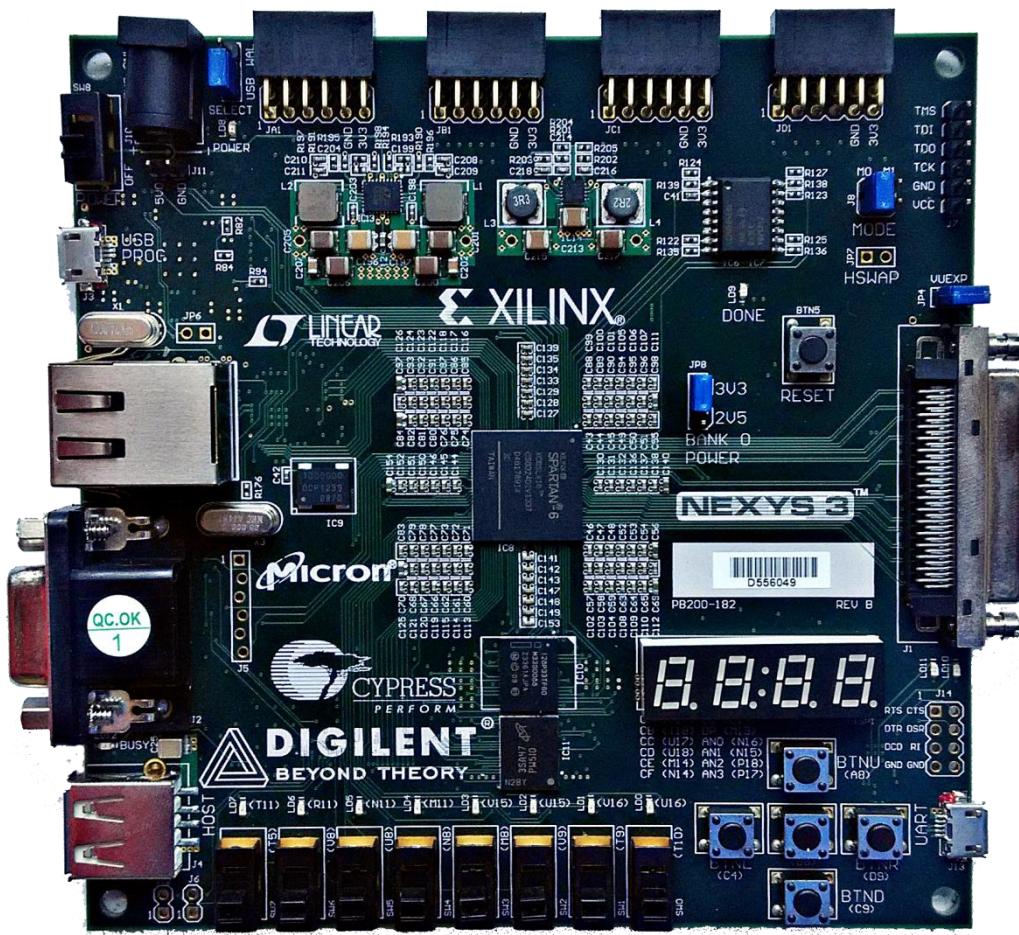


Modul Praktikum

FIELD PROGRAMMABLE GATE ARRAY

Universitas Gunadarma



Daftar Isi

BRIEFING	5
BAB 1	
VHDL	38
1.1 Mengenal VHDL.....	39
1.2 Library.....	40
1.3 Entity	41
1.4 Ports.....	42
1.5 Generics.....	42
1.6 Berkas.....	43
1.7 Packages.....	44
1.8 Loops.....	46
1.8.1 For	47
1.8.2 Exit.....	47
1.8.3 Next.....	48
1.9 Basic Variable Types and Operators	48
1.9.1 Constants.....	48
1.9.2 Signals.....	48
1.9.3 Variables	49
1.9.4 Boolean Operators	49
1.9.5 Arithmatic Operators	50
1.9.6 Shifting Functions	50
1.9.7 Concatenation	51
1.10 Type data dalam VHDL	51
1.10.1 Tipe Data Standar	51
1.10.2 Tipe Data Pengguna	52
1.11 Process dalam VHDL.....	53
1.12 Architecture dalam VHDL.....	54
1.13 Bagian Pendeklarasian Architecture	56
1.14 Bagian Pernyataan Architecture	57



BAB 2

CLOCK.....76

2.1 Mengenal Clock	77
2.2 Timer dan Counter	78
2.2.1 Tujuan Penggunaan Timer	78
2.2.2 Prescaler	78
2.3 Pulse Width Modulation(PWM)	79
2.3.1 Konsep Dasar PWM.....	80

BAB 3

FSM(Finite State Machine).....89

3.1 Mengenal Finite State Machine	90
3.2 Representasi FSM	91
3.3 Contoh Pengaplikasian FSM	92

BAB 4

SEVEN SEGMENT 106 |

4.1 Pengertian Seven Segment.....	107
4.2 LED Seven Segment	108
4.3 Prinsip Kerja Dasar Driver System pada LED Seven Segmen	110
4.4 Metode Scanning.....	111
4.4.1 Metode Scanning pada Seven Segment.....	111
4.4.2 Contoh Metode Scanning.	112

BAB 5

PUSH BUTTON.....129

5.1 Mengenal Push Button	139
5.2 Pengertian Push Button	129
5.3 Fungsi dan Penggunaan Push Button	130
5.4 Edge Detection	130

BAB 6

KEYBOARD 144 |

6.1 Keyboard USB	145
6.2 Scan Code.....	146
6.3 Sistem Bilangan Biner.....	146



BAB 7

VGA (VIDEO GRAPHIC ARRAY)	158
7.1 VGA	159
7.2 Resolusi VGA	160
7.3 DAC (Digital Analog Converter)	162

BAB 8

KOMUNIKASI SERIAL (UART)	179
8.1. Komunikasi Serial	180
8.1.1 Jenis-Jenis Komunikasi Serial	181
8.2. UART	182
8.3. Cara Kerja UART.....	184
8.4. FTDI Chip	184
8.5. PuTTY.....	185
8.6. Kode ASCII.....	185
8.6.1 Kode Standard ASCII	185
8.6.2 Kode Extended ASCII	186



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal FPGA
- Praktikan Dapat Mengetahui Cara Mengkonfigurasi FPGA
- Praktikan Dapat Memahami Pemrogramman Verilog pada FPGA
- Praktikan Mengenal

II. Dasar Teori

- Pengenalan Program Desain Schematic
- Aplikasi Perancangan Program Desain Schematic
- Pengenalan Pemrogramman Dasar Verilog

III. Peralatan

- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC

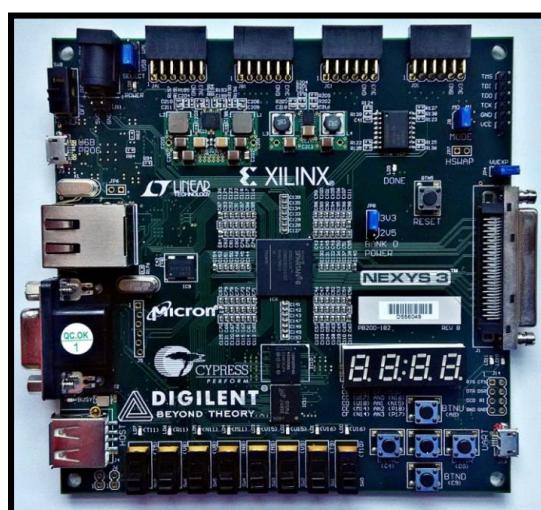


B.1 Field Programmable Gate Array (FPGA)

Field Programmable Gate Array (FPGA) merupakan sebuah IC digital yang sering digunakan untuk mengimplementasikan rangkaian digital. Bila dilihat dari segi namanya, Field Programmable dapat diartikan bahwa FPGA ini bersifat dapat dirancang sesuai dengan keinginan dan kebutuhan user/pemakai tanpa melalui tahap “burn” di laboratorium atau di “hardwire” oleh pabrik piranti. Sedangkan Gate Array artinya bahwa FPGA ini terdiri atas gerbang-gerbang digital dimana interkoneksi masing-masing gerbang tersebut dapat dikonfigurasikan antara satu sama lainnya.



Gambar B.1 Salah satu FPGA buatan Altera



Gambar B.2 Salah satu FPGA buatan Xilinx



Perlu diingat bahwa FPGA merupakan sebuah IC digital yang bersifat programmable. User/pemakai dapat memakai IC digital ini secara berulang-ulang untuk menyesuaikan program apa yang akan didownload ke dalam FPGA ini. Program tersebut nantinya akan dibuat oleh user menggunakan software yang ada untuk kemudian disimulasikan. Setelah simulasi berjalan lancar dan berhasil, program tersebut siap untuk didownload ke dalam FPGA, mudah, efisien dan sederhana. Jika program tersebut gagal untuk disimulasikan, maka user/pemakai hanya perlu menyusun ulang program dalam komputer sesuai yang dibutuhkan dan ketika siap, download lagi program tersebut ke FPGA, begitu untuk seterusnya.

Selain itu, perlu diingat juga bahwa FPGA ini bersifat volatile, yang artinya ketika sumber daya yang menyuplainya dicabut maka secara otomatis FPGA akan kehilangan fungsinya. Jadi FPGA ini tidak mampu menyimpan program ketika supply tenaganya dicabut. User/pemakai harus mendownload ulang program ke dalam FPGA lagi untuk mengimplementasikan program tersebut.

Hal-hal yang bisa digunakan dengan FPGA:

- Bisa mengisi-ulang (memprogram-ulang) FPGA sebanyak yang Anda inginkan tidak terbatas - dengan berbagai macam fungsi logik yang Anda inginkan.
- Jika melakukan kesalahan pada rancangan, cukup perbaiki kesalahan tersebut, lakukan kompilasi ulang kemudian unduh (download) lagi.
- Rancangan bisa bekerja lebih cepat dibandingkan dengan rancangan yang dibuat dengan komponen-komponen biasa, karena, dengan FPGA, hampir semua rangkaian terimplementasi di dalam chip.
- FPGA (secara umum, kecuali yang dilengkapi Flash PEROM) akan kosong saat tidak dikenai catu-daya (seperti RAM). Harus mengunduh ulang rangkaian agar bisa berfungsi kembali seperti semula.

B.2 Sejarah Singkat FPGA

FPGA ini dikembangkan sejak tahun 1984 oleh perusahaan Xilinx yang berbasis di San Jose, CA. Perkembangan selanjutnya, FPGA ini mulai diproduksi oleh beberapa perusahaan



misalnya, Altera, Lattice, dan Quicklogic. Diantara perusahaan-perusahaan tersebut, terdapat 2 perusahaan yang mendominasi produksi FPGA di seluruh dunia yaitu Xilinx dan Altera.

Ada 5 perusahaan besar yang memproduksi FPGA. Dua yang pertama merupakan pemain utama di pasar FPGA:

- Xilinx yang punya nama besar dalam dunia FPGA, masih memimpin dalam densitas dan teknologi.
- Altera merupakan pemain kedua terkenal di dunia FPGA, terkenal dengan namanya.
- Lattice, Actel, Quicklogic adalah perusahaan-perusahaan yang lebih kecil dan punya “pasar khusus”.

B.3 Isi FPGA

Seperti yang diketahui, Field Programmable Gate Array atau yang lebih dikenal dengan FPGA merupakan sebuah IC digital yang dapat diprogram sesuai dengan kehendak pemakainya. FPGA berkembang sejak sekitar tahun 1980-an. Awal mula dari adanya FPGA ini tidak lepas dari adanya alat elektronik, yaitu transistor. Transistor merupakan alat semikonduktor yang dipakai sebagai penguat, sebagai komponen pemutus dan penyambung(switching), dan stabilisasi tegangan. Pada umumnya, transistor ini berfungsi sebagai saklar arus listrik yang hendak masuk ke suatu rangkaian elektronik.

Perkembangan selanjutnya, adalah munculnya IC (Integrated Circuit) sekitar tahun 1950-an. Komponen elektronik ini lebih dikenal sebagai otak dari sebuah peralatan elektronika seperti televisi, handphone, computer, dsb. Kemudian berturut-turut muncul komponen lain, seperti DRAM, SRAM, microprocessor, ASIC hingga sampai pada FPGA. Bila dilihat dari segi bentuknya, FPGA tak berbeda jauh dengan bentuk IC-IC lainnya. Hanya saja, bila dilihat dari isinya FPGA memiliki bagian yang berbeda dengan komponen IC pada umumnya.

Berikut isi dari FPGA pada umumnya:

1. Configure Logic Blocks (CLB). Bisa dikatakan, bagian inilah yang akan memproses segala bentuk rangkaian logika yang dibuat oleh user/pemakai.
2. I/O Blocks. Sebagai interface antara external pin dari device dan internal user logic
3. Programmable interconnect. Bagian ini berisi wire segments dan programmable switches, selain itu bagian ini juga akan menghubungkan antara CLB satu dengan CLB



lainnya.

Sebagian besar FPGA memiliki arsitektur seperti di atas. Entah itu FPGA produksi Xilinx ataupun dari Altera. Bila berbeda pun tak akan sampai menimbulkan perbedaan yang terlalu signifikan.

B.4 Konfigurasi FPGA

Sebuah FPGA (Field Programmable Gate Array) dapat terbagi atas dua kondisi yaitu kondisi awal (**configuration mode**) dan kondisi pemakai (**user mode**). Ketika pertama kali FPGA dihidupkan, maka otomatis keadaan FPGA ini berada dalam kondisi awal. Hal ini disebabkan karena FPGA masih dalam keadaan awal, fresh dan belum terdapat suatu program apa pun di dalamnya. Untuk menggunakan FPGA tersebut maka perlu adanya proses download program oleh pemakainya (user). Proses mendownload program ke dalam FPGA bertujuan untuk mengirimkan berkas-berkas bilangan biner (0 dan 1) melalui beberapa pin khusus. Setelah proses download dilakukan dan FPGA kemudian telah siap digunakan, maka inilah yang sering disebut kondisi pemakai. Pada kondisi ini FPGA telah aktif dan program yang didownload ke dalamnya dapat digunakan. Inilah yang dinamakan konfigurasi atau pengaturan pada FPGA.

Bila pada kondisi pemakai, maka otomatis di dalam FPGA telah terdapat program. Program tersebut dibuat, diuji dan disimulasikan dahulu menggunakan PC (komputer). Setelah itu baru proses **download** program dilakukan. Pada umumnya terdapat 3 cara yang biasa digunakan untuk mendownload program ke dalam FPGA yaitu :

1. Pemakai dapat mendownload langsung program ke dalam FPGA menggunakan kabel yang disambungkan ke PC. Program tersebut dibuat dan diolah menggunakan PC (komputer) dan ketika siap digunakan, baru didownload ke FPGA melalui kabel.
2. Pemakai dapat menggunakan mikrokontroler pada board yang ada, dengan firmware yang cukup guna mengirimkan data ke dalam FPGA.
3. Pemakai dapat menggunakan “boot-PROM” pada board yang ada, yang dihubungkan ke FPGA, dan mengatur FPGA tersebut supaya secara otomatis bekerja tanpa proses download dulu ke dalamnya (Perusahaan-perusahaan FPGA pada umumnya memiliki spesifikasi khusus untuk tambahan boot-PROMS).

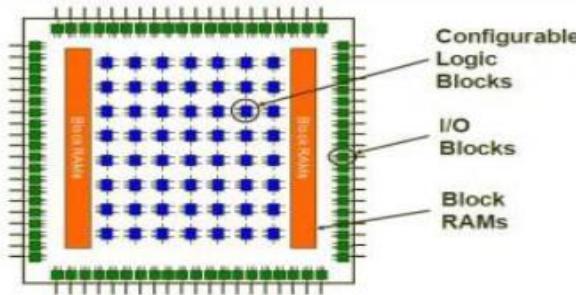


Dari ketiga cara di atas, cara pertama adalah yang paling sering dan banyak digunakan. Selain itu, cara pertama cukup mudah dan efisien untuk dilakukan. Jadi, ketika pemakai telah mendownload program ke dalam FPGA maka tidak lagi diperlukan PC (komputer). FPGA akan bekerja sesua dengan program yang telah didownload ke dalamnya tanpa perlu lagi terhubung dengan PC (komputer).

Proses konfigurasi FPGA untuk buatan Xilinx maupun Altera hampir sama. Perbedaanya, hanya masalah pemberian nama pada pin-pin FPGA tersebut (nama pin dan jenis operasinya berbeda). Namun, sebagian besar fungsi dan kegunaannya sama.

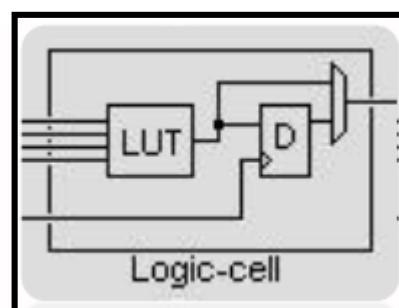
B.5 Cara kerja FPGA

FPGA (Field Programmable Gate Array) merupakan sebuah IC digital sering digunakan untuk implementasi rangkaian digital. IC digital ini pada umumnya terdiri atas 3 bagian yaitu configure logic blocks (CLB), I/O Blocks, dan Programmable Interconnect. Baik FPGA buatan Xilinx maupun Altera memiliki 3 bagian seperti yang disebutkan di atas.



Gambar B.3 Isi FPGA

Sebuah FPGA tersusun dari sebuah bagian yang bernama “*logic-cell*” (Logic Blocks), yang kemudian pada perkembangannya saling terhubung satu sama lain. Kumpulan-kumpulan dari *logic cell* ini berjumlah ratusan bahkan ribuan dan membentuk saatu fungsi yang kompleks. Sebuah logic cell pada dasarnya terdiri atas sebuah *lookup table* (LUT), D flip-flop, dan sebuah multiplekser 2 ke 1.



Gambar B.4 Isi logic cell



1. **Look Up Table** (LUT) merupakan sejenis RAM (**Random Acces Memory**) yang berkapasitas kecil. Di dalam FPGA, LUT ini memegang peranan penting dalam proses implementasi fungsi-fungsi logika. Selain itu, LUT ini berciri khas memiliki input sejumlah 4 buah.
2. D Flip Flop. Seperti yang diketahui, flip-flop (**Bistable Multivibrator**) adalah suatu rangkaian sel biner yang memiliki dua buah output yang saling berkebalikan keadaannya (0 atau 1). Di dalam FPGA, terdapat sebuah jenis flip-flop yaitu D flip-flop atau Data flip flop. Rangkaian D flip-flop ini berfungsi sebagai rangkaian logika sekuensial dimana di dalamnya terdapat peralatan memori dan pewaktu.
3. Multiplekser 2 ke 1. Sebuah multipleser adalah piranti digital yang bekerja sebagai switch (saklar) yang menghubungkan data dari n masukkan ke sebuah keluaran. Multiplekser berfungsi untuk memilih beberapa input untuk hanya menjadi 1 output saja. Di dalam FPGA, terdapat rangkaian multiplekser 2 ke 1 yang artinya, multiplekser tersebut memiliki 2 buah input dan 1 buah output.

Setiap **logic cell** tersebut dapat dihubungkan dengan logic cell lainnya melalui jalur/koneksi yang ada. Setiap cell hanya mampu bekerja secara sederhana dan ringkas, Namun bila antara satu cell saling terhubung satu sama lain sebuah fungsi-fungsi logika yang kompleks pun dapat terbentuk.

B.6 Design Software FPGA

Field Programmable Gate Array (FPGA) merupakan sebuah IC digital yang sering digunakan untuk mengimplementasikan rangkaian digital. Jadi, bentuk utama sebuah FPGA adalah Integrated Circuit (IC). Dua buah perusahaan yang cukup terkenal sebagai pembuat FPGA adalah Xilinx dan Altera. Pada umumnya, perusahaan tersebut tidak menjual FPGA dalam bentuk terpisah yakni keping IC saja melainkan sudah terintegrasi dengan komponen-komponen elektronik lain seperti kristal, LED, resistor, kapasitor, push button, dsb. Semuanya itu biasanya menjadi satu dan disebut papan pengembang FPGA (**FPGA development Board**). Bentuk inilah yang biasanya dijual oleh para vendor pembuat FPGA.

Perusahaan-perusahaan pembuat FPGA kemudian tidak hanya papan pengembang FPGA saja. Akan tetapi, perusahaan tersebut juga menjual sekaligus perangkat lunak (**software**) yang akan mendukung penggunaan. Jadi ketika kita membeli FPGA maka yang



dimaksud adalah membeli ***development board FPGA*** beserta perangkat lunak pendukungnya. Kedua bagian ini tidak bisa terpisah. Tidak bisa hanya membeli papan pengembang FPGA saja atau sebaliknya, perangkat lunaknya saja. Kedua bagian tersebut merupakan satu kesatuan utuh yang saling mendukung proses kerja satu sama lain. Secara garis besar, perancangan perangkat lunak (software) terdiri atas 4 tahap yakni :

- ***Design-entry.***
- ***Simulation.***
- ***Synthesis and place-and-route.***
- ***Programming through special cables (JTAG).***

B.7 Software FPGA

Perusahaan pembuat FPGA pada umumnya memberikan perangkat lunak secara cuma-cuma alias gratis. Software ini digunakan untuk mendukung proses ***design entry, simulation, synthesis and place-and-route***, dan ***Programming through special cables (JTAG)***. Biasanya software yang dibagikan secara gratis hanya untuk jenis FPGA tingkat rendah-menengah atau hanyalah berupa demo saja. Sedangkan untuk jenis FPGA tingkat atas maka software pendukungnya tidaklah gratis melainkan berbayar. Berikut beberapa software pendukung yang gratis atau demo bagi FPGA :

- Perusahaan Xilinx terkenal dengan software miliknya yang bernama **ISE WebPack**
- Perusahaan Altera terkenal dengan software miliknya yang bernama **Quartus II Web Edition**

Software di atas cukup bagus dan baik untuk memulai belajar menggunakan FPGA karena software tersebut memiliki fungsi yang hampir sama dengan yang berbayar. Selain itu, saat ini dengan fungsi yang cukup dan memadai maka tidak perlu membeli software yang ada secara lengkap. Berikut cara memulai memrogram FPGA:

1. Download software yang dibutuhkan.
2. Instal software tersebut
3. Meminta license untuk mengaktifkan software melalui email. Tanpa adanya license maka software yang telah terinstal tidak akan dapat digunakan.

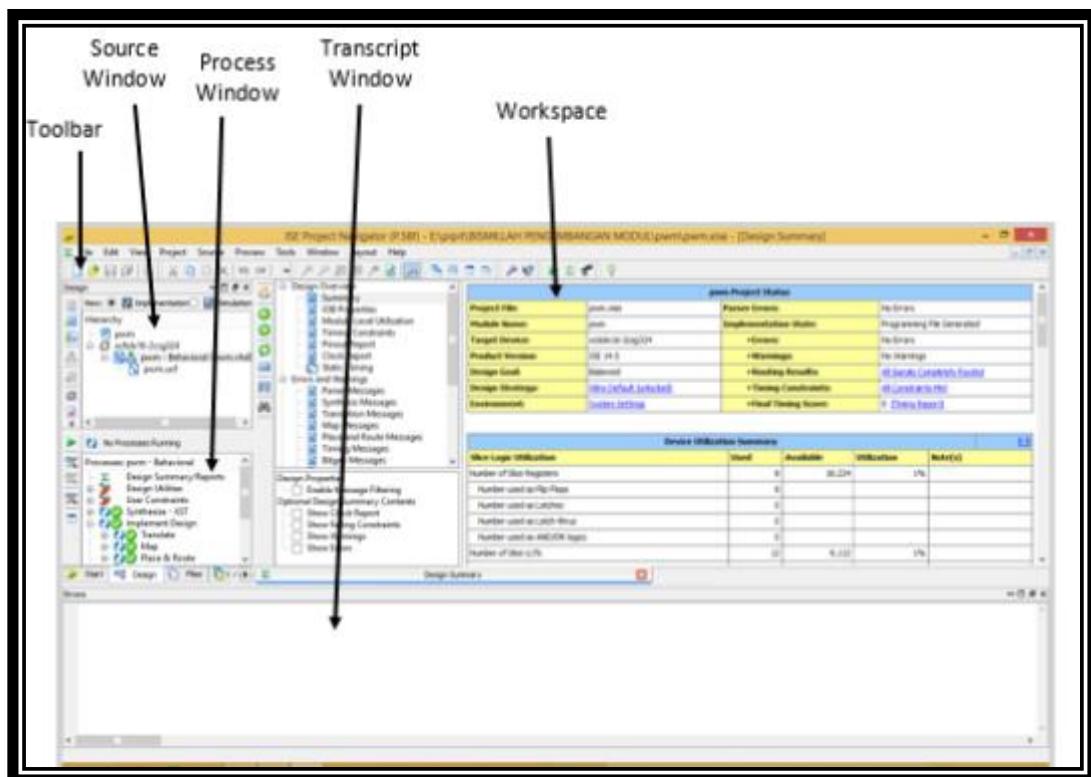


B.7.1 Perangkat Lunak Xilinx

Xilinx (Xilinx Foundation Series) adalah suatu perangkat lunak yang berguna untuk merancang dan mensimulasikan suatu rangkaian digital. Dengan menggunakan Xilinx proses perancangan suatu alat atau rangkaian digital melalui proses simulasi rangkaian yang telah dirancang untuk melihat apakah rancangan yang telah dibuat sudah benar atau masih mengandung kesalahan. Untuk perancangan rangkaian digital, Xilinx mempunyai tiga cara, yaitu dengan menggunakan State Diagram, HDL (Hardware Description Language) dan Schematic. Dalam perancangan bisa menggunakan salah satu cara saja atau menggabungkan ketiga cara tersebut. Untuk HDL, Xilinx dapat menggunakan dua bahasa pemrograman yaitu ABEL dan VHDL.

B.7.2 Integrated Software Environment (ISE) Overview

Gambar berikut menampilkan jendela Proyek Navigator utama, yang memungkinkan Anda untuk mengatur desain Anda dimulai dari entri desain melalui konfigurasi perangkat.



Gambar B.5: Jendela proyek navigator pada aplikasi ISE Design Suite 14.5



B.7.2.1 Jenis – jenis Proses

➤ Tasks

Ketika Anda menjalankan proses task, perangkat lunak ISE berjalan dalam "modus batch," yaitu, perangkat lunak proses file sumber Anda, tetapi tidak membuka perangkat lunak tambahan di Workspace itu. Output dari proses muncul di jendela Transkrip.

➤ Report

Tugas yang paling menyertakan laporan sub-proses, yang menghasilkan laporan ringkasan atau status, misalnya, Laporan Sintesis atau Peta Laporan. Ketika Anda menjalankan proses laporan, laporan itu muncul dalam Workspace tersebut.

➤ Tools

Ketika Anda menjalankan proses alat, alat terkait meluncurkan dalam mode standalone atau muncul dalam Workspace mana Anda dapat melihat atau memodifikasi file source desain Anda.

B.7.2.2 Proses Status

➤ Running

Ikon ini menunjukkan bahwa proses sedang berjalan

➤ Up-to-date

Ikon ini menunjukkan bahwa proses berhasil berjalan dengan tidak ada kesalahan atau peringatan dan tidak perlu mengulangi. Jika ikon di samping proses laporan, laporan yang up-to-date, namun, tugas-tugas terkait dapat memiliki peringatan atau kesalahan. Jika hal ini terjadi, Anda dapat membaca laporan tersebut untuk menentukan penyebab dari peringatan atau kesalahan.

➤ Warning reported

Ikon ini menunjukkan bahwa proses itu berjalan berhasil tetapi ada peringatan yang muncul.

➤ Error reported

Ikon ini menunjukkan bahwa proses itu berjalan , tetapi mengalami error dan harus di perbaiki.



➤ Out-of-date

Ikon ini menunjukkan bahwa Anda membuat perubahan desain, yang mengharuskan proses dijalankan ulang. Jika ikon ini berada di samping proses laporan, Anda dapat menjalankan kembali proses tugas yang terkait untuk membuat versi up-to-date laporan.

B.8.2 Metode Menggunakan *Hardware Language Description/HDL*

Metode yang lain untuk perancangan rangkaian adalah metode menggunakan Bahasa Deskripsi Perangkat Keras (*Hardware Description Language*/HDL). Nantinya tiap-tiap komponen serta jalur yang menghubungkannya akan dideskripsikan lewat tulisan atau kode tertentu. Seperti yang telah disinggung pada artikel sebelumnya, tiap vendor FPGA memiliki aturan mengenai penggunaan kode dalam hal implementasi di dalam FPGA. Namun, sejak sekitar 10 tahun lalu, telah muncul kode baru yang dapat diimplementasikan ke dalam semua jenis FPGA buatan vendor manapun. Kode baru tersebut ada 2 yakni **verilog** dan **VHDL**. Baik verilog maupun VHDL ternyata lebih terkenal karena mudah dipahami dan dimengerti. Selanjutnya dua kode ini kemudian menjadi acuan utama dalam proses implementasi rancangan rangkaian ke dalam FPGA (apapun jenis vendornya). Hingga saat ini, metode perancangan menggunakan HDL (baik verilog maupun VHDL) lebih banyak digunakan daripada metode *schematic*. Selanjutnya, mempelajari HDL sangatlah penting dan dibutuhkan terutama bagi mereka yang serius ingin terjun di dalam dunia FPGA. Selain itu, HDL kini telah menjadi acuan utama dalam dunia industri sehingga tidak ada ruginya bila kita ingin mempelajarinya.

B.8.2.1 VHDL

VHDL (Very high speed integrated Hardware Description Language) adalah sebuah bahasa pemrograman VHSIC (Very High Speed Intregated Circuit) yang dikembangkan oleh IEEE (Institute of Electrical and Electronic Engineering). Pada VHDL, konsep serta *syntax* banyak diperlukan untuk mengerti bagaimana rancangan VHDL sebagai bagian dari pemrograman FPGA. Dalam kebanyakan kasus, keputusan memilih dan menggunakan kode



VHDL daripada kode **Verilog** atau **SystemC**, sangat tergantung pada pilihan perancang itu sendiri dan lebih kepada ketersediaan software pendukung serta kebutuhan perusahaan.

B.8.2.2 VERILOG

Verilog adalah sebuah bahasa yang termasuk HARDWARE DESCRIPTION LANGUAGE (HDL). Dimana bahasa yang digunakan untuk menggambarkan sistem digital pada suatu perangkat keras. Verilog seperti bahasa deskripsi perangkat keras lainnya, memungkinkan desainer untuk merancang sebuah desain dalam dua metodologi, yaitu Bottom-up dan top-down metodologi.

- **Bottom-Up Design**

Pada Bottom-Up, Setiap desain dilakukan pada level gerbang menggunakan gerbang dasar standar. Dengan meningkatnya kompleksitas desain rancangan digital saat ini dengan sistem baru yang terdiri dari mikroprosesor dengan kompleksitas ribuan transistor teknik ini hampir mustahil untuk dipertahankan.

- **Top-Down Design**

Top-down design gaya desain yang hampir selalu dipakai dari semua desainer program verilog. Top-down sesungguhnya memungkinkan pengujian desain awal, mudah melakukan perubahan teknologi yang berbeda, memungkinkan sebuah desain sistem terstruktur dan menawarkan banyak keuntungan lain. Tapi sangat sulit untuk mengikuti murni desain top-down. Karena fakta ini sebagian besar programmer Verilog menggunakan desain campuran dari kedua metode, dan menerapkan beberapa elemen penting dari kedua desain.

Verilog mendukung desain pada berbagai tingkat abstraksi yaitu:

1. Behaviorial Level (Tingkat perilaku)

Tingkat ini menggambarkan sebuah sistem menggunakan algoritma bersamaan (Perilaku). Tiap algoritma itu sendiri adalah berurutan, yang berarti terdiri dari satu set instruksi yang dilaksanakan satu demi satu.

2. Register-Transfer Level (RTL)

Desain menggunakan Register-Transfer level menentukan karakteristik rangkaian dari operasi dan transfer data antara register. Desain RTL berisi kemungkinan waktu yang tepat. Operasi dijadwalkan untuk terjadi pada waktu-waktu tertentu.

3. Gate Level



Dalam tingkat logika karakteristik dari sistem dijelaskan oleh logis link dan sifat waktu sistem atau Semua sinyal diskrit sistem. Sistem hanya memiliki nilai-nilai logis tertentu ('0', '1', 'X', 'Z'), Operasi yang dapat dimanfaatkan standar logika primitif (AND, OR, NOT dll gerbang)

B.8.2.2.1 Pemrogramman Verilog

Verilog adalah sebuah bahasa yang termasuk **HARDWARE DESCRIPTION LANGUAGE (HDL)**. Dimana bahasa yang digunakan untuk menggambarkan sistem digital pada suatu perangkat keras. Verilog seperti bahasa deskripsi perangkat keras lainnya, memungkinkan desainer untuk merancang sebuah desain dalam dua metodologi, yaitu Bottom-up dan top-down metodologi.

- **Bottom-Up Design**

Pada Bottom-Up, Setiap desain dilakukan pada level gerbang menggunakan gerbang dasar standar. Dengan meningkatnya kompleksitas desain rancangan digital saat ini dengan sistem baru yang terdiri dari mikroprosesor dengan kompleksitas ribuan transistor teknik ini hampir mustahil untuk dipertahankan.

- **Top-Down Design**

Top-down design gaya desain yang hampir selalu dipakai dari semua desainer program verilog. Topdown sesungguhnya memungkinkan pengujian desain awal, mudah melakukan perubahan teknologi yang berbeda, memungkinkan sebuah desain sistem terstruktur dan menawarkan banyak keuntungan lain. Tapi sangat sulit untuk mengikuti murni desain top-down. Karena fakta ini sebagian besar programmer Verilog menggunakan desain campuran dari kedua metode, dan menerapkan beberapa elemen penting dari kedua desain.

Verilog mendukung desain pada berbagai tingkat abstraksi yaitu:

- **Behavioral Level (Tingkat perilaku)**

Tingkat ini menggambarkan sebuah sistem menggunakan algoritma bersamaan (Perilaku). Tiap algoritma itu sendiri adalah berurutan, yang berarti terdiri dari satu set instruksi yang dilaksanakan satu demi satu.

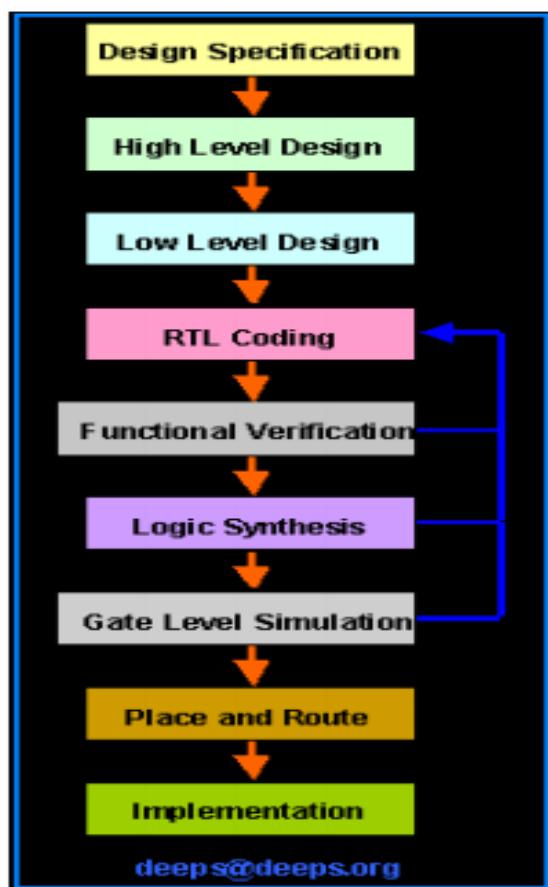
- **Register-Transfer Level (RTL)**



Desain menggunakan Register-Transfer level menentukan karakteristik rangkaian dari operasi dan transfer data antara register. Desain RTL berisi kemungkinan waktu yang tepat. Operasi dijadwalkan untuk terjadi pada waktu-waktu tertentu.

