prioriti encoder dan decoder yang ditulis dalam ***if*** statement dan ***case*** statement.

a. if Statement

Program yang sama prioriti encoder dan decoder yang ditulis dengan *if statement* ditunjukan pada *script* 4.1 dan 4.2. *if statement* harus dimasukan di dalam blok *process.*

```
architecture if_arch of prio_encoder is
begin
      process(r)
      begin
            if (r(4)='1' then pcode <= "100";
            elsif (r(3)='1' then pcode <= "011";
            elsif (r(2)='1' then pcode <= "010";
            elsif (r(1)='1' then pcode <= "001";
            else pcode <= "000";
            end if;
      end process;
end if_arch;
```

Script 4. 1 Priority encoder menggunakan *If Statement*

```
architecture if_arch of decoder_2_4 is
begin                                  lii
```

```
      process(en,a)
      begin
            if (en='0') then y <="0000";
            elsif (a="00") then y <= "0001";
            elsif (a="01") then y <= "0010";
            elsif (a="00") then y <= "0100";
            else <= "1000";
            end if;
      end process;
end if_arch;
```

Script 4. 2 Decoder menggunakan *If Statement*

b. Case Statement

Program yang sama prioriti encoder dan decoder yang ditulis dengan *case statement* ditunjukan *script* 4.3 dan 4.4. Kode-kode tersebut mencantumkan semua kombinasi input yang mungkin dan nilai output yang sesuai.

```
architecture case_arch of prio_encoder is
begin
      process(r)
      begin
            case r is
                  when "1000" | "1001" | "1010" | "1011"
                        "1100" | "1101" | "1110" | "1111" =>
                        pcode <= "100";
                  when "0100" | "0101" | "0110" | "0111" =>
                        pcode <= "011";
                  when "0010" | "0011" =>
                        pcode <= "010";
                  when "0001" =>
                        pcode <= "001";
                  when others =>
                        pcode <= "000";
            end case;
      end process;
end case_arch;
```

Script 4. 3 Priority Encoder menggunakan Case statement

```
architecture case_arch of decoder_2_4 is
      signal s: std_logoc_vector(2 downto 0);
begin
      s <= en & a;
```

liii

```
      process(a)
      begin
            case s is
                  when "000" | "001" | "010" | "01" | =>
                        y <= "0001";
                  when "100" =>
                        y <= "0001";
                  when "101" =>
                        y <= "0010";
                  when "110" =>
                        y <= "0100";
                  when others =>
                        y <= "1000";
            end case;
      end process;
end case_arch;
```

Script 4. 4 Decoder menggunakan Case Statement

3. Alat dan Bahan

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

4. Latihan

a. Buatlah program decoder 2to4 dengan if dan case statement.

b. Buatlah program Multiplex 4x1; 4 bit dengan if statement.

c. Buatlah program simulasinya.

d. Tampilkan hasil timing diagram simulasinya.

5. Bentuk dan Format Luaran

1. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.
2. Anatomi laporan :

   Judul, Nama, NRP, Kelas

   Screen capture program VHDL, program simulasi, RTL Schematics.

   Screen capture hasil simulasi dengan semua kombinasi input.
3. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas4-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkan.

Percobaan 5 – For Loop Statement

1. Tujuan

Dapat melakukan pemrogaman dasar VHDL menggunakan *for statement*

Dapat membuat rangkaian logika sederhana menggunakan *for statement*

2. Teori

***For loop* Statement**

VHDL for loop digunakan untuk mendeskripsikan behavior dari beberapa system digital. Ekspresi for loop digunakan untuk sebuah proses perulangan. Bentuk umum dari foor loop adalah

```
[label:] -- optional label
For <identifier> in <range> loop
      <sequential statement>
end loop[label]; -- optional label
```

Dalam *for* loop statement iterasi *counter* dan *range* di spesifikasikan. *Statement* yang berada di dalam *loop body* dieksekusi sedangkan *counter* berada dalam *range* tertentu. Setelah iterasi, *counter* ditugaskan ke nilai selanjutnya dari range. *Range* nya bisa jadi *ascending* (urutan naik), dispesifikasikan oleh kata kunci *to,* atau *descending* (urutan turun) yang dispesifikasikan oleh kata kunci ***downto.*** *Range* ini dapat juga di spesifikasikan sebagai tipe atau *sub*-tipe pencacahan, ketika batasan *range* tidak disebutkan secara spesifik ***for loop*** *statement*. Batasan *range* ditentukan oleh *compiler* dari tipe atau *sub*-tipe deklarasi.

for loop *statement* di sript 5.1 menghitung kuadrat dari nilai integer antara 1 sampai 10, dan menyimpan mereka dalam array ***i_square***.

```
for i in 1 to 10 loop
      i_square (i) <= i * i;
end loop;
```

Script 5. 1 for loop statement

pada contoh ini, iterasi *count* secara default bertipe ***integer***, karena tipe mereka belum didefinisikan secara eksplisit. Bentuk lengkap dari *domain* deklarasi untuk iterasi *count* adalah sama dengan tipenya:

```
for i in integer range 1 to 10 loop
```

Di beberapa bahasa pemrogaman, dalam nilai *loop* bisa jadi ditugaskan untuk iterasi *count* (dalam contoh sebelumnya, i). bahasa VHDL, bagaimanapun, tidak mengijinkan untuk memberi nilai pada iterasi *count* atau untuk menggunakan nya sebagai *input* atau *output* parameter dari sebiah prosedur. *Counter* bisa jadi digunakan di sebuah ekspresi asalkan nilainya tidak dimodifikasi. Aspek lain berhubungan dengan iterasi *count* yang tidak membutuhkan mendeklarasikan nya secara eksplisit di dalam prosesnya. *Counter* ini dideklarasikan secara implisit sebagai variable lokal dari ***loop*** *statement* Jika ada variable lain dengan nama yang sama dalam prosesnya, mereka akan diperlakukan sebagai bariabel pemisah.

Interpretasi sintetis dari ***for loop*** *statement* adalah bahwa *copy* baru digenerasi untuk *circuit* yang dideskripsikan oleh *statement* di setiap iterasi dari *loop.* Penggunaan ***for loop*** *statement* untuk me-generate *circuit* yang diilustrasikan pada script 5.2

```
entity match_bits is
 port( a, b: in bit_vector (7 downto 0);
       matches: out bit_vector (7 downto 0));
end match_bits;
architecture functional of match_bits is
begin
 process (a, b)
    begin
      for i in 7 downto 0 loop
         matches (i) <= not (a(i) xor b(i));
      end loop;
 end process;
end functional;
```

Script 5. 2 for loop statement match bits

Proses dari contoh sebelumnya yang me-generate satu set 1-bit *comparator* untuk membandingkan bit dari urutan vector a dan b yang sama. Hasilnya disimpan di dalam vector yang cocok, yang akan bernilai ‘1’ dimanapun bit dari dua vektor cocok dan ‘0’ untuk kondisi selain itu.

proses dari contoh sebelumnya setara dengan proses berikut:

```
process (a, b)
begin
 matches (7) <= not (a(7) xor b(7));
 matches (6) <= not (a(6) xor b(6));
 matches (5) <= not (a(5) xor b(5));
```

lvi

```
  matches (4) <= not (a(4) xor b(4));
  matches (3) <= not (a(3) xor b(3));
  matches (2) <= not (a(2) xor b(2));
  matches (1) <= not (a(1) xor b(1));
  matches (0) <= not (a(0) xor b(0));
end process;
```

a. Decoder 3 to 8

Ekspresi for loop untuk 3-to-8 Decoder

Tabel 5. 1 *: 3 to 8 Decoder*

A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_unsigned.all;

entity Decode_for is
      port(
            a :inSTD_LOGIC_VECTOR(2 downto 0);
            y :outSTD_LOGIC_VECTOR(0 downto 7)
      );
end Decode_for;

architecture Behavioral of Decode_for is
begin
      process(a)
         variable j :integer;
      begin
            j := conv_integer(a);
            for i in 0 to 7loop
                  if(i = j) then
                        y(i)<= '1';
                  else
```

lvii

```
                        y(i)<= '0';
                    end if;
                end loop;
            end process;
end Decode_for;
```

Script 5. 3 *Decoder 3 to 8*

b. Binary to BCD Converter

Shift and Add 3 Algorithm

Salah satu cara untuk melakukan konversi biner ke BCD adalah dengan metode shift and Add 3 Algorithm, yang step-stepnya adalah sebagai berikut :

a. Menggeser satu bit bilangan biner.
b. Jika 8 pergeseran telah terjadi, jumlah BCD adalah di kolom ratusan, puluhan dan satuan.
c. Jika nilai biner dalam salah satu kolom BCD adalah lebih besar dari 4, tambahkan 3 pada nilai di kolom tersebut.
d. Kembali ke step a

Algoritma secara detail dapat diilustrasikan pada tabel 5.2.

Tabel 5. 2 ***: Algoritma shift and add 3 binary to BCD***

Operation	Hundreds	Tens	Units	Binary	
B				7 4	3 0
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		
P	9 8	7 4	3 0		
Z	17 16	15 12	11 8	7 4	3 0

lviii

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity Bin2BCD is
      port(
            b:in STD_LOGIC_VECTOR(7 downto 0);
            p:out STD_LOGIC_VECTOR(9 downto 0)
      );
end Bin2BCD;

architecture Behavioral of Bin2BCD is
begin
      bcd1:process(b)
      variable z :STD_LOGIC_VECTOR(17 downto 0);
      begin
            for i in 0 to 17 loop
                  z(i):='0';
            endloop;
            z(10 downto 3):=b;
            for i in 0 to 4 loop
                  if z(11 downto 8)> 4 then
                        z(11downto8):= z(11 downto 8)+3;
                  endif;
                  if z(15 downto 12)>4 then
                        z(15 downto 12):= z(15 downto 12)+3;
                  endif;
                  z(17 downto 1):= z(16 downto 0);
            end loop;
            p <= z(17 downto 8);
      end process bcd1;
end Behavioral ;
```

Script 5. 4 ***Ekspresi for untuk biner to BCD Decoder menggunakan shift and Add 3 Algorithm***

c. Gray Code Converter - Biner to Gray

Berikut ini adalah hubungan antara 3 bit binary dengan 3 bit gray code.

Binary code {0..7} : {000, 001, 010, 011, 100, 101, 110, 111}

Gray code {0..7} : {000, 001, 011, 010, 110, 111, 101, 100}

Konversi Biner ke Gray :

- Salin semua bit

- untuk setiap i : ***g(i) = b(i+1)^b(i)***

Konversi Gray ke Biner :

lix

- Salin semua bit
- Untuk setiap i terkecil : ***b(i)= b(i+1) ^ g(i)***

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BinToGray is
	port(
		b:in STD_LOGIC_VECTOR(3 downto 0);
		g:out STD_LOGIC_VECTOR(3 downto 0)
	);
end BinToGray;

architecture Behavioral of BinToGray is
begin
	process(b)
	begin
		g(3)<= b(3);
		for i in 2 downto 0 loop
			g(i)<= b(i+1)xor b(i);
		end loop;
	end process;
end Behavioral;
```

Script 5. 5 *Konversi Biner ke Gray code*

d. Gray Code Converter - Gray to Biner

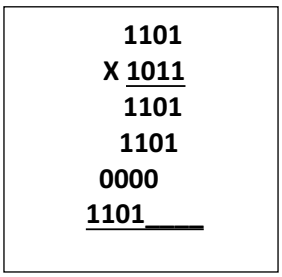
```
libraryIEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity GrayToBin is
	port(
		g:in STD_LOGIC_VECTOR(3 downto 0);
		b:out STD_LOGIC_VECTOR(3 downto 0)
	);
end GrayToBin;

architecture Behavioral of GrayToBin is
begin
	process(g)
		variable bv :STD_LOGIC_VECTOR(3 downto 0);
	begin
		bv(3):= g(3);
		for i in 2 downto 0 loop
			bv(i):= bv(i+1)xor g(i);
		end loop;
```

```
            b <= bv;
      end process;
end Behavioral ;
```

Script 5. 6 *Konversi Gray Code ke Biner*

e. Multiplier

Fungsi loop juga bisa digunakan untuk fungsi multiplier. Pada script 5.7 adalah aplikasi loop untuk perkalian 4 bit dan hasilnya ditampung dalam 8 bit.

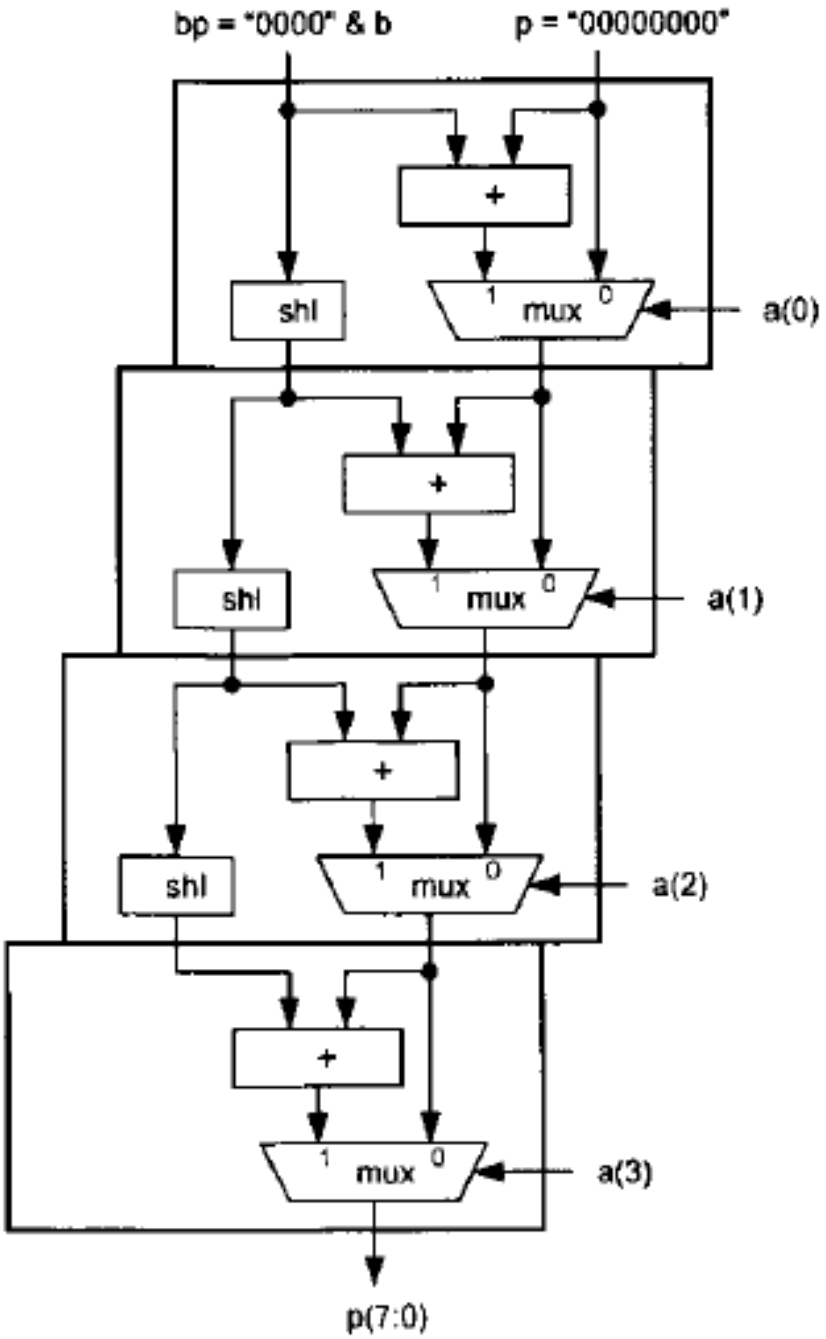


Gambar 5. 1 *Binary multiplication*

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.all;

entity MulShift is
      port(
            a:in STD_LOGIC_VECTOR(3 downto 0);
            b:in STD_LOGIC_VECTOR(3 downto 0);
            p:out STD_LOGIC_VECTOR(7 downto 0)
      );
end MulShift;

architecture Behavioral of MulShift is
begin
      process(a,b)
      variable pv,bp :STD_LOGIC_VECTOR(7 downto 0);
      begin
            pv :="00000000";
            bp :="0000"& b;
            for i in 0 to 3 loop
                  if a(i)= '1' then
                        pv := pv + bp;
                  endif;
                  bp := bp(6 downto 0) & '0';
            endloop;
            p <= pv;
      endprocess;
```

```
end Behavioral ;
```