- **Gate Level**

Dalam tingkat logika karakteristik dari sistem dijelaskan oleh logis link dan sifat waktu sistem atau semua sinyal diskrit sistem. Sistem hanya memiliki nilai-nilai logis tertentu ('0 ',' 1', 'X ',' Z '), Operasi yang dapat dimanfaatkan standar logika primitif (AND, OR, NOT, dll). Menggunakan tingkat gerbang pemodelan mungkin bukan ide yang baik untuk setiap tingkat desain logika. Tingkat Gate kode yang dihasilkan oleh alat bantu seperti alat sintesis dan netlist ini digunakan untuk tingkat gerbang simulasi dan untuk backend.



Gambar B.6 Contoh desain alur rancangan program Verilog

- **Simulasi**

Simulasi adalah proses verifikasi karakteristik fungsional model pada setiap tingkat abstraksi. Kami menggunakan simulator untuk mensimulasikan Hardware model. Untuk



menguji apakah kode RTL memenuhi persyaratan fungsional spesifikasi dan melihat apakah semua blok RTL fungsional benar. Untuk mencapai ini kita perlu menulis testbench, yang menghasilkan CLK, reset dan tes yang diperlukan vektor

- **Sintesis**

Sintesis adalah proses di mana desain alat seperti compiler atau Synplify mengambil RTL di Verilog atau VHDL. Sintesis adalah alat pemetaan RTL untuk setiap program, juga melakukan minimal waktu analisis untuk melihat apakah desain yang dipetakan memenuhi persyaratan atau tidak.

- **Verilog HDL Syntax and Semantics**

Konvensi leksikal dasar yang digunakan oleh Verilog HDL serupa dengan yang diprogrammam bahasa C. contoh bahasa Verilog:

<pre>module addbit(a,b,ci,sum,co); input a,b,ci;output sum co; wire a,b,ci,sum,co;</pre>	<pre>module addbit (a, b, ci, sum, co); input a; input b; input ci; output sum; output co; wire a; wire b; wire ci; wire sum; wire co;</pre>
Never write code like this.	Nice way to write code.

Gambar B.7 Contoh desain alur rancangan program Verilog

- **Identifier**

Identifier adalah nama yang digunakan untuk memberikan suatu objek, misalnya mendaftar atau modul, nama sehingga dapat direferensikan dari tempat-tempat lain dalam deskripsi. Ciri-ciri Identifier:

- Identifier harus dimulai dengan sebuah karakter abjad atau menggaris bawahi karakter (a-z A-Z _)



- Pengidentifikasi dapat mengandung huruf, angka karakter, dengan underscore, dan tanda dolar (az AZ 0-9 _ \$)
- Pengidentifikasi dapat berisi hingga 1024 karakter lama.

Verilog HDL memungkinkan setiap karakter yang akan digunakan dalam suatu pengenal dapat melarikan diri dari pengenal (tidak teridentifikasi). Lelos pengidentifikasi menyediakan sarana termasuk salah satu karakter ASCII dicetak dalam sebuah identifier (nilai desimal 33 melalui 126, atau 21 melalui 7E dalam heksadesimal). Hal-hal yang membuat karakter tidak teridentifikasi adalah:

- Seluruh pengidentifikasi lolos oleh slash belakang (\) karena dapat terdeteksi sebagai keterangan
- Lelos identifikasi diakhiri dengan spasi, Karakter seperti koma, tanda kurung, dan titik koma menjadi bagian dari pengenal lelos kecuali didahului oleh spasi.
- Mengakhiri lelos pengidentifikasi dengan spasi, kalau tidak karakter yang harus mengikuti pengenal dianggap sebagai bagian dari program.'

Hal-hal yang menyebabkan suatu program lolos dari identifikasi dapat terjadi. Sehingga berhasil di check sintaks. Namun pada saat di Implementasi kemungkinan menjadi error. Contoh : karakter yang lolos atau tidak teridentifikasi:

```
\486_up \Q~ \1,2,3 \reset*
module \486 (q,\q~,d,clk,\reset*);
```

B.8.2.2 Angka dalam Verilog

Kita dapat menetapkan angka konstan dalam desimal, heksadesimal, oktal, atau biner format. Angka negatif direpresentasikan dalam 2's complement. Pada karakter underscore (_) harus digunakan di mana saja dalam nomor kecuali sebagai karakter pertama, di mana ia diabaikan.

➤ Angka Integer

Verilog HDL memungkinkan angka integer untuk ditetapkan sebagai:

- Ukuran atau angka unsized (Unsized ukuran adalah 32 bit)
- Dalam radix biner, oktal, desimal, atau heksadesimal
- Kasus dan radix adalah digit hex (a, b, c, d, e, f) tidak peka
- Spasi antara ukuran, radix dan nilai Sintaks: <size> '<radix> <value>



Contoh angka integer:

Integer	Stored as	Description
1	00000000000000000000000000000001	unsized 32 bits
8'hAA	10101010	sized hex
6'b10_0011	100011	sized binary
'hF	00000000000000000000000000000001111	unsized hex 32 bits

Gambar B.8 Contoh angka Integer

➤ **Strings**

String adalah deretan karakter yang dilingkupi oleh tanda kutip ganda dan semua terdapat pada satu baris. String digunakan sebagai Operand dalam ungkapan yang digunakan sebagai urutan delapan-bit nilai ASCII, dengan sebuah delapan-bit nilai ASCII mewakili satu karakter. Verilog tidak menyimpan string penghentian karakter. String dapat dimanipulasi dengan menggunakan standar operator.

B.8.2.2.3 Operator Verilog

Berikut ini adalah operator yang digunakan dalam bahasa pemrograman Verilog :

➤ **Operator Arithmetic**

- Binary: +, -, *, /, % (modulus operator)
- Unary: +, -
- Pembagian integer memotong setiap bagian fraksional
- Hasil dari operasi modulus mengambil tanda operand pertama
- Jika ada nilai bit operand adalah nilai yang diketahui x, maka seluruh hasil nilainya x

➤ **Operator Relational**

- $a < b$ yang kurang b
- $a > b$ yang lebih besar daripada b
- $a \leq b$ kurang dari atau sama dengan b
- $a \geq b$ yang lebih besar dari atau sama dengan b
- Hasilnya adalah nilai skalar \square
- Jika relasi adalah palsu
- 1 jika hubungan benar
- x jika salah satu dari Operand telah diketahui x bit



- Catatan: Jika nilai x atau z, maka hasil tes itu palsu

➤ Operator Relational

- $a == b$ a sama dengan b, termasuk x dan z
- $a != b$ yang tidak sama dengan b, termasuk x dan z
- $a == b$ a sama dengan b, sehingga kemungkinan tidak diketahui
- $a != b$ yang tidak sama dengan b, hasil kemungkinan tidak diketahui
- Operand dibandingkan sedikit demi sedikit, dengan mengisi nol jika kedua Operand tidak memiliki panjang yang sama
- Hasilnya adalah 0 (false) atau 1 (true)
- Untuk $==$ and $!=$ Operator yang hasilnya adalah x, jika salah satu operand berisi nilai x atau z
- Untuk $==$ and $!=$ Operator
- bit dengan x dan z yang dimasukkan dalam perbandingan dan harus
- cocok untuk hasil benar hasilnya selalu 0 atau 1

➤ Operator Logika

- $!$ Logika negasi
- $\&$ & Logis and
- $\|$ Logis atau
- Ekspresi terhubung oleh $\&$ & and $\|$ dievaluasi dari kiri ke kanan
- Evaluasi berhenti segera setelah diketahui hasilnya
- Hasilnya adalah nilai scalar:
- 0 jika relasi adalah False
- 1 jika relasi adalah True
- x jika salah satu dari Operand telah diketahui x bit

➤ Operator Bit - wise

- \sim Negasi
- $\&$ And
- $\|$ Inklusif Or



- \wedge Eksklusif Or
- $\wedge \sim$ Or $\sim \wedge$ eksklusif Nor
- Perhitungan bit tidak diketahui, dengan cara sebagai berikut:
- $x = x$
- $0 \& x = 0$
- $1 \& x = x \& x = x$
- $1 | x = 1$
- $0 | x = x | x = x$
- $0 \wedge x = 1 \wedge x = x \wedge x = x$
- $0 \wedge \sim x = 1 \wedge \sim x = x \wedge \sim x = x$

➤ Operator Reduksi

$\&$	and
$\sim \&$	nand
$ $	or
$\sim $	nor
\wedge	xor
$\wedge \sim \text{ or } \sim \wedge$	xnor

Gambar B.9 Contoh Gerbang Logika

- Pengurangan operator unary.
- Mereka melakukan sedikit-operasi bit-wise pada satu operand untuk menghasilkan hasil bit tunggal.
- Pengurangan NAND dan NOR unary operator beroperasi sebagai AND dan OR, tetapi dengan menegaskan output.
- Unknown bit diperlakukan seperti yang dijelaskan sebelumnya.

➤ Operator Shift (pergeseran)

- << Left shift (pergeseran kekiri)
- >> Right shift (pergeseran kekiri)
- Operand kiri digeser oleh jumlah posisi bit yang diberikan oleh operand kanan.
- Bit posisi yang ditinggalkan diisi dengan nol.



➤ Operator Concatenation (rangkaian)

- Concatenations dinyatakan menggunakan penjepit karakter (dan), dengan koma memisahkan ekspresi dalam

➤ Contoh:

(a, b [3:0], c, 4'b1001) // jika a dan c adalah 8-bit, memiliki hasil 24 bit

- Angka konstan Unsized tidak diperbolehkan dalam concatenations
- Pengulangan pengganda yang konstan dapat digunakan:
(3 (a)) // ini setara dengan (a, a, a)
- Concatenations nested (rangkaian penggabungan) yang mungkin: (b, (3 (c, d))) // ini setara dengan (b, c, d, c, d, c, d)

➤ Operator Kondisional

- Operator Kondisional memiliki format seperti berikut: cond_expr? true_expr: false_expr
- True_expr atau false_expr dievaluasi dan digunakan sebagai hasil tergantung pada apakah cond_expr mengevaluasi untuk benar atau salah

Contoh :

out = (enable)? data: 8'bz; // Tri state buffer

➤ Operator Precedence

Operator	Symbols
Unary, Multiply, Divide, Modulus	+ - ! ~ * / %
Add, Subtract, Shift.	+, -, <<, >>
Relation, Equality	<,>,<=,>=,==,!=,====,!====
Reduction	&, !&, ^, ^~, , ~
Logic	&&,
Conditional	?:

Gambar B.10 Penulisan Operator dalam Verilog

- Model Verilog Behavioral

B.8.2.2.4 Tingkat Abstraksi HDL Verilog

- Behavioral Models: tingkat tinggi di mana perilaku pemodelan logika adalah model.



- RTL Model: Logika mendaftar dimodelkan pada tingkat
- Structural Model: Logika dimodelkan pada kedua tingkat mendaftar dan gerbang tingkat.

B.8.2.2.5 Blok Prosedural

Kode perilaku Verilog adalah prosedur di dalam blok, tetapi ada pengecualian, beberapa kode perilaku juga ada prosedur di luar blok. Kita bisa melihat hal ini secara rinci ketika kita membuat kemajuan. Ada dua jenis prosedural blok di Verilog:

- Initial: Initial blok mengeksekusi hanya sekali pada waktu nol (mulaipada waktu pelaksanaan nol).
- Always: always blok loop untuk mengeksekusi berulang kali, Dengan kata lain selalu dijalankan.

Contoh : initial dan always

<pre>initial begin clk = 0; reset = 0; enable = 0; data = 0; end</pre>	<pre>always @ (posedge clk) begin : D_FF if (reset == 1) q <= 0; else q <= d; end</pre>
--	---

Gambar B.11 Penulisan Initial dan Always pada Verilog

B.8.2.2.6 Statements (pernyataan)

➤ Pernyataan Sequential group

- Kelompok beberapa pernyataan dilakukan bersama.
 - Menyebabkan pernyataan dievaluasi dalam secara berurutan (satu per satu waktu).
 - Setiap waktu dalam kelompok-kelompok berurut relatif terhadap pernyataan sebelumnya.
1. Penundaan dalam urutan menumpuk (setiap penundaan akan ditambahkan ke penundaan sebelumnya)
 2. Blok selesai setelah pernyataan terakhir di blok.



➤ **Pernyataan Parallel group**

3. Kelompok beberapa pernyataan dilakukan bersama.
4. Menyebabkan pernyataan untuk dievaluasi secara paralel (pada saat yang sama).
5. Timing dalam kelompok paralel mutlak ke awal kelompok.
6. Blok selesai setelah pernyataan terakhir selesai

➤ **Pernyataan Kondisi (if – else)**

Pernyataan *if – else* mengendalikan pelaksanaan pernyataan lainnya serta aliran program berdasarkan sebuah kondisi yang terpenuhi.

if (kondisi)

statements;

atau

if (kondisi)

statements;

else

statements;

atau

if (kondisi)

statements;

else if (kondisi)

statements;

.....

.....

else

statements;

Contoh Penggunaan If:

// Simple if statement



```

if (enable)
    q <= d;

// One else statement
if (reset == 1'b1)
    q <= 0;
else
    q <= d;

// Nested if-else-if statements
if (reset == 1'b0)
    counter <= 4'b0000;
else if (enable == 1'b1 && up_en == 1'b1)
    counter <= counter + 1'b1;
else if (enable == 1'b1 && down_en == 1'b1);
    counter <= counter - 1'b0;
else
    counter <= counter;

```

➤ Pernyataan Case

Pernyataan **Case** adalah sebuah ekspresi untuk membandingkan serangkaian kasus dan mengeksekusi pernyataan atau kelompok pernyataan yang terkait dengan yang pertama (kasus pencocokan)

7. Mendukung pernyataan Case satu atau beberapa pernyataan.
8. Kelompok pernyataan multiple memulai dan berakhir menggunakan kata kunci.

Contoh :

```

case (<expression>)
    <case1> : <statement>
    <case2> : <statement>
    ....
    default : <statement>
endcase

```



➤ Pernyataan Perulangan

Pernyataan Looping (perulangan) muncul di dalam sebuah blok prosedural saja, Verilog memiliki empat pernyataan perulangan seperti bahasa pemrograman lain: **Forever**, **Repeat**, **While**, dan **For**.

Contoh penggunaan perulangan:

- **Forever** : Mengeksekusi loop yang terus-menerus (loop tidak pernah berakhir)

```
initial begin
```

```
    clk = 0;
```

```
    forever #5 clk = !clk;
```

```
end
```

- **Repeat** : Mengeksekusi loop pernyataan yang berulang-ulang.

```
if (opcode == 10) //perform rotate
```

```
repeat (8) begin
```

```
    temp = data[7];
```

```
    data = {data<<1,temp};
```

```
end
```

- **While** : Mengeksekusi loop sementara.

```
loc = 0;
```

```
if (data = 0) // example of a 1 detect shift value
```

```
    loc = 32;
```

```
else while (data[0] == 0); //find the first set bit
```

```
begin
```

```
    loc = loc + 1;
```

```
    data = data << 1;
```

```
end
```

- **For** : looping yang dilakukan oleh for telah diketahui batas awal, syarat looping dan perubahannya. Selama kondisi terpenuhi, maka pernyataan akan terus dieksekusi.

```
for(i=0;I,=63;i=i+1)
```

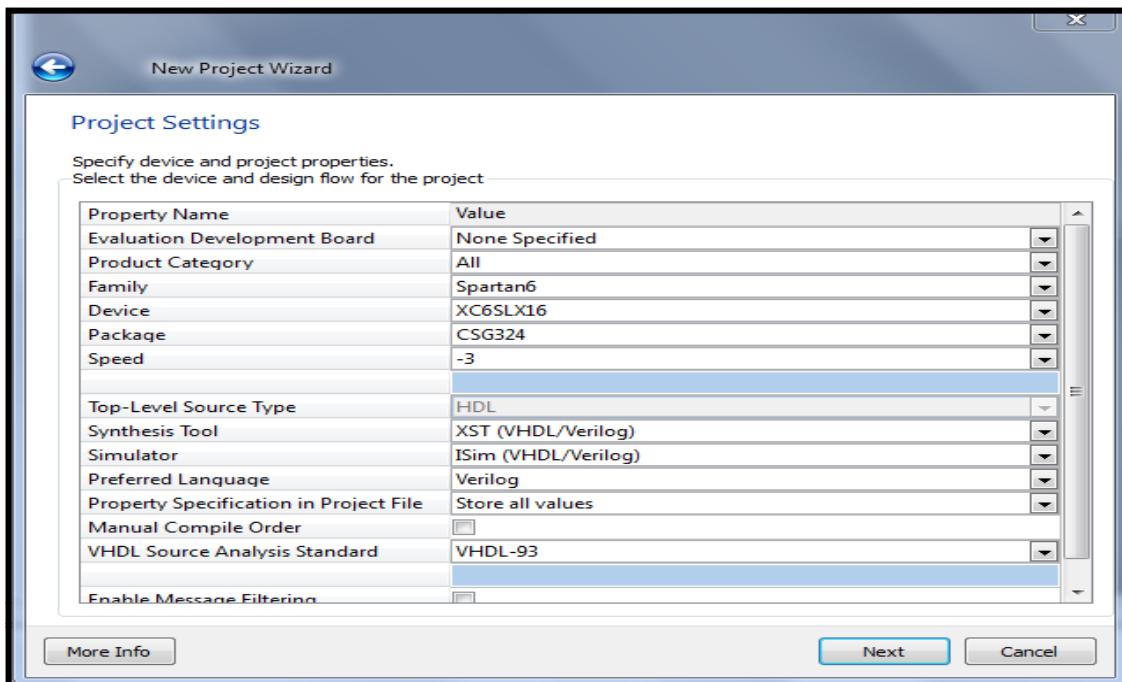
```
ram[i]<=0;//Inialize the RAM with 0.
```



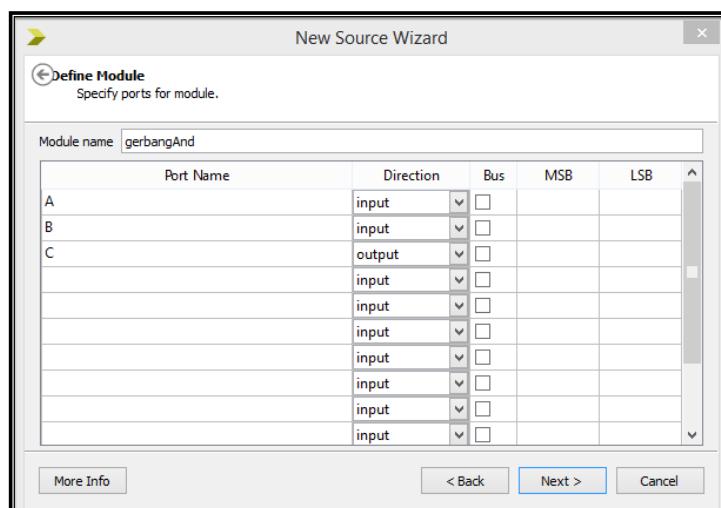
Percobaan 1 : Gerbang AND

Langkah-Langkah:

1. Buka Xilinx ISE 14.5
2. Klik File -> New Project
3. Ketik nama project dengan nama gerbangAnd lalu klik **next**
4. Atur Device Properties seperti gambar dibawah ini :



5. Klik next terus hingga Finish -> pilih yes
6. Klik kanan pada subname **gerbangAnd** yang ada pada tab hierarchy pilih **new source** -> ketik nama file namanya **gerbangAnd** pilih **Verilog Module** -> klik **next** -> masukan input dan outputnya seperti dibawah ini:



7. Klik next -> lalu Finish
8. Cek source code nya seperti dibawah ini :

AND	OR
<pre> 20 /// 21 module gerbangAND(22 input A, 23 input B, 24 output C 25); 26 and (C,A,B); 27 28 endmodule 29 </pre>	<pre> 20 /// 21 module gerbangAND(22 input A, 23 input B, 24 output C 25); 26 or (C,A,B); 27 28 endmodule 29 </pre>

9. Lalu **Save**
10. Pada bagian tab processes pilih **Synthesize –XST** klik tanda +klik kanan **CheckSyntax** pilih run atau cukup klik 2x pada **Synthesize –XST**
11. Klik tanda + pada user constraints pilih **I/O pinPlanning (PlanAhead) – Pre Synthesis**, klik kanan pilih Run -> pilih yes (**Jika ingin menggunakan PlanAhead**)
12. Jika tidak ingin menggunakan PlanAhead, maka selanjutnya klink kanan pada subname yang diberi nama gerbangAnd-> pilih newsource, ketik nama file namanya gerbangAnd pilih **Implementation Constraints File** -> klik next. (**Menggunakan UCF**)
13. Sebelum masuk text editor, klik tanda + pada project gerbangAnd, di hierarchy -> double klik gerbangAnd.ucf. Pada text editor ketiklah pin FPGA seperti dibawah ini : (**Setelah selesai jangan lupa untuk save**)

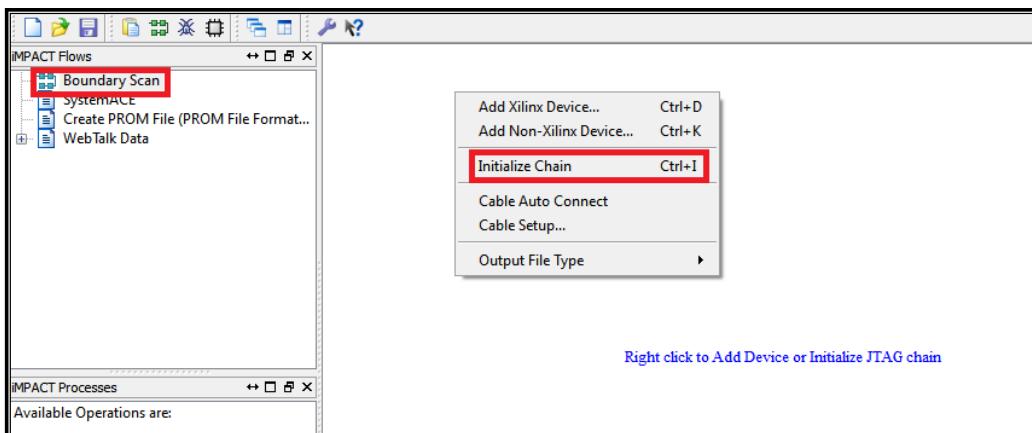
```

1 NET "A" LOC = "T10"; #SW1 A
2 NET "B" LOC = "T9"; #SW2 B
3 NET "C" LOC = "U16"; #LEDO

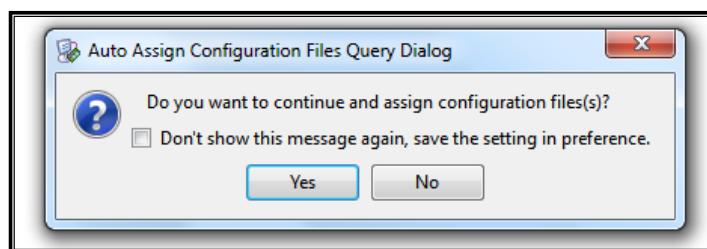
```

9. Selanjutnya klik 2x **Implement Desain** (**Sebelum klik 2x Implement Desain, jangan lupa untuk menghubungkan dengan FPGA terlebih dahulu**).
10. Lalu klik tanda + pada **Configure Target Device**, Pilih **Manage Configuration Project**
11. Pilih boundary scan, lalu klik kanan pada area “right click to add device or initialize JTAG chain”, kemudian pilih **Initialize Chain** seperti gambar dibawah ini :



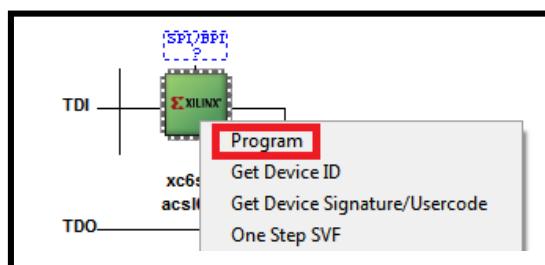


12. Pilih Yes pada Auto Assign Configuration Files Query Dialog



13. Pilih file gerbangAND.bit pada **Assign New Configuration** file lalu pilih open > gerbangAND.bit

14. Pada tampilan pop up selanjutnya pilih Cancel -> kemudian pilih OK, maka akan muncul tampilan seperti gambar dibawah ini, setelah itu klik kanan pada gambar IC -> klik **Program**



TABEL KEBENARAN GERBANG AND, OR, NOT :

A	B	Output AND	Output OR	Output NOT

KESIMPULAN:

Ikuti Langkah-langkah seperti pada Gerbang And, OR, Not untuk mengerjakan Gerbang Half Adder dan Decoder.



Percobaan 2 : Half Adder

```
20 //////////////////////////////////////////////////////////////////
21 module halfadder(
22     input A,
23     input B,
24     output S,
25     output C
26 );
27
28 xor (S,A,B);
29 and (C,A,B);
30 endmodule
```

Input		Output	
A	B	Carry	Sum

KESIMPULAN:



Percobaan 2 : Decoder

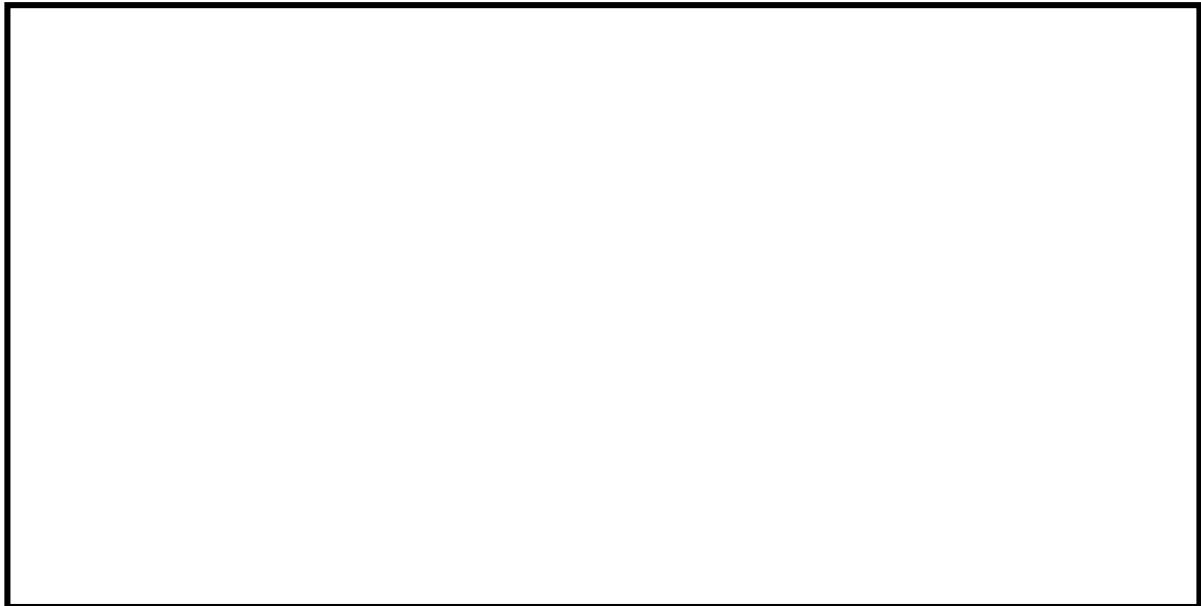
```
20 //////////////////////////////////////////////////////////////////
21 module acsl(
22     input [2:0] sw,
23     output [7:0] led
24 );
25 assign led = (sw == 3'b000) ? 8'b00000001 :
26     (sw == 3'b001) ? 8'b00000010 :
27     (sw == 3'b010) ? 8'b000000100 :
28     (sw == 3'b011) ? 8'b000001000 :
29     (sw == 3'b100) ? 8'b000100000 :
30     (sw == 3'b101) ? 8'b001000000 :
31     (sw == 3'b110) ? 8'b010000000 :
32     8'b10000000;
33
34 endmodule
35
```

```
1 NET "led[0]" LOC = U16;
2 NET "led[1]" LOC = V16;
3 NET "led[2]" LOC = U15;
4 NET "led[3]" LOC = V15;
5 NET "led[4]" LOC = M11;
6 NET "led[5]" LOC = N11;
7 NET "led[6]" LOC = R11;
8 NET "led[7]" LOC = T11;
9 NET "sw[0]" LOC = T10;
10 NET "sw[1]" LOC = T9;
11 NET "sw[2]" LOC = V9;
```

HASIL:

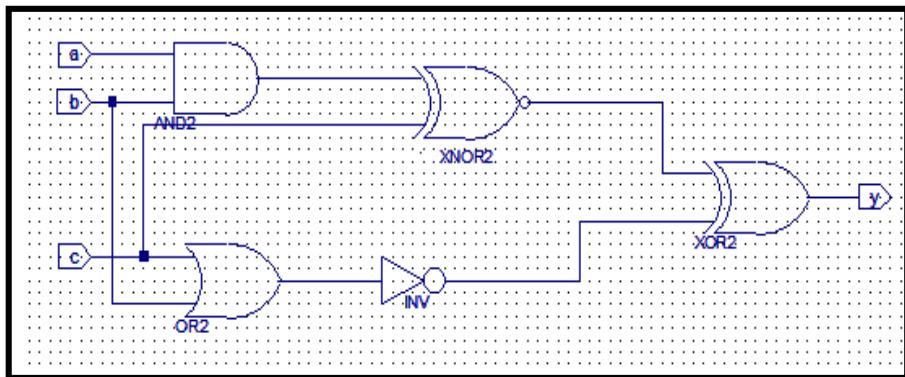


KESIMPULAN:



Percobaan Mandiri I :

Buatlah program verilog dari rangkaian dibawah ini :



Source Code :

Hasil Output dan Kesimpulan :



Percobaan Mandiri II :

Buatlah program verilog :untuk Downcounter

Source Code :

Hasil Output dan Kesimpulan:



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal dan Memahami pemrograman VHDL pada FPGA
- Praktikan Dapat Merancang Program Desain VHDL pada Pemrograman FPGA
- Praktikan Dapat Memahami Penggunaan gerbang logika dalam Pemrograman Desain VHDL

II. Dasar Teori

- Pengenalan Program VHDL
- Merancang Desain Program menggunakan VHDL
- Aplikasi Perancangan Program VHDL

III. Peralatan

- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC

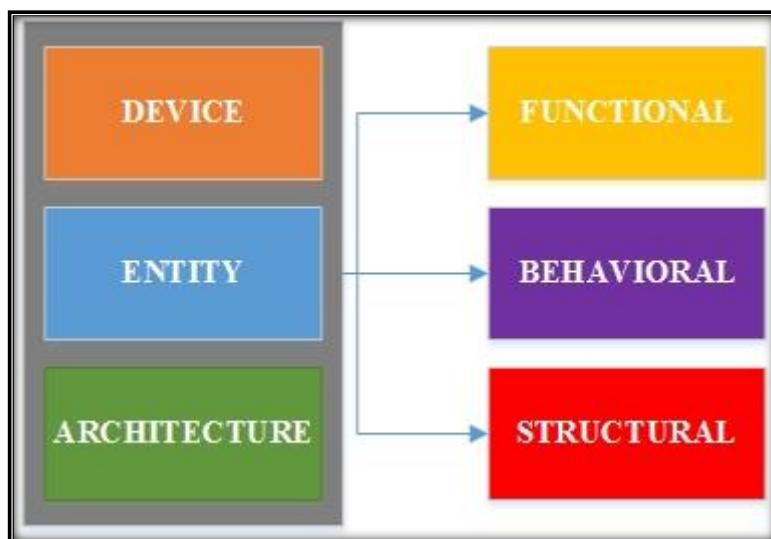


1.1 Mengenal VHDL

VHDL (Very High Speed Integrated Hardware Description Language). Konsep serta syntax banyak diperlukan untuk mengerti bagaimana rancangan VHDL sebagai bagian dari pemrograman FPGA. Dalam kebanyakan kasus, keputusan memilih dan menggunakan kode VHDL daripada kode Verilog atau SystemC, sangat tergantung pada pilihan perancang itu sendiri dan lebih kepada ketersediaan software pendukung serta kebutuhan perusahaan.

Verilog berasal dari tradisi “bottom-up” yang telah sering digunakan dalam industri IC dalam hal rancangan dasar IC. Sedangkan kode VHDL dikembangkan lebih kepada persepektif “top-down”. Tentu saja, banyak perbedaan umum dan luas dalam konteks saat ini. Namun, secara jelas dan nyata, perbedaannya dapat terlihat pada syntax dasar dan metode dari kedua kode tersebut.

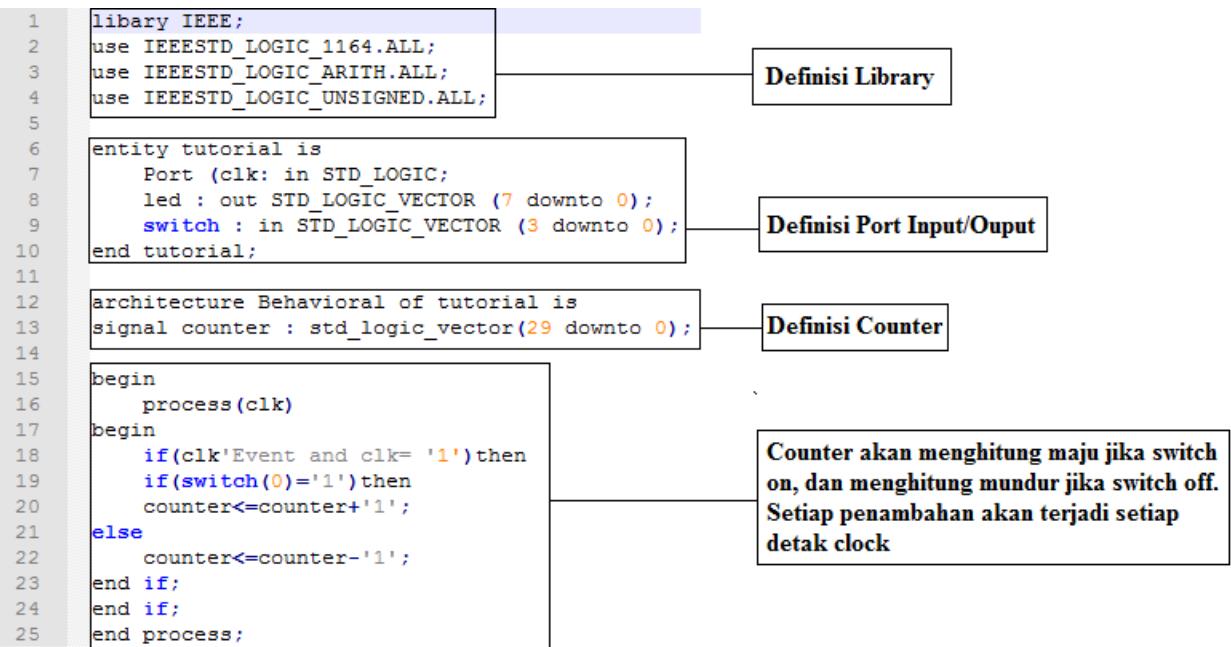
Kemampuan dari VHDL itu sendiri untuk menggunakan gabungan level dari model yang memiliki arsitektur yang berbeda seperti ditunjukkan pada gambar berikut:



Gambar 1.1 Contoh Arsitektur Model VHDL

Hal tersebut memang bukanlah keunikan atau ciri khas VHDL. Namun, pada kenyataannya kode Verilog juga memiliki konsep sama walaupun hanya terdapat dalam sebuah “module”. Dan untuk memperjelas konsep dari VHDL ini sendiri bisa di lihat pada contoh gambar source code berikut :





Gambar 1.2 Contoh Penggunaan Progam Dengan Menggunakan VHDL

VHDL secara praktis digunakan bersama oleh rancangan multi-level dalam VHDL. Pembagian sebuah model ke dalam beberapa bagian juga merupakan keunggulan lain dari VHDL. Misalnya, bagian interface (dalam VHDL dikenal sebagai “entity”) dan bagian kelakuan atau behaviour (dalam VHDL dikenal sebagai “architecture”).