Script 5. 7 Multiplier dengan operator (&)



Gambar 5. 2 Logika Diagram Multiplier

Untuk menggunakan multiplication operator(*) kita harus menggunakan tambahan library :

use *IEEE.STD_LOGIC_**unsigned**.all;*

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.all;
```

lxii

```
entity Mul is
	port(
		a:in STD_LOGIC_VECTOR(3downto0);
		b:in STD_LOGIC_VECTOR(3downto0);
		p:out STD_LOGIC_VECTOR(7downto0)
	);
end Mul;

architecture Behavioral of Mul is
begin
		p <= a*b;

end Behavioral;
```

Script 5. 8 *Multiplier dengan operator (*)*

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.all;

entity MulFunc is
	port(
		sw:in STD_LOGIC_VECTOR(7 downto 0);
		ld:out STD_LOGIC_VECTOR(7 downto 0)
	);
end MulFunc;

architecture Behavioral of MulFunc is

function mul(a,b :in STD_LOGIC_VECTOR)
		return STD_LOGIC_VECTOR is
variable pv,bp :STD_LOGIC_VECTOR(7 downto 0);

begin
	pv :="00000000";
	bp :="0000"& b;
	for i in 0 to 3 loop
		if a(i)='1' then
			pv := pv + bp;
		endif;
		bp := bp(6 downto 0)& '0';
	end loop;
	return pv;
end function;
```

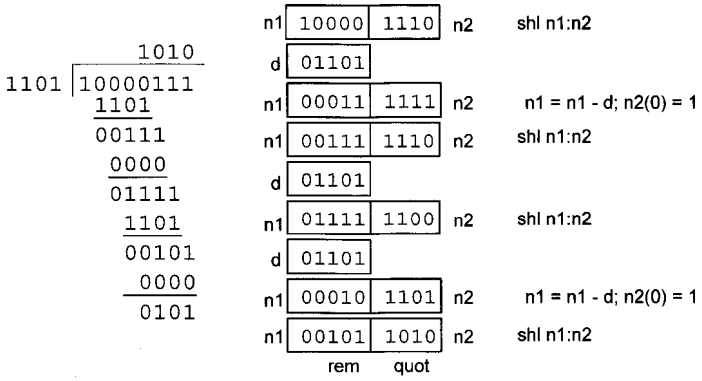
lxiii

```
signal a1, a2:STD_LOGIC_VECTOR(3 downto 0);
begin
      a1 <= sw(7 downto 4);
      a2 <= sw(7 downto 4);
      ld <= mul (a1,a2);
end Behavioral ;
```

Script 5. 9 *Multiplication Function*

f. Divider

Ekspresi for loop untuk fungsi divider :



Gambar 5. 3 Contoh fungsi pembagian

Algoritma divider sebagai berikut :

1. Simpan pembilang dalam gabungan dari *n1:n2*
2. Simpan denominator di *d*
3. Ulangi 4 kali :

   Geser n1:n2 kekiri 1 bit

   If n1>d

   n1 = n1 – d;

   n2(0) =1;
4. quot = n2;

   rem = n1(3:0);

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
Use IEEE.STD_LOGIC_unsigned.all;

entity Devide is
      port(
```

```
            numer  :in STD_LOGIC_VECTOR(7 downto 0);
            denom :in STD_LOGIC_VECTOR(3 downto 0);
            quotient  :out STD_LOGIC_VECTOR(3 downto 0);
            remainder  :out STD_LOGIC_VECTOR(3 downto 0)
      );
end Devide;

architecture Behavioral of Devide is
begin
      process(numer,denom)
      variable d, n1 :STD_LOGIC_VECTOR(4 downto 0);
      variable n2 :STD_LOGIC_VECTOR(3 downto 0);
      begin
            d  := '0' & denom;
            n2 := numer(3 downto 0);
            n1 := '0' & numer (7 downto 4);
            for i in 0 to 3 loop
                  n1 := n1(3 downto 0)& n2(3);
                  n2 := n2(2 downto 0)& '0';
                  if n1 >= d then
                        n1 := n1-d;
                        n2(0):= '1';
                  end if;
            end loop;
            quotient <= n2;
            remainder <= n1(3 downto 0);
      end process;
end Behavioral;
```

Script 5. 10 *Divider*

3. Alat dan Bahan

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

4. Langkah Percobaan

1. Tuliskan program perkalian dan pembagian.
2. Dengan menggunakan *function*, buatlah program sesuai dengan persamaan x = (a * b)/2.
3. Tampikan RTL Schematicnya.
4. Buatlah program simulasinya.

5. Tampilkan hasil simulasi, jika nilai a=7; b=4

5. Bentuk dan Format Luaran

1. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.
2. Anatomi laporan :

   Judul, Nama, NRP, Kelas

   Screen capture program VHDL, program simulasi, RTL Schematics.

   Screen capture hasil simulasi.
3. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas5-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkan.

Referensi

1. 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.

Percobaan 6 – Structural Design

1. Tujuan

Dapat membuat program dengan model disain structural.

Dapat mengimplementasikan algoritma multiplexer pada disain structural.

2. Teori

Sebuah pemodelan terstruktur untuk menjelaskan jalur komponen dan koneksinya. Setiap komponen diasumsikan telah didefinisikan sebelumnya dan dapat dijelaskan secara terstruktur, sebuah *behavior* atau model *dataflow*. Pada hirarki terendah setiap komponen dideskripsikan sebagai pemodelan *behavior* menggunakan operator logika dasar di *VHDL*. Secara umum pemodelan terstruktur sangat bagus untuk mendeskripsikan sistem digital yang kompleks, meskipun seperangkat komponen dalam model hirarki.

Sebuah penjelasan struktur dapat dibandingkan secara terbaik terhadap diagram blok skematik yang dapat dijelaskan dengan komponen dan interkoneksi. VHDL memberikan cara sebagai berikut :

- Deklarasikan kompoen yang digunakan
- Deklarasikan sinyal yang menentukan *nets* yang menghubungkan dengan komponen
- Melabelkan kompoenen yang sama berkali – kali sehingga setiap contoh didefinisikan secara unik.

komopnen dan sinyal dideklarasikan di dalam arsitektur *body.*

```
architecture architecture_name of NAME_OF_ENTITY is
      -- Declarations
      component declarations
      signal declarations
begin
      -- Statements
      component instantiation and connections;
end architecture_name;
```

Deklarasi Komponen

Sebelum komponen dapat dipakai mereka butuh di deklarasikan di dalam sesi deklarasi arsitektur atau dalam paket deklarasi. Deklarasi komponen terdiri dari nama komponen dan *interface* nya *(ports).* Sintak nya sebagai berikut:

```
component component_name
      port (      port_signal_names: mode type;
                  port_signal_names: mode type;
```

```
                    :
                    port_signal_names: mode type);
    end component ;
```

Nama komponen merujuk pada nama dari *entity* yang didefinisikan di *library* juga sebuah *entity* yang secara jelas didefinisikan di dalam *file VHDL.*

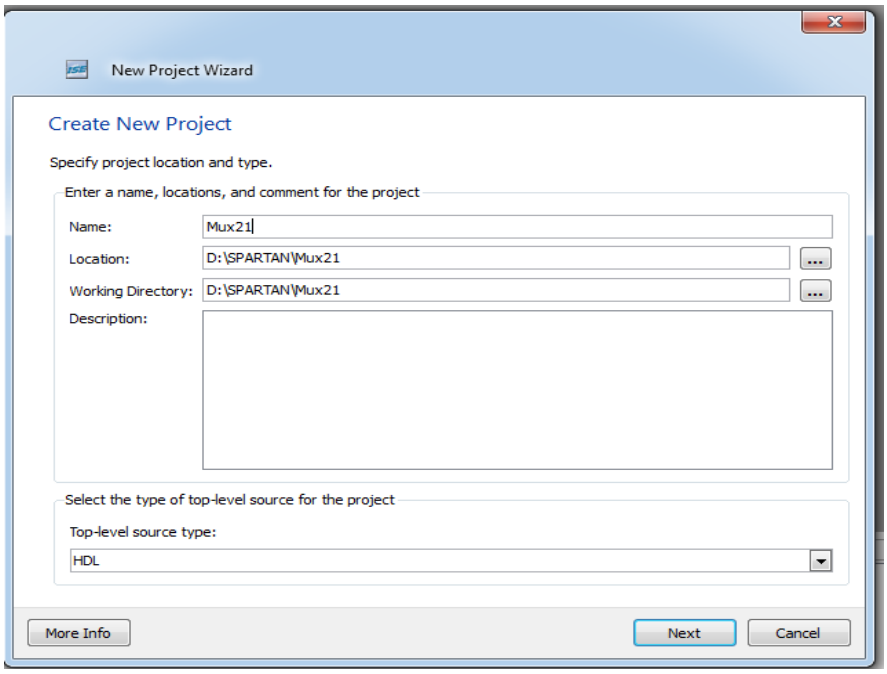
3. Alat dan Bahan

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

4. Langkah Percobaan

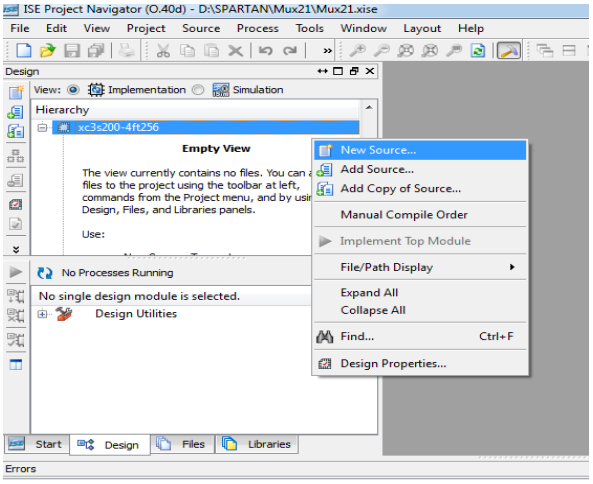
A. Membuat File Top Modul

1. Membuat project baru dengan nama folder Mux21, top level source HDL.



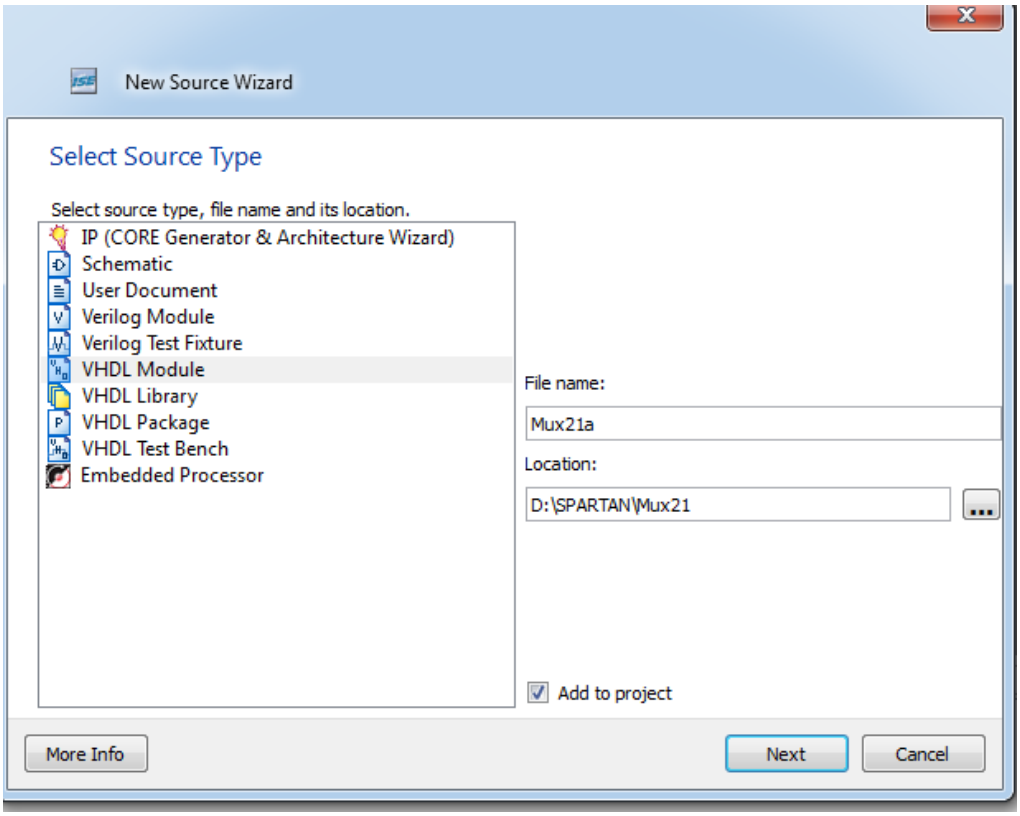
Gambar 6. 1 Membuat project baru dengan nama folder

2. Setelah selesai membuat folder, klik kanan pada type IC kemudian pilih **New Source** untuk membuat file .vhdl



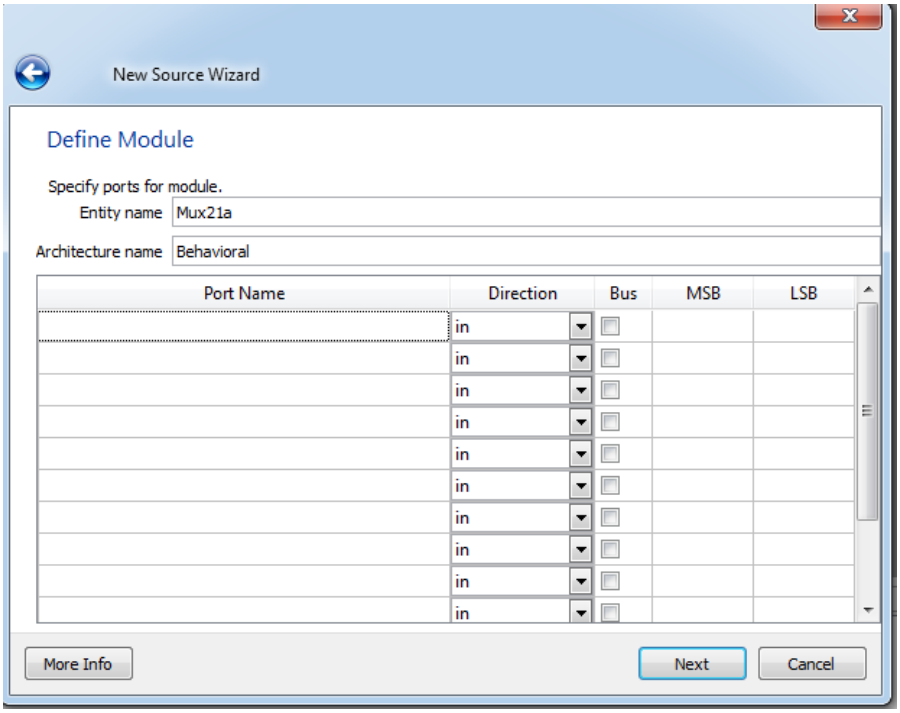
Gambar 6. 2 New Source

3. Pada **New Source Wizard -** Pilih **VHDL Module** dan beri nama file misalnya Mux21a kemudian klik **next.**



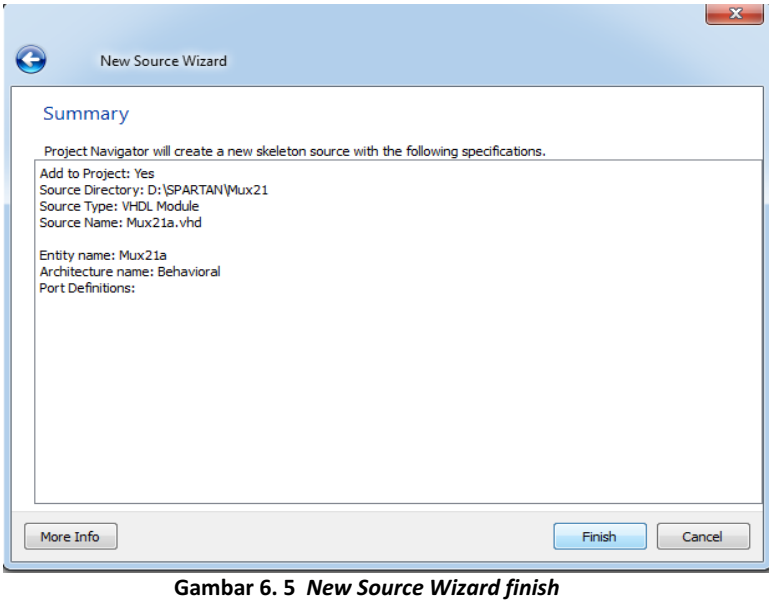
Gambar 6. 3. Memberi nama file

4. Pada sesi **Define Module** biarkan kosong dan lewati saja, kemudian klik **next.**



Gambar 6. 4. Define module

5. Pilih Finish



Gambar 6. 5 ***New Source Wizard finish***

6. Lengkapi program pada file Mux21a seperti pada skrip 6.1

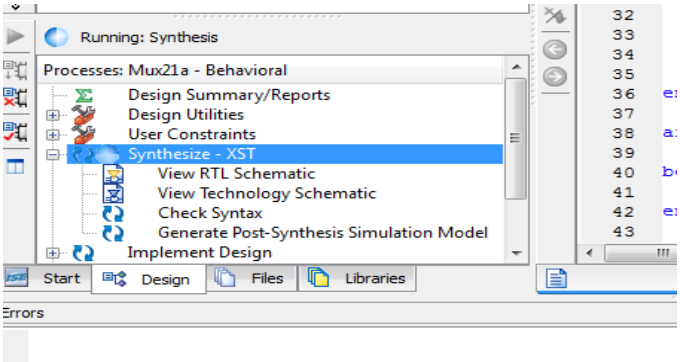
```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux21a is
      Port(
            a :inSTD_LOGIC;
```

lxx

```
            b :in STD_LOGIC;
            s :in STD_LOGIC;
            y :out STD_LOGIC
      );
end Mux21a;

architecture Behavioral of Mux21a is
begin
      y <=(not s and a)or(s and b);
end Behavioral;
```

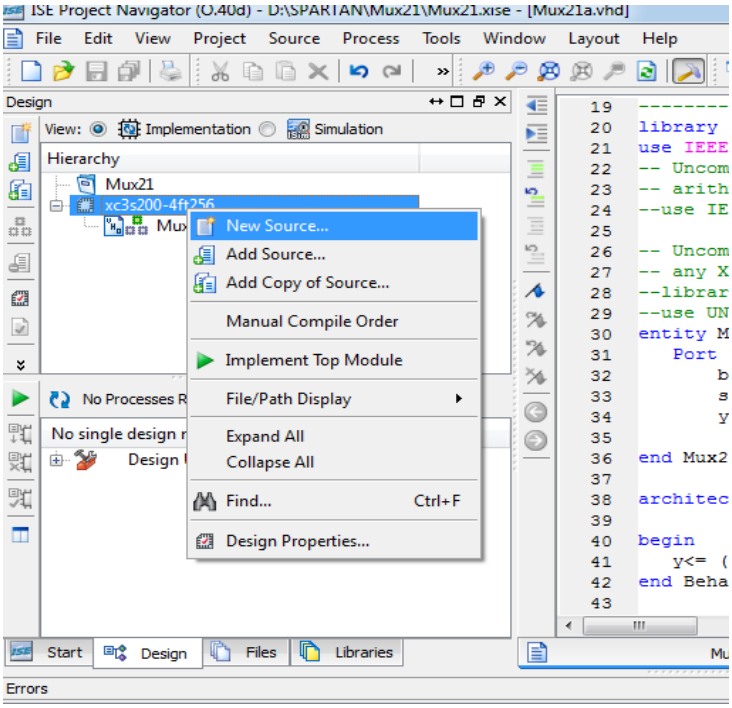
Skrip 6. 1 Program Mux21a

7. Lakukan double klik pada **Synthesize-xst** untuk check Syntax.



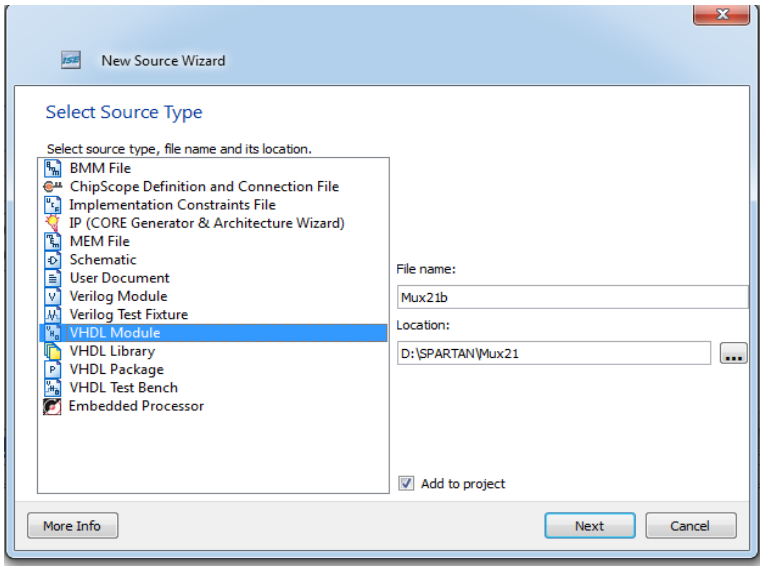
Gambar 6. 6 Double klik Synthesize –XST

8. Membuat file kedua dengan cara yang sama, klik kanan pada nama IC kemudian pilih **New-Source**



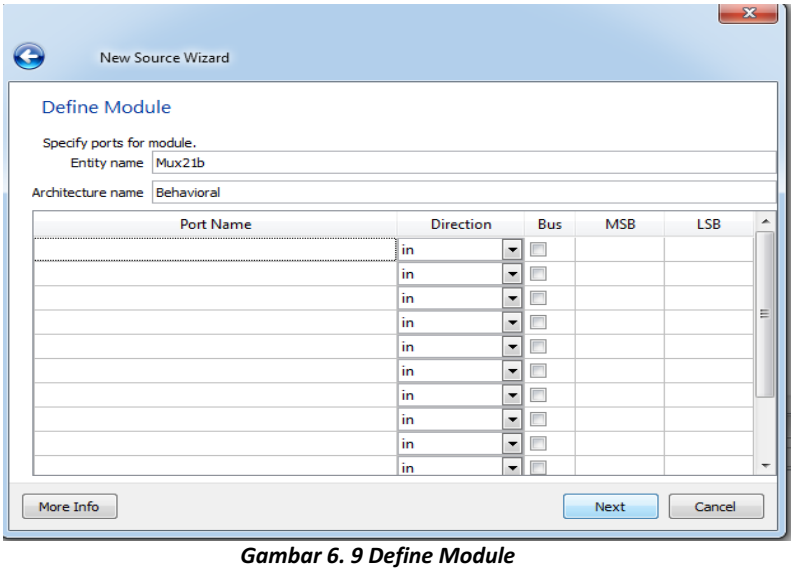
Gambar 6. 7 Membuat file baru .vhdl

9. Pada New Source Wizard bagian Select Source Type, pilih **VHDL Module** dan beri nama file misalnya **Mux21b**, kemudian klik **next**



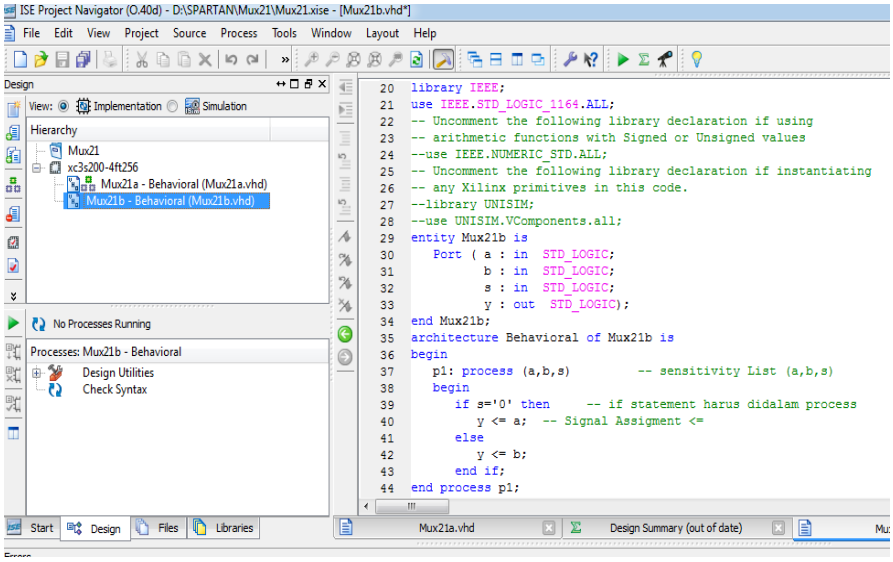
Gambar 6. 8 Membuat file baru Mux21b.vhdl

10. Pada New Source Wizard bagian Define Module, biarkan dan lewati saja bagian ini kemudian klik **next.** Dan selanjutnya klik Finish.



Gambar 6. 9 Define Module

11. Lengkapi program pada file Mux21b seperti gambar 6.10.

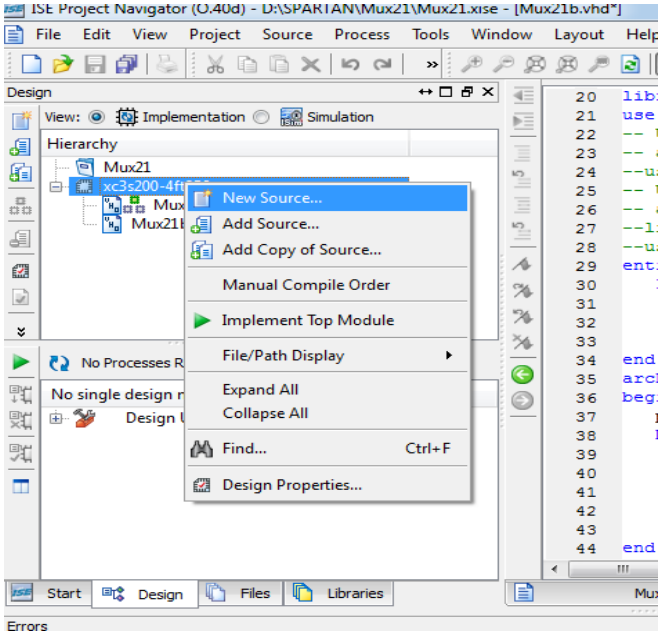


Gambar 6. 10 Program file Mux21b.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux21b is
      Port(
            a:inSTD_LOGIC;
            b:inSTD_LOGIC;
            s:inSTD_LOGIC;
            y:outSTD_LOGIC;
      );
end Mux21b;

architecture Behavioral of Mux21b is
begin
p1 :process(a,b,s)
      begin
            if s='0' then
                  y <= a;
            else
                  y <= b;
            endif;
end process p1;
```