Entity merupakan salah satu contoh bentuk satu kesatuan dan terpisah dengan bagian yang lain. Untuk selengkapnya, beberapa uraian berikut akan menjelaskan tentang entity serta hal-hal yang terkait di dalamnya. Ada beberapa dasar-dasar serta teknik penulisan kode dalam VHDL. Beberapa diantaranya adalah entity, architcture, basic type variables and operator, decisions, loops, dsb. Terdapat pula berkas-berkas yang nantinya berkas ini dapat digunakan sewaktu-waktu untuk berbagai keperluan tanpa harus menulisnya ulang. Biasanya berkas ini langsung disertakan menjadi satu dalam sebuah library.

Istilah library sendiri dikenal sebagai sekumpulan koleksi bermacam-macam berkas kode. Bila suatu berkas kode disimpan di dalam library maka berkas kode tersebut digunakan serta dibagikan dengan rancangan yang lain.

1.2 Library

Pada pemrograman dikenal pula istilah library atau pustaka yang bias terdapat pada bahasa pemrograman yang lain seperti C atau header pada Pascal. Library berfungsi



untuk memudahkan prorammer untuk menyelesaikan pekerjaannya karena dalam library tersebut terdapat fungsi-fungsi dan tipe data yang sudah didefinisikan sebelumnya untuk digunakan berulang-ulang. Di dalam library tersebut terdapat sub-tree yang disebut sebagai paket, diantaranya :

```
1 LIBRARY IEEE :  
2 use IEEE.std_logic_1164.all;  
3 use IEEE.std_logic_textio.all;  
4 use IEEE.std_logic_arith.all;  
5  
6 use IEEE.numeric_bit.all;  
7 use IEEE.numeric_std.all;  
8  
9 use IEEE.std_logic_signed.all;  
10 use IEEE.std_logic_unsigned.all;  
11 use IEEE.math_real.all;  
12  
13 use IEEE.math_complex.all;  
14  
15 LIBRARY STD :  
16 use STD.standard;  
17 use STD.textio;  
18 LIBRARY WORK :
```

Gambar 1.3 Isi dari library

Paket yang didefinisikan oleh user sendiri, yang isinya terdiri dari tipe data, architecture dan behavioral. Isi paket tidak harus memenuhi syarat dan kesepakatan IEEE.

1.3 Entity

Entity memberikan arti tentang bagaimana sebuah bagian rancangan dideskripsikan di VHDL dalam hubungannya dengan model VHDL lain dan juga memberikan nama untuk model tersebut. Di dalam entity juga diperbolehkan untuk mendefinisikan beberapa parameter yang mengambil model menggunakan hierarki. Kerangka dasar untuk sebuah entity digambarkan sebagai berikut :

```
entity <name> is  
  ...  
end entity <name>;
```



Misalkan sebuah entity diberi nama “test”, maka kerangka entity tersebut akan menjadi

:

```
entity test is  
end entity test;  
  
atau  
  
entity test is  
end test;
```

1.4 Ports

Sebuah cara atau metode untuk menghubungkan entity secara bersama adalah menggunakan PORTS. Hal ini didefinisikan bahwa entity menggunakan metode sebagai berikut:

```
port (  
...list of port declarations...  
);
```

Deklasi port ini mendefinisikan jenis dari koneksi dan arah yang sesuai. Misalnya, deklarasi port untuk sebuah input bit adalah 1, maka digambarkan sebagai berikut :

```
in1 : in bit;  
port (  
in1, in2 : in bit;  
out1 : out bit);
```

Dengan menngunakan ports maka titik koneksi diantara entities akan berlangsung dengan efektif dalam hal proses koneksi entities satu sama lain. Selain itu, dengan menggunakan ports akan menjadikan sinyal yang ada menjadi efektif serta cocok digunakan dalam model VHDL

1.5 Generics

Jika sebuah contoh memiliki sebuah parameter, maka contoh tersebut dapat didefinisikan menggunakan generics. Deklarasi umum dari generics ditunjukan berikut:



```
generic (
...list of generic declarations...
);
```

Pada beberapa kasus dari generics, deklarasinya mirip dari sebuah constant dengan bentuk yang ditunjukkan sebagai berikut:

```
param1 : integer := 4;
```

Misalnya saja sebuah model yang memiliki dua generics (gain(integer) dan time_delay (time)), keduanya dapat didefinisikan dalam sebuah entity sebagai berikut :

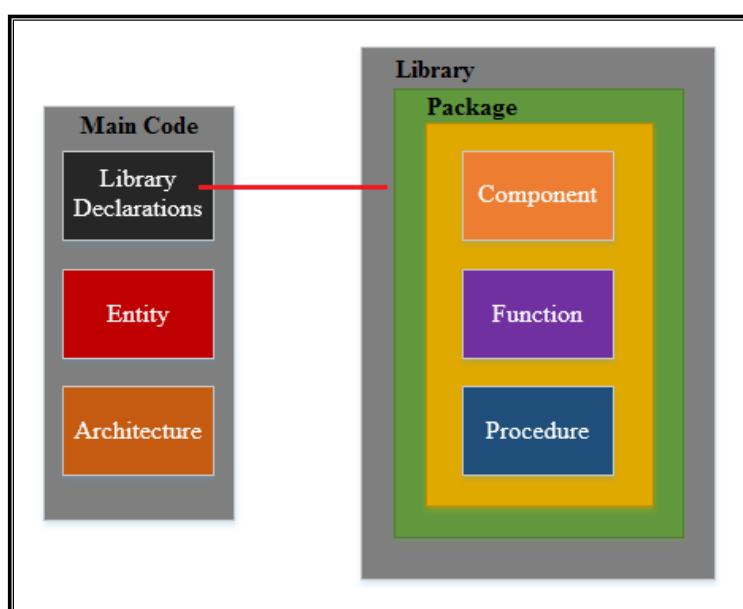
```
generic (
    gain : integer := 4;
    time_delay : time = 10 ns
);
```

1.6 Berkas

Selanjutnya ada 4 cara pembuatan berkas yang nantinya dapat disimpan dalam library.

Keempat cara tersebut adalah :

1. Function
2. Packages
3. Components
4. Procedures



Gambar 1.4 Contoh Arsistektur Model VHDL



Seperti terlihat pada gambar, keempat berkas tersebut biasanya menjadi satu dalam sebuah library. Selanjutnya, library tersebut akan digunakan dalam kode utama yang terdiri atas entity dan architecture. Inilah yang disebut rancangan bertingkat (hierarchical design) dalam VHDL. Fungsi utama dari berkas-berkas tersebut adalah untuk membuat efisien dan efektif dalam penulisan kode VHDL.

Bayangkan bila tiap rangkaian dalam sebuah rancangan sistem harus dideklaraskan satu per satu. Hal itu tentu memerlukan resource yang banyak dan waktu yang lama. Belum lagi jika terdapat kesalahan. Maka proses pengecekan dan pemeriksaan kode juga butuh waktu yang tak sedikit. Oleh karena itu, dibuatlah berkas-berkas sederhana untuk menghindari hal tersebut. Selain itu dengan mempelajari berkas-berkas ini maka sebuah library dapat dibuat sesuai dengan kepentingan dan kebutuhan perancang sistem.

1.7 Packages (Paket)

Bagian pertama berkas yang akan dibahas yaitu PACKAGES. Seperti terlihat pada gambar di atas, packages merupakan bagian berkas yang luas karena bisa terdiri atas Berkas-berkas lain seperti component, function, dan procedures. Semua berkas yang terdapat dalam packages, termasuk berkas packages sendiri nantinya akan dikumpulkan menjadi satu dan disimpan dalam sebuah library. Sedangkan struktur untuk aturan penulisan packages yaitu:

```
package name is  
...package header contents  
end package;  
  
package body name is  
... package body contents  
end package body;
```

Seperti yang terlihat pada aturan penulisan, sebuah packages terdiri dari 2 bagian yaitu header dan body. Header adalah tempat untuk mendeklarasikan keseluruhan namanama berkas (baik function, procedures, atau component) yang diperlukan. Sementara body adalah tempat untuk menjabarkan lebih lanjut mengenai berkas-berkas yang tertulis di header. Perlu diingat bahwa antara haeder dan body packages haruslah memiliki nama yang sama. Sebagai contoh, berikut ada packages yang berisi function untuk logika yang sederhana. Bagian header packages tersebut dapat ditulis :



```

package new_functions is

function and10 (a,b,c,d,e,f,g,h,i,j : bit) return bit;

end

```

Sedangkan untuk bagian body, yaitu tempat untuk menjabarkan lebih lanjut deklarasi pada header, dapat ditulis:

```

package body new_functions is

function and10 (a,b,c,d,e,f,g,h,i,j : bit) return
bit is
begin
return a and b and c and d and e and f and g and h
and i and j;
end;

```

Bila antara bagian header dan body digabung menjadi satu, maka inilah yang disebut packages. Dan secara keseluruhan dapat ditulis berikut :

```

package new_functions is

function and10 (a,b,c,d,e,f,g,h,i,j : bit) return bit;
end;

package body new_functions is

function and10 (a,b,c,d,e,f,g,h,i,j : bit) return
bit is
begin
return a and b and c and d and e and f and g and h and i
and j;
end;
end;

```

Selanjutnya packages dapat dijadikan satu untuk kemudian disimpan dalam library. Untuk bisa membuatnya demikian, perlu ditambahkan beberapa kode dalam penulisan VHDL



nya, yakni menambahkan kalimat USE (USE work.my_package.all;) di kode utama VHDLitu. Syntaxnya sebagai berikut :

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE work.my_package.all;  
ENTITY...  
...  
ARCHITECTURE...
```

1.8 Loops (Perulangan)

1.8.1 For

Selain algoritma percabangan, di dalam VHDL juga dikenal algoritma lain yaitu pengulangan. Pengulangan digunakan untuk menjalankan satu atau beberapa pernyataan sebanyak beberapa kali. Dengan kata lain, pengulangan dapat digunakan untuk menjalankan beberapa pernyataan hanya dengan menuliskan pernyataan tersebut satu kali saja. Untuk VHDL, algoritma pengulangan yang sering dipakai adalah pernyataan “for”. Aturan dasar untuk pernyataan “for” sebagai berikut :

```
for loopvar in start to finish loop  
... loop statements  
end loop;
```

Selanjutnya, algoritma pengulangan juga dapat digunakan untuk mencacah turun kemudian kembali lagi ke awal. Penulisannya sebagai berikut :

```
for loopvar in start downto finish loop  
... loop statements  
end loop;
```

Bentuk pengulangan dapat dijadikan satu dengan susunan nilai bit demi bit. Misalnya sebagai berikut :



```
signal a : std_logic_vector(7 downto 0);  
  
for i in 0 to 7 loop  
  
a(i) <= '1';  
  
end loop;
```

1.8.2 Exit

Perintah “exit” mengijinkan bentuk pengulangan “for” untuk diselesaikan secara lengkap dan utuh. Perintah ini digunakan ketika sebuah kondisi dicapai dan pengulangan sudah tidak diperlukan lagi. Syntax untuk perintah “exit” ditunjukkan sebagai berikut :

```
for i in 0 to 7  
loop  
  
if (i = 4) then  
exit;  
endif;  
endloop;
```

1.8.3 Next

Sedangkan perintah “next” mengijinkan pengulangan untuk diselesaikan bila adakondisi yang telah terpenuhi. Hal ini sedikit berbeda dengan perintah “exit” yang digunakan untuk menyelesaikan pengulangan saat semua kondisi terpenuhi. Perintah “next” dapat menghentikan pengulangan walaupun secara umum, kondisi yang diharapkan belum terpenuhi. Kemudian, pengulangan akan dilanjutkan kepada pengulangan yang lain. Hal ini sangat bermanfaat ketika sebuah kondisi telah tercapai dan pengulangan sudah tidak lagi dibutuhkan. Contoh pengulangan untuk perintah “next” ditunjukkan berikut :



```
for i in 0 to 7 loop  
  
    if (i = 4) then  
  
        next;  
  
    endif;  
  
endloop;
```

1.9 Basic Variable Types and Operators

1.9.1 Constants

Dalam sebuah simulasi, ada sebuah nilai yang perlu diatur sedemikian rupa agar nilaitersebut tidak berubah atau tetap. Jenis nilai yang bersifat tetap demikan dinamakan sebagai constant. Constant ini sering digunakan untuk memberikan nilai awal dalam sebuah parameter. Selain itu, constant juga digunakan sebagai perbandingan untuk nilai-nilai yangterdapat pada sejumlah register. Sebuah constant dapat dideklarasikan untuk beberapa tipedalam VHDL. Berikut beberapa contoh penggunaan constant :

```
constant a : integer := 1;  
  
constant b : real := 0.123;  
  
constant c : std_logic := '0';
```

1.9.2 Signals

Dalam sebuah process, sebuah signal berperan dalam menghubungkan cara kerja serta apa yang akan dikerjakan oleh process tersebut. Selain itu, signal merupakan jalur yang efektif dalam suatu rangkaian, dimana signal tersebut akan menghubungkan bagian satu sama lain dalam rangkaian itu. Ketika proses simulasi dilakukan, simulator akan memeriksa apakah sinyal-sinyal yang tersedia sudah memiliki nilai atau belum. Bila belum, maka akan diinisialisasikan sejumlah nilai untuk sinyal-sinyal tersebut. Jika sebaliknya, nilai sinyal akan senantiasa diperiksa dan diperbarui (update) sesuai dengan kondisi proses simulasi saat itu. Hal



ini diperlukan agar simulator dapat mengetahui, apakah terdapat perubahan atau tidak didalam process yang tengah berlangsung Atau dengan kata lain, simulator akan memeriksa processmanakah yang aktif atau tidak.

Suatu singal dapat diberikan nilai yang tepat atau dapat ditambah dengan waktu tunda (delay). Dengan adanya delay, maka process yang hendak berjalan dapat dijadwalkan dimasa yang akan datang. Inilah salah satu perbedaan signal dengan urutan kumpulan kode program (seperti dalam C). Contoh pernyataan dan penggunaan signal ditunjukkan berikut :

```
signal sig1 : integer := 0;  
signal sig2 : integer := 1;  
  
sig1 <= 14;  
sig1 <= sig2;  
  
sig1 <= sig2 after 10 ns;
```

1.9.3 Variables

Dalam VHDL, dikenal istilah variable yakni sebuah nilai yang tidak tetap. Variable ini hanya digunakan dalam jenis proses sekuensial/beurutan. Variable ini berbeda dengan sinyal-sinyal pada umumnya yang terjadi secara serentak. Variable digunakan dalam sebuah proses dan cara pendeklarasiannya sebagai berikut:

```
variable var1 : integer := 0;  
  
variable var2 : integer := 1;  
  
var1 := var2;
```

Sekedar catatan, bahwa tidak ada konsep delay dalam penggunaan variable (jika memerlukan proses yang perlu penjadwalan, lebih baik digunakan signal saja).

1.9.4 Boolean Operators

VHDL juga memiliki sejumlah operator Boolean. Daftar operator boolean yang ada didalam VHDL adalah AND, OR, NAND, NOT, NOR, dan XOR. Operator tersebut dapat diaplikasikan ke dalam BIT, BOOLEAN, atau jenis logika seperti contoh berikut ini :

```
out1 <= in1 and in2;  
  
out2 <= in3 or in4;  
  
out5 <= not in5;
```



1.9.5 Arithmatic Operators

Selain operator boolean, terdapat pula operator aritmetika yang terdapat dalam VHDL. Operator tersebut dijelaskan seperti tabel berikut ini :

Operator	Description	Example
+	Addition	out1 <= in1 + in2;
-	Subtraction	out1 <= in1 - in2;
*	Multiplication	out1 <= in1 * in2;
/	Division	out1 <= in1/in2;
abs	Absolute Value	absin1 <= abs(in1);
mod	Modulus	modin1 <= mod(in1);
rem	Remainder	remin1 <= rem(in1);
**	Exponent	out1 <= in1 ** 3;

Tabel 1.1 Operator Aritmatika

VHDL juga memiliki sejumlah aturan baku untuk membandingkan nilai satu dengan nilai yang lain. Daftar operator pembanding tersebut adalah $=$, $/=$, $<$, $<=$, $>$, $=>$. Jenis operator tersebut dapat digunakan seperti berikut ini :

<i>in1 < 1</i>
<i>in1 /= in2</i>
<i>in2 >= 0.4</i>

1.9.6 Shifting Functions

VHDL juga memiliki 6 fungsi “logical shift” yang terangkum sebagai berikut :

Operator	Description	Example
sll	Shift Left Logical	reg <= reg sll 2;
srl	Shift Right Logical	reg <= reg srl 2;
sla	Shift Left Arithmetic	reg <= reg sla 2;
sra	Shift Right Arithmetic	reg <= reg sra 2;
rol	Rotate Left	reg <= reg rol 2;
ror	Rotate Right	reg <= reg ror 2;

Tabel 1.2 Perintah Shifting



1.9.7 Concatenation

Fungsi rentetan (concatenation) dalam VHDL disimbolkan dengan “&” dan penggunaanya seperti berikut ini :

```
A <= '1111';
B <= '000';
out1 <= A & B & '1'; — out1 =
'11110001';
```

1.10 Type data dalam VHDL

Untuk menuliskan kode VHDL secara efisien, sangatlah penting untuk mengetahui tipe-tipe data yang diperbolehkan, bagaimana, serta kapan penggunaannya. Artikel berikut dan beberapa artikel mendatang akan membahas tipe-tipe data apa saja yang terdapat dalam kode VHDL.

1.10.1 Tipe Data Standar

Kode VHDL mengandung sederetan tipe-tipe data yang telah ditentukan melalui aturan standar IEEE 1076 dan IEEE 1164. Untuk lebih jelas, beberapa tipe data telah tercantum ke dalam masing-masing jenis library/packages yaitu :

- Package standard of library std. Tipe data yang masuk jenis ini adalah tipe bit, boolean, integer, dan real.
- Package std_logic_1164 of library ieee. Tipe data yang masuk jenis ini adalah std_logic dan std_ulogic.
- Package std_logic_arith of library ieee. Tipe data yang masuk ke dalam library ini adalah signed, unsigned serta beberapa konversi fungsi, seperti conv_integer(p), conv_unsigned(p, b), conv_signed(p, b), and conv_std_logic_vector(p, b).

Perhatikan, berikut beberapa aturan tipe data std dalam VHDL :

- Tipe data Boolean : True, False
- Tipe data Integer : berisi 32 bit integer (mulai dari -2.147.483.647 s/d +2.147.483.647)
- Tipe data natural : adalah bilangan non negatif dari integer (mulai dari 0 s.d



+2,147,483,647)

1.10.2 Tipe Data Pengguna

Selain tipe data yang telah ditentukan secara baku oleh aturan IEEE, ternyata VHDL juga memperbolehkan pengguna (user) untuk menentukan tipe data yang diinginkan. Ada 2 kategori untuk tipe data yang bisa ditentukan sendiri oleh user yaitu :

1. Tipe Integer

- Tipe integer yang memiliki jangkauan -2.147.483.647 s/d +2.147.483.647 (mirip dengan tipe data standar).
- Tipe natural yang memiliki jangkauan 0 s/d +2.147.483.647 (mirip dengan tipe data standar).
- Tipe my_integer yang memiliki jangkauan -32 s/d 32 (tipe data yang ditentukan sendiri oleh user).
- Tipe nilai_murid yang memiliki jangkauan 0 s/d 100 (tipe data yang ditentukan sendiri oleh user).

2. Tipe Enumerated

- Tipe bit yaitu ‘0’ dan ‘1’
- Tipe my_logic yaitu ‘0’, ‘1’, dan ‘Z’ (ditentukan sendiri oleh user)
- Tipe keadaan yaitu idle, stop, backward, forward (ditentukan sendiri oleh user)
- Tipe warna misalnya merah, hijau, putih (ditentukan sendiri oleh user)

1.11 Process dalam VHDL

Process adalah sebuah mekanisme atau cara untuk mengesekusi susunan pernyataan (statement) dengan urutan yang terstruktur. Pernyataan (statement) tersebut biasanya lebih dari satu dan eksekusinya terjadi secara bersamaan.

Setiap process memiliki cara pendeklarasian tertentu dan biasanya berbeda satu sama lain, tergantung pernyataan yang hendak dikesekusi. Selain itu, lokasi process biasanya terletak di dalam bagian architecture. Aturan dasar untuk penulisan process adalah sebagai berikut :



```
process sensitivity_list is  
    ... declaration part  
begin  
    ... statement part  
end process;
```

Dalam process, suatu kumpulan (list) pernyataan akan dijalankan/aktif ketika terjadi perubahan nilai sinyal. Misalnya, sebuah process memiliki sinyal “clock” dan “reset”. Process tersebut akan dijalankan ketika terjadi perubahan sinyal yang melibatkan sinyal sinyal “clk” dan “rst”.

```
process (clk, rst) is  
begin  
    ... process statements  
end process;
```

Pada contoh berikut ini, process hanya bisa berjalan/aktif terjadi perubahan nilai “clk” atau “rst”. Selain itu, process juga masih bisa aktif, yaitu dengan cara menggunakan wait statement. Bila menggunakan wait statement, process akan otomatis dijalankan sekali kemudian menunggu terjadinya perubahan nilai sinyal lain yang melibatkan sinyal “clk” atau sinyal “rst”. Bila itu terjadi, process akan berjalan/aktif lagi.

```
process  
begin  
    ... process statements  
wait on clk, rst;  
end process;
```



Pada umumnya, lokasi dari sebuah wait statement tidaklah begitu penting. Hal ini karena dalam simulasi VHDL, setiap process berjalan setiap siklus sehingga wait statement dapat diletakan di awal atau di akhir. Hal tersebut tidak berpengaruh pada jalannya process karena sifatnya sama saja. Untuk bagian pendeklarasian dalam process, variabel dan sinyal-sinyal yang digunakan dapat didefinisikan saat itu juga atau bisa sebelumnya, seperti contoh berikut ini :

```
process (a) is
    signal na : bit;
begin
    na <= not a;
end process;
```

Listing di atas menggambarkan process “a”. Di dalam process “a”, terdapat sinyal “na” yang bertipe bit. Sinyal “na” akan dijalankan ketika terjadi perubahan nilai pada sinyal “a”. Sinyal “a” sendiri telah dideklarasikan di luar process tersebut, bisa dideklarasikan di bagian architecture atau bagian lainnya.

1.12 Architecture dalam VHDL

Jika sebuah entity digambarkan sebagai sebuah interface serta parameter sebagai bagian dari sebuah model, maka architecture akan menggambarkannya sebagai watak (behaviour) dari model tersebut. Maksudnya, entity secara umum menggambarkan keseluruhan kerangka model. Sedangkan architecture akan menggambarkan watak dari model tersebut. Misalnya,

```
entity multiplexor is
    port (I1, I2, E,S : in std_logic;
          o : out std_logic);
end Multiplexor;
```



Listing program diatas menggambarkan sebuah entity bernama multiplexor yang memiliki 4 input dan 1 output.

```
architecture RTL of multiplexor is
begin
process
begin
o <= (I1 and S and E) or
(I2 and not(S) and E);

end process;

end RTL;
```

Listing program diatas menggambarkan watak architecture. Dapat dikatakan bahwa sifat atau watak output entity multiplexor ditentukan oleh output dari I1, S, dan E.

Hingga saat ini dikenal beberapa tipe dari architecture VHDL. Selain itu, VHDL juga memperbolehkan adanya architecture yang berbeda dalam sebuah entity yang sama. Artinya sebuah entity boleh memiliki architecture lebih dari satu walaupun sifat atau wataknya tersebut berbeda satu sama lain.

Akibatnya, hal tersebut menjadikan sebuah bentuk yang ideal untuk mengembangkan watak, Register Transfer Level (RTL), serta architecture pada level gerbang. Selanjutnya, semuanya itu dapat dijadikan satu ke dalam sebuah rancangan dan dapat diuji coba menggunakan test bench yang sama. Seperti digambarkan pada listing pertama, pendekatan dasar untuk menggambarkan sebuah architecture sebagai berikut :



```
architecture behaviour of test is  
..architecture declarations  
begin  
...architecture contents  
end architecture behaviour;  
atau  
architecture behaviour of test is  
..architecture declarations  
begin
```

1.13 Bagian Pendeklarasian Architecture

Setelah digambarkan bagaimana pendeklarasian dari sebuah nama architecture, selanjutnya beberapa sinyal atau variabel harus dideklarasikan. Pendeklarasian ini dilakukan sebelum memulai membuat statement. Seperti contoh, jika terdapat 2 sinyal pada sebuah architecture (misal, sig1 dan sig2), maka keduanya dapat dideklarasikan dalam sebuah model sebagai berikut :

```
architecture behaviour of test is  
signal sig1, sig2 : bit;  
begin
```

Sifat dari arsitektur bernama test yang memiliki sinyal 1 dan sinyal 2 (keduanya bertipe bit); Nantinya kedua sinyal tersebut akan digunakan dalam sebuah bagian statement di bagian isi dari model yang hendak digunakan.



1.14 Bagian Pernyataan Architecture

Architecture dalam VHDL dapat memiliki variasi dari sebuah struktur untuk mencapai fungsi yang berbeda satu sama lain. Selanjutnya, output yang ingin diinginkan dapat dibuat dengan menggabungkan sinyal-sinyal atau variabel yang ada. Misalnya :

```
out1 <= in1 and in2 after 10 ns;  
  
out1 <= in1 or in2 after 10 ns;
```

Dua buah output dapat dibuat dengan input yang sama. Output 1 ditentukan dengan meng AND kan input 1 dengan input 2. Sedangkan output 2 ditentukan dengan mengOR kan input 1 dan input 2. Secara keseluruhan, contoh dari sebuah architecture dituliskan di bawah ini :

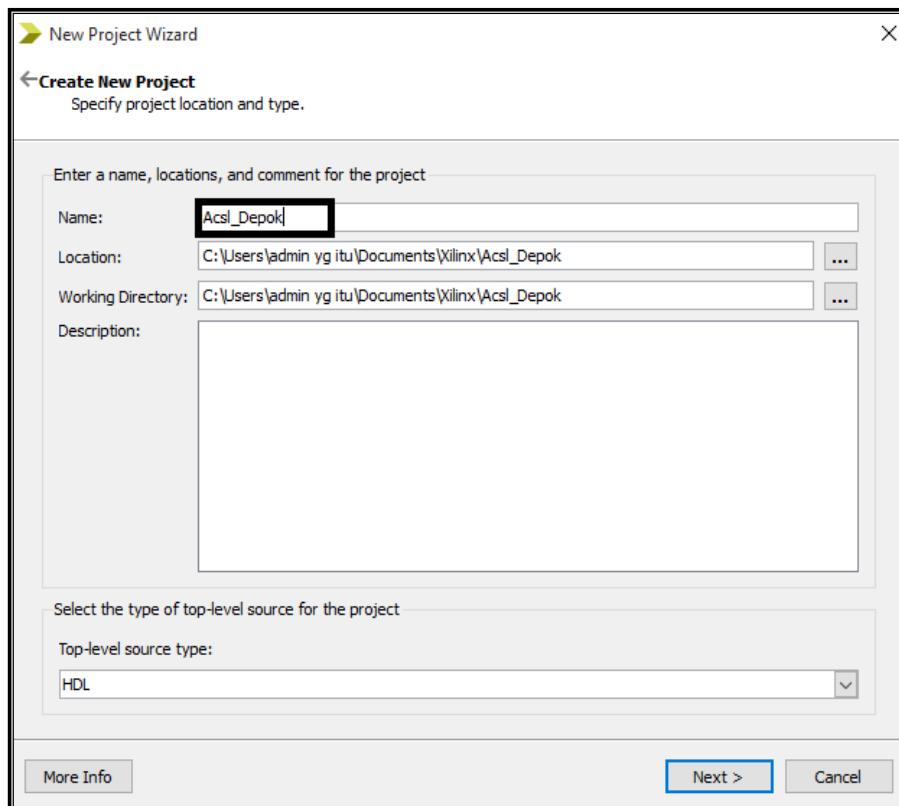
```
architecture behavioural of test is  
signal int1, int2 : bit;  
  
begin  
  
int1 <= in1 and in2;  
  
int2 <= in3 or in4;  
  
out1 <= int1 xor int2;  
  
end architecture behavioural
```



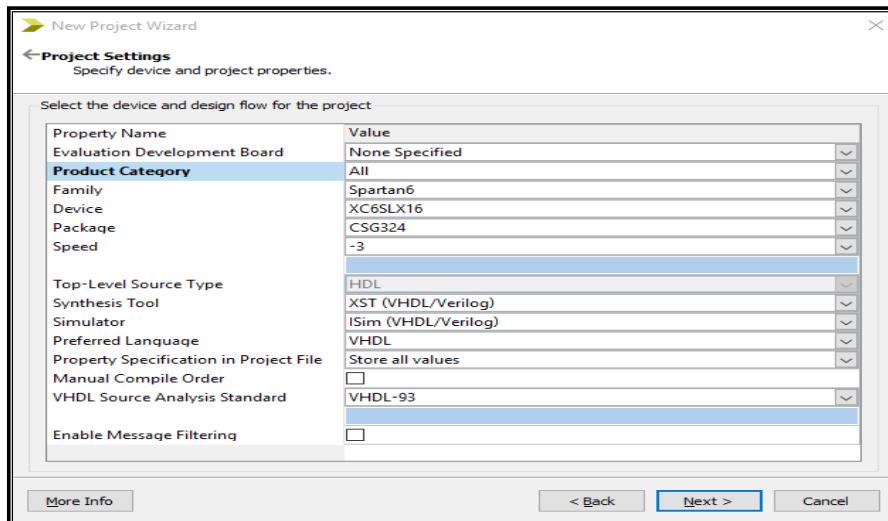
Percobaan 1 : Membuat gerbang AND, OR, NAND, NOR dengan program VHDL

Langkah - langkah :

1. Buka aplikasi ISE Design Suite 14.5
2. Buka Menu File > New Project
3. Buat nama project lalu klik next seperti gambar di bawah ini :

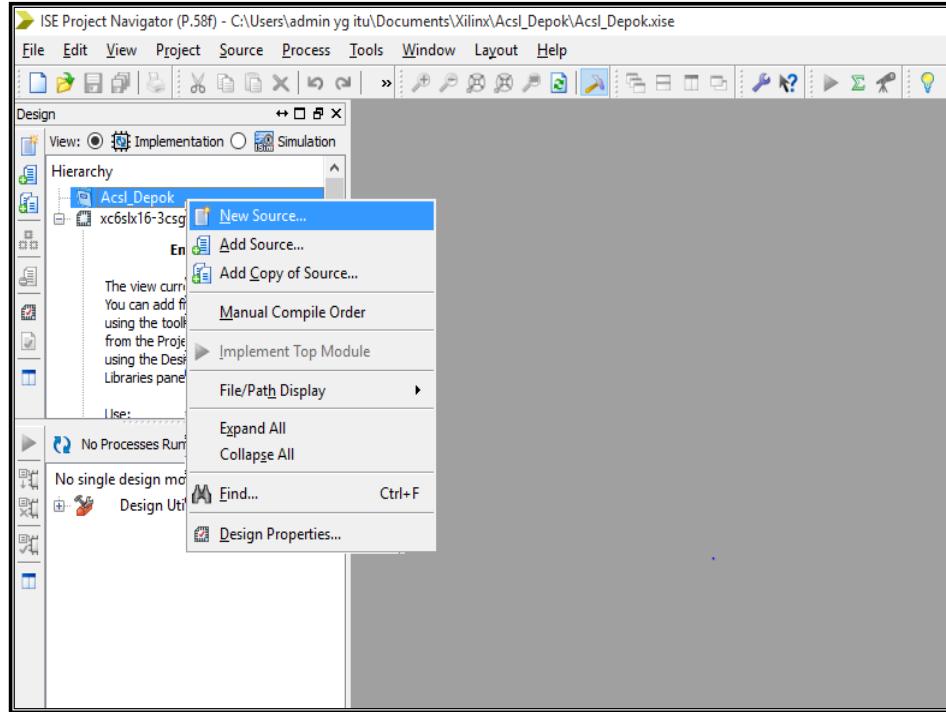


4. Setelah itu akan muncul project settings dan ikuti aturannya seperti gambar di bawah ini :

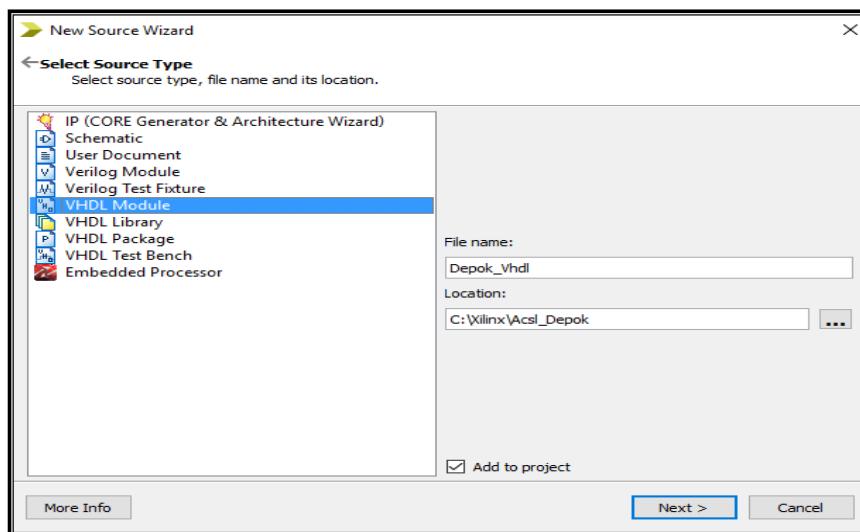


5. Klik **Next >** lalu klik **Finish**

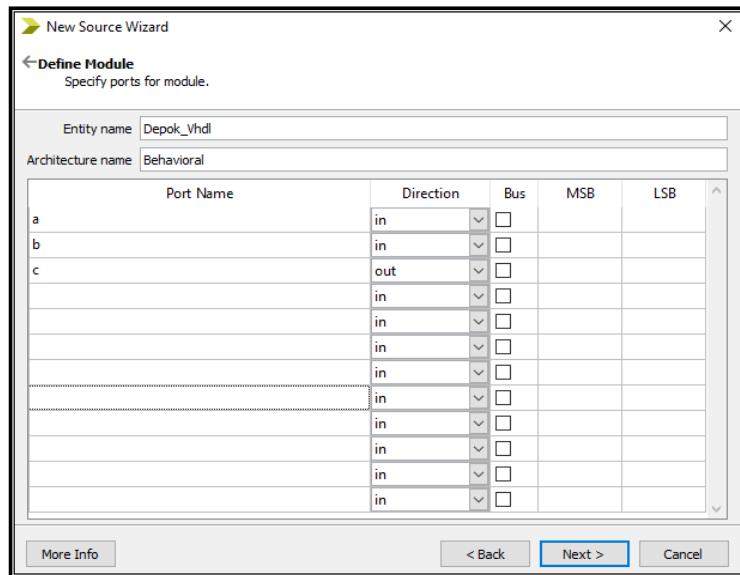
6. Lalu akan muncul halam form untuk project, setelah itu **klik kanan** pada nama **Acs1_Depok** > Pilih **New Source** seperti gambar di bawah ini :



7. Pilih bagian **VHDL Module** lalu isikan nama project dengan **Depok_Vhdl**, lalu klik **Next** seperti gambar di bawah ini :



8. Selanjutnya masukan I/O untuk bagian port name yang digunakan setelah itu klik **Next** dan **Finish**



9. Masukan source code seperti di bawah ini dan **Save** :

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Depok_Vhdl is
5     Port ( a : in STD_LOGIC;
6             b : in STD_LOGIC;
7             c : out STD_LOGIC);
8 end Depok_Vhdl;
9
10 architecture Behavioral of Depok_Vhdl is
11
12 begin
13     process (a,b)
14     begin
15         c <= (a and b);
16     end process
17 end Behavioral;
18