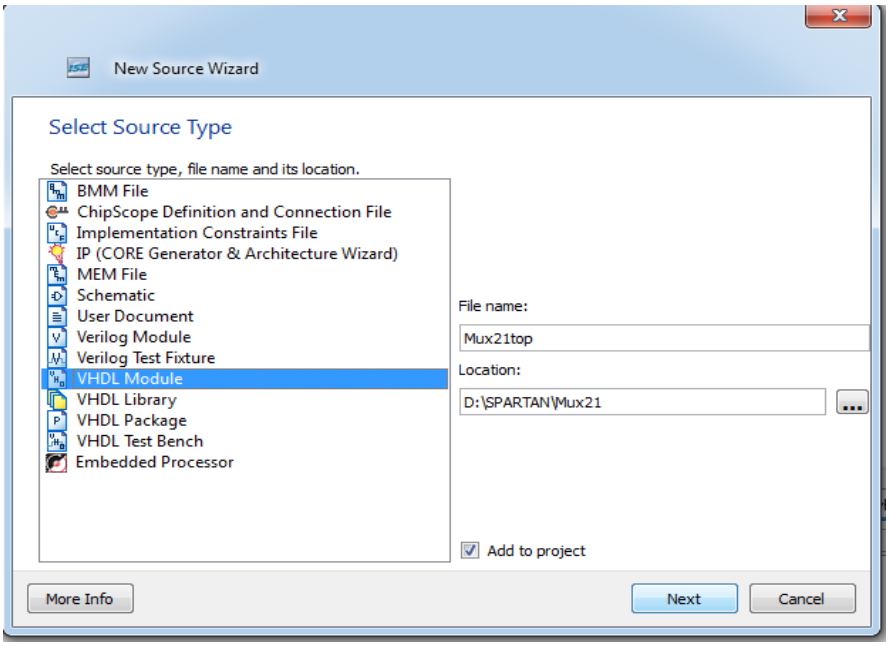
Skrip 6. 2 Program file Mux21b.vhdl

12. Kemudian buat file baru yang ketiga, dengan cara klik kanan pada nama IC-pilih **New Source**



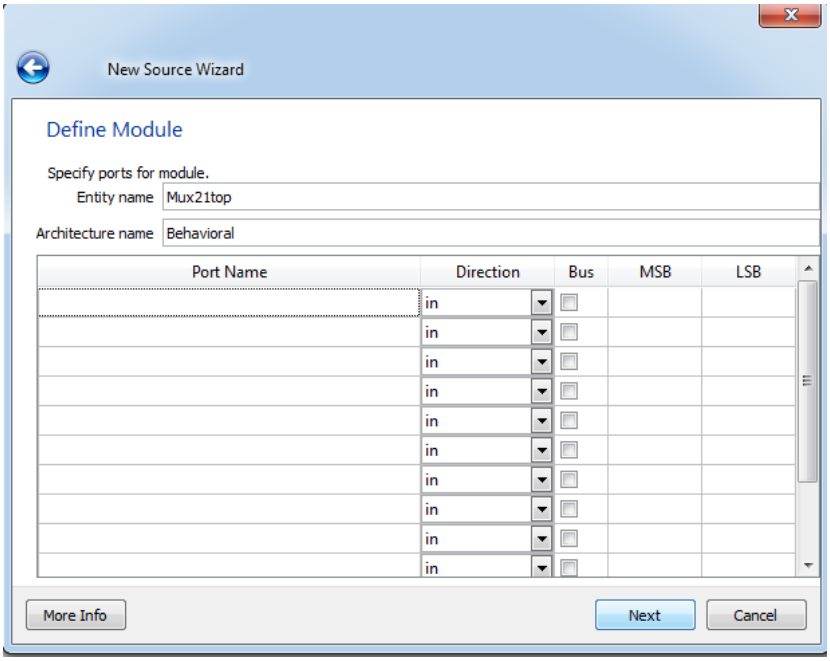
Gambar 6. 11 Membuat file baru

13. Pada New Source Wizard bagian Select Source Type pilih VHDL Module dan beri nama file Mux21top.



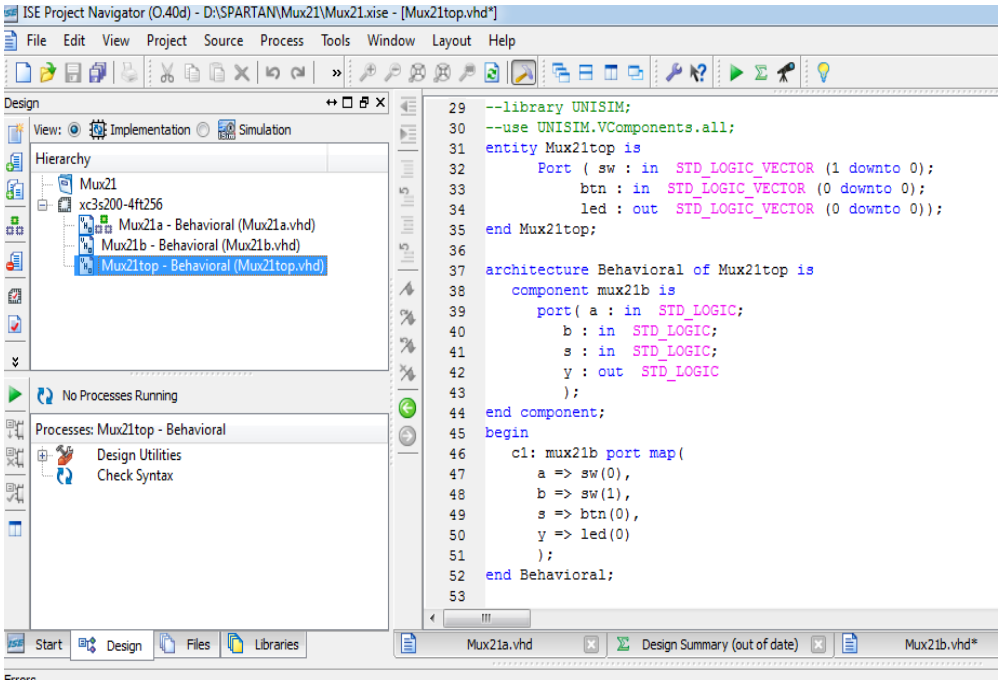
Gambar 6. 12 Membuat file baru .vhdl

14. Pada bagian Define Module biarkan kosong dan lewati saja, kemudian klik **next,** dan selanjutnya klik Finish.



Gambar 6. 13 Define Module

15. Lengkapi kode program file Mux21top seperti pada gambar 6.13.



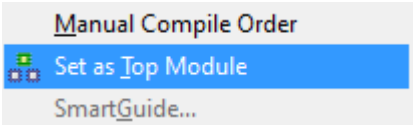
Gambar 6. 14. Program file Mux21top

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;

entity Mux21top is
      Port(
            sw :in STD_LOGIC_VECTOR(1 downto 0);
            btn :in STD_LOGIC_VECTOR(0 downto 0);
```

```
            led    :out STD_LOGIC_VECTOR(0 downto 0)
      );
end Mux21top;

architecture Behavioral of Mux21top is
component mux21b is
      port(
            a :in STD_LOGIC;
            b :in STD_LOGIC;
            s :in STD_LOGIC;
            y :out STD_LOGIC
      );
endcomponent;

begin
      c1:mux21b portmap(
            a => sw(0),
            b => sw(1),
            s => btn(0),
            y => led(0);
      );
end Behavioral;
```

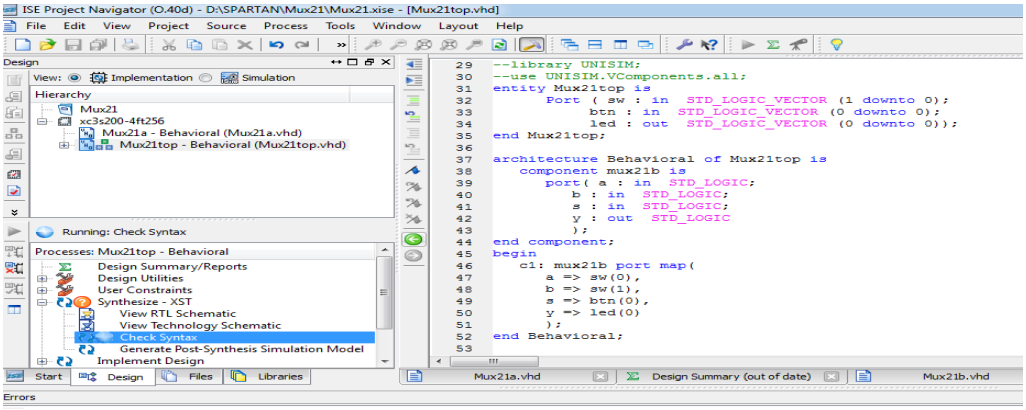
Skrip 6. 3 Kode program file Mux21top.vhdl

Saat kita membuat pertama kali sebuah file project .vhdl, maka secara otomatis ISE akan menempatkan file tersebut sebagai top modul. Sedangkan yang kita ingin jadikan sebagai top modul adalah file Mux21top. Sehingga untuk membuat file Mux21top menjadi top model caranya adalah klik kanan pada nama file, pilih **Set as Top Module.**



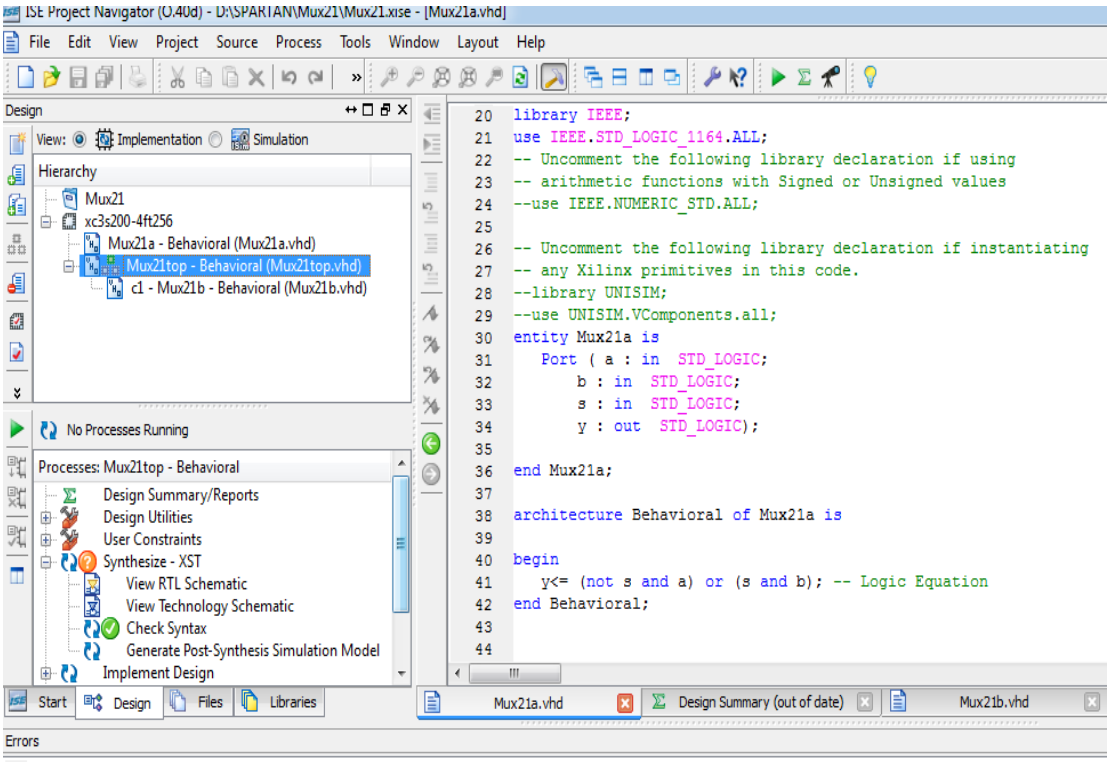
Gambar 6. 15 Kode program file Mux21top.vhdl

16. Lakukan Check Syntax pada file Mux21b, untuk cek kode program. (Secara otomatis file Mux21b akan pindah menjadi bagian dari file Mux21top, dan secara otomatis pula file Mux21b menjadi top module. Hal ini karena program dari file Mux21b sudah di-call oleh program Mux21top)



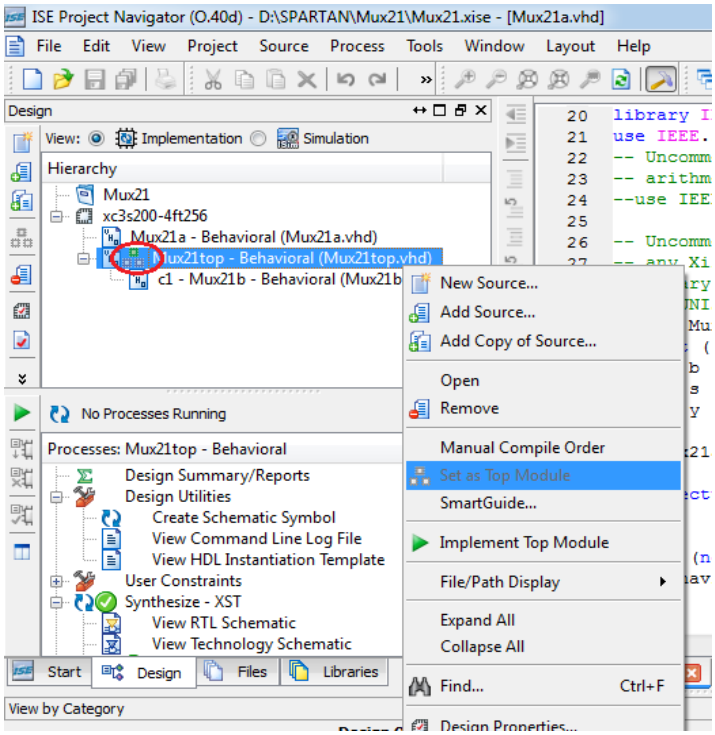
Gambar 6. 16 Check Syntax

17. Karena program Mux21a tidak di-call oleh file Mux21top, maka program Mux21a tidak berada di akar program Mux21top. Kemudian secara otomatis, Mux21top akan menjadi top module yang di dalamnya ada module Mux21b.



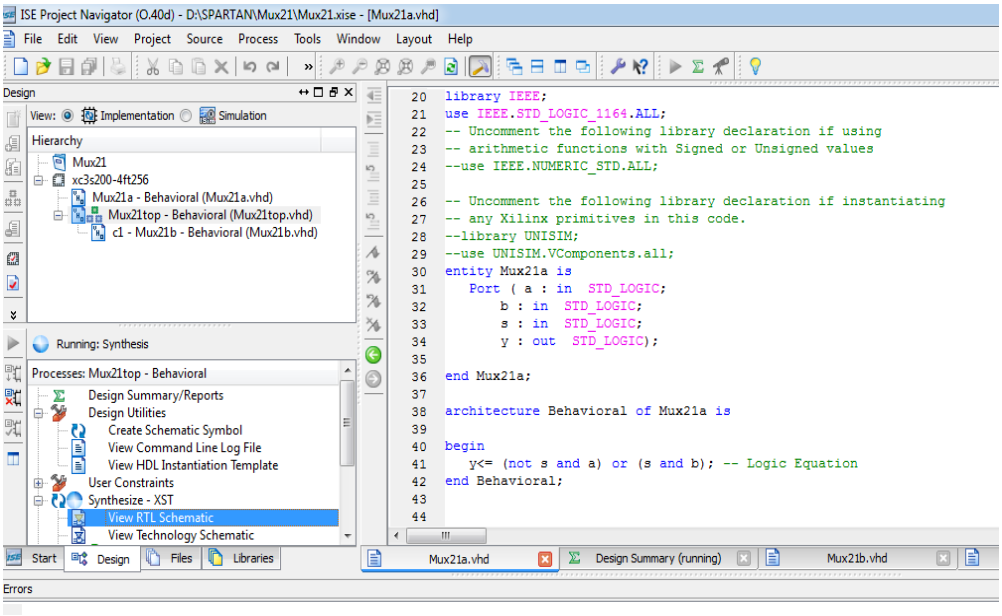
Gambar 6. 17 Top Module

Untuk membuat top module secara manual, caranya sebagai berikut :



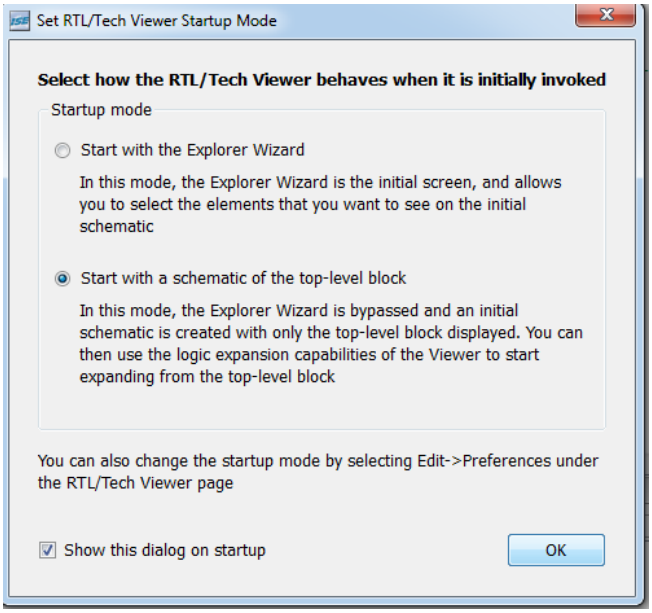
Gambar 6. 18 Tanda Top Module dan cara membuat file menjadi Top Module

18. View RTL Schematic dengan double klik **View RTL Schematic** pada Synthesize-XST



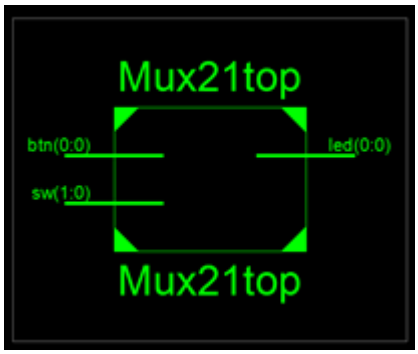
Gambar 6. 19 Double Klik View RTL Schematic

19. Pilih **Start with a schematic of the top-level block** pada dialog Set RTL



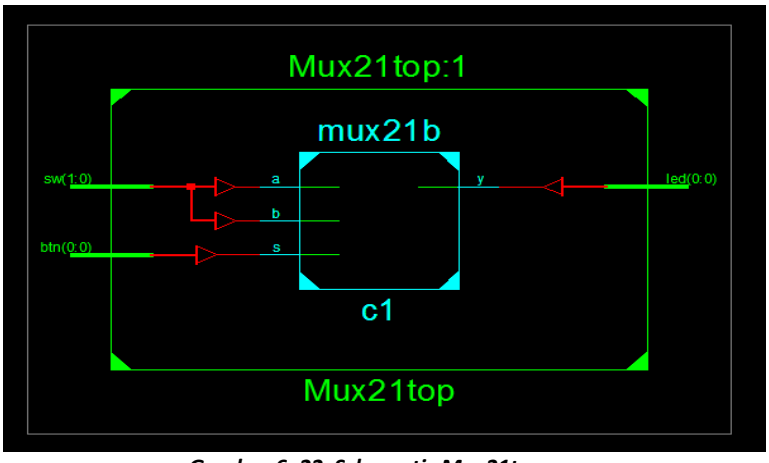
Gambar 6. 20 Set RTL Start with a schematic

20. Akan muncul diagram blok dari **top module** dengan IO, untuk melihat isi bagian dalam top module, klik 2x pada diagram blok top module.



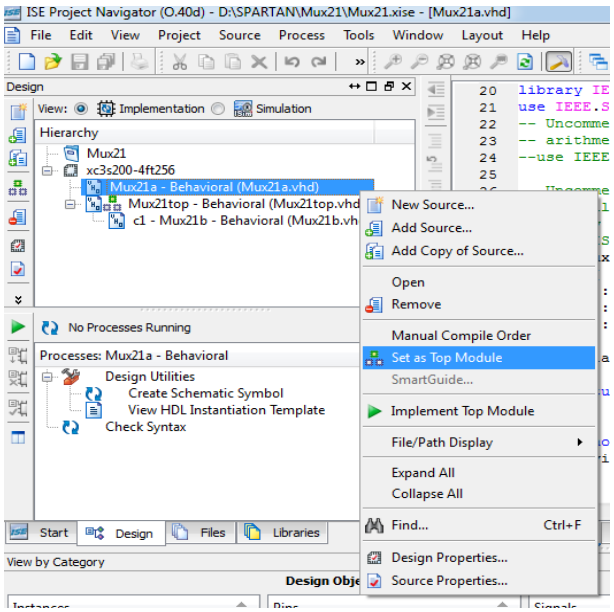
Gambar 6. 21 Diagram Blok top module

21. Akan diperlihatkan shcematic dari top module Mux21top.



Gambar 6. 22 Schematic Mux21top

22. Bila ingin menjadikan file lain sebagai top module, caranya adaah klik kanan pada nama file-pilih **Set as top module**



Gambar 6. 23 Membuat top module

5. Latihan

Membangun full adder 1 bit untuk membangun full adder 4 bit.

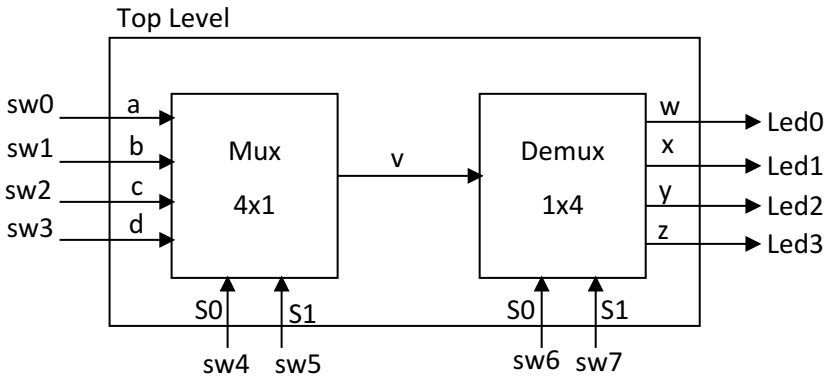
1. Buat program full adder 1 bit dan beri nama FA_1.vhd dan full ader 4 bit dengan nama FA_4.

```
---------------- FA_1.vhd ---------------------
entity FA_1 is
    Port (  A : in  STD_LOGIC;
            B : in  STD_LOGIC;
                Cin : in  STD_LOGIC;
            S : out  STD_LOGIC;
            Cout : out  STD_LOGIC);
end FA_1;

architecture Behavioral of FA_1 is
begin
      S <= A xor B xor Cin;
      Cout <= (A and B) or (A and Cin) or (B and Cin);
end Behavioral;
```

```
---------------- FA_4.vhd ---------------------
entity FA_4 is
    Port (  a : in  STD_LOGIC_VECTOR (3 downto 0);
            b : in  STD_LOGIC_VECTOR (3 downto 0);
            cin : in  STD_LOGIC;
                  |^^^
```

```
            sum : out  STD_LOGIC_VECTOR (3 downto 0);
            cout  : out  STD_LOGIC);
end FA_4;

architecture Behavioral of FA_4 is
component fA_1
      port(       A, B, Cin : in std_logic;
            S, Cout  : out std_logic
      );
end component;
signal ci  : std_logic_vector(4 downto 0);


begin
      ci(0) <= cin;
      u0 : FA_1   port map(A => a(0), B => b(0), Cin => ci(0), S => sum
(0), Cout  => ci(1));
      u1 : FA_1   port map(A => a(1), B => b(1), Cin => ci(1), S => sum
(1), Cout  => ci(2));
      u2 : FA_1   port map(A => a(2), B => b(2), Cin => ci(2), S => sum
(2), Cout  => ci(3));
      u3 : FA_1   port map(A => a(3), B => b(3), Cin => ci(3), S => sum
(3), Cout  => ci(4));
      cout  <= ci(4);
end Behavioral;
```

2. Membangun multipelex dan demultiplex
3. Buatlah program Multiplexer 4x1 menggunakan if statement
4. Buatlah program Demultiplexer 1x4 menggunakan if statement.
5. Buatlah program top_level-nya seperti pada gambar dibawah ini



6. Bentuk dan Format Luaran

1. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.

2. Anatomi laporan :

   Judul, Nama, NRP, Kelas

   Screen capture program VHDL, program simulasi, RTL Schematics.

   Screen capture hasil simulasi.

   Foto hasil demo program mutiplex dan demultiplex.

3. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas6-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkan

Referensi

1. 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.

Percobaan 7 – Decoder 7 segmen

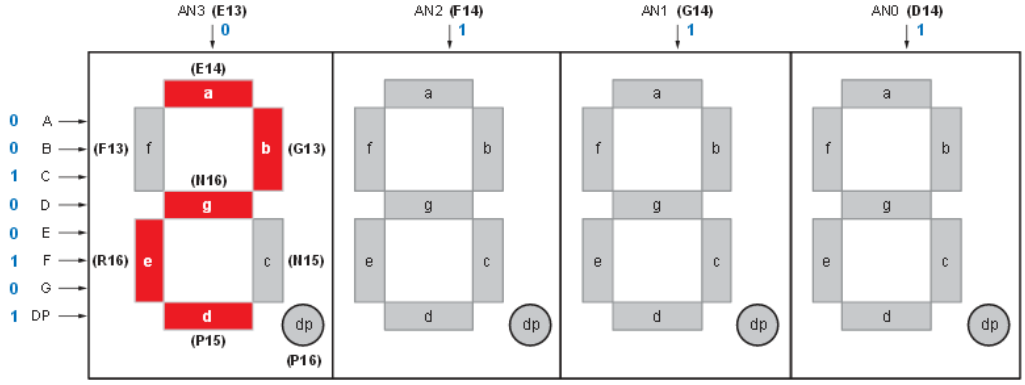
1. Tujuan

Mampu mengimplementasikan algoritma decoder 7 segmen.

Dapat menampilkan nilai masukan keluaran pada 7 segmen dalam modul FPGA.

2. Teori

Pada percobaan ini akan mempelajari beberapa rangkaian kombinasional yaitu BCD to 7'segmen. Rangkaian BCD to 7'segmen adalah rangkaian kombinasional yang mengkodekan bilangan biner menjadi bilangan desimal, selanjutnya bilangan tersebut di konfigurasi menjadi tampilan *seven segmen*. Konfigurasi seven segmen pada board spartan 3 ditunjukan pada gambar 7.1.



Gambar 7. 1 Konfigurasi seven segmen pada board spartan3

Terdapat 4 buah sevensegmen yang disusun secara pararel dengan data a-g dan dot di rangkai menjadi satu. Sedangkan kendali tampilan di atur melalu pin AN di setiap segmen.

3. Alat dan Bahan

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

4. Langkah Percobaan

A. 7-Segmen Display.

Percobaan ini akan membuat membuat program dengan masukan switch dan keluaran 7segmen.

1. Buatlah ***new project*** dengan nama **Lab7A**

2. Tambahkan program dibawah ini.

a. 7-segmen Decoder dengan Logic Equation.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity hex7seg is
    Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
           a_to_g : out  STD_LOGIC_VECTOR (6 downto 0));
end hex7seg;
architecture hex7seg_le of hex7seg is
begin
      a_to_g(6)<=  (not x(3) and not x(2) and not x(1) and x(0))  --a
            or (not x(3) and x(2) and not x(1) and not x(0))
            or ( x(3) and x(2) and not x(1) and x(0))
            or ( x(3) and not x(2) and x(1) and x(0));
      a_to_g(5)<=  (x(2) and x(1) and not x(0))                   --b
            or ( x(3) and x(1) and x(0))
            or (not x(3) and x(2) and not x(1) and x(0))
            or ( x(3) and x(2) and not x(1) and not x(0));
      a_to_g(4)<= (not x(3) and not x(2) and x(1) and not x(0))   --c
            or (x(3) and x(2) and x(1))
            or (x(3) and x(2) and not x(0));
      a_to_g(3)<=  (not x(3) and not x(2) and not x(1) and x(0))  --d
            or (not x(3) and x(2) and not x(1) and not x(0))
            or (x(3) and not x(2) and x(1) and not x(0))
            or (x(2) and x(1) and x(0));
      a_to_g(2)<=  (not x(3) and x(0))                            --e
            or (not x(3) and x(2) and not x(1))
            or (not x(2) and not x(1) and x(0));
      a_to_g(1)<=  (not x(3) and not x(2) and x(0))               --f
            or (not x(3) and not x(2) and x(1))
            or (not x(3) and x(1) and x(0))
            or (x(3) and x(2) and not x(1) and x(0));
      a_to_g(0)<=  (not x(3) and not x(2) and not x(1))           --g
            or (x(3) and x(2) and not x(1) and not x(0))
            or (not x(3) and x(2) and x(1) and x(0));
end hex7seg_le;
```

Script 7. 1. 7-Segmen Logic Equation

b. 7-Segmen Decoder Case Statement

```
entity hex7segb is
      port (x : in  STD_LOGIC_VECTOR (3 downto 0);
            a_to_g : out  STD_LOGIC_VECTOR (6 downto 0));
end hex7segb;

                                    lvvvvv
architecture hex7segb of hex7segb is
```

```
begin
	process(x)
	begin
		case x is                              --  abcdefg
			when x"0" => a_to_g <= "0000001";--0
			when x"1" => a_to_g <= "1001111";--1
			when x"2" => a_to_g <= "0010010";--2
			when x"3" => a_to_g <= "0000110";--3
			when x"4" => a_to_g <= "1001100";--4
			when x"5" => a_to_g <= "0100100";--5
			when x"6" => a_to_g <= "0100000";--6
			when x"7" => a_to_g <= "0001111";--7
			when x"8" => a_to_g <= "0000000";--8
			when x"9" => a_to_g <= "0000100";--9
			when x"A" => a_to_g <= "0001000";--a
			when x"B" => a_to_g <= "1100000";--b
			when x"C" => a_to_g <= "0110001";--c
			when x"D" => a_to_g <= "1000010";--d
			when x"E" => a_to_g <= "0110000";--e
			when others => a_to_g <= "0111000"; --f
		end case;
	end process;
end hex7segb;
```

Script 7. 2. 7-Segmen Case Statement

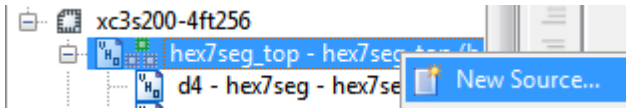
c. 7-Segmen Top Level

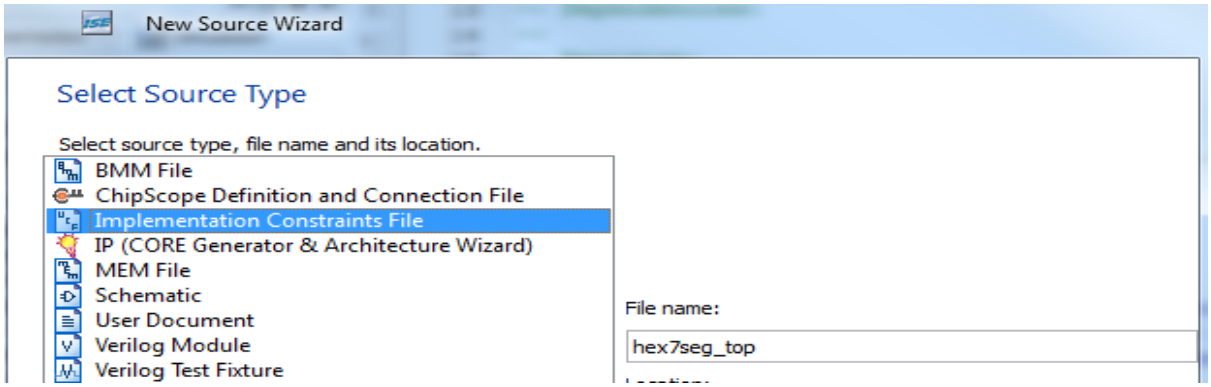
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity hex7seg_top is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
           a_to_g : out  STD_LOGIC_VECTOR (6 downto 0);
           an : out  STD_LOGIC_VECTOR (3 downto 0);
           dp : out  STD_LOGIC);
end hex7seg_top;