```

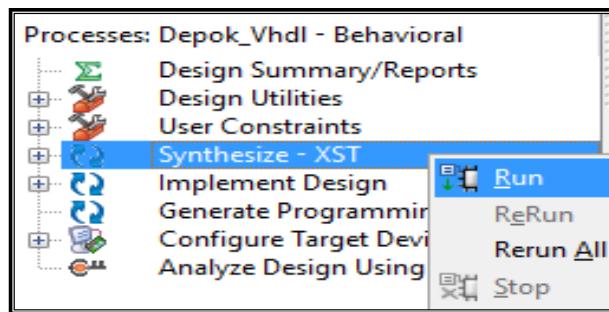
standar library yang sudah ditentukan IEEE;
seperti library yang digunakan untuk variable

bagian ditentukannya I/O untuk memproses program vhdl, dengan menggunakan
variable a,b,dan c, dengan nama project Depok_Vhdl

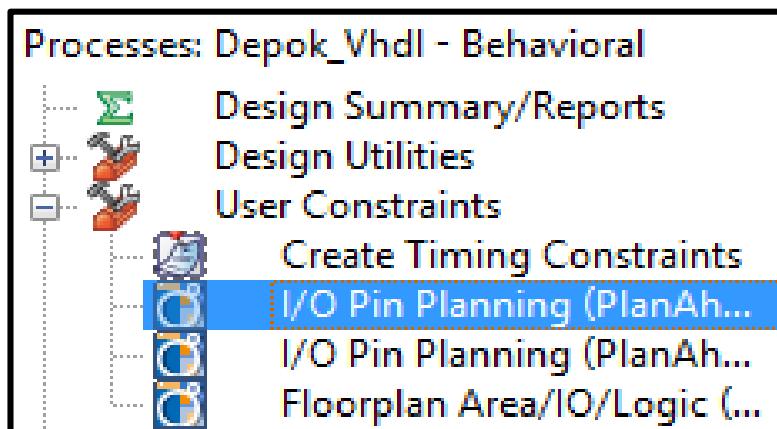
proses untuk membuat sebuah gerbang logika dengan
menggunakan program VHDL

10. Pada bagian process pilih **Synthesize – XST** lalu klik kanan dan pilih **Run**

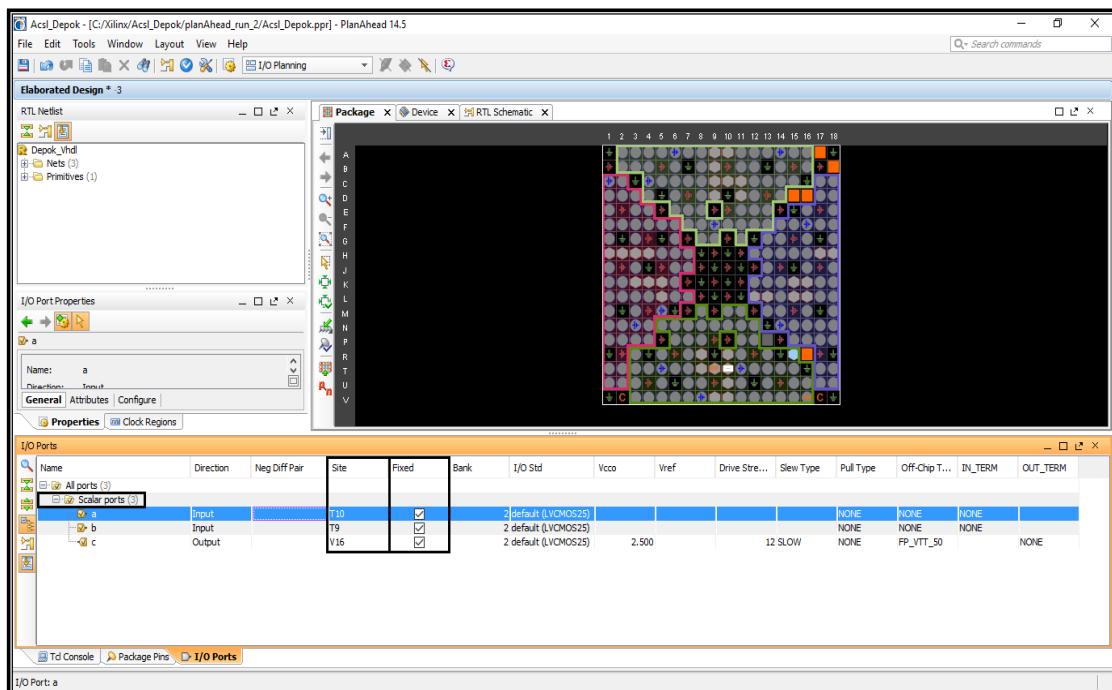




11. Klik tanda + pada user constraints pilih **I/O Pin Planning (PlanAhead) – Pre Synthesis**, klik kanan pilih **Run** -> pilih Yes

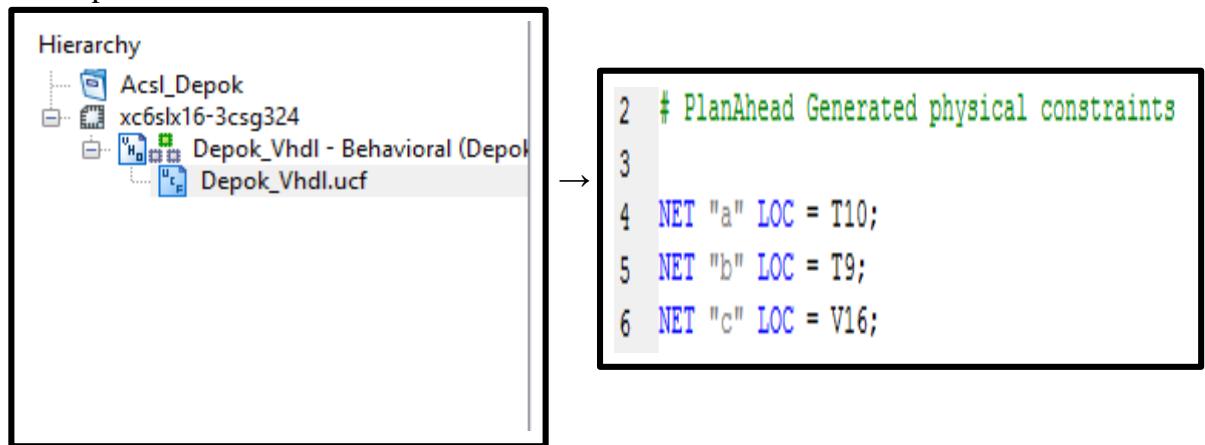


12. Setelah muncul tampilan seperti ini pilih bagian **Scalar Ports** > klik tanda + di bagian scalar ports masukan bagian port yang ingin digunakan sesuai **I/O** yang ada di FPGA, isi di bagian **Site** dan pastikan sudah terceklis **I/Onya** di bagian **Fixed** > lalu **Save**.

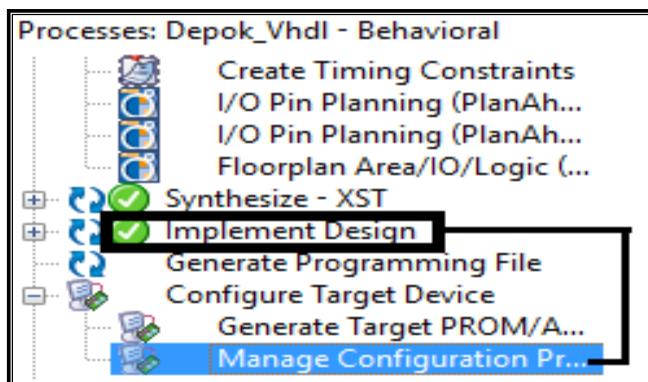


13. Dibagian hierarchy klik tanda + di bagian file **Depok_Vhdl – Behavioral**

(Depok_Vhdl.vhd) > double klik pada bagian file **Depok_Vhdl.ucf dan pastikan lokasi dan I/Onya sudah tepat**

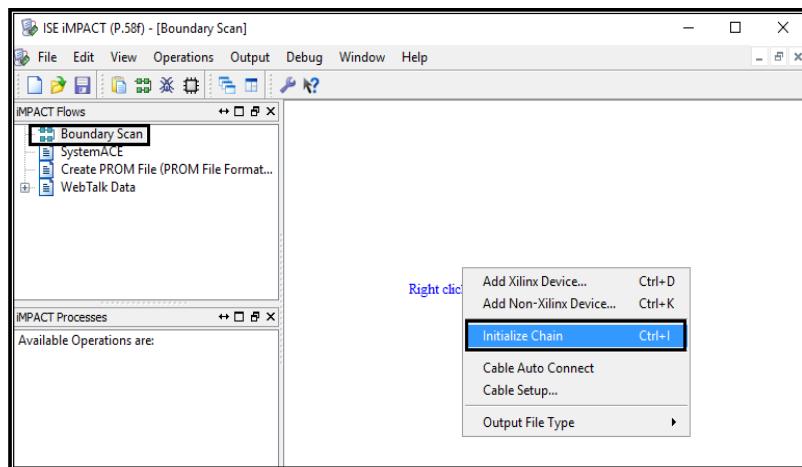


14. Selanjutnya pilih **Implement Desain lalu klik tanda + pada **Configure Target Device** dan Pilih Manage Configure Project (iMPACT).**

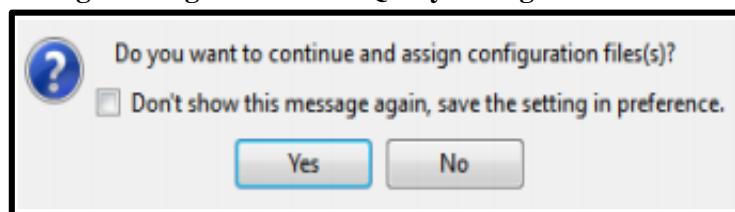


15. Tampilan selanjutnya akan muncul pilih **Boundary Scan > pilih klik kanan pada form “Right click to Add Device or Initialize JTAG chain” > pilih **Initialize Chain** seperti gambar di bawah ini :**

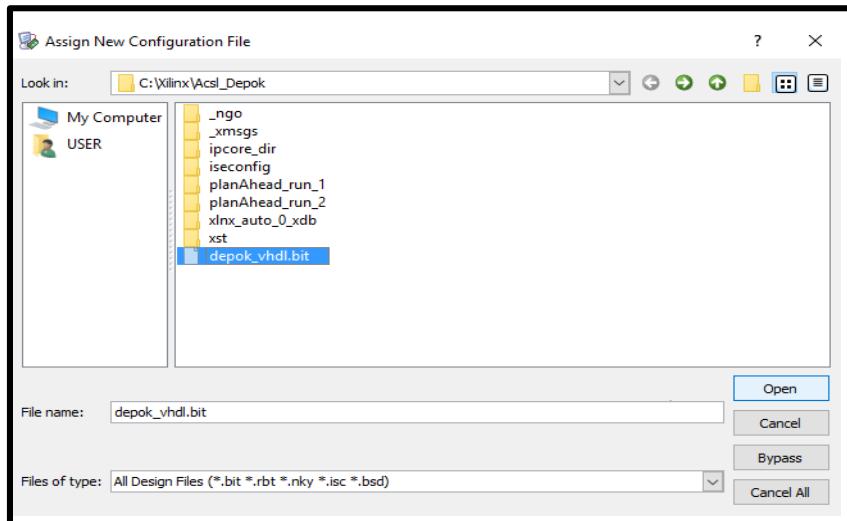




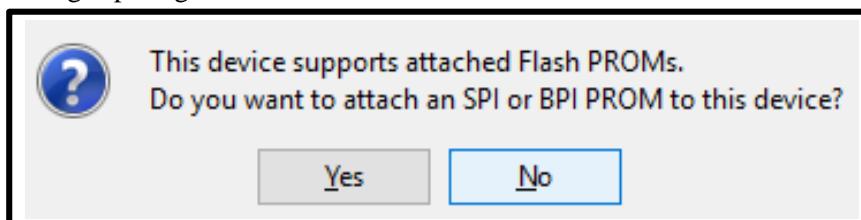
16. Pilih Yes pada Auto Assign Configuration Files Query Dialog



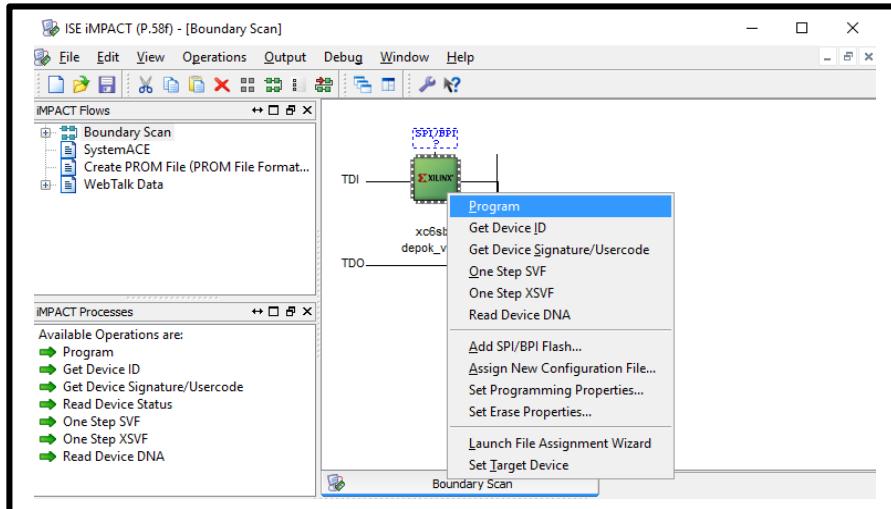
17. Jika muncul tampilan seperti gambar di bawah ini pilih file yang ber extensi .bit > klik Open



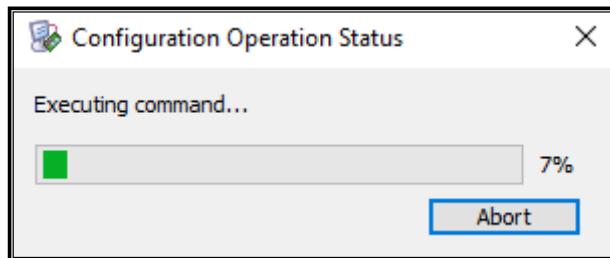
18. Pilih No pada dialog seperti gambar di bawah ini > dan Pilih OK :



19. Klik kanan di bagian gambar IC FPGA dan Pilih **Program**.



20. Tunggu Proses seperti gambar di bawah ini hingga selesai :



21. Setelah **100%** lihat hasilnya di FPGA kalian dan untuk melihat hasilnya, geser switch 0 (T10) dan switch 1 (T9) kemudian amati lah keluarannya berupa LED.



Tabel Kebenaran Gerbang AND, OR, NAND, NOR

AND		OR		NAND		NOR	
Input	Output	Input	Output	Input	Output	Input	Output

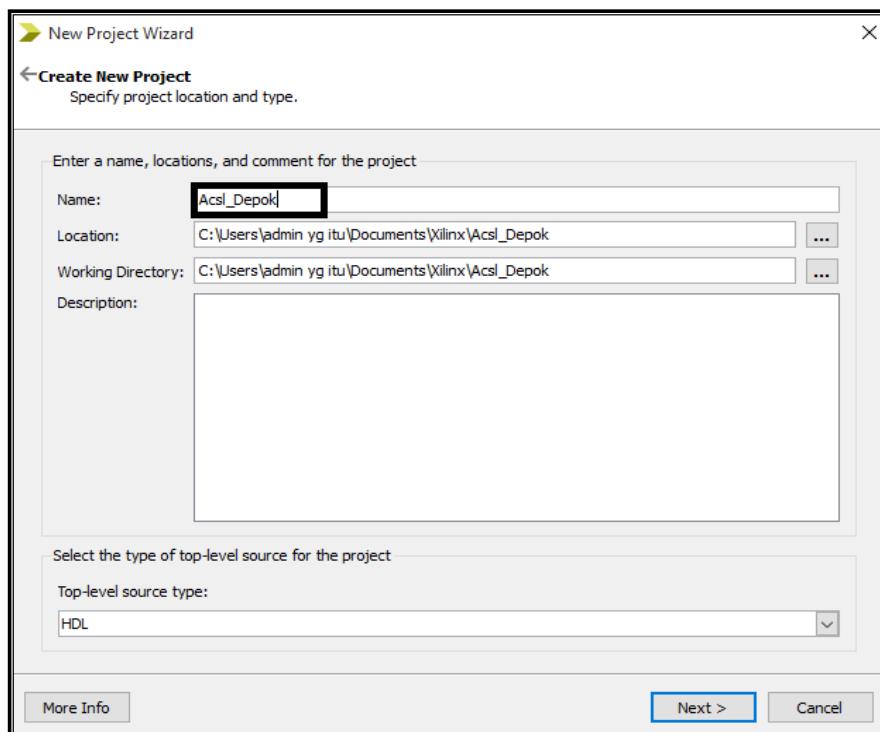
KESIMPULAN:



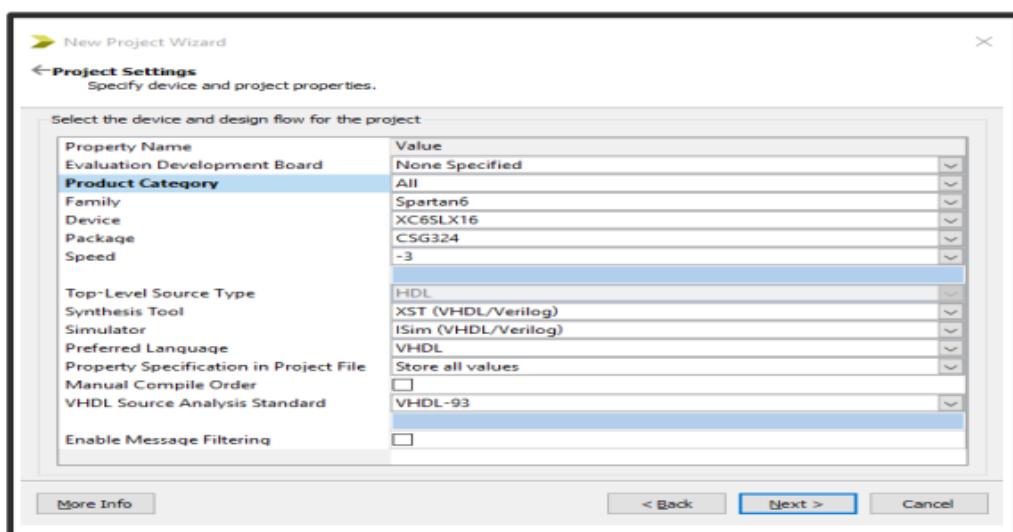
Percobaan 2 : Membuat Rangkaian Gerbang Logika dengan Menggunakan VHDL

Langkah – langkah :

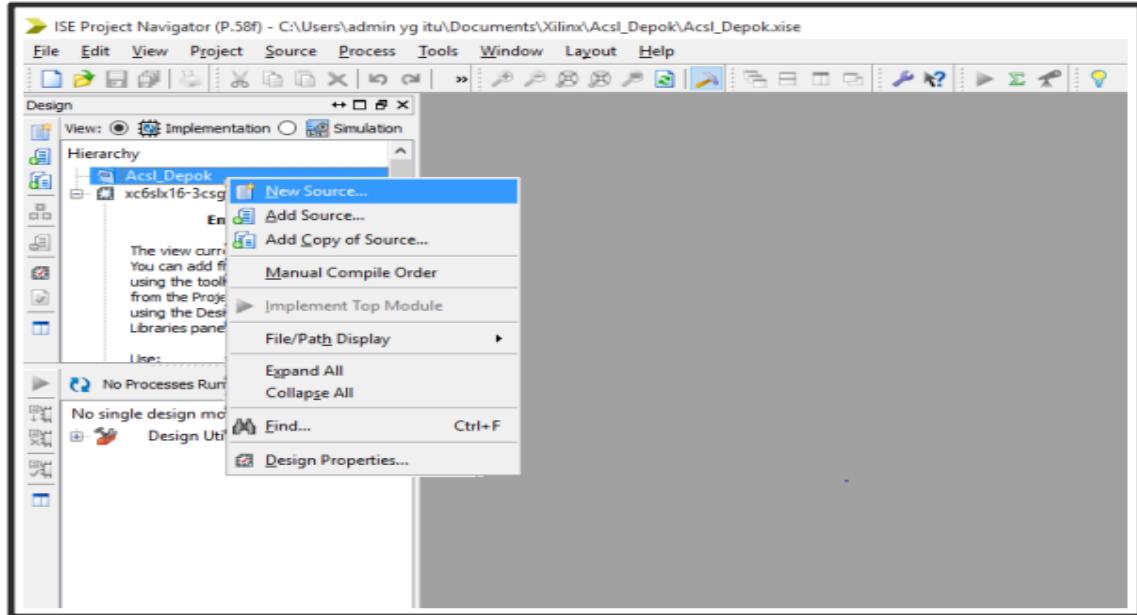
1. Buka aplikasi ISE Design Suite 14.5
2. Buka Menu File > New Project
3. Buat nama project lalu ,pada bagian Top-level source type klik next seperti gambar di bawah ini :



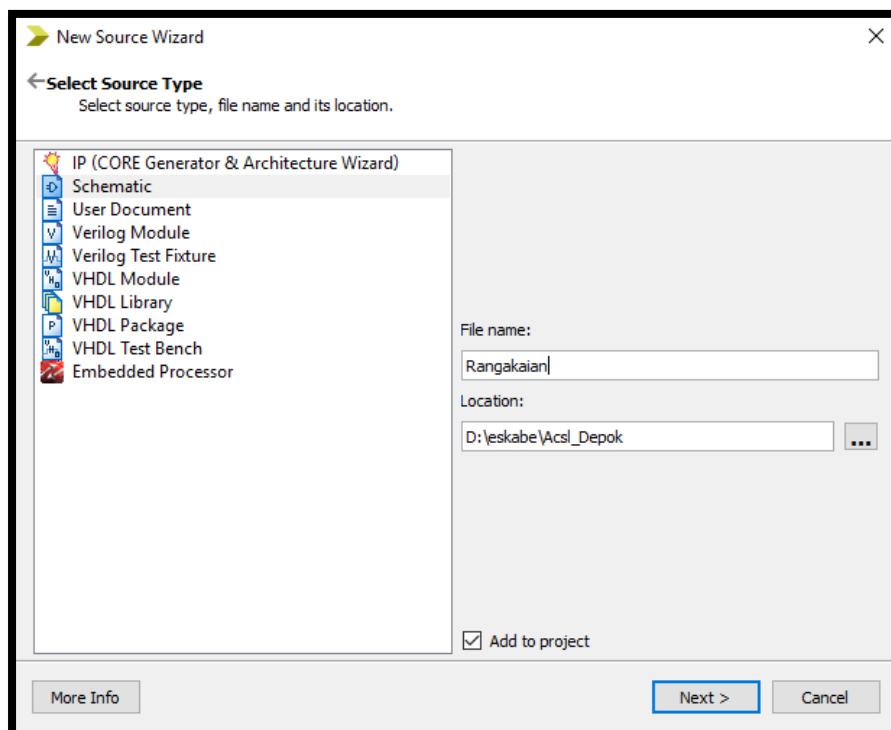
4. Setelah itu akan muncul project settings dan ikuti aturannya seperti gambar di bawah ini :



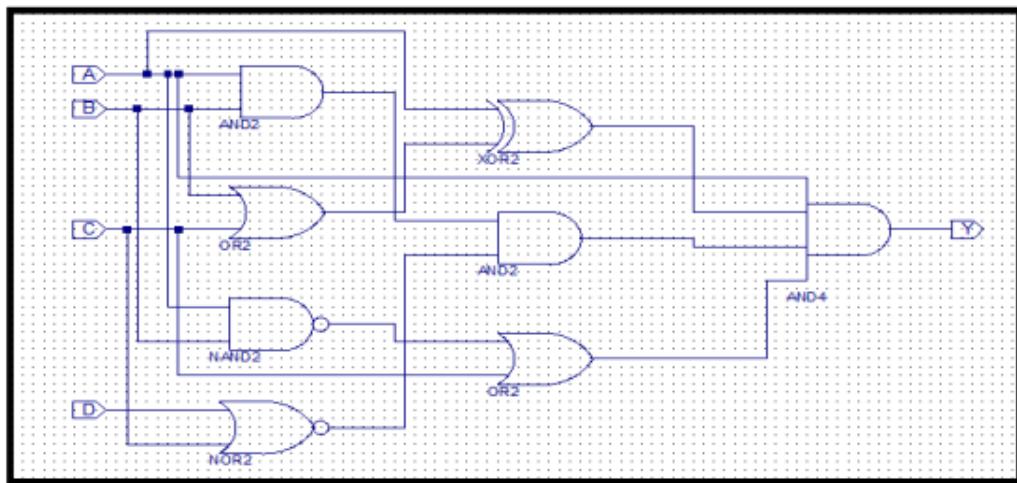
5. Klik **Next >** lalu klik **Finish**
6. Lalu akan muncul halam form untuk project, setelah itu **klik kanan** pada nama **Acsl_Depok > Pilih New Source** seperti gambar di bawah ini :



7. Pilih bagian **Schematic** lalu isikan nama project dengan **Rangkain**, lalu klik **Next** seperti gambar di bawah ini :



8. Buat source code untuk gambar rangkaian gerbang logika seperti dibawah ini:



SOURCE CODE:

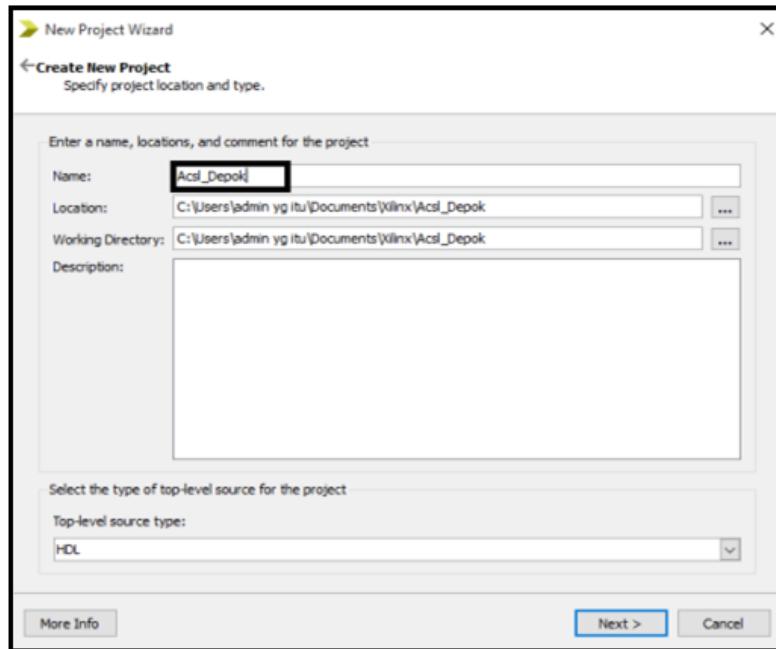
OUTPUT & KESIMPULAN:



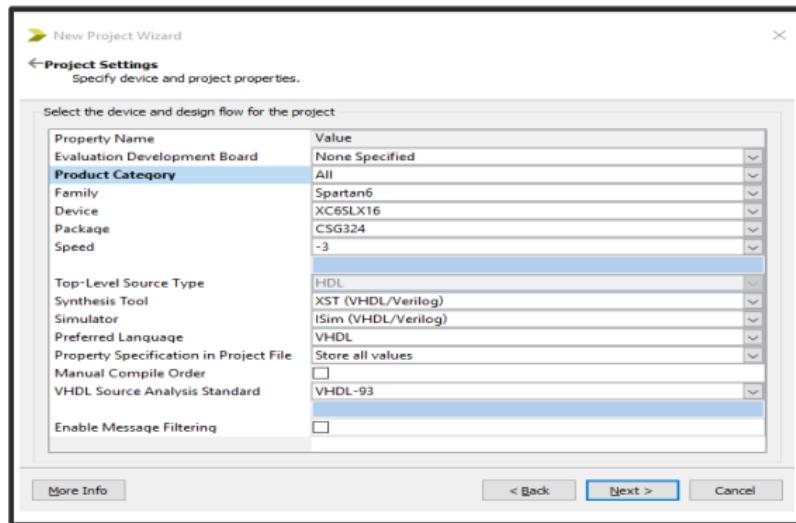
Percobaan 3 : Membuat program I/O dasar dengan VHDL

Langkah - langkah :

1. Buka aplikasi **ISE Design Suite 14.5**
2. Buka Menu **File > New Project**
3. Buat nama project lalu klik next seperti gambar di bawah ini :



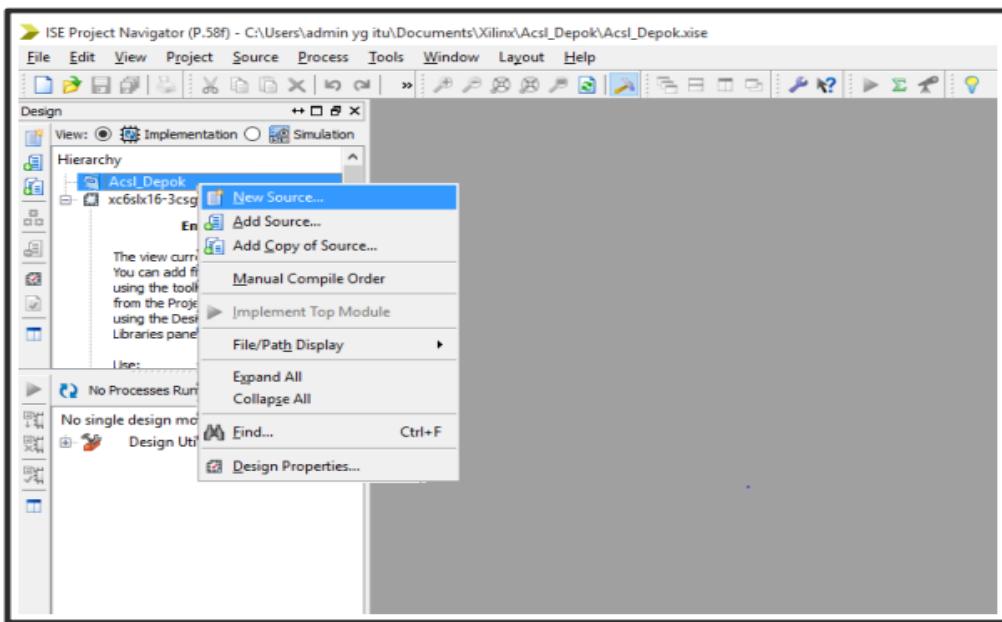
4. Setelah itu akan muncul project settings dan ikuti aturannya seperti gambar di bawah ini :



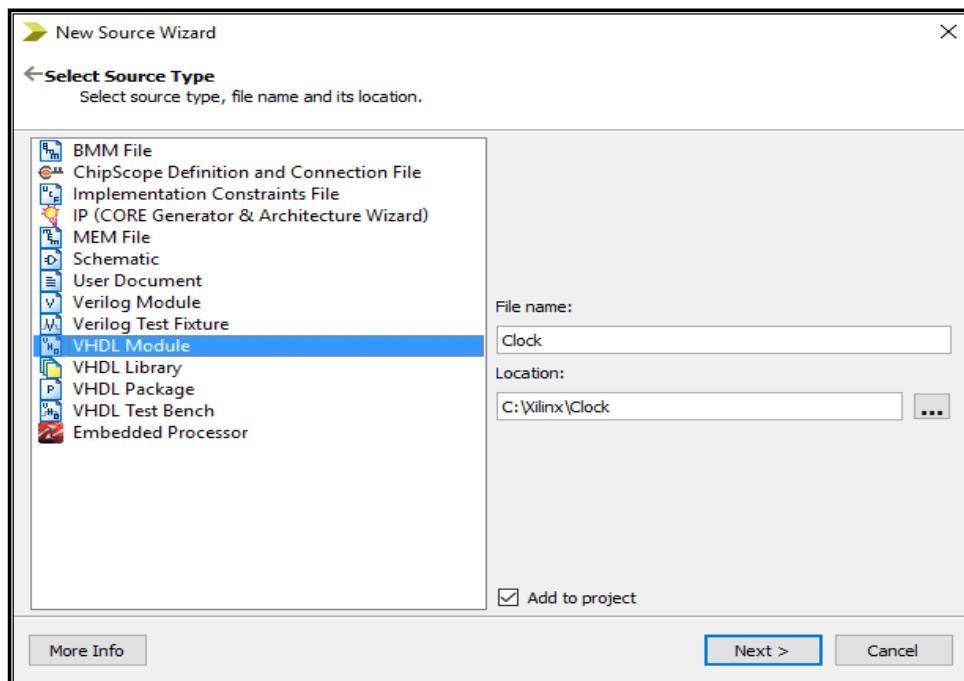
5. Klik **Next >** lalu klik **Finish**



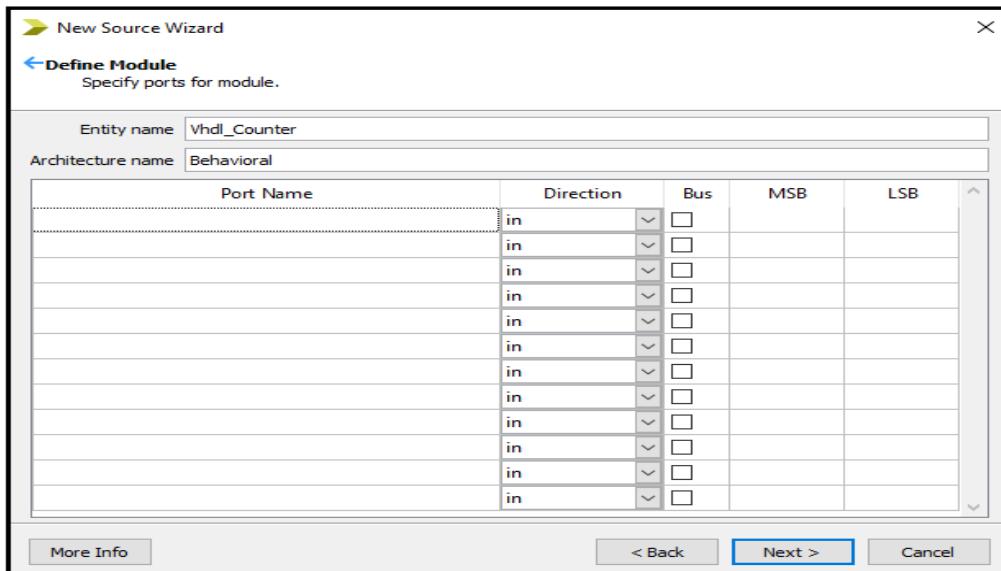
6. Lalu akan muncul halam form untuk project, setelah itu **klik kanan** pada nama **AcsI_Depok > Pilih New Source** seperti gambar di bawah ini :