architecture hex7seg_top of hex7seg_top is
	component hex7segb is
		port(
			x: in STD_LOGIC_VECTOR (3 downto 0);
		   a_to_g : out  STD_LOGIC_VECTOR (6 downto 0)
		);
	end component;
begin
	an <= "0000";		-- all digit on
	dp<= '1';		-- dp off
	d4: hex7seg port map (x=>sw, a_to_g=>a_to_g);
```

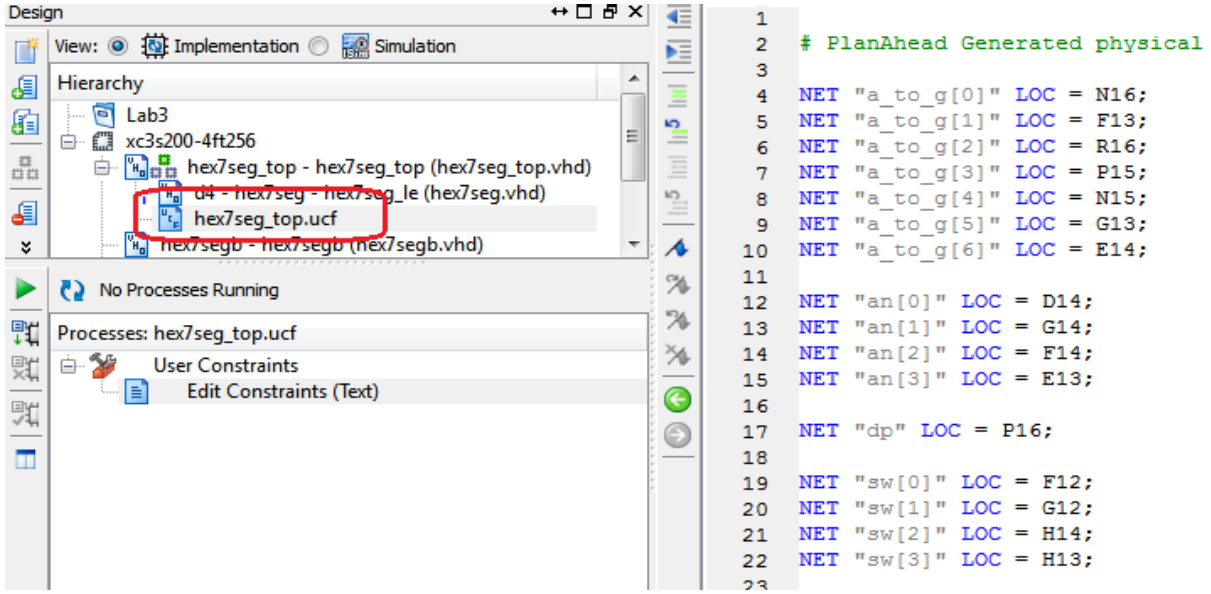
```
end hex7seg_top;
```

Script 7. 3. 7-Segmen Top Level

3. Atur agar program ***hex7seg_top*** menjadi ***Top modul***.
4. Lakukan ***Synthezises-XST***, pastikan tidak ada error.
5. Tambahkan konfigurasi pin dengan cara tambahkan **new source| Implementation Contraints File.** Beri nama **hex7seg_top.**







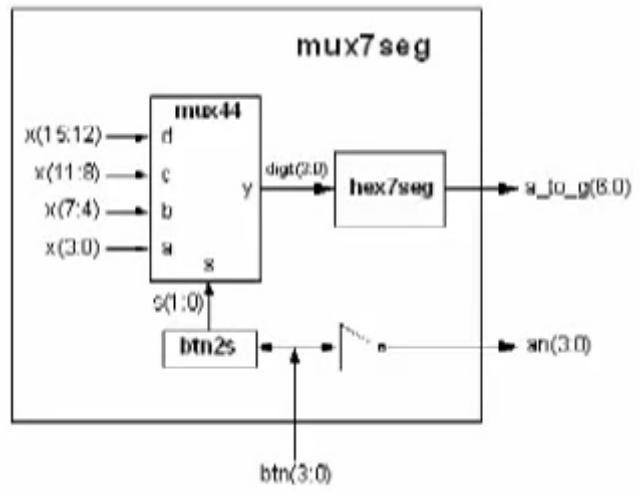
6. Klik Pada jendela **hex7seg_top.ucf** dan isikan konfigurasi pin seperti gambar diatas.
7. Lakukan ***Synthezises-XST | Implement Design | Generate programming File*** sampai dengan ***programming*** ke *board* Spartan-3 FPGA.

8. Atur switch sesuai dengan 1 digit angka terakhir nrp anda.
9. Foto hasil tampilan seven segmen.

B. Multiplexing 7-Segmen

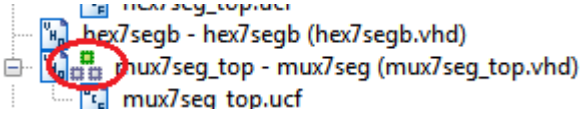
Percobaan ini untuk menampilkan data angka "1234" ke 7segmen dengan pengaturan tampilan dari enable 7segmen melaui *push botton*.

Blok program ditunjukan pada gambar 7.2.



Gambar 7. 2 Blok program mux7seg

1. Buatlah ***new project*** dengan nama **Lab7B**.
2. Tambahkan Program baru dengan nama ***mux7seg_top.***
3. Pastikan ***top modul*** pada mux7seg_top.vhd.



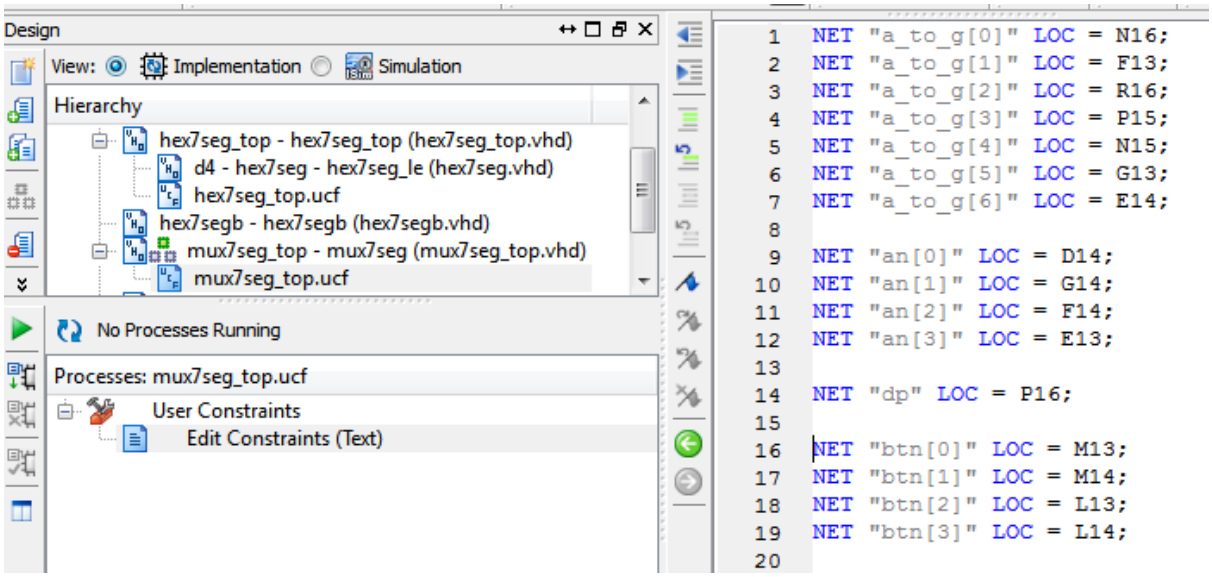
4. Tulis program mux7seg_top.vhd berikut ini.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux7seg_top is
    Port ( btn : in  STD_LOGIC_VECTOR (3 downto 0);
           a_to_g : out  STD_LOGIC_VECTOR (6 downto 0);
           an : out  STD_LOGIC_VECTOR (3 downto 0);
           dp : out  STD_LOGIC);
end mux7seg_top;

architecture mux7seg of mux7seg_top is
     signal x: STD_LOGIC_VECTOR (15 downto 0);
     signal s: STD_LOGIC_VECTOR (1 downto 0);
     signal digit: STD_LOGIC_VECTOR (3 downto 0);
```

```
begin
	x <= x"1234";
	an <= not btn;
	s(1) <= btn(2) or btn(3);
	s(0) <= btn(1) or btn(3);
	dp <='1';

-- quad 4 to 1 mux: mux44
	process(s)
	begin
		case s is
			when "00" => digit <= x(3 downto 0);
			when "01" => digit <= x(7 downto 4);
			when "10" => digit <= x(11 downto 8);
			when others => digit <= x(15 downto 12);
		end case;
	end process;

	process(digit)
	begin
		case digit is			--  abcdefg
			when x"0" => a_to_g <= "0000001";	--0
			when x"1" => a_to_g <= "1001111";	--1
			when x"2" => a_to_g <= "0010010"; --2
			when x"3" => a_to_g <= "0000110";	--3
			when x"4" => a_to_g <= "1001100"; --4
			when x"5" => a_to_g <= "0100100";	--5
			when x"6" => a_to_g <= "0100000"; --6
			when x"7" => a_to_g <= "0001111";	--7
			when x"8" => a_to_g <= "0000000";	--8
			when x"9" => a_to_g <= "0000100";	--9
			when x"A" => a_to_g <= "0001000";	--a
			when x"B" => a_to_g <= "1100000"; --b
			when x"C" => a_to_g <= "0110001";	--c
			when x"D" => a_to_g <= "1000010"; --d
			when x"E" => a_to_g <= "0110000";	--e
			when others => a_to_g <= "0111000"; --f
		end case;
	end process;
end mux7seg;
```

Program 4. Multiplexing 7-Segmen

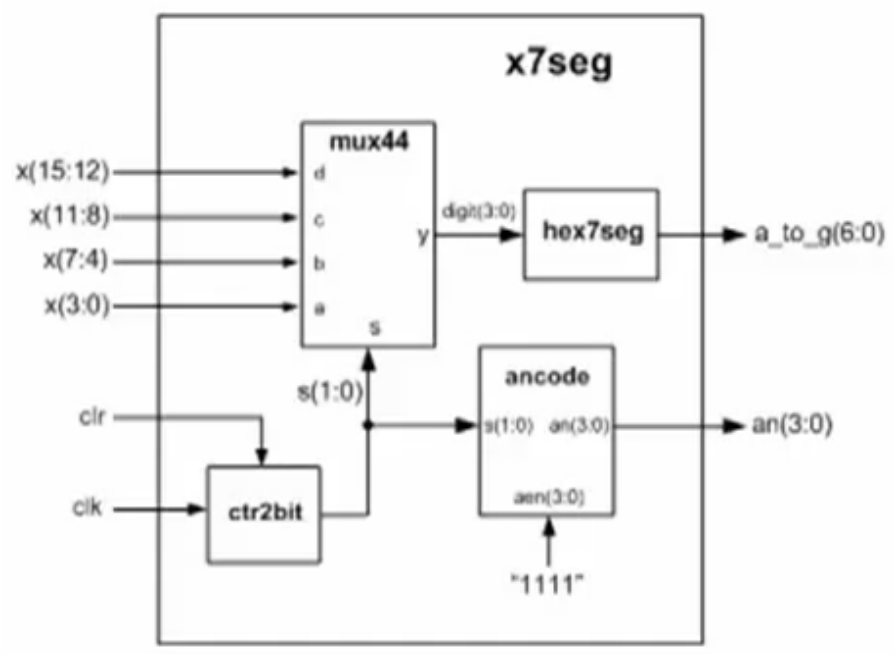
5. Tambahkan file ***.ucf*** untuk konfigurasi pin. Seperti dibawah ini.



6. Jalankan programnya dan upload program ke dalam chip FPGA-nya
7. Ubah program agar tampilkan di seven segmen adalah 4 digit terakhir NRP anda.
8. Fotokan hasil tampilannya.

C. Multiplexing 7segmen Display dengan ClockDevider

Tampilan dari empat 7segmen akan diatur secara otomatis oleh ClockDevider. Ada 2 model program yang akan ditampilkan yaitu ***x7seg*** dan ***x7segb***.



Gambar 7. 3. Blok program x7seg

lxxxix

x7seg.

1. Buatlah ***new project*** dengan nama **Lab7C1.**
2. Tambahkan program dibawah ini.
3. Seperti pada langkah sebelumnya. Tambahkan program vhdl berikut ini.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity x7seg is
    Port ( x : in  STD_LOGIC_VECTOR (15 downto 0);
           clk : in  STD_LOGIC;
           clr : in  STD_LOGIC;
           a_to_g : out  STD_LOGIC_VECTOR (6 downto 0);
           an : out  STD_LOGIC_VECTOR (3 downto 0);
           dp : out  STD_LOGIC);
end x7seg;

architecture x7seg of x7seg is
    signal s : STD_LOGIC_VECTOR (1 downto 0);
    signal digit : STD_LOGIC_VECTOR (3 downto 0);
    signal aen : STD_LOGIC_VECTOR (3 downto 0);
    signal clkdiv : STD_LOGIC_VECTOR (20 downto 0);
begin
    s <= clkdiv(20 downto 19);
    aen <= "1111";
    dp <= '1';
-- Quad 4 to 1 mux
    process(s,x)
    begin
          case s is
                when "00" => digit <=x(3 downto 0);
                when "01" => digit <=x(7 downto 4);
                when "10" => digit <=x(11 downto 8);
                when others => digit <=x(15 downto 12);
          end case;
    end process;

-- 7-segment decoder: hex7seg
    process(digit)
    begin
          case digit is
                when x"0" => a_to_g <= "0000001";   --0
                when x"1" => a_to_g <= "1001111";   --1
                when x"2" => a_to_g <= "0010010";   --2
```

```
                when x"3" => a_to_g <= "0000110";   --3
                when x"4" => a_to_g <= "1001100";   --4
                when x"5" => a_to_g <= "0100100";   --5
                when x"6" => a_to_g <= "0100000";   --6
  when x"7" => a_to_g <= "0001111"; --7
                when x"8" => a_to_g <= "0000000";   --8
                when x"9" => a_to_g <= "0000100";   --9
                when x"A" => a_to_g <= "0001000";   --a
                when x"B" => a_to_g <= "1100000";   --b
                when x"C" => a_to_g <= "0110001";   --c
                when x"D" => a_to_g <= "1000010";   --d
                when x"E" => a_to_g <= "0110000";   --e
                when others => a_to_g <= "0111000";       --f
            end case;
    end process;

  --  Digit select: ancode
    process(s,aen)
    begin
            an <= "1111";
            if aen(conv_integer(s))='1' then
                an(conv_integer(s))<='0';
            end if;
    end process;

  -- Clock devider
    process(clk, clr)
    begin
            if clr='1' then
                clkdiv <= (others => '0');
            elsif clk'event and clk='1' then
                clkdiv <= clkdiv+1;
            end if;
    end process;

  end x7seg;
```

Program 5. Multiplexing 7-Segmen dengan ClockDevider A

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity x7seg_top is
    Port ( mclk : in  STD_LOGIC;
           btn : in  STD_LOGIC;
```

xci

```
            a_to_g : out  STD_LOGIC_VECTOR (6 downto 0);
            an : out  STD_LOGIC_VECTOR (3 downto 0);
            dp : out  STD_LOGIC);
end x7seg_top;

architecture x7seg_top of x7seg_top is
      component x7seg is
            Port ( x : in  STD_LOGIC_VECTOR (15 downto 0);
                  clk : in  STD_LOGIC;
                  clr : in  STD_LOGIC;
                  a_to_g : out  STD_LOGIC_VECTOR (6 downto 0);
                  an : out  STD_LOGIC_VECTOR (3 downto 0);
                  dp : out  STD_LOGIC);
      end component;
      signal x: STD_LOGIC_VECTOR (15 downto 0);
begin
      x <= x"1234";
      x1 : x7seg port map
      (x=>x, clk=>mclk, clr=>btn, a_to_g=>a_to_g,an=>an, dp=>dp);
end x7seg_top;
```

Program 6. Multiplexing 7-Segmen Top Modul

4. Jalankan programnya dan upload program ke dalam chip FPGA-nya
5. Ubah program agar tampilkan di seven segmen adalah “Kelas(A/B)” – “0” - 2 digit terakhir NRP anda.
6. Fotokan hasil tampilannya.

x7segb.