7. Pilih bagian **VHDL Module** lalu isikan nama project dengan **Clock**, lalu klik **Next** seperti gambar di bawah ini :



8. Selanjutnya untuk bagian I/O tidak usah diisi > selanjutnya klik Next dan Finish.



9. Masukan source code seperti di bawah ini dan Save :

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5 entity clock is
6 port (
7
8     reset : in std_logic;
9     clock : in std_logic;
10    saklar : in std_logic_vector (3 downto 0);
11    led : out std_logic_vector (3 downto 0));
12 end clock;
13
14 architecture Behavioral of clock01 is
15
16 begin
17 process(reset,clock)
18
19     begin
20
21         if (reset = '1')
22 then
23
24             led <= (led' range => '0');
25
26         elsif (clock' event and clock = '1') then
27
28             led <= saklar;
29
30         end if;
31
32     end process;
33
34 end Behavioral;
```

standar library yang sudah ditentukan oleh IEEE;
seperti library yang digunakan untuk variable,
perhitungan aritmatika, dll

bagian dimana I/O ditentukan contohnya seperti pada program ini yaitu
input reset, clock, saklar, dan output berupa led.
Untuk saklar digunakan 3 downto 0 karena menggunakan 4 switch dan
4 led

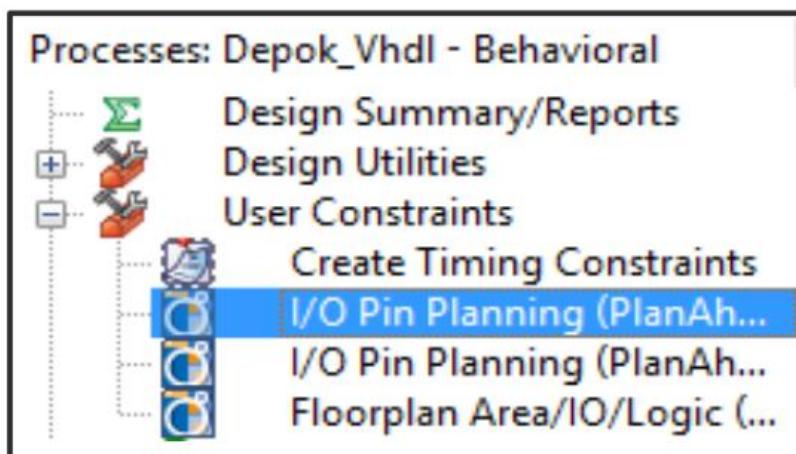
program ini menjelaskan kondisi untuk switch
dan led yang digunakan sebagai input / output
pada saat switch di geser ke atas akan bernilai 1
dan led juga akan bernilai 1 dan meneruskan
hasil input dari switch yaitu berupa nyala led

10. Pada bagian process pilih Synthesize – XST lalu klik kanan dan pilih Run

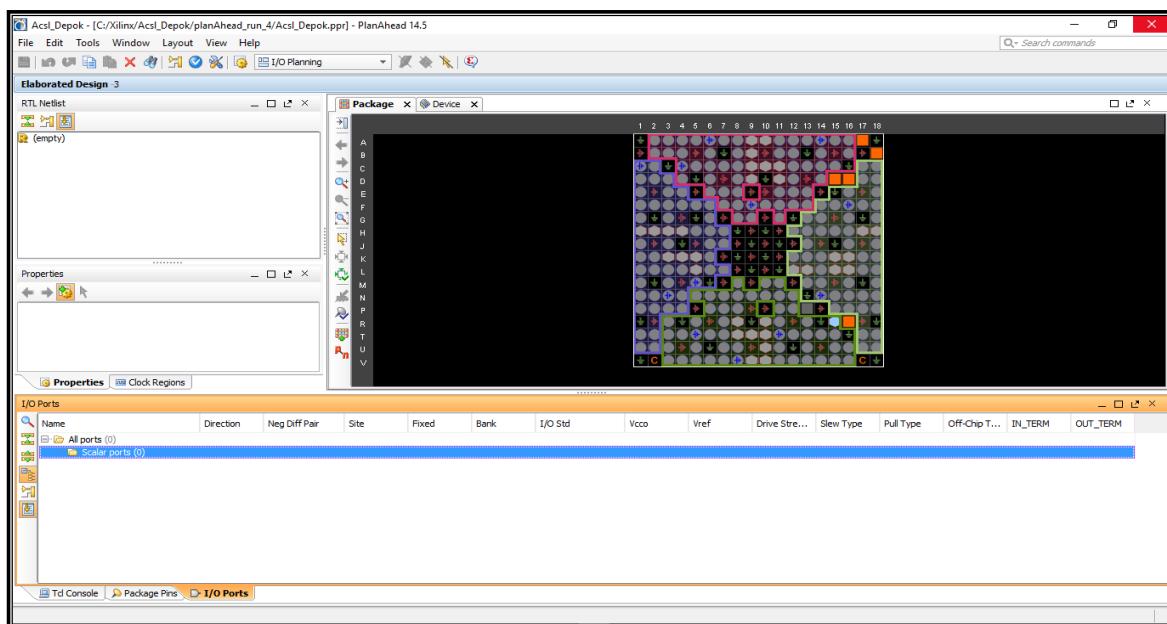




11. Klik tanda + pada user constraints pilih **I/O Pin Planning (PlanAhead) – Pre Synthesis**, klik kanan pilih **Run** -> pilih **Yes**



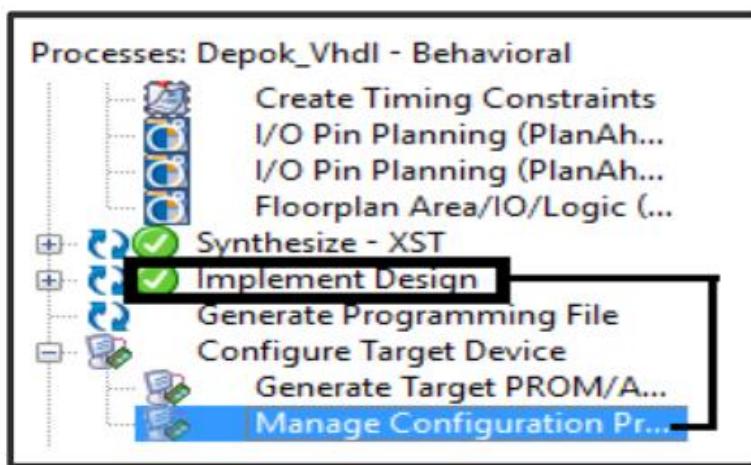
12. Setelah muncul tampilan seperti ini > **Save** dan **Close**. Selanjutnya cek dibagian hierarchy akan muncul file yang bernama Clock.ucf



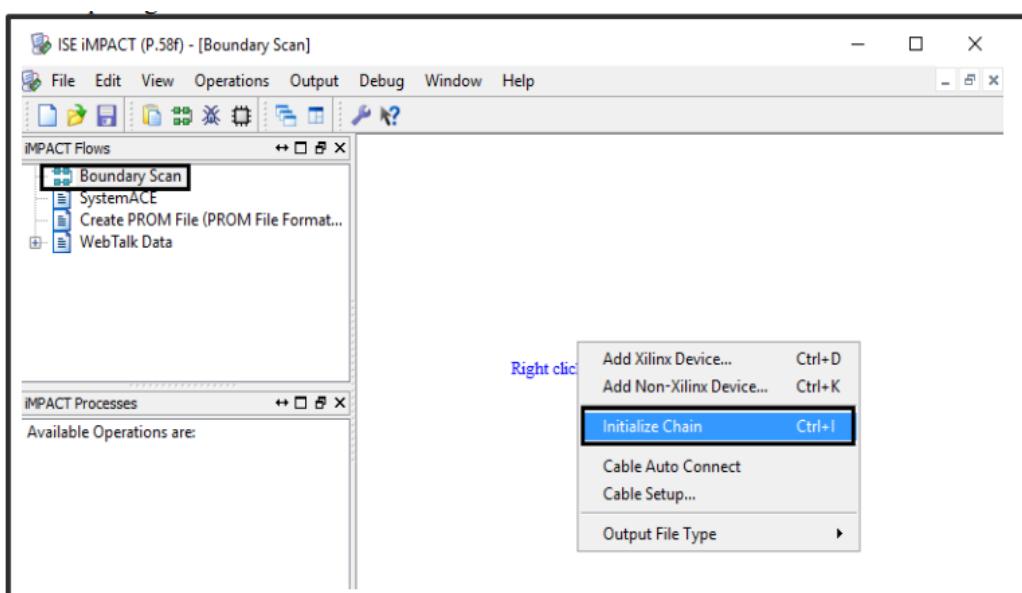
13. Klik dan ketik dibagian text editor seperti gambar dibawah ini :

```
4  NET "saklar[3]" LOC = T10;
5  NET "saklar[2]" LOC = T9;
6  NET "saklar[0]" LOC = M8;
7  NET "led[2]" LOC = V16;
8  NET "saklar[1]" LOC = V9;
9  NET "led[3]" LOC = U16;
10 NET "led[1]" LOC = U15;
11 NET "led[0]" LOC = V15;
12 NET "reset" LOC = B8;
13 NET "clock" LOC = V10;
```

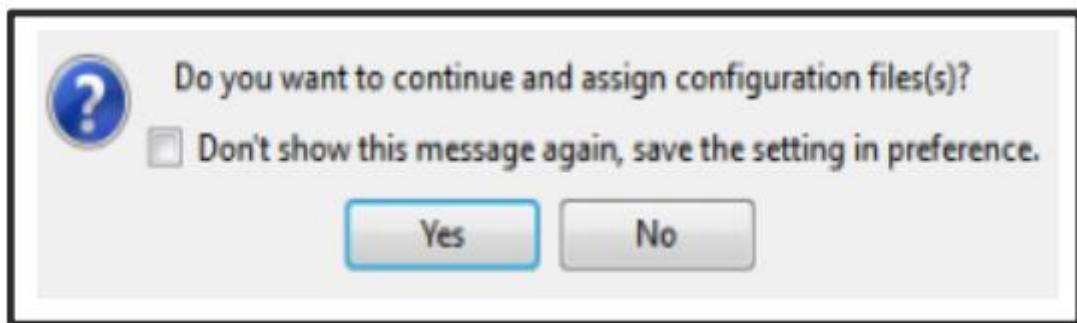
14. Selanjutnya pilih **Implement Desain** lalu klik tanda + pada **Configure Target Device** dan Pilih Manage Configure Project (iMPACT).



15. Tampilan selanjutnya akan muncul pilih **Boundary Scan** > lilih klik kanan pada form “Right click to Add Device or Initialize JTAG chain” > pilih **Initialize Chain** seperti gambar di bawah ini :

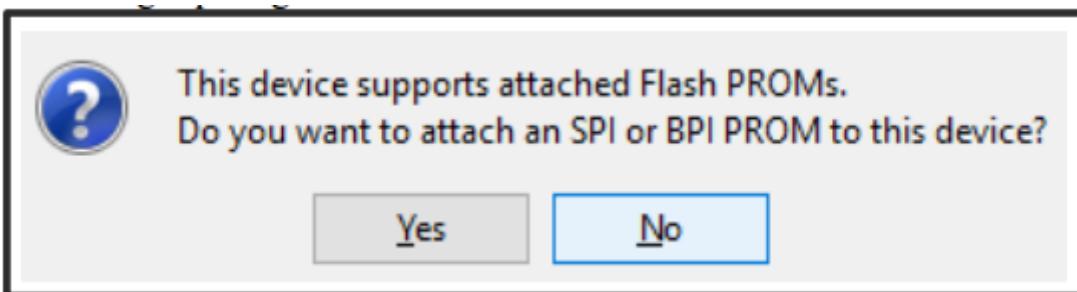


16. Pilih Yes pada Auto Assign Configuration Files Query Dialog

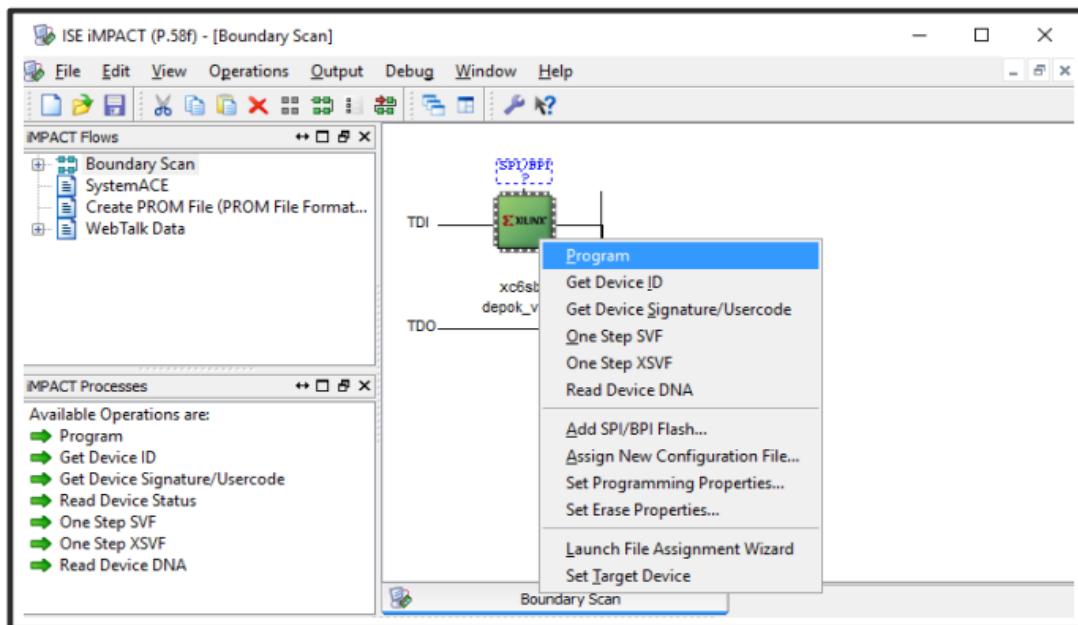


17. Jika muncul tampilan untuk mencari file bit pilih file yang ber extensi .bit > klik Open

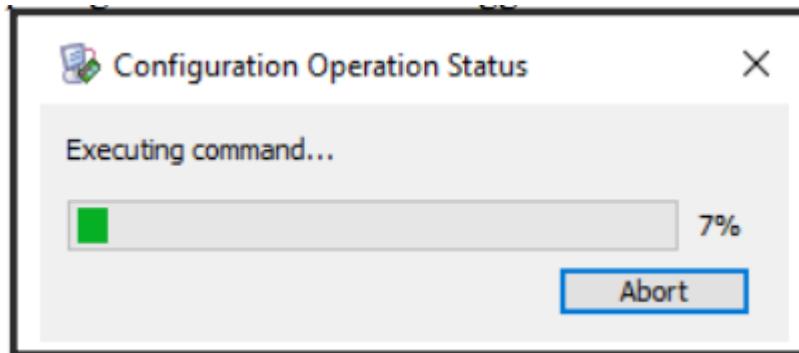
18. Pilih No pada dialog seperti gambar di bawah ini > dan Pilih OK :



19. Klik kanan di bagian IC FPGA dan Pilih Program.



20. Tunggu Proses seperti gambar di bawah ini hingga selesai :



21. Setelah **100%** lihat hasilnya di FPGA kalian.

OUTPUT & KESIMPULAN:



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal dan Memahami Penggunaan dari Clock
- Praktikan Dapat Memahami Pembuatan Timer dan PWM pada FPGA
- Praktikan Dapat Memahami Pengaplikasian Timer dan PWM

II. Dasar Teori

- Pengenalan Teori dan Perhitungan Clock
- Merancang Desain Program Pembuatan Timer dan PWM
- Aplikasi Perancangan Program Desain

III. Peralatan

- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC



2.1 Mengenal Clock

Clock merupakan sinyal listrik yang berupa suatu denyutan dan berfungsi untuk mengkoordinasikan atau mensinkronisasikan setiap aksi-aksi atau proses-proses yang dilakukan oleh setiap komponen di dalam perangkat elektronika. Bagaimana proses A, bagaimana proses B,... bagaimana proses X berjalan bersama proses A, bagaimana proses Z berjalan dengan proses B, ... dst. Oleh karena itu, nilai clock sangat penting artinya agar perangkat elektronika dapat berfungsi sebagaimana mestinya.

Ada beberapa istilah penting yang berkaitan dengan Clock yaitu :

- Cycle : adalah satuan yang digunakan untuk menandakan selesainya satu siklus clock, mulai dari denyutan dikeluarkan kemudian naik hingga nilainya mencapai 1 lalu mulai turun nilainya hingga 0
- Cycle Time (T) : adalah jumlah waktu yg diperlukan oleh sinyal clock untuk menyelesaikan satu (1) siklus clock
- Rise Time : adalah waktu yang dibutuhkan untuk perubahan nilai clock dari 0 ke 1
- Fall Time : adalah waktu yang dibutuhkan untuk perubahan nilai clock dari 1 ke 0
- Clock Frequency (F) : adalah besaran untuk menilai kemampuan suatu sinyal clock dalam menciptakan satu siklus denyutan setiap detiknya alias berapa banyak cylce per detik yang dapat di hasilkan oleh sinyal clock. Sesuai standra internasional, Satuan yang digunakan untuk mengukurnya adalah Hertz = Hz, dimana 1Hz sama dgn satu cycle per detik.

Sebagai contoh, jika sinyal clock membutuhkan waktu 10ms (micro second) dalam menyelesaikan satu siklus denyutan (cycle) maka clock frequency = $1/0,001 = 1000 \text{ Hz} = 1\text{KHz}$

$$F = 1/T \rightarrow T = 1/F$$



2.2 Timer dan Counter



Timer & Counter merupakan fitur yang telah tertanam pada mikrokontroler yang memiliki fungsi terhadap waktu. Akan tetapi penggunaan Timer & Counter pada FPGA harus ditambahkan sendiri. Fungsi waktu yang dimaksud disini adalah penentuan kapan program tersebut dijalankan, tidak hanya itu saja fungsi timer yang lainnya adalah PWM, ADC, dan Oscillator. Prinsip kerja timer dengan cara membagi frekuensi (prescaler) pada clock yang terdapat pada FPGA sehingga timer dapat berjalan sesuai dengan frekuensi yang dikehendaki.

Timer merupakan fungsi waktu yang sumber clocknya berasal dari clock internal. Sedangkan counter merupakan fungsi perhitungan yang berasal dari clock tersebut maupun eksternal. Salah satu contoh penggunaan fungsi timer yaitu pada jam digital yang sumber clocknya bisa menggunakan crystal oscillator dan contoh penggunaan counter pada penghitung barang pada konveyor yang sumber clocknya berasal dari sensor yang mendeteksi barang tersebut.

2.2.1 Tujuan Penggunaan Timer

1. Melaksanaan tugas secara ber-ulang
2. Mengendalikan kecepatan motor DC (PWM)
3. Melakukan perhitungan (Counter)
4. Membuat penundaan waktu (delay)

2.2.2 Prescaler

Pada dasarnya Timer hanya menghitung pulsa clock. Frekuensi pulsa clock yang dihitung tersebut bisa sama dengan frekuensi crystal yang digunakan atau dapat diperlambat menggunakan prescaler dengan faktor sebagai missal, 8, 64, 256 atau 1024.

Contoh penggunaan prescaler :

Suatu mikrokontroler menggunakan crystal dengan frekuensi 8 MHz dan timer yang digunakan adalah Timer 1 (16 Bit) maka maksimum waktu Timer yang bisa dihasilkan adalah :

Diketahui :



- f_{CLK} (frekuensi CLOCK) = 8 MHz
- FFFFh (timer yang digunakan) = 16 Bit
- N (prescaler) = 1024

Jawab :

$$T_{MAX} = 1/f_{CLK} \times FFFFh$$

$$= 0.125 \mu s \times 65536$$

$$= 0.008192 s$$

Untuk menghasilkan waktu Timer yang lebih lama dapat menggunakan prescaler 1024, maka waktu Timer yang bisa dihasilkan adalah :

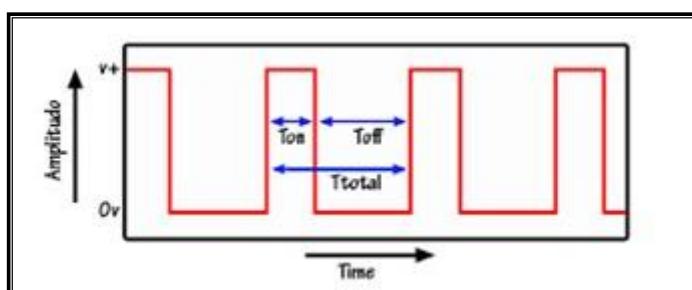
$$T_{MAX} = 1/f_{CLK} \times FFFFh \times N$$

$$= 0.125 \mu s \times 65536 \times 1024$$

$$= 8.388736 s$$

2.3 Pulse Width Modulation (PWM)

Pulse Width Modulation (PWM) secara umum adalah sebuah cara memanipulasi lebar sinyal yang dinyatakan dengan pulsa dalam satu periode, untuk mendapatkan tegangan rata-rata yang berbeda. Penggunaan PWM ini mengantikan output analog dikarenakan keterbatasan sebuah IC yang pada umumnya hanya dapat mengeluarkan sinyal digital. Beberapa contoh aplikasi PWM adalah pemodulasi data untuk telekomunikasi, pengontrolan daya atau tegangan yang masuk ke beban, regulator tegangan, audio effect dan penguatan, serta aplikasi-aplikasi lainnya.



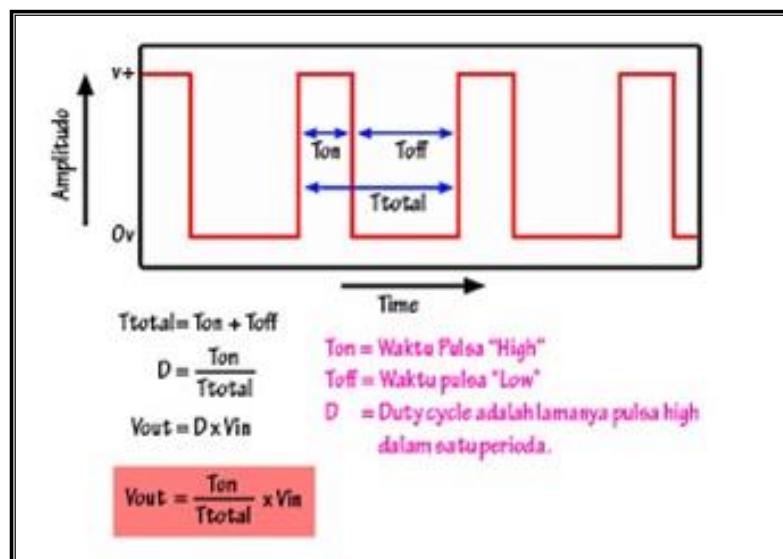
Gambar 2.1: Grafik PWM



Aplikasi PWM biasanya berupa pengendalian kecepatan motor DC, pengendalian motor servo, dan pengaturan nyala terang LED. Oleh karena itu diperlukan pemahaman terhadap konsep PWM itu sendiri.

2.3.1 Konsep Dasar PWM

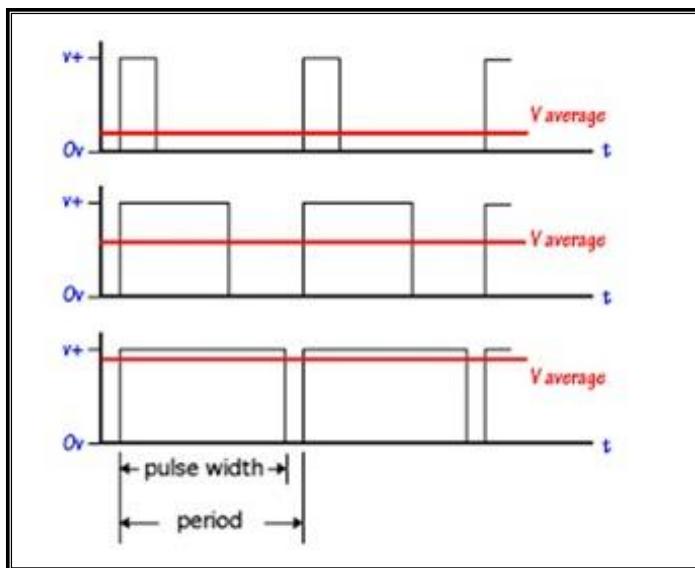
Sinyal PWM pada umumnya memiliki amplitudo dan frekuensi dasar yang tetap, namun memiliki lebar pulsa yang bervariasi. Lebar pulsa PWM berbanding lurus dengan amplitudo sinyal asli yang belum termodulasi. Artinya, sinyal PWM memiliki frekuensi gelombang yang tetap namun duty cycle bervariasi antara 0% hingga 100%.



Gambar 2.2: Perhitungan V_{out} PWM

Dari persamaan diatas, diketahui bahwa perubahan duty cycle akan merubah tegangan output atau tegangan rata-rata seperti gambar dibawah ini.

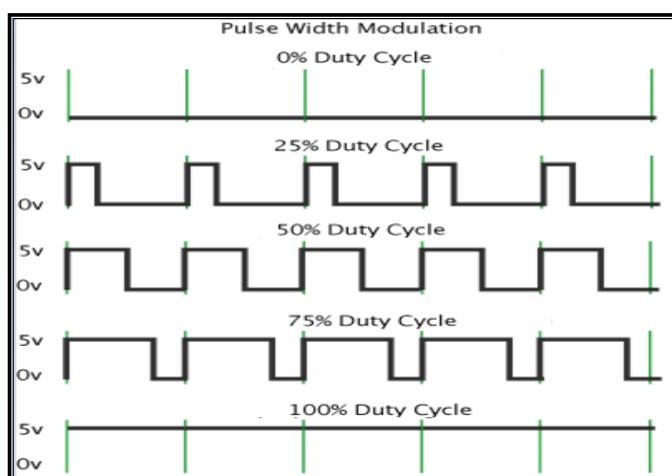




Gambar 2.3: Hasil Vout berdasarkan panjang pulsa

Untuk mengatur nilai *duty cycle* pada timer bit 8, nilai pada parameter berkisar antara 0 hingga 255. Bila kita hendak mengeset *duty cycle* ke 0%, maka kita set nilai parameter ke 0, dan untuk *duty cycle* 100%, maka kita set nilai parameter ke 255. Jadi bila misalkan kita hendak mengeset *duty cycle* ke 50%, berarti nilai yang harus kita set adalah 127 ($50\% \times 255$).

Analog input dihasilkan oleh teknik yang dikenal dengan istilah PWM atau *Pulse Width Modulation*. PWM memanipulasi keluaran digital sedemikian rupa sehingga menghasilkan sinyal analog. Mikrokontroler mengeset output digital ke HIGH dan LOW bergantian dengan porsi waktu tertentu untuk setiap nilai keluarannya. Durasi waktu untuk nilai HIGH disebut *pulse width* atau panjang pulsa. Variasi nilai output analog didapatkan dari perubahan panjang pulsa yang diberikan pada satu periode waktu dan dilakukan berulang-ulang. Untuk lebih jelasnya perhatikan ilustrasi berikut:



Gambar 2.4: Grafik duty cycle PWM



Kondisi HIGH adalah kondisi ketika sinyal berada di atas grafik (5V) dan LOW adalah ketika sinyal berada di bawah (0V). *Duty cycle* adalah persentasi panjang pulsa HIGH dalam satu periode sinyal. Ketika *duty cyclenya* 0% atau sinyal LOW penuh, maka nilai analog yang dikeluarkan adalah 0V atau setara dengan GND. Ketika *duty cyclenya* 100% atau sinyal HIGH penuh maka sinyal yang dikeluarkan adalah 5V.

3.4 Penggunaan PWM

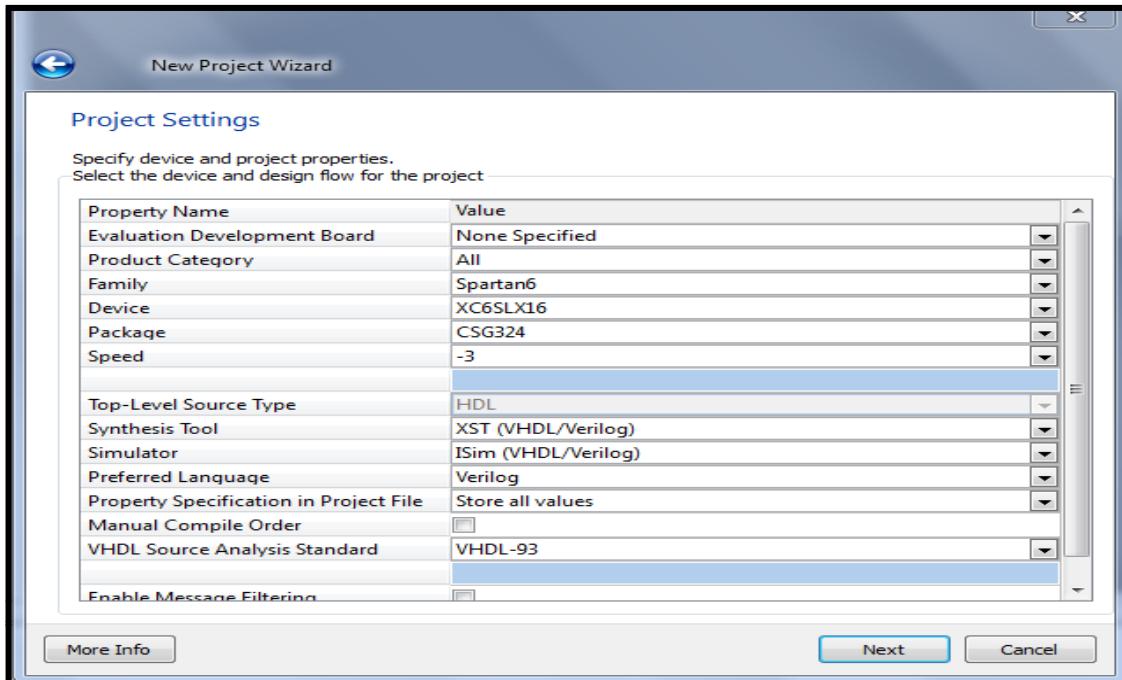
1. PWM sebagai data keluaran suatu perangkat. PWM dapat digunakan sebagai data dari suatu perangkat, data direpresentasikan dengan lebar pulsa positif (T_p).
2. PWM sebagai data masukan kendali suatu perangkat. Selain sebagai data keluaran, PWM pun dapat digunakan sebagai data masukan sebagai pengendali suatu perangkat. Salah satu perangkat yang menggunakan data PWM sebagai data masukannya adalah Motor Servo. Motor Servo itu sendiri memiliki dua tipe: 1. Kontinyu, 2. Sudut. Pada tipe 1., PWM digunakan untuk menentukan arah Motor Servo, sedangkan pada tipe 2., PWM digunakan untuk menentukan posisi sudut Motor Servo.
3. PWM sebagai pengendali kecepatan Motor DC bersikat. Motor DC bersikat atau Motor DC yang biasa ditemui di pasaran yang memiliki kutub A dan kutub B yang jika diberikan beda potensial diantara kedua-nya, maka Motor DC akan berputar. Pada prinsipnya Motor DC jenis ini akan ada waktu antara saat beda potensial diantara keduanya dihilangkan dan waktu berhentinya. Prinsip inilah yang digunakan untuk mengendalikan kecepatan Motor DC jenis ini dengan PWM, semakin besar lebar pulsa positif dari PWM maka akan semakin cepat putaran Motor DC. Untuk mendapatkan putaran Motor DC yang halus, maka perlu dilakukan penyesuaian Frekuensi (Perioda Total) PWM-nya.

Percobaan 1 : Timer

Langkah-Langkah:



1. Buka Xilinx ISE 14.5
2. Klik **File -> New Project**
3. Ketik nama project dengan nama **Timer** lalu klik **next**
4. Atur Device Properties seperti gambar dibawah ini :



5. Klik **Next -> Next -> Finish**
6. Klik kanan pada subname **Clock** yang ada pada tab hierarchy, pilih **new source** -> ketik nama file namanya **Counter** pilih **VHDL Module** -> klik **next**
7. Copy source code yang diberikan Asisten
8. Lalu **Save**
9. Pada bagian tab processes pilih **Synthesize -XST** klik tanda +klik kanan **CheckSyntax** pilih run atau cukup klik 2x pada **Synthesize -XST**
10. Maka selanjutnya klink kanan pada subname yang diberi nama **Counter**->pilih newsource, ketik nama file namanya **Counter** pilih **Implementation Contrains File** -> klik next. (**Menggunakan UCF**)
11. Sebelum masuk text editor, klik tanda + pada project gerbangAnd, di hierarchy -> double klik **Timer.ucf**. Pada text editor ketiklah pin FPGA seperti dibawah ini : (**Setelah selesai jangan lupa untuk save**)

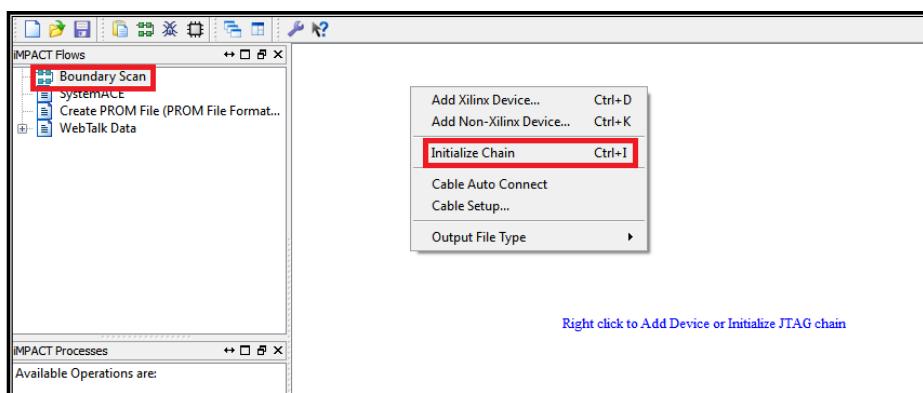