1. Buatlah ***new project*** dengan nama **Lab7C2.**
2. Tambahkan program dibawah ini.
3. Seperti pada langkah sebelumnya. Tambahkan program vhdl berikut ini.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
```

```
entity x7segb is
    Port ( x : in  STD_LOGIC_VECTOR (15 downto 0);
           clk : in  STD_LOGIC;
           clr : in  STD_LOGIC;
           a_to_g : out  STD_LOGIC_VECTOR (6 downto 0);
           an : out  STD_LOGIC_VECTOR (3 downto 0);
           dp : out  STD_LOGIC);
end x7segb;

architecture x7seg of x7segb is
      signal s: STD_LOGIC_VECTOR (1 downto 0);
      signal digit: STD_LOGIC_VECTOR (3 downto 0);
      signal aen: STD_LOGIC_VECTOR (3 downto 0);
      signal clkdiv: STD_LOGIC_VECTOR (20 downto 0);
begin
      s <= clkdiv(20 downto 19);
      dp <= '1';

-- set aen(3 downto 0) for leading blenks
      aen(3) <= x(15) or x(14) or x(13) or x(12);
      aen(2) <= x(15) or x(14) or x(13) or x(12) or
                x(11) or x(10) or x(9)  or x(8);
      aen(1) <= x(15) or x(14) or x(13) or x(12) or
                x(11) or x(10) or x(9)  or x(8)  or
                x(7)  or x(6)  or x(5)  or x(4);
      aen(0) <= '1'; --digit 0 always on

--Quad 4 to 1 mux
      process(s,x)
      begin
            case s is
                  when "00" => digit <= x(3 downto 0);
                  when "01" => digit <= x(7 downto 4);
                  when "10" => digit <= x(11 downto 8);
                  when others => digit <= x(15 downto 12);
            end case;
      end process;

-- 7-segment decoder: hex7seg
      process(digit)
      begin
            case digit is
                  when x"0" => a_to_g <= "0000001";   --0
                  when x"1" => a_to_g <= "1001111";   --1
                  when x"2" => a_to_g <= "0010010"; --2
```

```
                when x"3"  => a_to_g <= "0000110";   --3
                when x"4"  => a_to_g <= "1001100";   --4
                when x"5"  => a_to_g <= "0100100";   --5
                when x"6"  => a_to_g <= "0100000";   --6
                when x"7"  => a_to_g <= "0001111";   --7
                when x"8"  => a_to_g <= "0000000";   --8
                when x"9"  => a_to_g <= "0000100";   --9
                when x"A"  => a_to_g <= "0001000";   --a
                when x"B"  => a_to_g <= "1100000";   --b
                when x"C"  => a_to_g <= "0110001";   --c
                when x"D"  => a_to_g <= "1000010";   --d
                when x"E"  => a_to_g <= "0110000";   --e
                when others => a_to_g <= "0111000"; --f
            end case;
      end process;

-- Digit select: ancode
      process(s, aen)
      begin
            an <= "1111";
            if aen(conv_integer(s)) = '1' then
                  an(conv_integer(s))<='0';
            end if;
      end process;

-- Clock Devider
      process(clk,clr)
      begin
            if clr='1' then
                  clkdiv <=(others =>'0');
            elsif  clk'event and clk='1' then
                  clkdiv <= clkdiv + 1;
            end if;
      end process;
end x7seg;
```

Program 7. Multiplexing 7-Segmen dengan ClockDevider B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity x7segb_top is
    Port ( clk : in  STD_LOGIC;
           btn : in  STD_LOGIC_VECTOR (3 downto 0);
           sw : in  STD_LOGIC_VECTOR (7 downto 0);
           a_to_g : out  STD_LOGIC_VECTOR (6 downto 0);
```

```
            an : out  STD_LOGIC_VECTOR (3 downto 0);
            dp : out  STD_LOGIC);
end x7segb_top;

architecture x7seg_top of x7segb_top is
      component x7segb is
            Port (      x : in  STD_LOGIC_VECTOR (15 downto 0);
                        clk : in  STD_LOGIC;
                        clr : in  STD_LOGIC;
                        a_to_g : out  STD_LOGIC_VECTOR (6 downto 0);
                        an : out  STD_LOGIC_VECTOR (3 downto 0);
                        dp : out  STD_LOGIC);
      end component;
      signal x : STD_LOGIC_VECTOR (15 downto 0);
begin
-- concatenate switch and 3 bottons
      x <= sw & btn(2 downto 0) & "01010";            -- digit 0 =A
      x2 : x7segb port map
                  (x=>x, clk=>clk, clr=>btn(3), a_to_g=>a_to_g,an=>an,
dp=>dp);

end x7seg_top;
```

Program 8. Multiplexing 7-Segmen Top Modul B

4. Jalankan programnya dan upload program ke dalam chip FPGA-nya
5. Ubah program agar tampilkan di seven segmen adalah “CE” – 2 digit terakhir NRP anda.
6. Fotokan hasil tampilan seven segmen dan konfigurasi switch.

5. Bentuk dan Format Luaran

1. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.
2. Anatomi laporan :

   Judul, Nama, NRP, Kelas

   Screen capture program VHDL, program simulasi, RTL Schematics.

   Foto hasil demo program.

   Analisa dari hasil tiap project, project Lab7A, Lab7B, Lab7C1 dan Lab7C2
3. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas7-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkan

6. Mini Project

xcv

1. Buatlah program dengan masukan switch dan keluaran pada 7segmen. Dengan ketentuan sbb:
    a. Masukan biner 4 bit dari switch akan ditampilkan pada 2 buah 7segmen.
    b. 2 7segmen yang lain akan memanpilkan masukan switch + 1.

Percobaan 8 – Counter

1. Tujuan

Dapat menjelaskan rangkaian pembagi clock.

Dapat membuat rangkaian Counter modulus 10k.

2. Teori

Clock Divider

Frekuensi dan periode dari Xilinx Spartan 3 adalah 24-bit counter dengan clock sebesar 50MHz. Counter dapat digunakan untuk membagi frekuensi *f* dari clock, dimana frekuensi output *q(i)* adalah $f_i=f/2^{i+1}$. Tabel 2.1 menyatakan pembagian frekuensi.

Tabel 8.1 Clock divide frequencies

q(i)	Frekuensi (Hz)	Periode(ms)
i	50000000.00	0.00002
0	25000000.00	0.00004
1	12500000.00	0.00008
2	6250000.00	0.00016
3	3125000.00	0.00032
4	1562500.00	0.00064
5	781250.00	0.00128
6	390625.00	0.00256
7	195312.50	0.00512
8	97656.25	0.01024
9	48828.13	0.02048
10	24414.06	0.04096
11	12207.03	0.08192
12	6103.52	0.16384
13	3051.76	0.32768
14	1525.88	0.65536
15	762.94	1.31072
16	381.47	2.62144
17	190.73	5.24288
18	95.37	10.48576
19	47.68	20.97152
20	23.84	41.94304
21	11.92	83.88608
22	5.96	167.77216
23	2.98	335.54432

Counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity Counter is
    generic(N : integer :=8);
       Port ( clk : in  STD_LOGIC;
              clr : in  STD_LOGIC;
              q : out  STD_LOGIC_VECTOR (N-1 downto 0));
end Counter;

architecture Behavioral of Counter is
      signal count : std_logic_vector (N-1 downto 0);
begin
      process (clk, clr)
      begin
            if clr ='1' then
                  count <= (others => '0');
            elsif clk'event and clk='1' then
                  count <= count + 1;
            end if;
            q <= count;
      end process;
end Behavioral;
```

```
--Example 4
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;

entity clkdiv is
      port(
            mclk : in STD_LOGIC;
            clr : in STD_LOGIC;
            clk25 : out STD_LOGIC;
            clk190 : out STD_LOGIC;
            clk3 : out STD_LOGIC
      );
end clkdiv;
architecture clkdiv of clkdiv is
signal q : STD_LOGIC_VECTOR(23 downto 0);
begin
      --clock divider
      process(mclk, clr)
      begin
```

```
            if clr ='1' then
                  q <= X"000000";
            elsif mclk'event and mclk = '1' then
                  q <= q + 1;
            end if;
      end process;

      clk25 <= q(0);            --25Mhz
      clk190 <= q(17);   --190Hz
      clk3 <= q(23);            --3Hz

end clkdiv;
```

Skrip 8.1 Clock Divider

Pada skrip 8.1 adalah contoh 24-bit counter yang mempunyai 3 output, yaitu 25Mhz clock (clk25), clock 190Hz (clk190), dan clock 3Hz (clk3). Untuk memodifikasi component *clkdiv* untuk bisa menghasilkan output frekuensi yang berbeda dengan menggunakan tabel 8.1.

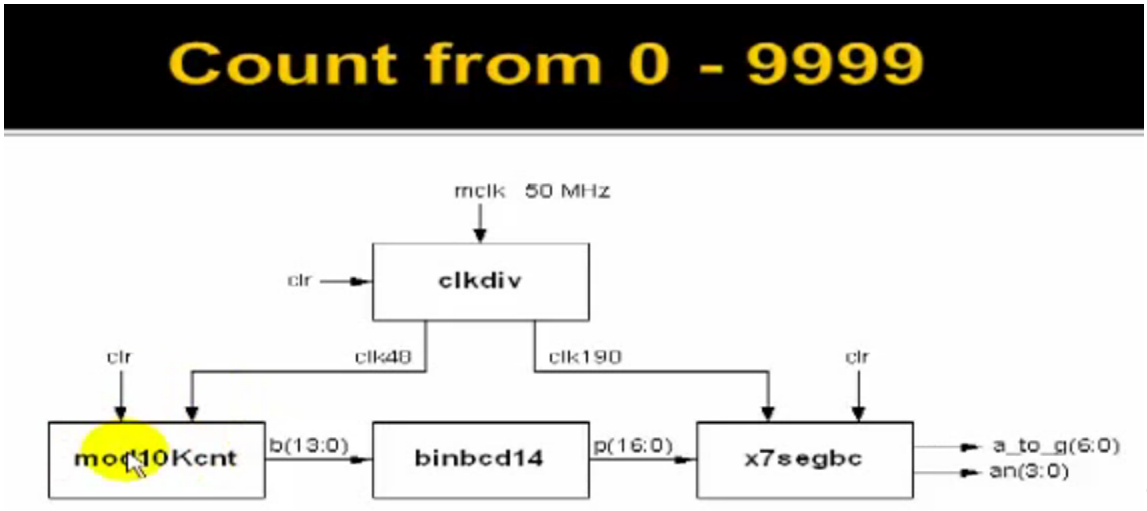
3. Peralatan

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

4. Langkah Percobaan

Pada percobaan ini akan mendisain counter 0-9999. Masukan rangkaian ini adalah mclk 50Mhz dan button Reset. Keluaran di seven segmen(a_to_g) dan enable(an(3:0)).

1. Buatlah ***new project*** dengan nama **Lab8.**
2. Tambahkan program VHDL dibawah ini.



Gambar 2.2 Counter 0-9999

Clock Devider

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Clockdiv is
	port (	Clk25Mhz: in STD_LOGIC;
		Clk: out STD_LOGIC);
	end Clockdiv;

architecture Behavior of Clockdiv is
	constant max: integer := 25000000;
	constant half: integer := max/2;
	signal count: integer range 0 to max;
begin
	process
	begin
		wait until Clk25Mhz'EVENT and Clk25Mhz = '1';
		if count < max then
			count <= count + 1;
		else
			count <= 0;
		end if;

		if count < half then
			Clk <= '0';
		else
			Clk <= '1';
		end if;
	end process;
end Behavior;
```

Clock devider 48Hz dan 190Hz

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity clkdiv is
    Port ( mclk : in  STD_LOGIC;
           clr : in  STD_LOGIC;
           clk190 : out  STD_LOGIC;
           clk48 : out  STD_LOGIC);
end clkdiv;

architecture Behavioral of clkdiv is
	signal q:std_logic_vector(23 downto 0);
begin
                                        c
	-- clock devider
```

```
	process(mclk, clr)
	begin
		if clr ='1' then
			q <= x"000000";			-- format hexadesimal
		elsif mclk'event and mclk ='1' then
			q <= q + 1;
		end if;
		clk48 <= q(19);
		clk190 <= q(17);
	end process;
end Behavioral;
```

Counter modulus 10K.

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity mod10kcnt is
    Port ( clr : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           q : out  STD_LOGIC_VECTOR (13 downto 0));
end mod10kcnt;

architecture Behavioral of mod10kcnt is
	signal count :STD_LOGIC_VECTOR (13 downto 0);
begin
	-- modulo 10k
	process(clk,clr)
	begin
		if clr = '1' then
				count <= (others => '0');
		elsif clk'event and clk ='1' then
			if	conv_integer(count) = 9999 then
				count <= (others => '0');
			else
				count <= count + 1;
			end if;
		end if;
	end process;
	q <= count;
end Behavioral;
```

Biner 14 bit to BCD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity binBCD14 is
    Port ( b : in  STD_LOGIC_VECTOR (13 downto 0);
           p : out  STD_LOGIC_VECTOR (16 downto 0));
```

```
end binBCD14;

architecture Behavioral of binBCD14 is

begin
	process(b)
		variable z : STD_LOGIC_VECTOR (32 downto 0);
	begin
		for i in 0 to 32 loop
			z(i) := '0';
		end loop;

		z(16 downto 3) := b;
		for i in 0 to 10 loop
			if z(17 downto 14) > 4 then
				z(17 downto 14) := z(17 downto 14) + 3;
			end if;
			if z(21 downto 18) > 4 then
				z(21 downto 18) := z(21 downto 18) + 3;
			end if;
			if z(25 downto 22) > 4 then
				z(25 downto 22) := z(25 downto 22) + 3;
			end if;
			if z(29 downto 26) > 4 then
				z(29 downto 26) := z(29 downto 26) + 3;
			end if;
			z(32 downto 1) := z(31 downto 0);
		end loop;
		p <= z(30 downto 14);
	end process;
end Behavioral;
```

x7segbc

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity x7segbc is
	port( x	: in std_logic_vector (15 downto 0);
		cclk : in std_logic;
		clr	: in std_logic;
		a_to_g : out std_logic_vector(6 downto 0);
		an	: out std_logic_vector (3 downto 0);
		dp	: out std_logic
		);
end x7segbc;
```

cii

```
architecture Behavioral of x7segbc is
      signal s :  std_logic_vector(1 downto 0);
      signal digit : std_logic_vector(3 downto 0);
      signal aen : std_logic_vector(3 downto 0);
begin
      dp <= '1';
      -- set aen(3 downto 0) for leading blanks
      aen(3) <= x(15) or x(14) or x(13) or x(12);
      aen(2) <= x(15) or x(14) or x(13) or x(12)
                  or x(11) or x(10) or x(9) or x(8);
      aen(1) <= x(15) or x(14) or x(13) or x(12)
                  or x(11) or x(10) or x(9) or x(8)
                  or x(7) or x(6) or x(5) or x(4);
      aen(0) <= '1';

-- Quad 4-to-1 MUX:      mux44
      process(s,x)
            begin
                  case s is
                        when "00" => digit <= x(3 downto 0);
                        when "01" => digit <= x(7 downto 4);
                        when "10" => digit <= x(11 downto 8);
                        when others => digit <= x(15 downto 12);
                  end case;
      end process;

-- 7-segment decoder: hex7seg
      process(digit)
      begin
            case digit is
                  when x"0" => a_to_g <= "0000001"; --0
                  when x"1" => a_to_g <= "1001111"; --1
                  when x"2" => a_to_g <= "0010010"; --2
                  when x"3" => a_to_g <= "0000110"; --3
                  when x"4" => a_to_g <= "1001100"; --4
                  when x"5" => a_to_g <= "0100100"; --5
                  when x"6" => a_to_g <= "0100000"; --6
                  when x"7" => a_to_g <= "0001111"; --7
                  when x"8" => a_to_g <= "0000000"; --8
                  when x"9" => a_to_g <= "0000100"; --9
                  when x"A" => a_to_g <= "0001000"; --a
                  when x"B" => a_to_g <= "1100000"; --b
                  when x"C" => a_to_g <= "0110001"; --c
                  when x"D" => a_to_g <= "1000010"; --d
```

```
                  when x"E"  => a_to_g <= "0110000"; --e
                  when others => a_to_g <= "0111000"; --f
            end case;
      end process;

-- Digit select: ancode
      process(s,aen)
      begin
            an <= "1111";
            if aen(conv_integer(s))='1' then
                  an(conv_integer(s))<='0';
            end if;
      end process;

-- 2 bit counter
      process(cclk, clr)
      begin
            if clr='1' then
                  s <= "00";
            elsif cclk'event and cclk='1' then
                  s <= s + 1;
            end if;
      end process;
end Behavioral;
```

Mod10Kcnt_top

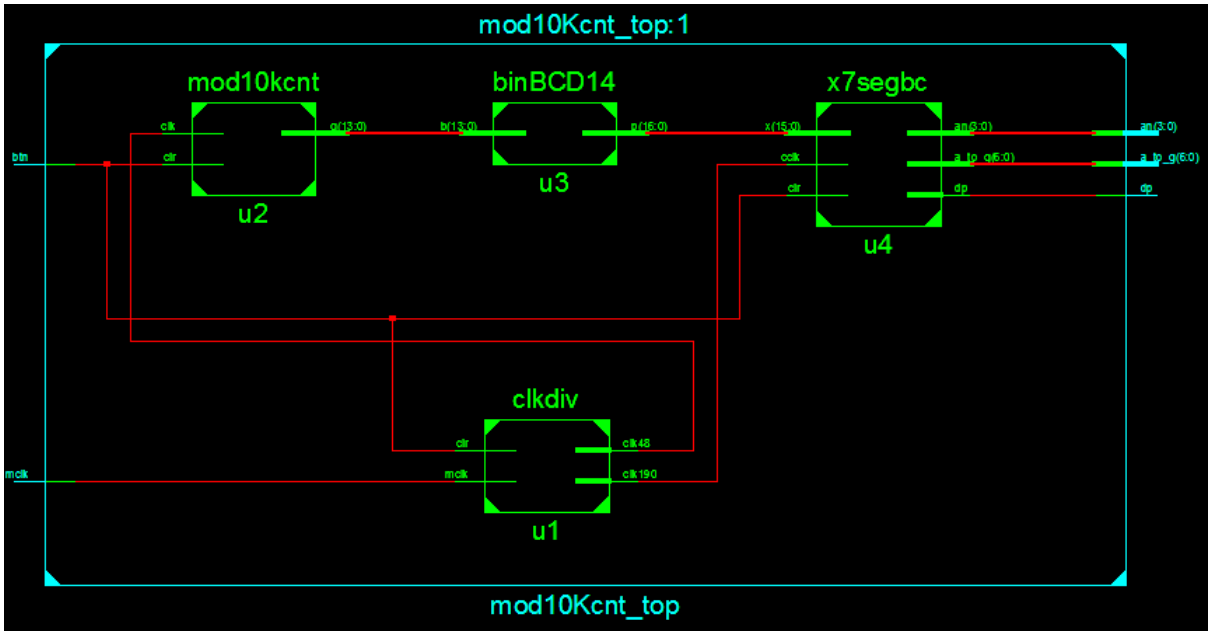
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mod10Kcnt_top is
      port( mclk  : in std_logic;
                  btn   : in std_logic;
                  a_to_g      :     out std_logic_vector(6 downto 0);
                  an    : out std_logic_vector(3 downto 0);
                  dp    : out std_logic
```

```
                );
end mod10Kcnt_top;

architecture Behavioral of mod10Kcnt_top is
      component clkdiv is
            port( mclk : in std_logic;
                  clr   : in std_logic;
                  clk190:     out std_logic;
                  clk48: out std_logic
                );
      end component;
      component mod10kcnt is
            port(       clr : in STD_LOGIC;
                        clk : in STD_LOGIC;
                        q : out STD_LOGIC_VECTOR (13 downto 0)
                        );
      end component;
      component binBCD14 is
      Port (      b : in STD_LOGIC_VECTOR (13 downto 0);
                  p : out STD_LOGIC_VECTOR (16 downto 0));
      end component;
      component x7segbc is
      port( x      : in std_logic_vector (15 downto 0);
            cclk : in std_logic;
            clr    : in std_logic;
            a_to_g : out std_logic_vector(6 downto 0);
            an     : out std_logic_vector (3 downto 0);
            dp     : out std_logic
            );
      end component;

      signal b : std_logic_vector(13 downto 0);
      signal p : std_logic_vector(16 downto 0);
      signal clr, clk48, clk190 : std_logic;

      begin
      clr <= btn;

      u1: clkdiv port map(
            mclk  => mclk,
            clr   => clr,
            clk190      => clk190,
            clk48 => clk48
      );                              cv
```

```
        u2:   mod10kcnt port map(
              clr => clr,
              clk => clk48,
              q => b
        );
        u3: binbcd14 port map(
              b => b,
              p => p
        );
        u4: x7segbc port map(
              x     => p(15 downto 0),
              cclk => clk190,
              clr    => clr,
              a_to_g => a_to_g,
              an    => an,
              dp => dp
        );

end Behavioral;
```

3. Pastikan program **Mod10Kcnt_top** menjadi top module.
4. Lakukan sintessis, sehingga menghasilkan RTL schematics seperti dibawah ini.

Hasil RTL Shcematics



5. Upload program ke board spartan, setelah sukses demokan hasilnya ke dosen pengampu.

6. Latihan

Buatlah program stop watch, dengan menggunakan push button untuk tombol start, pause, reset.

7. Bentuk dan Format Luaran

1. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.
2. Anatomi laporan :

   Judul, Nama, NRP, Kelas

   Screen capture program VHDL, program simulasi, RTL Schematics.

   Foto hasil demo program.
3. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas8-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkanLaporan

Percobaan 9 – FSM (Finite State Machine)

1. Tujuan

1. Menerapkan metode FSM dalam VHDL untuk sistem kendali pada simulasi trafic light
2. Menampilkan simulasi sistem kendali pada simulasi trafic light untuk mengetahui aliran setiap perubahan state.

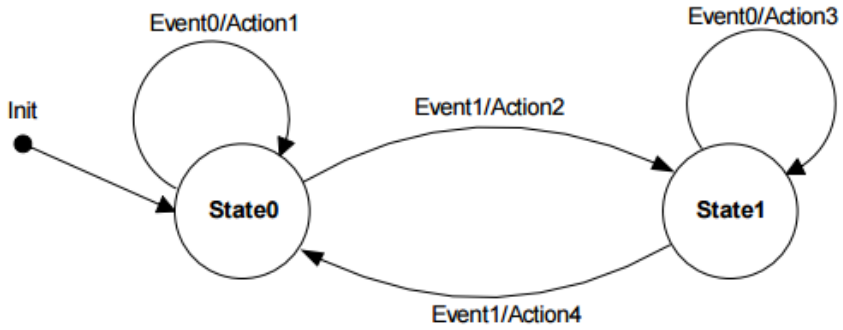
2. Teori

Finite State Machines (FSM) adalah sebuah metodologi perancangan sistem kontrol yang menggambarkan tingkah laku atau prinsip kerja sistem dengan menggunakan tiga hal berikut: State (Keadaan), Event (kejadian) dan action (aksi). Pada satu saat dalam periode waktu yang cukup signifikan, sistem akan berada pada salah satu state yang aktif. Sistem dapat beralih atau bertransisi menuju state lain jika mendapatkan masukan atau event tertentu, baik yang berasal dari perangkat luar atau komponen dalam sistemnya itu sendiri (misal interupsi timer). Transisi keadaan ini umumnya juga disertai oleh aksi yang dilakukan oleh sistem ketika menanggapi masukan yang terjadi. Aksi yang dilakukan tersebut dapat berupa aksi yang sederhana atau melibatkan rangkaian proses yang relative kompleks. Berdasarkan sifatnya, metode FSM ini sangat cocok digunakan sebagai basis perancangan perangkat lunak pengendalian yang bersifat reaktif dan real time. Salah satu keutungan nyata penggunaan FSM adalah kemampuannya dalam mendekomposisi aplikasi yang relative besar dengan hanya menggunakan sejumlah kecil item state. Selain untuk bidang kontrol, Penggunaan metode ini pada kenyataannya juga umum digunakan sebagai basis untuk perancangan protokol-protokol komunikasi, perancangan perangkat lunak game, aplikasi WEB dan sebagainya. Dalam bahasa pemrograman prosedural seperti bahasa C, FSM ini umumnya direalisasikan dengan menggunakan statemen kontrol switch case atau/dan if..then. Dengan menggunakan statemen-statemen kontrol ini, aliran program secara praktis akan mudah dipahami dan dilacak jika terjadi kesalahan logika.

DIAGRAM KEADAAN

Diagram keadaan pada dasarnya merupakan salah satu bentuk representasi dari FSM. Diagram ini secara visual menggambarkan tingkah laku yang dimiliki oleh sistem kontrol yang kompleks kedalam bentuk yang lebih sederhana dan relative mudah dipahami. Dalam diagram ini, state-state yang terdapat pada sebuah sistem digambarkan sebagai lingkaran yang diberi label unik, sedangkan transisi state yang diakibatkan oleh event tertentu direpresentasikan sebagai anak panah yang berasal dari state yang ditinggalkan menuju state yang aktif. Setiap transisi yang terjadi umumnya

juga diikuti oleh aksi yang dilakukan oleh sistem yang dirancang. Secara praktis setiap diagram state yang dirancang akan selalu memiliki sebuah transisi awal (inisial) yang menuju salah satu state sejak sistem kontrol tersebut mulai dihidupkan. Gambar 1. berikut memperlihatkan contoh penggambaran diagram state:



Gambar 1. Contoh Diagram State Sederhana

Traffic lights

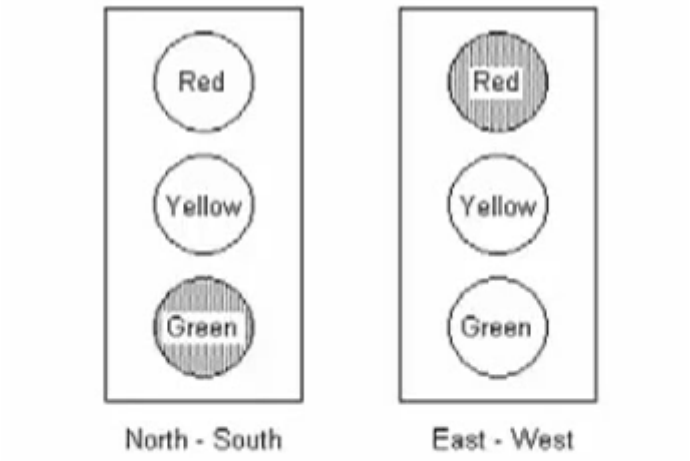
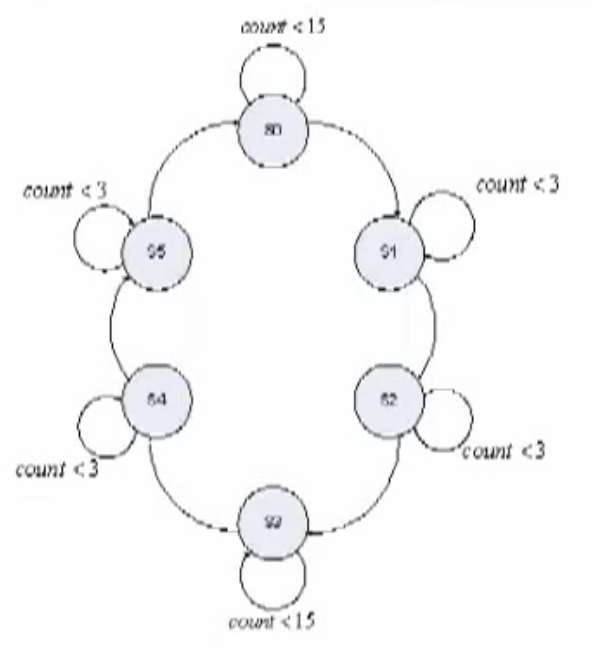


Table 8.2 Traffic Light States

State	North - South	East - West	Delay (sec.)
0	Green	Red	5
1	Yellow	Red	1
2	Red	Red	1
3	Red	Green	5
4	Red	Yellow	1
5	Red	Red	1

State Diagram for controlling Traffic Lights



3. Peralatan

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

4. Langkah percobaan

1. Buatlah ***new project*** dengan nama **Lab9.**
2. Tambahkan program VHDL dibawah ini.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity traffic is
	port( clr    : in STD_LOGIC;
		clk    : in STD_LOGIC;
		lights : out STD_LOGIC_VECTOR(5 downto 0)
	);
end traffic;

architecture Behavioral of traffic is
	type state_type is (s0,s1,s2,s3,s4,s5);
	signal state: state_type;
```

cx

```
	signal count: STD_LOGIC_VECTOR(3 downto 0);
	constant SEC5: STD_LOGIC_VECTOR(3 downto 0):= "1111";
	constant SEC1: STD_LOGIC_VECTOR(3 downto 0):= "0011";
begin
	process (clr,clk)
	begin
		if clr='1' then
				state <= s0;
				count <= X"0";
		elsif clk'event and clk='1' then
			case state is
				when s0 =>	if count < SEC5 then
							state <= s0;
							count <= count + 1;
						else
							state <= s1;
							count <= x"0";
						end if;
				when s1 => if count < SEC1 then
							state <= s1;
							count <= count + 1;
						else
							state <= s2;
							count <= x"0";
						end if;
				when s2 => if count < SEC1 then
							state <= s2;
							count <= count + 1;
						else
							state <= s3;
							count <= x"0";
						end if;
				when s3 => if count < SEC5 then
							state <= s3;
							count <= count + 1;
```

```
                                    else
                                        state <= s4;
                                        count <= x"0";
                                    end if;
                            when s4 => if count < SEC1 then
                                        state <= s4;
                                        count <= count + 1;
                                    else
                                        state <= s5;
                                        count <= x"0";
                                    end if;
                            when s5 => if count < SEC1 then
                                        state <= s5;
                                        count <= count + 1;
                                    else
                                        state <= s0;
                                        count <= x"0";
                                    end if;
                            when others =>      state <= s0;
                    end case;
            end if;
    end process;

    c2: process(state)
    begin
    case state is
            when s0 => lights <= "100001";
            when s1 => lights <= "100010";
            when s2 => lights <= "100100";
            when s3 => lights <= "001100";
            when s4 => lights <= "010100";
            when s5 => lights <= "100100";
            when others => lights <= "100001";
    end case;
    end process;
```

cxii

```
end Behavioral ;
```

3. Lakukan sintesis dan uplaod program ke board spartan-3, amati hasilnya dan dan demokan hasilnya ke dosen pengampu.

7. Latihan

1. Buatlah program dan blok diagramnya untuk kasus ini. Traffic light pada perempatan, lampu merah di tunjukan oleh segmen A, lampu kuning segmen G, dan lampu hijau segmen D,
2. Buatlah program simulasi vending machine. Masukan dari botton 1-4, botton 1=50, botton 2=100, botton 3=200,botton 4= 500. Hasil uang yang dimasukan akan ditampilkan ke seven segmen.

8. Bentuk dan Format Luaran

4. Tulisan dengan format A4, Margin 4-4-3-3, font Cambria 11, Spasi 1.15.
5. Anatomi laporan :

   Judul, Nama, NRP, Kelas

   Screen capture program VHDL, program simulasi, RTL Schematics.

   Foto hasil demo program.
6. Laporan di ditulis dalam format **.docx**, dng nama file: **Tugas8-NRP-NamaMHS**, dan diunggah pada laman edmodo.com, sesuai dg waktu yg telah ditetapkanLaporan