```

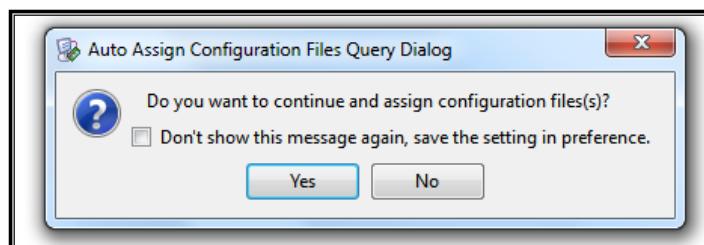
1  NET "count_out[0]" LOC = U16;
2  NET "count_out[2]" LOC = U15;
3  NET "count_out[3]" LOC = V15;
4  NET "count_out[1]" LOC = V16;
5  NET "clk" LOC = V10;
6  NET "pause" LOC = T5;
7  NET "reset" LOC = D9;

```

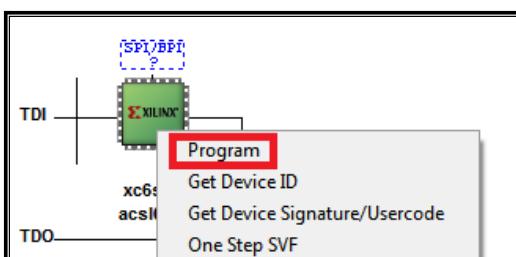
12. Selanjutnya klik 2x **Implement Desain** (Sebelum klik 2x **Implement Desain**, jangan lupa untuk menghubungkan dengan FPGA terlebih dahulu).
13. Lalu klik tanda + pada **Configure Target Device**, Pilih **Manage Configuration Project**
14. Pilih boundary scan, lalu klik kanan pada area “right click to add device or initialize JTAG chain”, kemudian pilih **Initialize Chain** seperti gambar di bawah ini :



15. Pilih Yes pada **Auto Assign Configuration Files Query Dialog**



16. Pilih file **Counter.bit** pada **Assign New Configuration** file lalu pilih open > **Timer.bit**
17. Pada tampilan pop up selanjutnya pilih Cancel -> kemudian pilih OK, maka akan muncul tampilan seperti gambar dibawah ini, setelah itu klik kanan pada gambar IC -> klik **Program**



SOURCE CODE:



OUTPUT & KESIMPULAN:



Percobaan Mandiri 1 :

Buatlah percobaan Timer 4 bit dengan Prescaler 8. Copy Source Code yang diberikan Asisten.

SOURCE CODE :

OUTPUT & KESIMPULAN:



Percobaan Mandiri 2 :

Buatlah percobaan Clock dengan PWM (Pulse Width Modulation) 8 bit. Copy Source Code yang diberikan Asisten.

SOURCE CODE:

OUTPUT & KESIMPULAN:



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal dan Memahami penggunaan FSM pada FPGA
- Praktikan Dapat Merancang Program Desain menggunakan FSM pada Pemrograman FPGA

II. Dasar Teori

- Pengenalan FSM
- Merancang Desain Program menggunakan FSM
- Aplikasi Perancangan Program Desain FSM

III. Peralatan

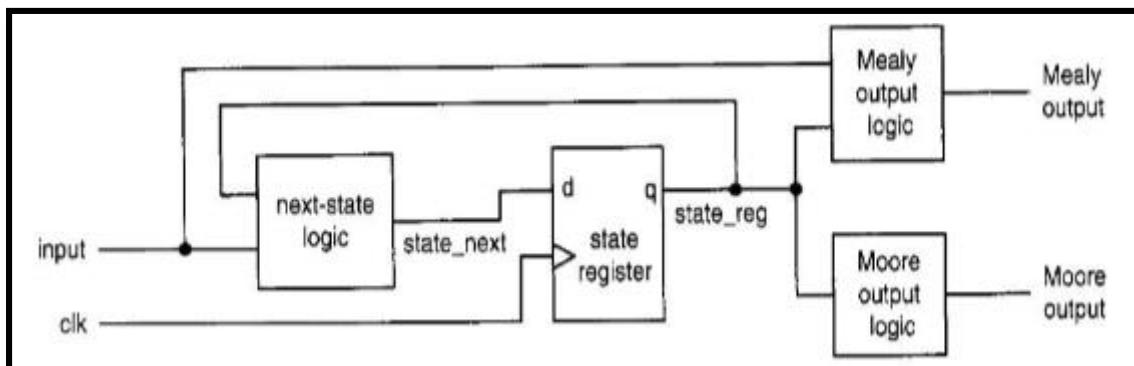
- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC



3.1 Mengenal Finite State Machine

Finite State Machine (FSM) adalah sebuah metodologi perancangan sistem kontrol yang menunjukkan dan menggambarkan suatu tingkah laku atau prinsip kerja pada sistem kontrol tersebut, yang menyangkut 3 hal berikut : State (Keadaan), Event (Kejadian), dan Action (Tindakan). Pada dasarnya cara kerja FSM ini mirip seperti Flowchart hanya saja terdapat perbedaan jika flowchart memerlukan input, proses dan output. Pada FSM ini sendiri terdapat 2 struktur yang membedakan prinsip kerjanya, yaitu struktur Mealy dan Moore.

Struktur Mealy ini untuk memberikan tindakan atau proses sehingga menghasilkan output bergantung pada input dan keadaan. Berbeda dengan Moore, yang tindakan untuk menghasilkan outputnya bergantung sepenuhnya pada keadaan.



Gambar 3.1: Blok Diagram dan Tabel Fungsi dari D Flip-Flop

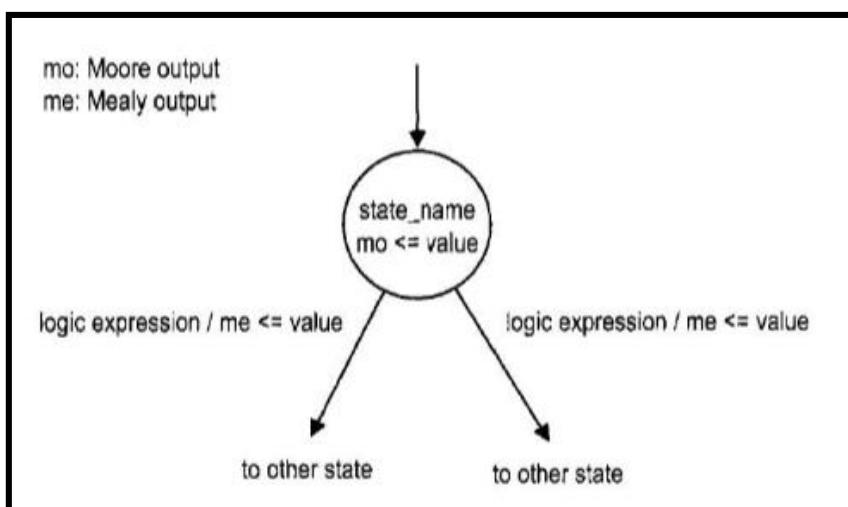
Salah satu contoh penggunaan metodologi FSM bisa dilihat pada sistem pengendali elevator. Prinsipnya hanya memilih lantai yang akan dituju, berhenti pada suatu lantai tertentu, menggerakkan motor agar kotak elevator naik atau turun menggunakan prinsip finite state machine Moore. Penjelasannya, jika lantai tersebut menjadi sebuah keadaan bagi prinsip kerjanya. Ketika keadaan telah ditentukan, maka elevator akan menentukan tindakan atau prosesnya. Contohnya, jika elevator berada di lantai 5, dan pengunjung ingin ke lantai 10 maka sesuai keadaan yang diinginkan, tindakan dari elevator tersebut adalah menggerakkan motor agar kotak elevator naik menuju lantai 10.



Sebuah FSM (keadaan mesin terbatas) digunakan untuk memodelkan sistem yang mentransmisikan antara jumlah terbatas dari keadaan internal. Transisi tergantung pada keadaan akhir dan inputan dari eksternal. Tidak seperti rangkaian sekuensial biasa, keadaan transisi dari suatu FSM tidak menunjukkan sebuah pola berulang sederhana. Dalam FSM sendiri terdapat next-state logic (keadaan selanjutnya) yang terdiri dari scratch (coretan) dan terkadang dikenal sebagai logika "acak", dimana logika ini berbeda dengan next-state logic yang terdapat pada rangkaian sekuensial biasa, yang didalamnya tersusun dari berbagai komponen yang terstruktur, seperti incrementors (penambah nilai) dan shifters (peralihan atau pemindahan). Pada bab ini, diberikan gambaran tentang karakteristik dan representasi tentang FSM. Prinsip kerja dasar pada setiap mesin atau perangkat yang tergantung pada inputan dan keadaan, maupun sepenuhnya bergantung pada keadaan.

3.2 Representasi FSM

Sebuah FSM biasanya ditentukan oleh state diagram yang abstrak atau grafik ASM (algorithmic state machine chart), keduanya mengambil input dari FSM, output, state, dan transisi dalam representasi secara grafik. Kedua representasi memberikan informasi yang sama. Representasi FSM lebih ringkas dan lebih baik untuk aplikasi sederhana. Representasi grafik ASM agak terlihat seperti flowchart dan lebih deskriptif untuk aplikasi dengan transisi yang kompleks kondisi dan tindakan. State Diagram, sebuah state diagram terdiri dari node, yang merupakan gambaran dari states dan digambarkan dengan sebuah lingkaran, serta dijelaskan transisinya menggunakan panah. Sebuah node tunggal dan anak panah ditunjukkan pada gambar 2.2.

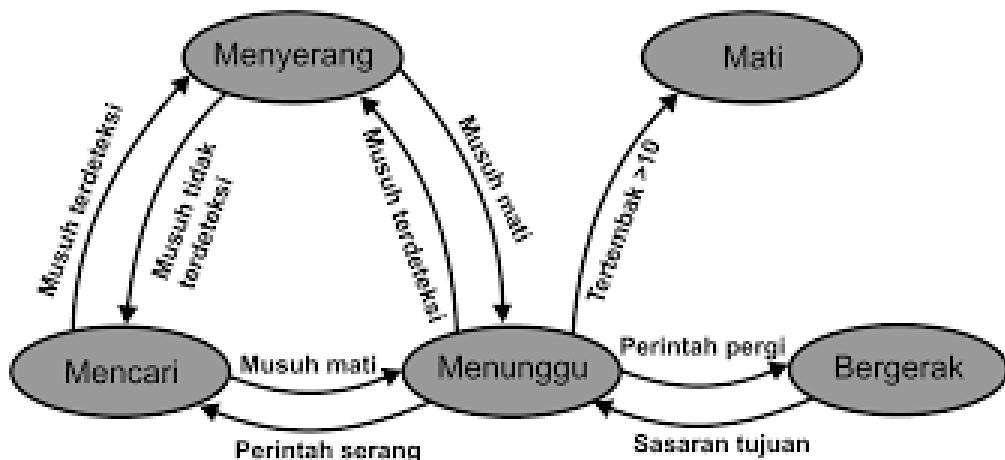


Gambar 3.2: Node Tunggal



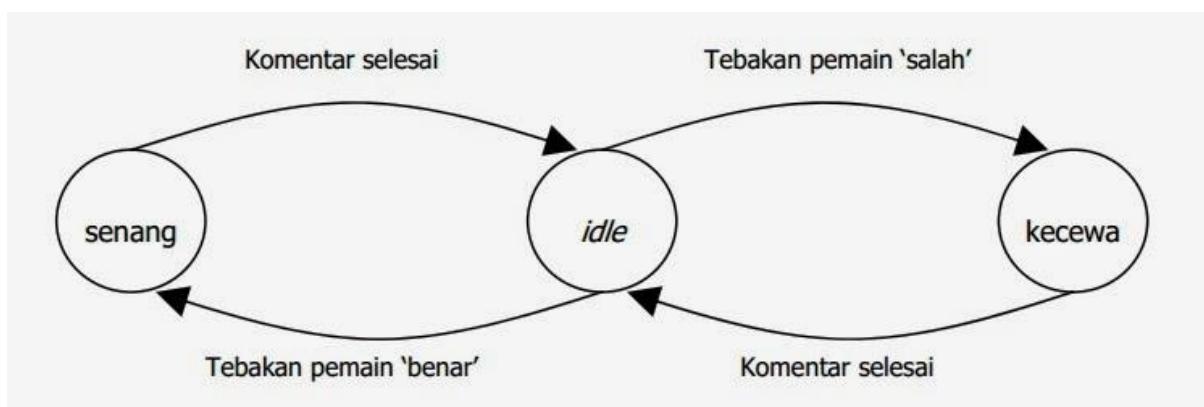
Sebuah ekspresi logika dinyatakan dalam sinyal masukan tergantung dengan masing - masing arah anak panah transisi dan mewakili kondisi tertentu. Anak panah diambil ketika ekspresi yang dievaluasi bernilai true. Hasil output dari Moore ditempatkan di dalam lingkaran karena bergantung pada current state. sedangkan hasil dari ouput Mealy bergantung pada kondisi dari arah anak panah karena hasil output didapatkan berdasarkan current state dan external input. Untuk mengurangi kekacauan dalam diagram, nilai output hanya yang terdaftar.

3.3 Contoh Pengaplikasian FSM



Gambar 3.3 : Contoh Pengaplikasian FSM dalam kehidupan

Dari gambar 3.3 ini merupakan contoh pengaplikasian FSM pada suatu peperangan yang melibatkan prajurit dengan musuh yang diawali dari state ‘Bergerak’.



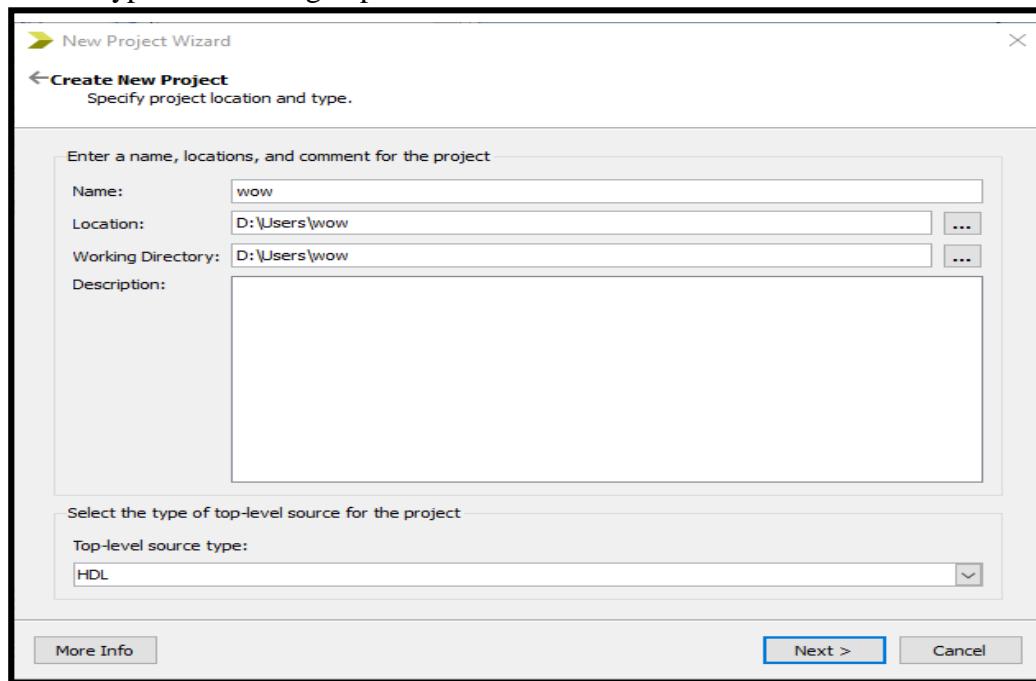
Gambar 3.4 : Contoh Pengaplikasian FSM dalam kehidupan

Pada gambar 3.4 , contoh pengimplementasian teori FSM pada suatu kompetisi cerdas cermat. Yang melibatkan pemain atau peserta dengan juri.

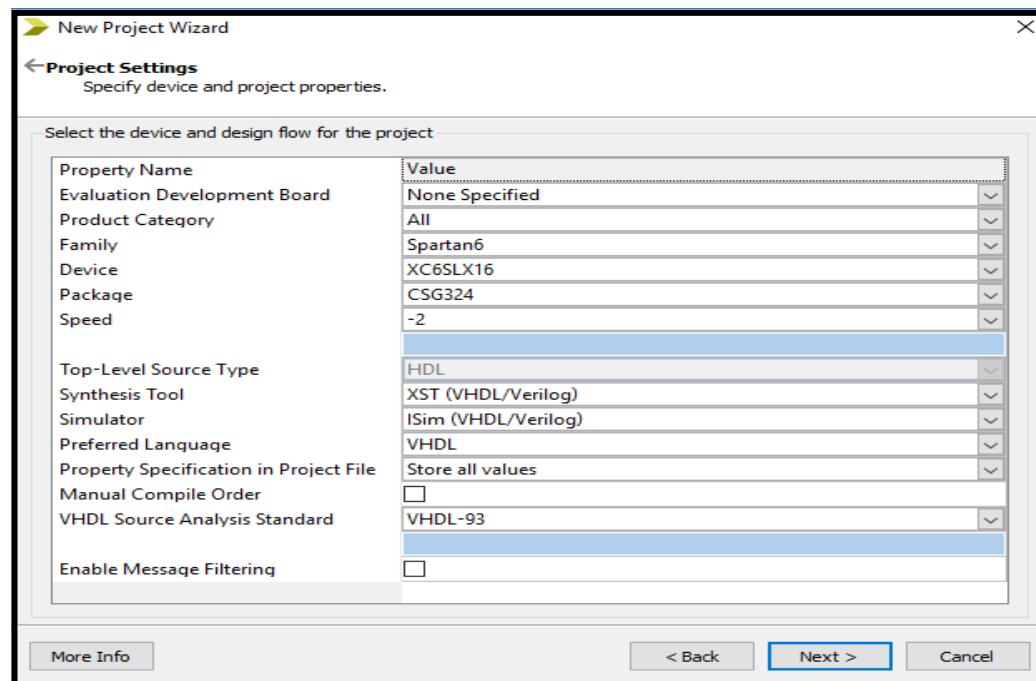


Percobaan 1: Moore FSM

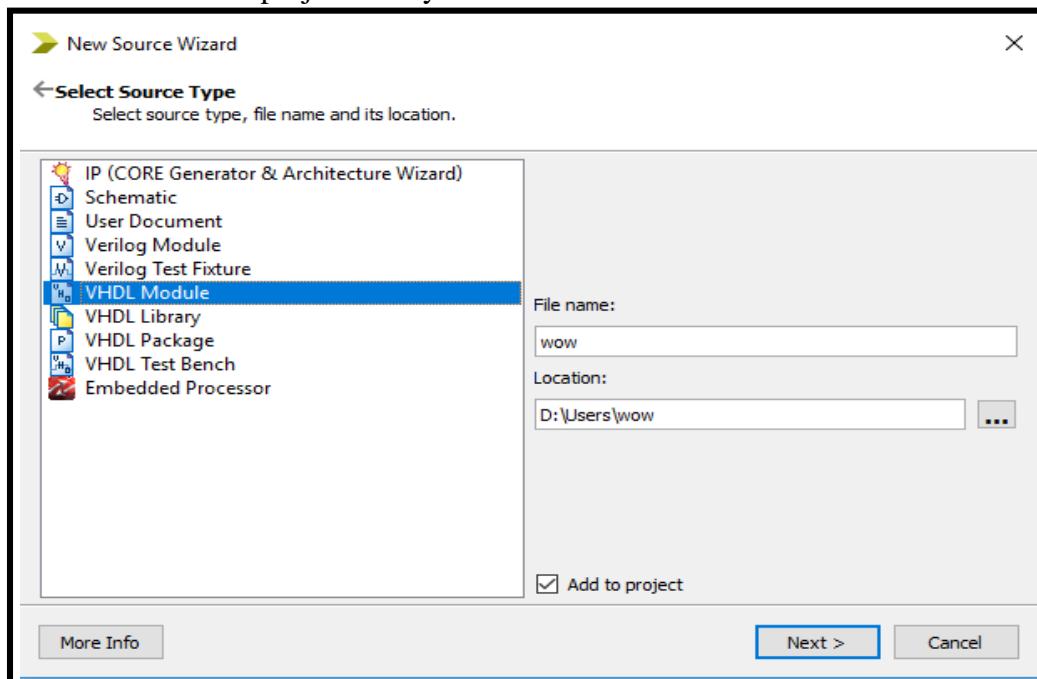
1. Buka software Xilinx ISE 14.5
2. Pilih menu **File** -> Kemudian pilih **New Project**
3. Lalu isikan nama project pada tampilan dengan nama *wow*, kemudian untuk Top-level source type kita isi dengan pilihan **HDL**. Setelah itu klik **Next**.



4. Atur device properties seperti pada gambar di bawah ini. Kemudian klik **Next**.



5. Klik Next lalu klik Finish
6. Klik kanan pada hierarchy lalu pilih new source -> Pada tampilan sebelah kiri gambar pilih *VHDL Module*. Kemudian untuk file name, kita isi dengan nama yang sama pada saat kita membuat project baru yaitu **Wow**. Lalu klik Next.



7. Pada define module, di next saja. Lalu klik finish.
8. Lalu ketikkan program berikut pada source code di setiap bagian yang telah ditentukan :

Bagian 1

```
port
(
    clk      : in std_logic;
    data_in  : in std_logic;
    reset   : in std_logic;
    data_out : out std_logic_vector(1 downto 0)
);
```



Bagian 2

```
begin

clk_division : process (clk, clk_divider)
begin
    if (clk = '1' and clk'event) then
        clk_divider<= clk_divider + 1;
    end if;
    slow_clk<= clk_divider(26);
end process;
```

Bagian 3

```
process (state)
begin
    case state is
        when s0 =>
            data_out <= "00";
        when s1 =>
            data_out <= "01";
        when s2 =>
            data_out <= "10";
        when s3 =>
            data_out <= "11";
    end case;
end process;
```

Kemudian CTRL + S , untuk save program.

9. Langkah berikutnya adalah kita pilih pilihan **Synthesize – XST**. Dengan cara klik dua kali. Untuk mengecek apakah program terdapat error.
10. Klik tanda + pada User Constraints pilih **I/O Pin Planning (PlanAhead) – Pre-Synthesis** dengan klik kanan pilih **Run** atau klik kiri 2x.
11. Nanti akan muncul software PlanAhead. Pada bagian I/O ports, silahkan kalian tentukan pin untuk LED dan SWITCH pada FPGA di bagian Site, jika sudah menentukan lalu enter dah pastikan pada bagian Fixed sudah tercentang.

Name	Direction	Neg Diff Pair	Site	Fixed
All ports (5)				
data_out (2)	Output			<input type="checkbox"/>
data_out[1]	Output			<input type="checkbox"/>
data_out[0]	Output			<input type="checkbox"/>
Scalar ports (3)				
dk	Input			<input type="checkbox"/>
data_in	Input			<input type="checkbox"/>
reset	Input			<input type="checkbox"/>



12. Selanjutnya klik tanda + pada project mains di Hierarchy -> double klik top.ucf -> pilih yes. Dan pastikan pin sudah benar sesuai dengan posisi pin LED dan SWITCH pada FPGA.
13. Lalu klik dua kali pada bagian **Implement Design**
14. Klik tanda + pada **Generate Programming File** -> pilih **Configure Device**
(iMPACT) klik kanan lalu Run
15. Klik 2 kali **Configure Target Device** ->Pilih **Boundary Scan** ->Klik kanan pilih **Initialize Chain**
16. Tunggu beberapa detik lalu **Pilih (- - - nama module- - -)**.bit project yang di buat klik **Open** -> Bila ada tanda warning klik **no**
17. klik **OK** -> Identify Succeeded
18. Klik kanan pada gambar IC Xilinx sebelah kiri lalu pilih **program**
19. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**

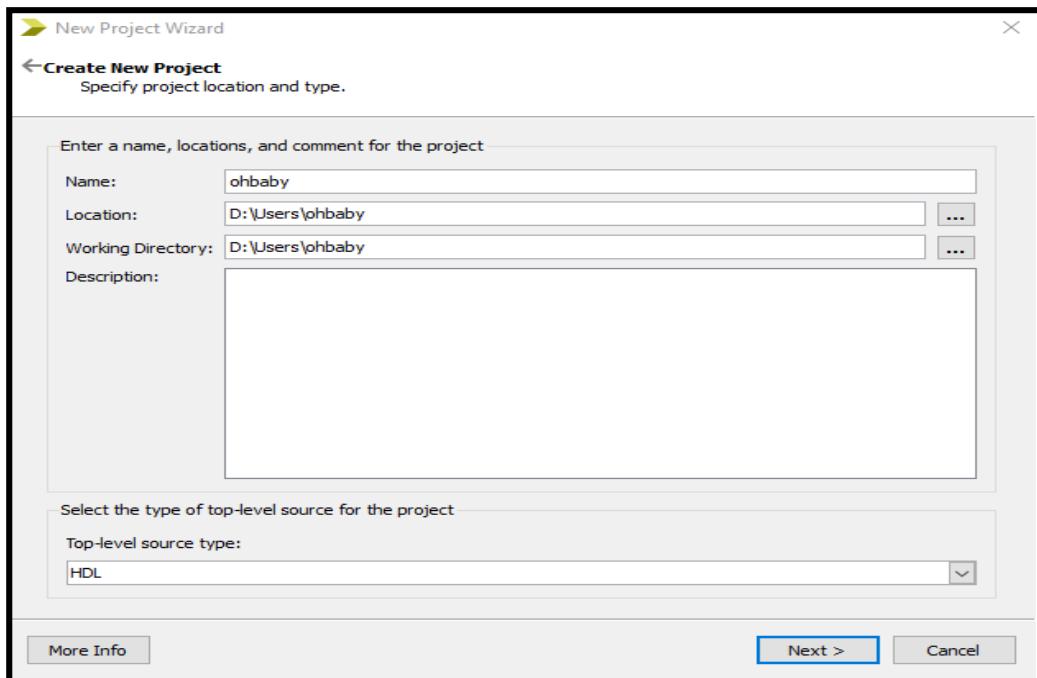
SOURCE CODE:

OUTPUT & KESIMPULAN:

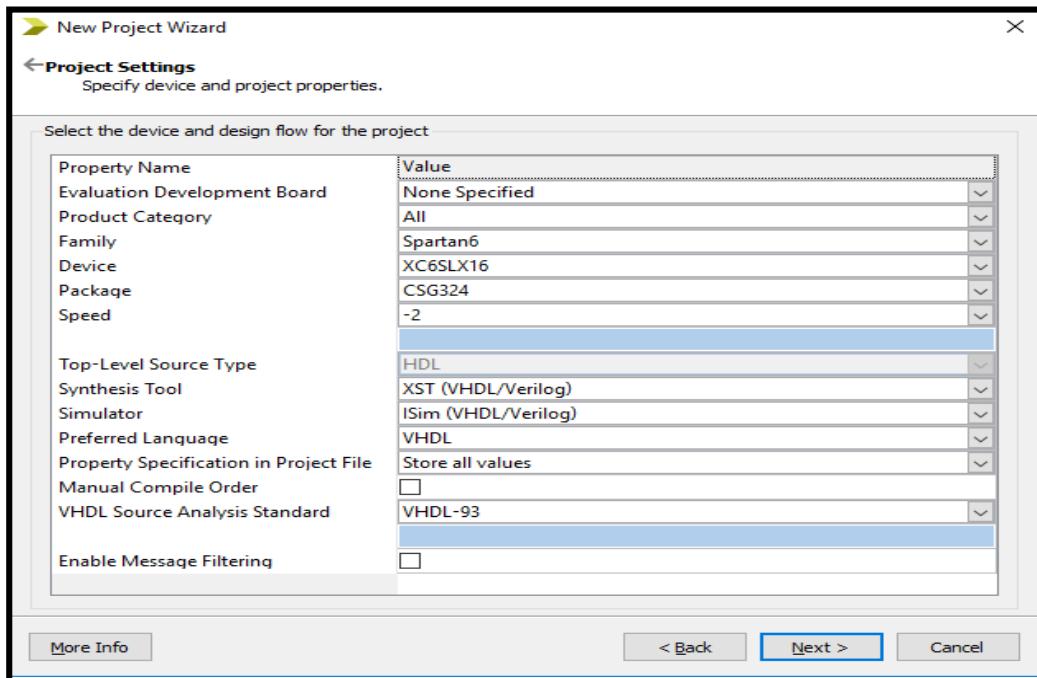


Percobaan 2 : Mealy FSM

1. Buka software Xilinx ISE 14.5
2. Pilih menu **File** -> Kemudian pilih **New Project**
3. Lalu isikan nama project pada tampilan dengan nama *ohbaby*, kemudian untuk Top-level source type kita isi dengan pilihan **HDL**. Setelah itu klik Next.



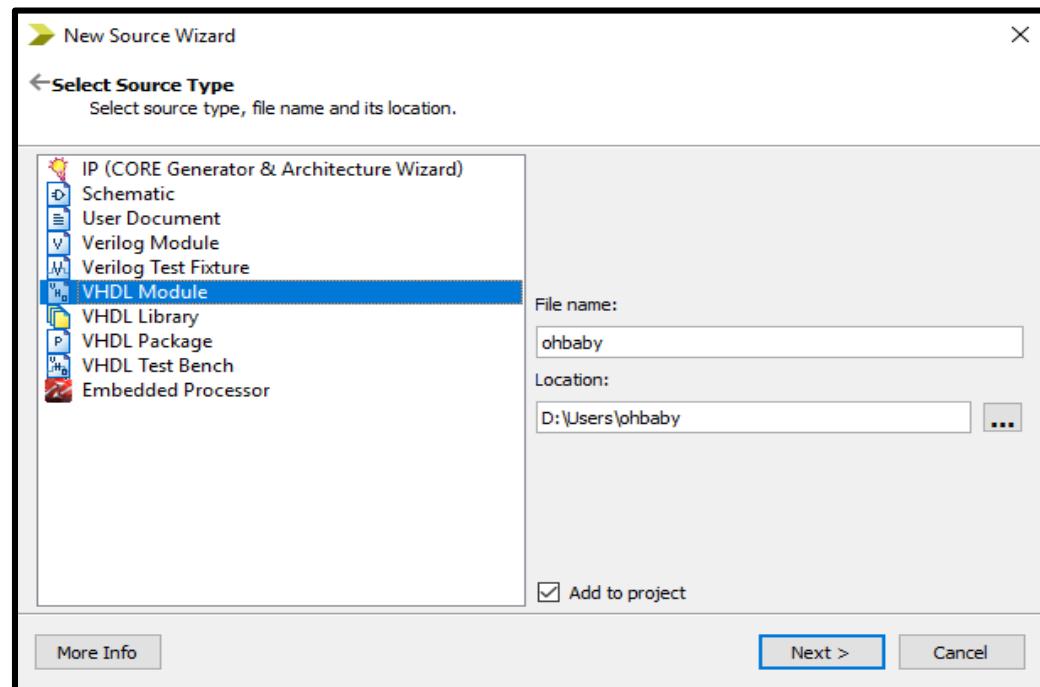
4. Atur Device Properties seperti gambar dibawah ini, kemudian pilih Next.



5. Klik Next, lalu klik Finish.



- Klik kanan pada hierarchy lalu pilih new source -> Pada tampilan sebelah kiri gambar pilih *VHDL Module*. Kemudian untuk file name, kita isi dengan nama yang sama pada saat kita membuat project baru yaitu **ohbaby**. Lalu klik Next.



- Pada define module, di next saja. Lalu klik finish.
- Lalu ketikkan program berikut pada source code di setiap bagian yang telah ditentukan

Bagian 1

```

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mealy_4s is

    port
    (
        clk      : in std_logic;
        data_in  : in std_logic;
        reset   : in std_logic;
        data_out : out std_logic_vector(1 downto 0)
    );

end entity;

```



Bagian 2

```
case state is
    when s0=>
        if data_in = '1' then
            data_out <= "00";
        else
            data_out <= "01";
        end if;
    when s1=>
        if data_in = '1' then
            data_out <= "01";
        else
            data_out <= "11";
        end if;
    when s2=>
        if data_in = '1' then
            data_out <= "10";
        else
            data_out <= "10";
        end if;
    when s3=>
        if data_in = '1' then
            data_out <= "11";
        else
            data_out <= "10";
        end if;
end case;
```

Kemudian CTRL + S , untuk save program.

9. Langkah berikutnya adalah kita pilih pilihan **Synthesize – XST**. Dengan cara klik dua kali. Untuk mengecek apakah program terdapat error.
10. Klik tanda + pada User Constraints pilih **I/O Pin Planning (PlanAhead) – Pre-Synthesis** dengan klik kanan pilih **Run** atau klik kiri 2x.

Nanti akan muncul software PlanAhead. Pada bagian I/O ports, silahkan kalian tentukan pin untuk LED dan SWITCH pada FPGA di bagian Site, jika sudah menentukan lalu enter dah pastikan pada bagian Fixed sudah tercentang.

Name	Direction	Neg Diff Pair	Site	Fixed
All ports (5)				
data_out (2)	Output			<input type="checkbox"/>
data_out[1]	Output			<input type="checkbox"/>
data_out[0]	Output			<input type="checkbox"/>
Scalar ports (3)				
clk	Input			<input type="checkbox"/>
data_in	Input			<input checked="" type="checkbox"/>
reset	Input			<input type="checkbox"/>



11. Selanjutnya klik tanda + pada project mains di Hierarchy -> double klik top.ucf -> pilih yes. Dan pastikan pin sudah benar sesuai dengan posisi pin LED dan SWITCH pada FPGA.
12. Lalu klik dua kali pada bagian **Implement Design**
13. Klik tanda + pada **Generate Programming File** -> pilih **Configure Device** (**iMPACT**) klik kanan lalu Run
14. Klik 2 kali **Configure Target Device** ->Pilih **Boundary Scan** ->Klik kanan pilih **Initialize Chain**
15. Tunggu beberapa detik lalu **Pilih (- - - nama module- - -)**.bit project yang di buat klik **Open** -> Bila ada tanda warning klik **no**
16. klik **OK** -> Identify Succeeded
17. Klik kanan pada gambar IC Xilinx sebelah kiri lalu pilih **program**
18. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**

SOURCE CODE:

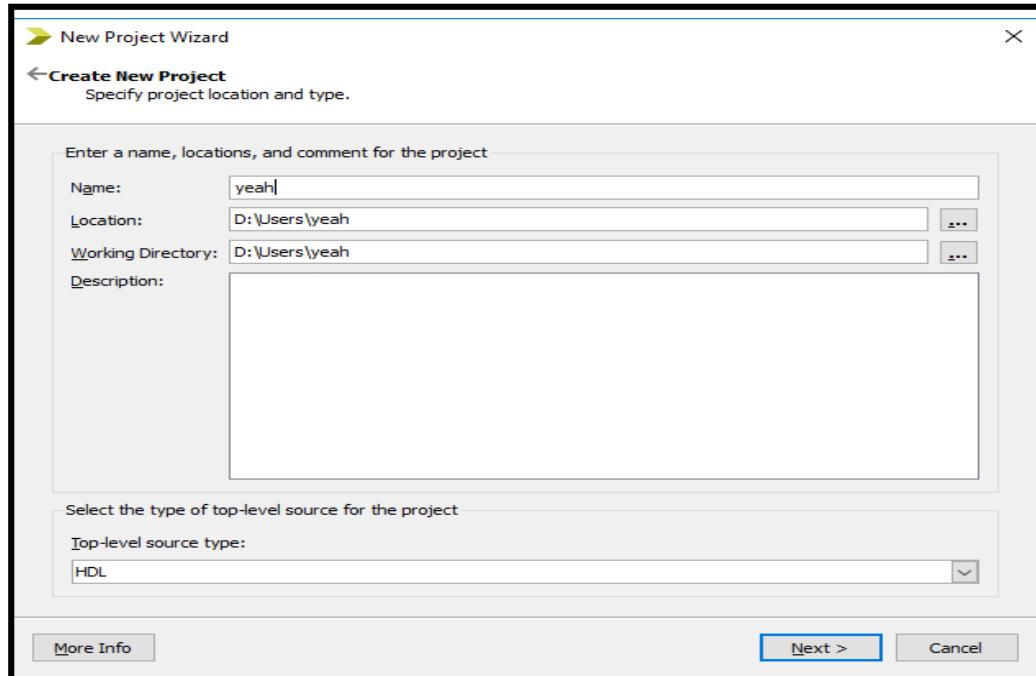


OUTPUT & KESIMPULAN :

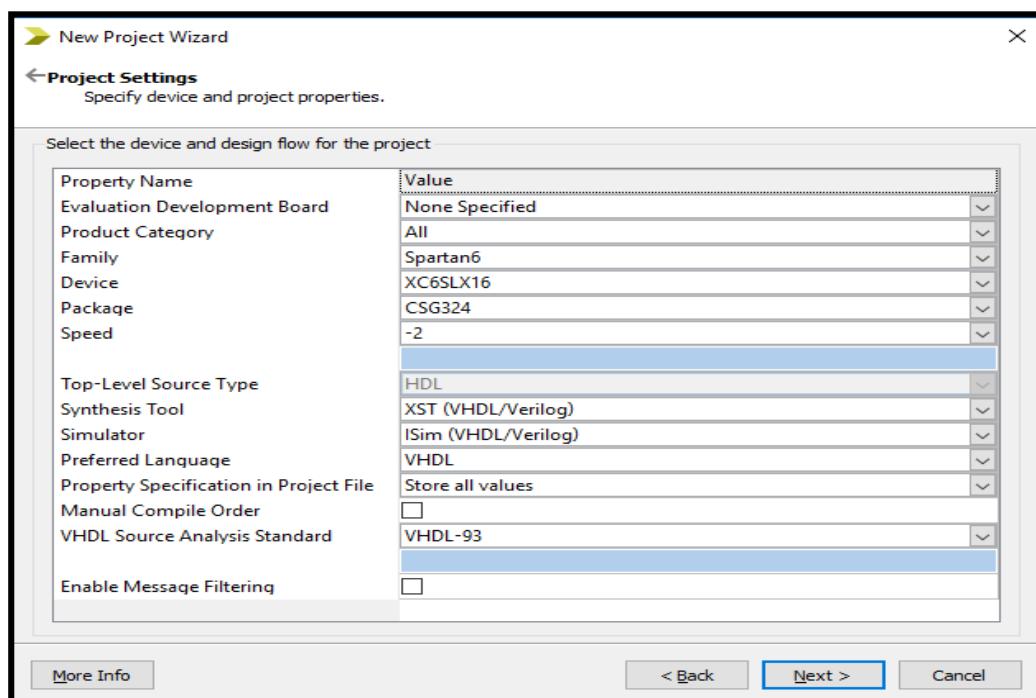


Percobaan 3: FSM 4 State dengan Gerbang Logika

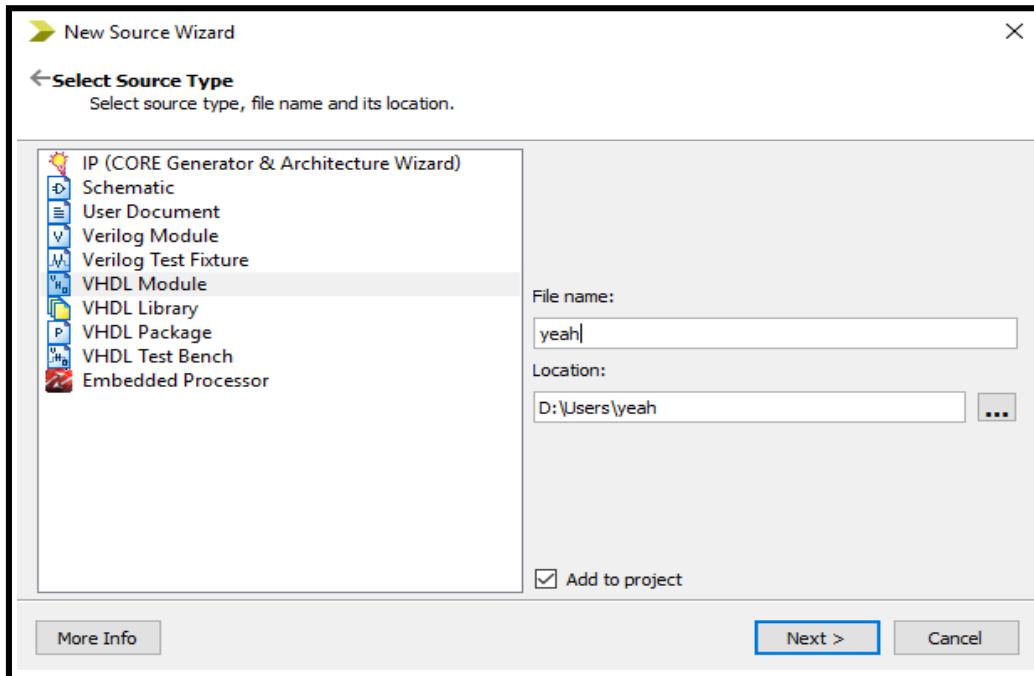
1. Buka software Xilinx ISE 14.5
2. Pilih menu **File** -> Kemudian pilih **New Project**
3. Lalu isikan nama project pada tampilan dengan nama *yeah*, kemudian untuk Top-level source type kita isi dengan pilihan **HDL**. Setelah itu klik **Next**.



4. Atur device properties seperti pada gambar di bawah ini. Kemudian klik **Next**.



5. Klik Next lalu klik Finish
6. Klik kanan pada hierarchy lalu pilih new source -> Pada tampilan sebelah kiri gambar pilih *VHDL Module*. Kemudian untuk file name, kita isi dengan nama yang sama pada saat kita membuat project baru yaitu **yeah**. Lalu klik Next.



7. Pada define module, di next saja. Lalu klik finish.
8. Lalu ketikkan program berikut pada source code di setiap bagian yang telah di tentukan

Bagian 1

```
port (clk : in std_logic;
      reset : in std_logic;
      input : in std_logic;
      Q : in std_logic;
      R : in std_logic;
      output : out std_logic
    );
  
```

Bagian 2

```
if (reset='1') then
  current_s <= s0;
elsif (rising_edge(clk)) then
  case current_s is
    when s0 =>
      if(input ='0') then
        output <= Q and R;
      else
        next_s <= s1;
      end if;
    end case;
  end if;
end process;
```

Bagian 3



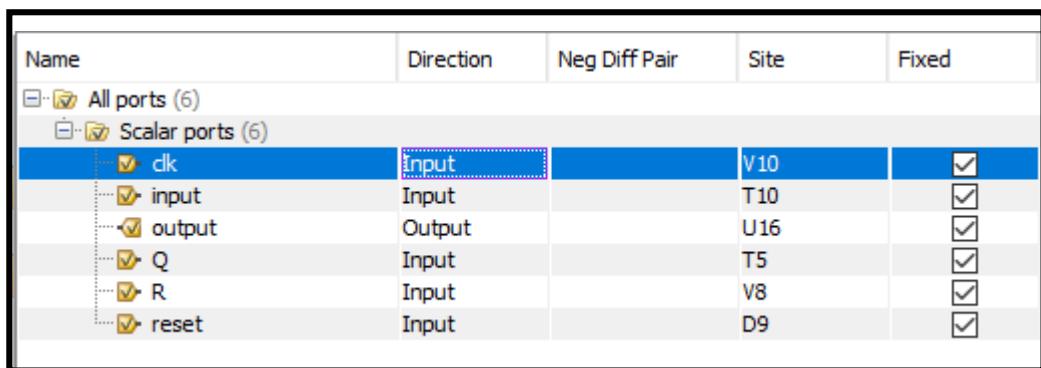
```

when s2 =>
if(input ='0') then
    output <= Q nand R;
else
    next_s <= s3;
end if;

```

Kemudian CTRL + S , untuk save program.

9. Langkah berikutnya adalah kita pilih pilihan **Synthesize – XST**. Dengan cara klik dua kali. Untuk mengecek apakah program terdapat error.
10. Klik tanda + pada User Constraints pilih **I/O Pin Planning (PlanAhead) – Pre-Synthesis** dengan klik kanan pilih **Run** atau klik kiri 2x.
11. Nanti akan muncul software PlanAhead. Pada bagian I/O ports, silahkan kalian tentukan pin untuk LED dan SWITCH pada FPGA di bagian Site, jika sudah menentukan lalu enter dah pastikan pada bagian Fixed sudah tercentang.



Name	Direction	Neg Diff Pair	Site	Fixed
All ports (6)				
Scalar ports (6)				
clk	Input		V10	<input checked="" type="checkbox"/>
input	Input		T10	<input checked="" type="checkbox"/>
output	Output		U16	<input checked="" type="checkbox"/>
Q	Input		T5	<input checked="" type="checkbox"/>
R	Input		V8	<input checked="" type="checkbox"/>
reset	Input		D9	<input checked="" type="checkbox"/>

12. Selanjutnya klik tanda + pada project mains di Hierarchy -> double klik top.ucf -> pilih yes. Dan pastikan pin sudah benar sesuai dengan posisi pin LED dan SWITCH pada FPGA.
13. Lalu klik dua kali pada bagian **Implement Design**
14. Klik tanda + pada **Generate Programming File** -> pilih **Configure Device (iMPACT)** klik kanan lalu Run
15. Klik 2 kali **Configure Target Device** ->Pilih **Boundary Scan** ->Klik kanan pilih **Initialize Chain**
16. Tunggu beberapa detik lalu **Pilih (- - - nama module- - -).bit** project yang di buat klik **Open** -> Bila ada tanda warning klik **no**
17. klik **OK** -> Identify Succeeded



18. Klik kanan pada gambar IC Xilinx sebelah kiri lalu pilih **program**

19. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**

SOURCE CODE:

OUTPUT & KESIMPULAN :



Percobaan Mandiri 1 :

Dari source code program diatas, buatlah sebuah program menggunakan 5 state dengan ketentuan berikut :

- State 1, terdapat gerbang NAND
- State 2, terdapat gerbang XOR
- State 3, terdapat gerbang NOR
- State 4, terdapat gerbang XNOR
- State 5, terdapat gerbang AND

SOURCE CODE:



OUTPUT & KESIMPULAN:



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal dan Memahami penggunaan seven segment pada FPGA
- Praktikan Dapat Merancang Program Desain menggunakan seven segment pada Pemrograman FPGA
- Praktikan Dapat Memahami Penggunaan seven segment dalam Pemrograman Desain VHDL

II. Dasar Teori

- Pengenalan Program Desain seven segment
- Merancang Desain Program menggunakan seven segment
- Aplikasi Perancangan Program Desain seven segment

III. Peralatan

- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC



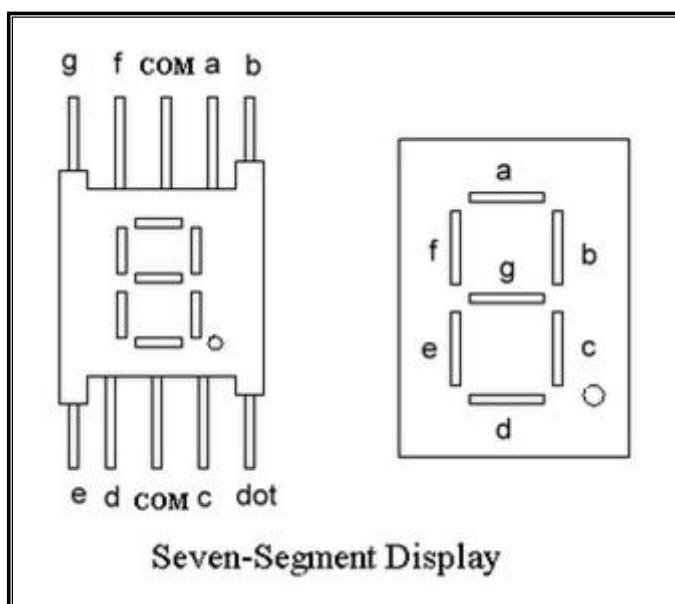
4

BAB 4 SEVEN SEGMENT

4.1 Pengertian Seven Segment

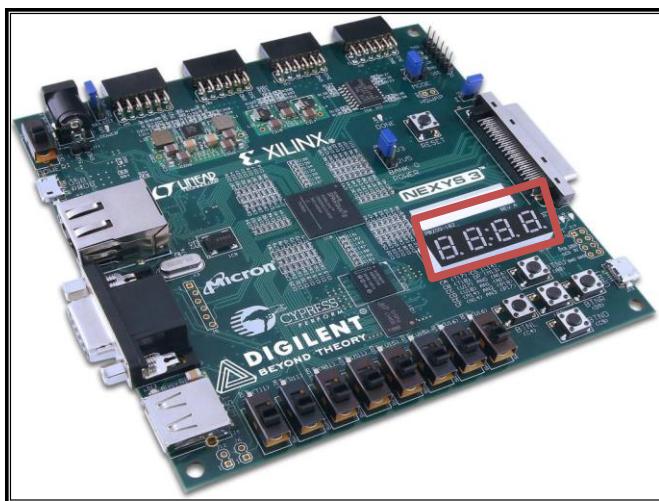
Seven Segment adalah salah satu perangkat layar untuk menampilkan sistem angka desimal yang merupakan alternatif dari layar dot-matrix. Layar tujuh segmen ini seringkali digunakan pada jam digital, meteran elektronik, dan perangkat elektronik lainnya yang menampilkan informasi numerik. Ide mengenai layar tujuh segmen ini sudah cukup tua. Pada tahun 1910 misalnya, sudah ada layar tujuh segmen yang diterangi oleh lampu pijar yang digunakan pada panel sinyal kamar ketel suatu pembangkit listrik.

Tujuh bagian dari layar dapat dinyalakan dalam bermacam-macam kombinasi untuk menampilkan angka Arab. Sering ketujuh segmen tersebut disusun dengan kemiringan tertentu, untuk memudahkan pembacaan. Pada sebagian besar penerapannya, ketujuh segmen ini memiliki bentuk dan ukuran yang hampir seragam (biasanya segienam panjang, walaupun trapesium dan persegi panjang juga dapat digunakan).



Gambar 4.1 Seven Segment Display





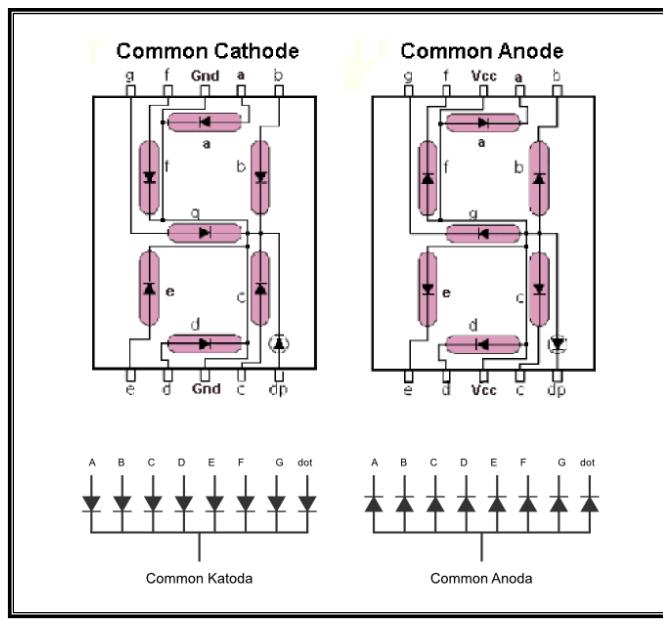
Gambar 4.2 Seven Segment pada FPGA Board Xilinx Spartan 6

Seven Segment memiliki 7 Segmen dimana setiap segmen dikendalikan secara ON dan OFF untuk menampilkan angka yang diinginkan. Angka-angka dari 0 (nol) sampai 9 (Sembilan) dapat ditampilkan dengan menggunakan beberapa kombinasi Segmen. Selain 0 – 9, Seven Segment juga dapat menampilkan Huruf Hexadecimal dari A sampai F. Segmen atau elemen-elemen pada Seven Segment diatur menjadi bentuk angka “8” yang agak miring ke kanan dengan tujuan untuk mempermudah pembacaannya. Pada beberapa jenis Seven Segment, terdapat juga penambahan “titik” yang menunjukkan angka koma decimal. Terdapat beberapa jenis Seven Segment, diantaranya adalah Incandescent bulbs, Fluorescent lamps (FL), Liquid Crystal Display (LCD) dan Light Emitting Diode (LED).

4.2 LED Seven Segment

LED Seven Segment umumnya memiliki 7 segmen atau elemen garis dan 1 segmen titik yang menandakan ‘koma’ decimal atau *dot product*. Jadi jumlah keseluruhan segmen atau elemen LED sebenarnya adalah 8. Terdapat 2 jenis LED Seven Segment, diantaranya adalah LED Seven Segment Common Cathoda dan LED Seven Segment Common Anoda, selengkapnya dijelaskan pada bab cara kerja SevSegment.





4.3 Cara Kerja Seven Segment

- ❖ Common Cathoda

Kaki Katoda pada semua segmen LED adalah terhubung menjadi 1 pin, sedangkan Kaki Anoda akan menjadi input untuk masing – masing Segmen LED. Kaki Katoda yang terhubung menjadi 1 pin ini merupakan terminal negative (-) atau ground, sedangkan sinyal kendali akan diberikan kepada masing – masing Kaki Anoda Segmen LED.

Cara kerja Seven Segment Common Cathoda akan aktif pada kondisi high ‘1’.

ANGKA h g f e d c b a HEXA	
0	0 0 1 1 1 1 1 1 3FH
1	0 0 0 0 0 1 1 0 06H
2	0 1 0 1 1 0 1 1 5BH
3	0 1 0 0 1 1 1 1 4FH
4	0 1 1 0 0 1 1 0 66H
5	0 1 1 0 1 1 0 1 6DH
6	0 1 1 1 1 1 0 1 7DH
7	0 0 0 0 0 1 1 1 07H
8	0 1 1 1 1 1 1 1 7FH
9	0 1 1 0 1 1 1 1 1 6FH

Gambar 4.4 Common Catoda/Aktif High



❖ Common Anoda

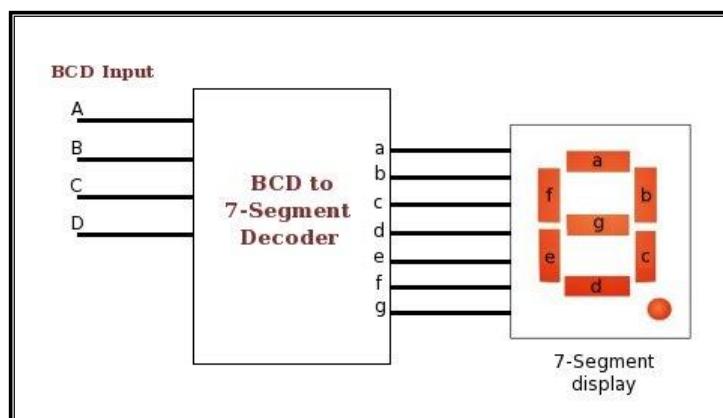
Kaki Anoda pada semua segmen LED adalah terhubung menjadi 1 pin, sedangkan Kaki Katoda akan menjadi input untuk masing – masing Segmen LED. Kaki Anoda yang terhubung menjadi 1 pin ini akan diberi tegangan positif (+) dan sinyal kendali akan diberikan pada masing – masing Kaki Katoda Segmen LED.

Cara kerja Seven Segment Common Anoda akan aktif pada kondisi low ‘0’

ANGKA h g f e d c b a HEXA	
0	1 1 0 0 0 0 0 0 C0H
1	1 1 1 1 1 0 0 1 F9H
2	1 0 1 0 0 1 0 0 A4H
3	1 0 1 1 0 0 0 B0H
4	1 0 0 1 1 0 0 1 99H
5	1 0 0 1 0 0 1 0 EDH
6	1 0 0 0 0 0 1 0 12H
7	1 1 1 1 1 0 0 0 F8H
8	1 0 0 0 0 0 0 0 10H
9	1 0 0 1 0 0 0 0 90H

Gambar 4.5 Common Anoda/Aktif Low

4.3 Prinsip Kerja Dasar Driver System pada LED Seven Segment



Gambar 4.3 Skematik Seven Segment

Blok Dekoder pada skematik diatas mengubah sinyal input yang diberikan menjadi 8 jalur yaitu a – g dan *dot product* (koma) untuk meng – ON kan segmen sehingga menghasilkan



angka atau digit yang diinginkan. Contohnya, jika output decoder adalah a, b, dan c, maka segmen LED akan menghasilkan angka ‘7’. Jika sinyal input adalah berbentuk analog, maka diperlukan ADC (Analog to Digital Converter) untuk mengubah sinyal analog menjadi sinyal digital sebelum masuk ke input decoder. Jika sinyal input sudah merupakan sinyal digital, maka decoder akan menanganinya sendiri tanpa harus menggunakan ADC.

Fungsi daripada Blok Driver adalah untuk memberikan arus listrik yang cukup kepada segmen/element LED untuk menyala. Pada tipe decoder tertentu, decoder sendiri dapat mengeluarkan tegangan dan arus listrik yang cukup untuk menyalakan segmen LED maka blok driver ini tidak diperlukan. Pada umumnya driver untuk menyalakan 7 segmen ini adalah terdiri dari 8 transistor switch pada masing – masing elemen LED.

4.4 Metode Scanning

Scanning digunakan untuk menghemat penggunaan pin I/O mikrokontroler. Pada prinsipnya *scanning* adalah teknik yang bergantian. Misalnya, kita ingin menampilkan 8 angka pada 7 segment, maka nyala lampu pada 7 segment dibuat bergantian sehingga jalur data dapat digunakan secara bersamaan oleh kedelapan 7 segment tersebut. Proses bergantian ini dilakukan sangat cepat sehingga mata manusia seolah-olah akan melihat kedelapan 7 segment menyala.

Secara program *scanning* dapat dilakukan dengan cara *polling* maupun *timer*. Cara *polling* dilakukan dengan menjalankan program *scanning* secara terus menerus dimana program *scanning* biasanya diletakkan di program utama. Jika pada program utama terdapat terdapat program lain yang membutuhkan waktu pengerjaan yang lama, maka program untuk *scanning* akan teganggu. Sedangkan, cara *timer* lebih efektif dan efisien sebab mikrokontroler dapat mengerjakan program lain tanpa mengganggu *scanning*.

4.4.1 Metode Scanning pada Seven Segment

Pada teknik scanning 7 segment, masing-masing pin a, b, c, d, e, f, g, dan dot pada semua 7 segment digabungkan menjadi satu kemudian dihubungkan ke port mikrokontroler. Masing-masing pin common 7 segment dihubungkan ke sebuah driver yang dikontrol oleh sebuah pin I/O mikrokontroler. Driver digunakan untuk memilih 7 segment mana yang akan dihidupkan. Jadi pin I/O mikrokontroler yang dibutuhkan sebanyak 8 pin untuk data (segment a, b, c, d, e, f, g, dan dot) dan ditambah jumlah common pada 7 segment yang akan digunakan.



Untuk menyalakan segment (a, b, c, d, e, f, g, dan dot) pada 7 segment dengan cara memberi tegangan dimana terdapat selisih tegangan yang positif antara kaki anode (A) dengan kaki cathode (K). Untuk menyalakan 7 segment common anode, yaitu dengan cara memberi tegangan yang mana terdapat selisih tegangan yang positif antara kaki common dengan kaki a, b, c, d, e, f, g, dan dot. Jika 7 segment dihubungkan ke mikrokontroler dibutuhkan 8 pin I/O. Semua pin a, b, c, d, e, f, g, dan dot langsung dihubungkan ke prt mikrokontroler melalui resistor pembatas arus, dengan common 7 segment anode dihubungkan ke VCC. Seandainya dibutuhkan 7 segment sebanyak 8 buah untuk penampil, maka port mikrokontroler tidak akan mencukupinya, untuk itu digunakan cara scanning. Teknik scanning digunakan apabila terdapat minimal 2 buah 7 segment.

4.4.2 Contoh Metode Scanning

Pada bagian ini akan dijelaskan bagaimana contoh metode scanning. Bagaimana cara kerja seven segment jika di-interface kan dengan mikrokontroler. Misalnya, jika ingin menampilkan angka 1234 pada seven segment, maka dibutuhkan (digit-1 = 1, digit-2 = 2, digit-3 = 3, digit-4 = 4) dan pada prinsipnya cara kerja seven segment dilakukan dengan metode scanning, yaitu “jika ingin menampilkan 1234, maka ditampilkan angka 1 terlebih dahulu pada sevsegment digit-1 dengan mematikan digit-2, digit-3, dan digit-4. Lalu jika ingin menampilkan angka 2 (digit -2) seven segment, maka mematikan digit-1, digit-3, dan digit-4, dan seterusnya.

Metode inilah yang dinamakan scanning, dengan melakukan scanning secara cepat (biasanya 25x dalam 1 detik), maka mata kita tidak akan bisa mengikuti scanning seven segment tersebut sehingga seolah-olah mata kita akan melihat bahwa semua seven segmen (digit-1 s/d digit-4) menyalा secara bersamaan.

Pertanyaannya, mengapa digunakan metode scanning? Jawabannya dikarenakan metode scanning dapat mengurangi konsumsi daya listrik dibandingkan dengan menyalakan semua seven segment. Dan yang paling penting yaitu dapat menghemat pemakaian pin-pin mikrokontroler, sehingga tidak banyak input/output yang terpakai untuk mengakses seven segment.

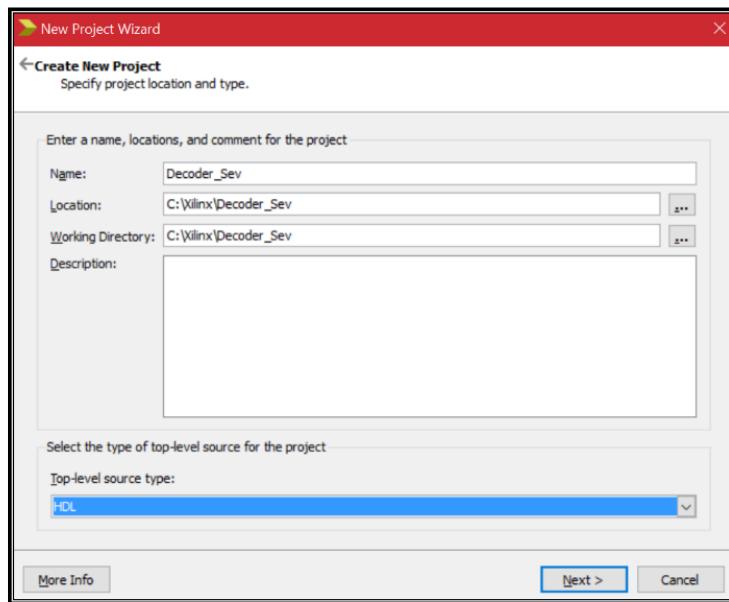
Percobaan 1 : Seven Segment

Langkah – langkah :

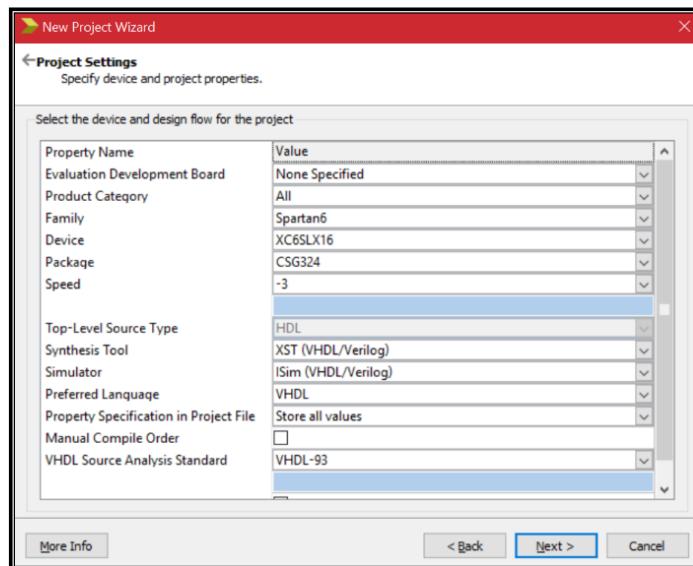
1. Buka software Xilinx ISE 14.5



2. Pilih menu **File → New Project**
3. Lalu isikan nama project pada tampilan dengan nama bebas, misalnya : **Decoder_Sev**
Kemudian untuk *Top – level Source Type* isi dengan pilihan **HDL**. Setelah itu klik **Next**.

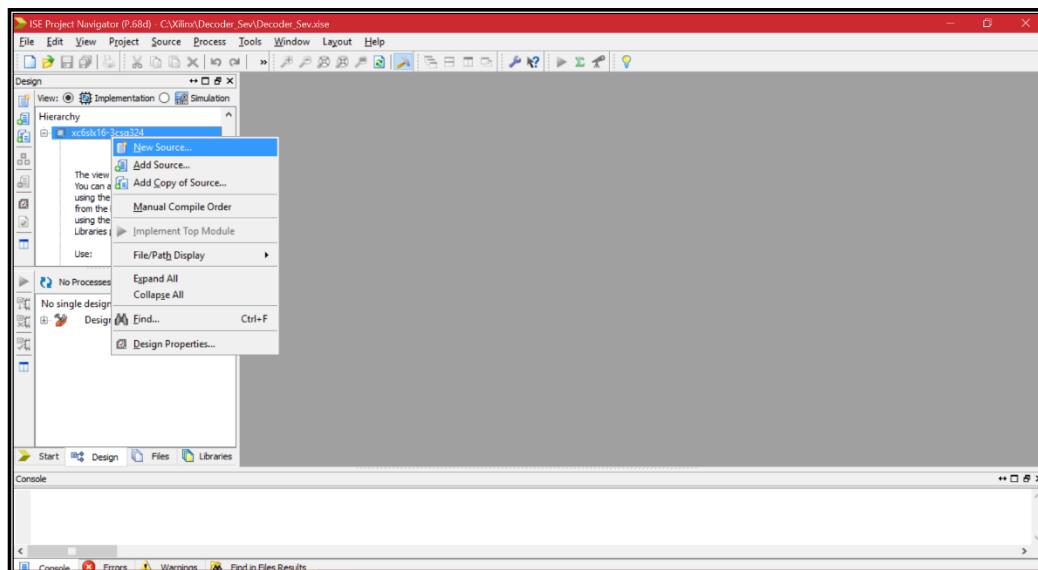


4. Atur *Device Properties* seperti gambar dibawah ini. Kemudian klik **Next → Next → Finish**

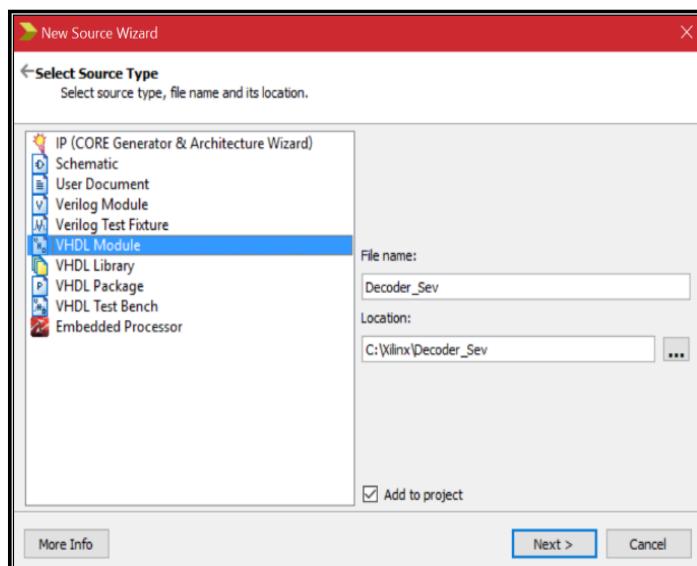


5. Lalu pada *Hierarchy*, klik kanan lalu **New Source**



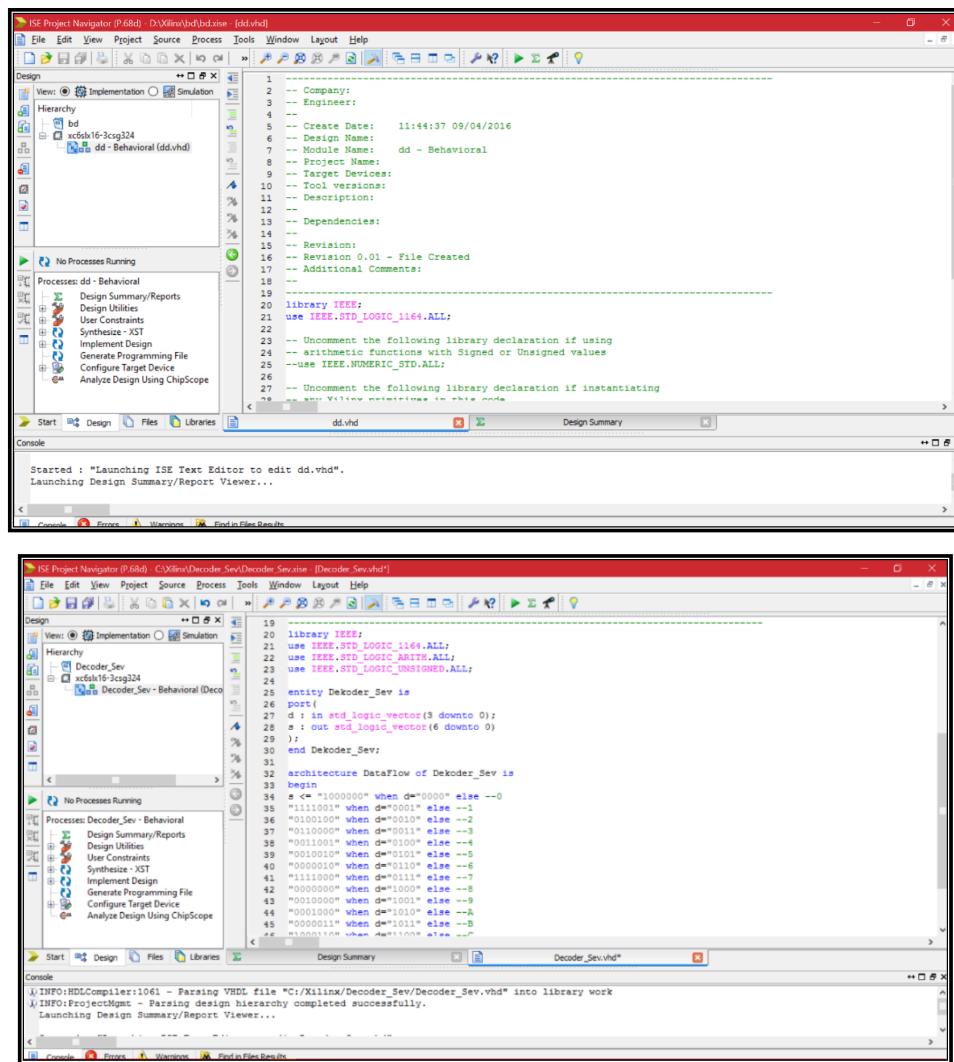


- Setelah itu, muncul tampilan seperti gambar dibawah. Tampilan tersebut untuk menentukan tipe asal (*source type*), nama file (*file name*), dan lokasi – nya (*its location*). *Source type* pilih **VHDL Module**, lalu *File name* isikan sesuai keinginan atau sesuai gambar, misalnya : **Decoder_Sev**, sedangkan *location* sudah **default** seperti pada gambar.



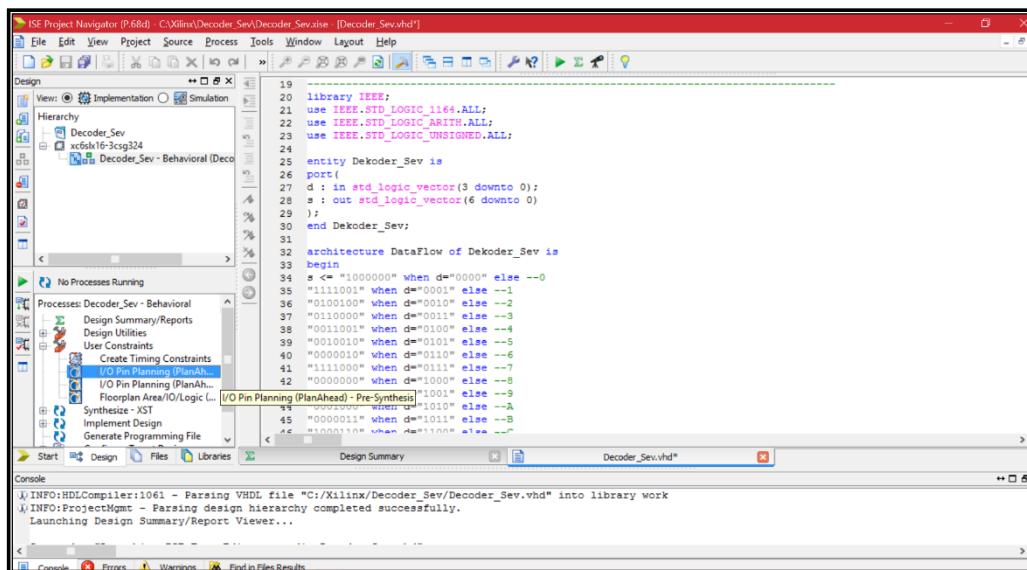
- Kemudian klik **Next → Next → Finish**
- Lengkapi *Source Code* yang diberikan asisten dengan *Source Code* dibawah ini :





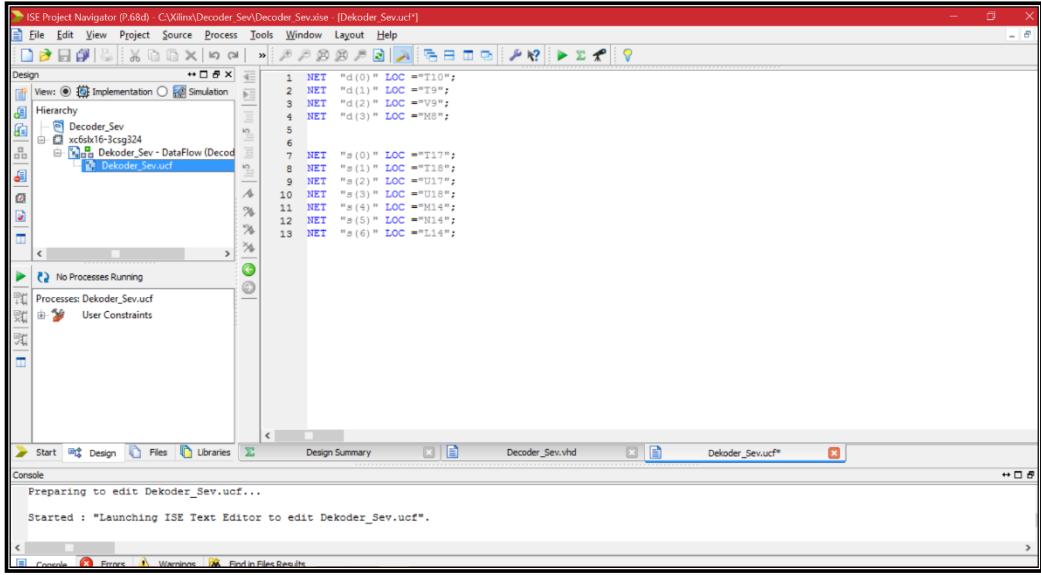
9. Kemudian Save.

10. Langkah berikutnya, klik *expand* (+) pada **User Constraints**, pilih **I/O Pin Planning (PlanAhead) – Pre – Synthesis**.

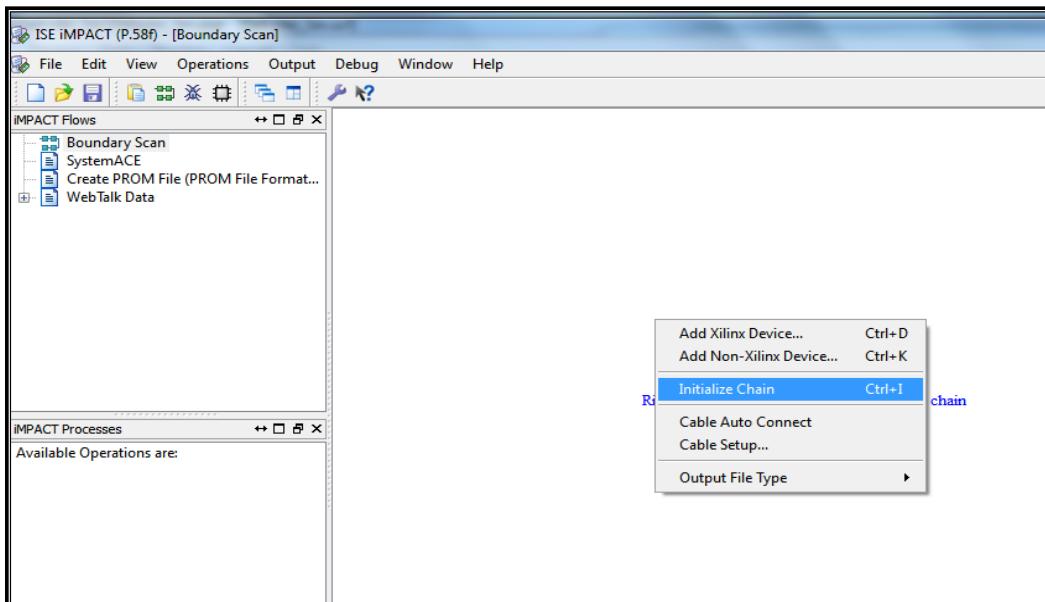


11. Lalu akan muncul *Text Editor* dengan format *UCF* dan lengkapi Code yang diberikan

Asisten.



12. Berikutnya, kembali ke tab **Decoder_Sev.vhd**. Lalu pilih **Synthesize – XST**
13. Klik **Implement Design**
14. Lalu, *double klik* pada **Generate Programming File**
15. Klik *expand* (+) pada **Configure Target Device** → **Generate Target PROM/ACE File**
→ klik kanan **Run**
16. Pada jendela ISE iMPACT, *double klik* pada **Boundary Scan**, lalu klik kanan pada *field Boundary Scan*, kemudian pilih **Initialize Chain**

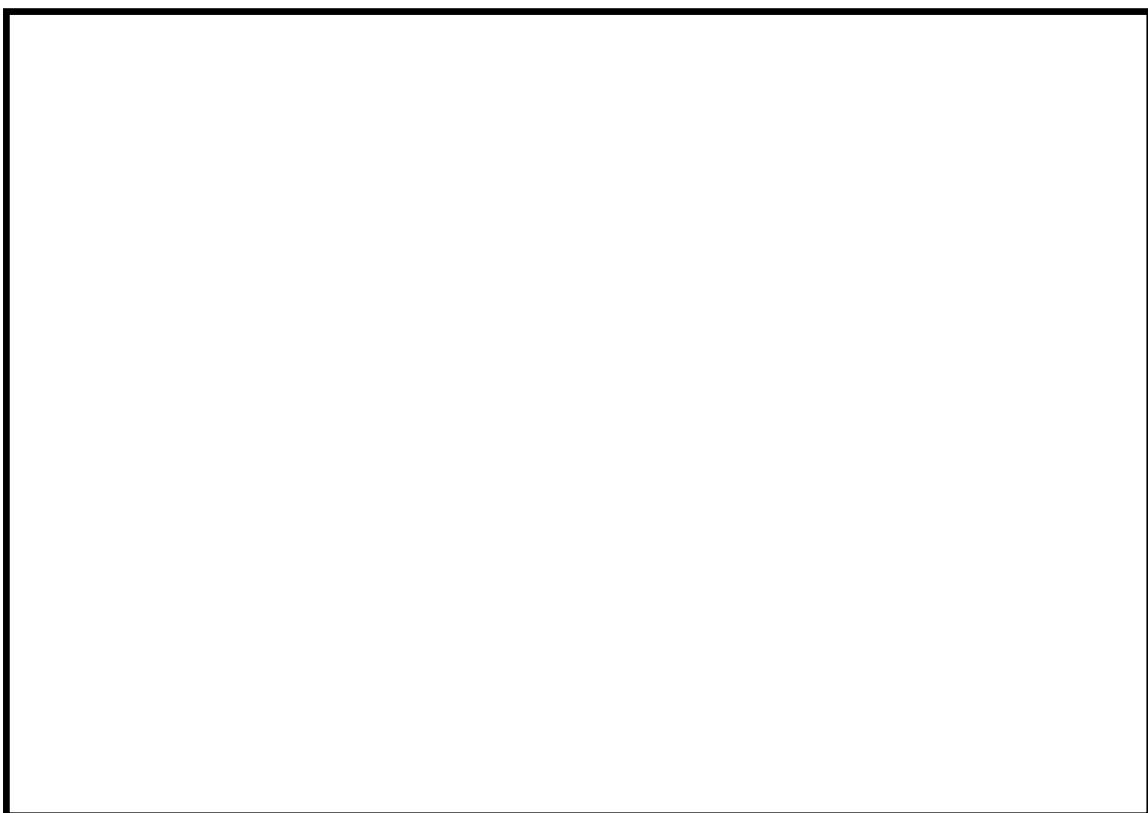


17. Pilih **(-- nama module--).bit** → **Open** → bila ada *warning* klik **Ok**
18. Pada tab **Xilinx Web Talk Dialog**, klik **No** → **Ok**
19. Klik kanan pada gambar **IC Xilinx (warna hijau)** lalu pilih **Program...** seperti pada gambar



20. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**

SOURCE CODE:



OUTPUT & KESIMPULAN:



Dengan menggunakan *source code* program 1 diatas, buatlah project baru dimana project tersebut menghasilkan output huruf G, H, P, L

SOURCE CODE:



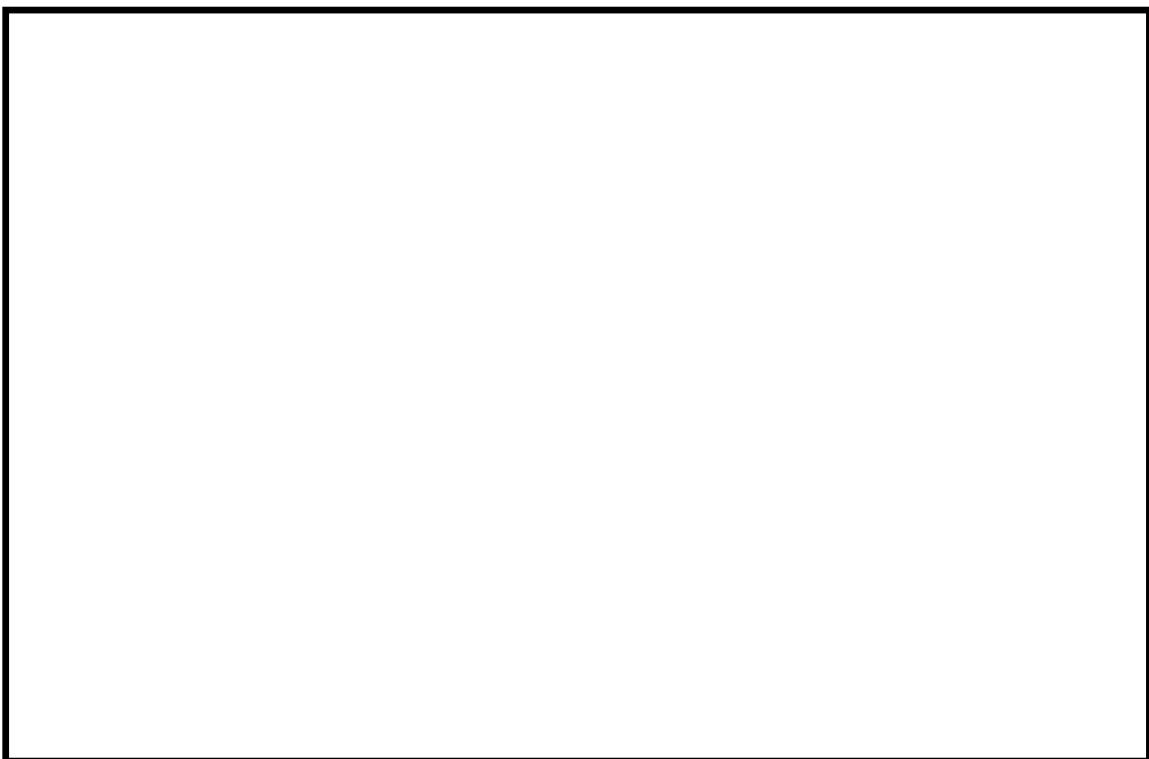
OUTPUT & KESIMPULAN:



Percobaan Mandiri 1B :

Bagaimana jika output huruf FPGA menggunakan *Seven Segment Common Catoda* (Aktif High)? Tulis program Seven Segment – nya !

SOURCE CODE:



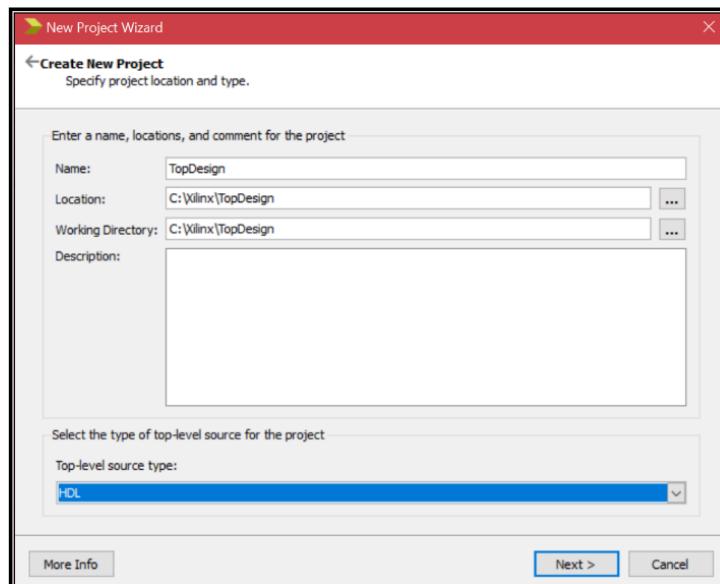
KESIMPULAN:



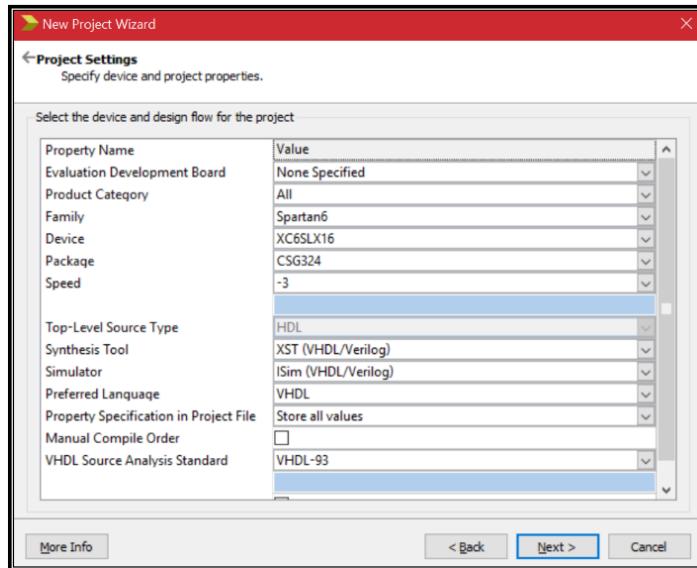
Percobaan 2 : Tampilan Pada 2 Seven Segment

Langkah – langkah :

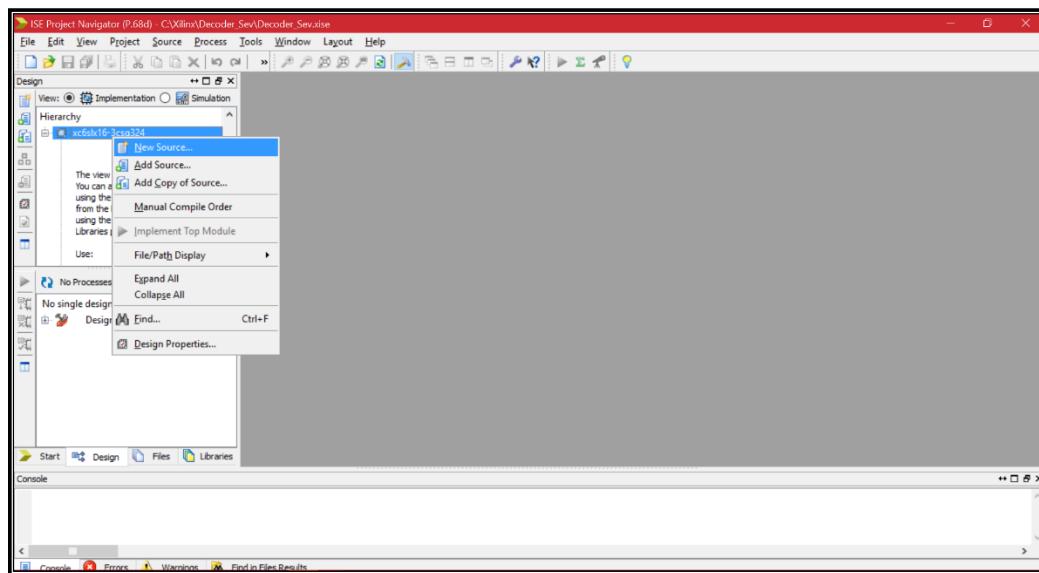
1. Buka software Xilinx ISE 14.5
2. Pilih menu **File → New Project**
3. Lalu isikan nama project pada tampilan dengan nama bebas, misalnya : **TopDesign**. Kemudian untuk *Top – level Source Type* isi dengan pilihan **HDL**. Setelah itu klik **Next**.



4. Atur *Device Properties* seperti gambar dibawah ini. Kemudian klik **Next → Next → Finish**

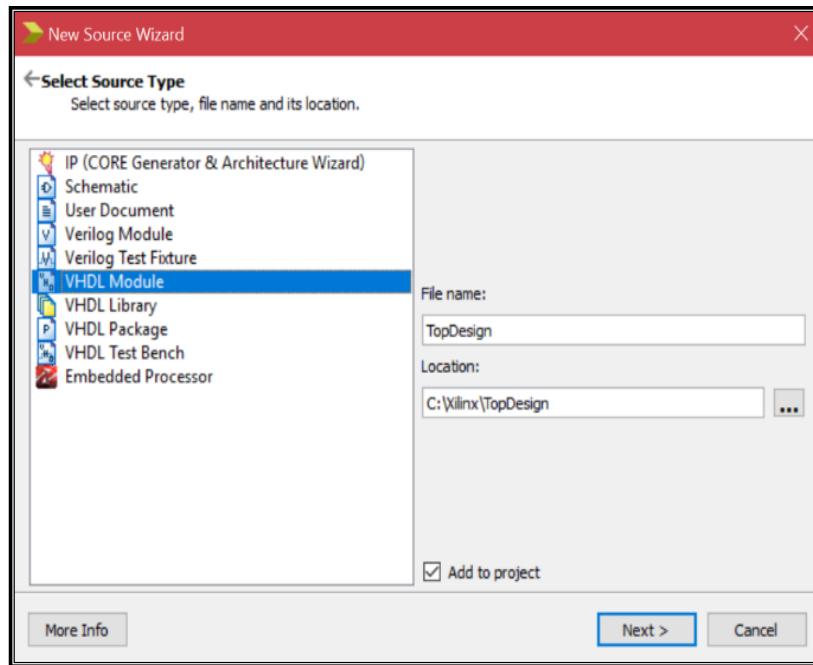


5. Lalu pada *Hierarchy*, klik kanan lalu **New Source**

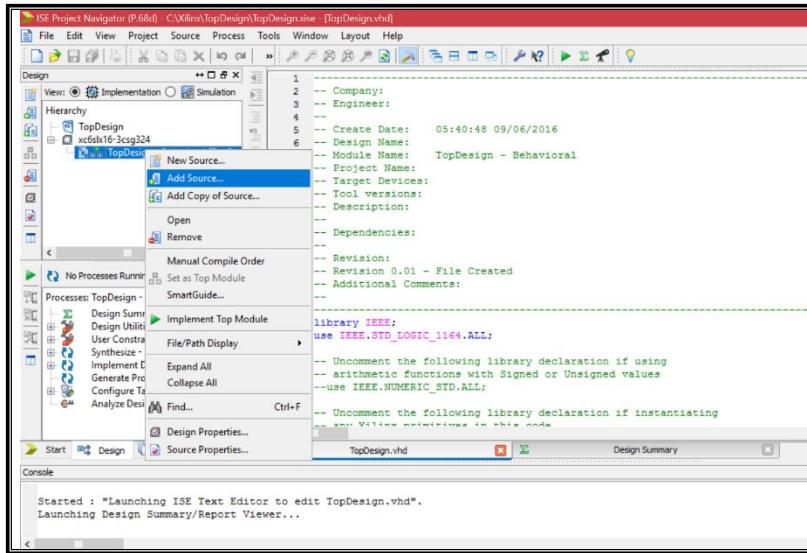


6. Setelah itu, muncul tampilan seperti gambar dibawah. Tampilan tersebut untuk menentukan tipe asal (*source type*), nama file (*file name*), dan lokasi – nya (*its location*). *Source type* pilih **VHDL Module**, lalu *File name* isikan sesuai keinginan atau sesuai gambar, misalnya : **TopDesign**, sedangkan *location* sudah **default** seperti pada gambar.



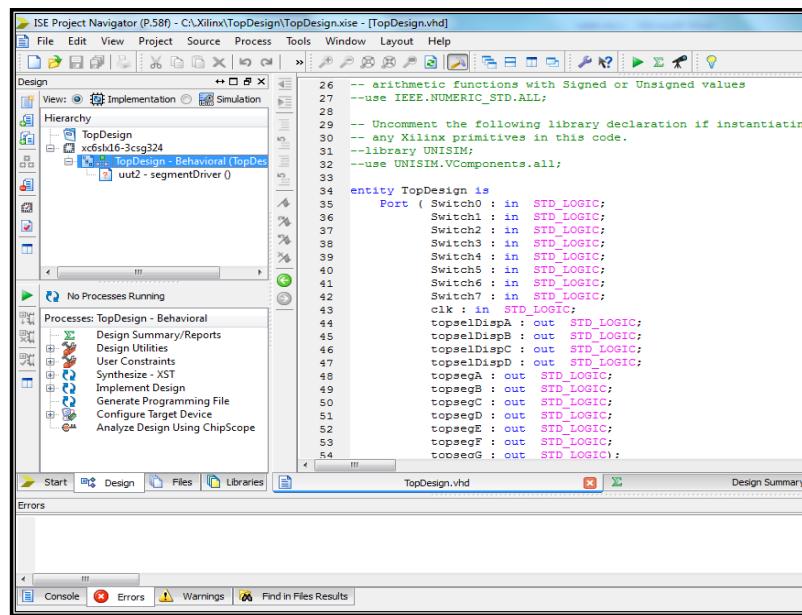


7. Kemudian klik **Next** → **Next** → **Finish**
8. Lalu pada *Hierarchy*, klik kanan lalu **Add Source**. Kemudian cari folder *source code* – nya dan tambahkan *source code* yang diberikan asisten :



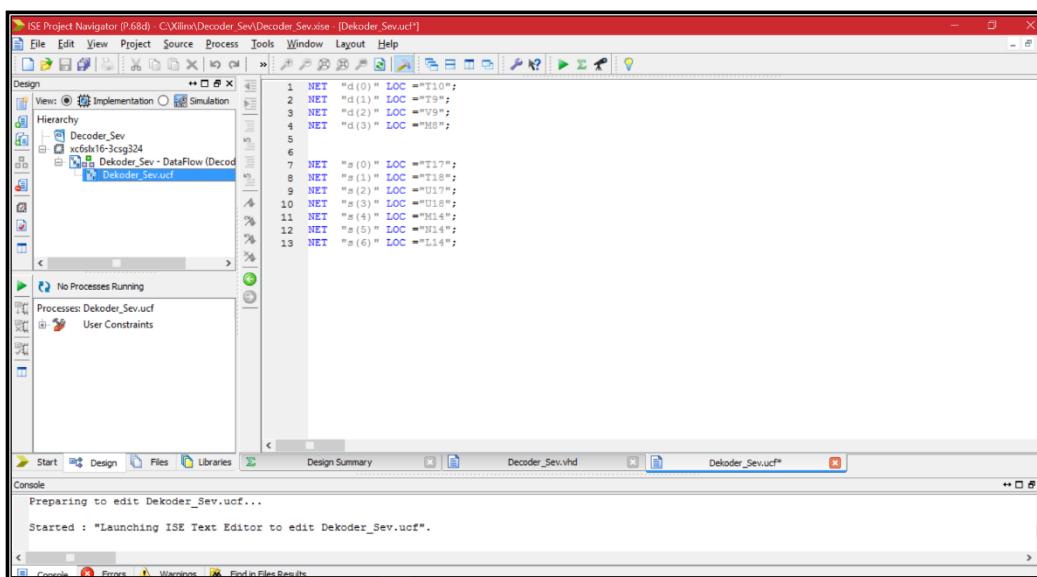
9. Terlihat pada gambar terdapat contoh *source code* “uut2 – segmentDriver()”. Ini adalah *source code* yang ditambahkan tadi. Lalukan hal yang sama pada *source code* “uut1 – clockdivider” dan “uutsegmentdecoder”





10. Kemudian Save

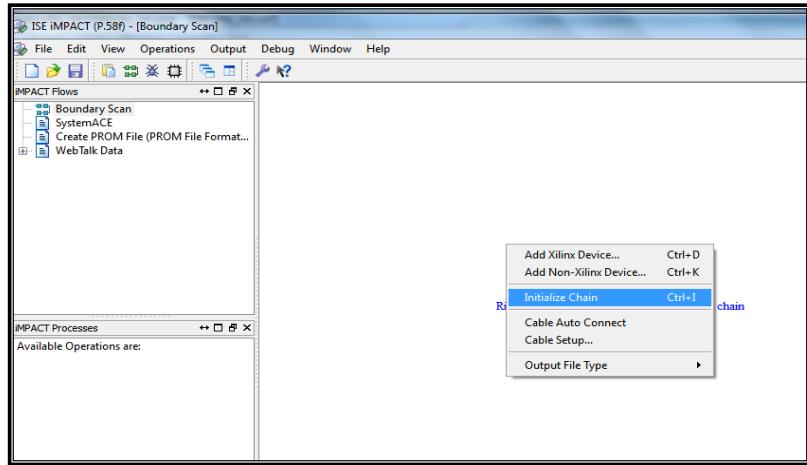
11. Langkah berikutnya, klik *expand (+)* pada **User Constraints**, pilih **I/O Pin Planning (PlanAhead) – Pre – Synthesis.**
12. Lalu akan muncul *Text Editor* dengan format **UCF** dan lengkapi Code yang diberikan Asisten.



13. Berikutnya, kembali ke tab **TopDesign.vhd**. Lalu pilih **Synthesize – XST**
14. Klik **Implement Design**
15. Lalu, *double click* pada **Generate Programming File**



16. Klik *expand* (+) pada **Configure Target Device** → **Generate Target PROM/ACE File**
 → klik kanan **Run**
17. Pada jendela ISE iMPACT, *double click* pada **Boundary Scan**, lalu klik kanan pada *field* **Boundary Scan**, kemudian pilih **Initialize Chain**



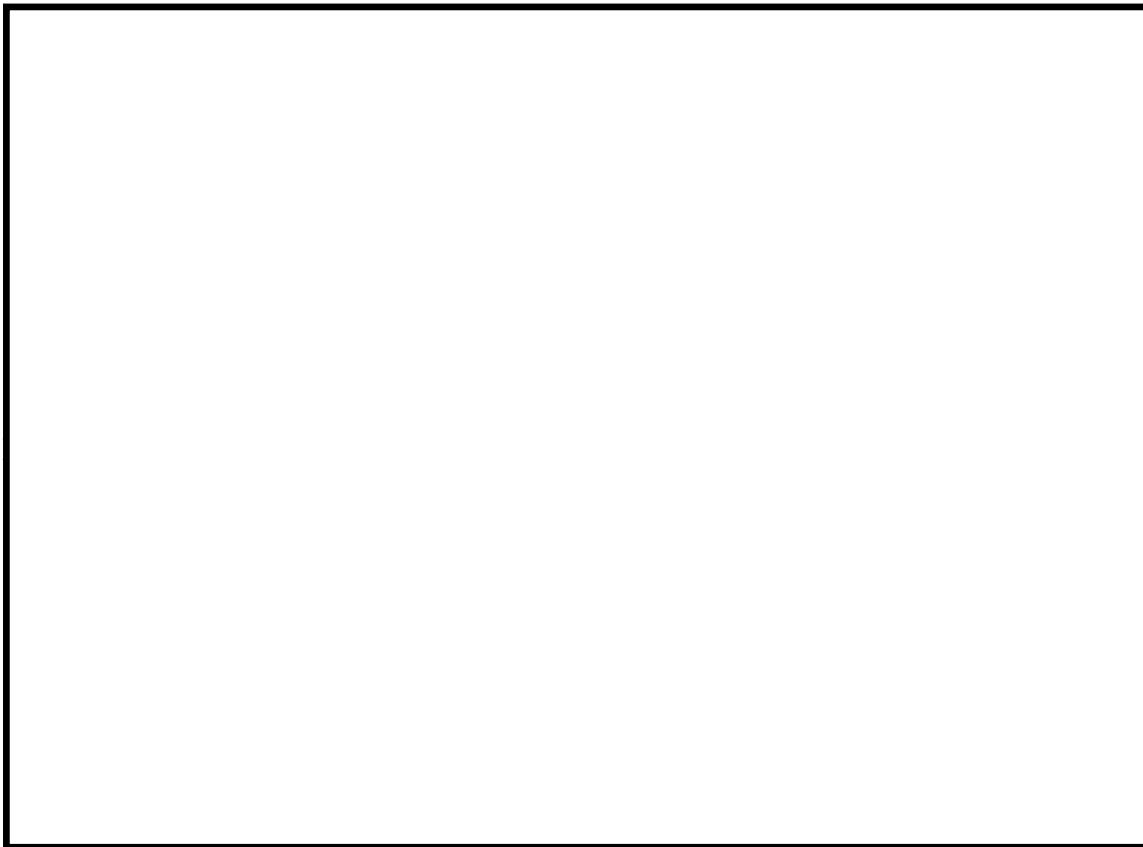
18. Pilih **(-- nama module--).bit** → **Open** → bila ada *warning* klik **Ok**
19. Pada tab *Xilinx Web Talk Dialog*, klik **No** → **Ok**
20. Klik kanan pada gambar **IC Xilinx (warna hijau)** lalu pilih **Program...** seperti pada gambar



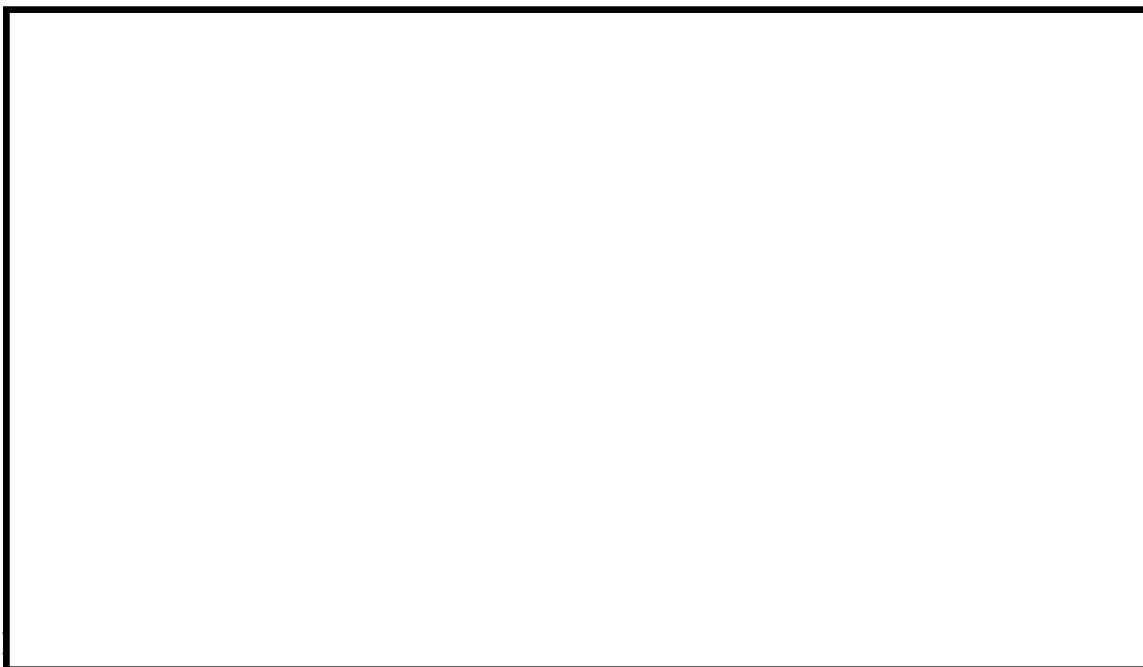
21. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**



SOURCE CODE:



OUTPUT & KESIMPULAN:



Dari *source code* pada percobaan diatas, buatlah program *seven segment* lainnya. Tampilan awal dari *seven segmen* FF00 ubah menjadi E0E0 dan *seven segment* yang bekerja seven segmen ke 2 dan 4.

SOURCE CODE:

OUTPUT & KESIMPULAN:



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal dan Memahami penggunaan Push Button pada FPGA
- Praktikan Dapat Merancang Program Desain menggunakan Push Button pada Pemrograman FPGA
- Praktikan Dapat Memahami Penggunaan Push Button dalam Pemrograman Desain VHDL

II. Dasar Teori

- Pengenalan Program Desain Push Button
- Merancang Desain Program menggunakan Push Button
- Aplikasi Perancangan Program Desain Push Button

III. Peralatan

- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC

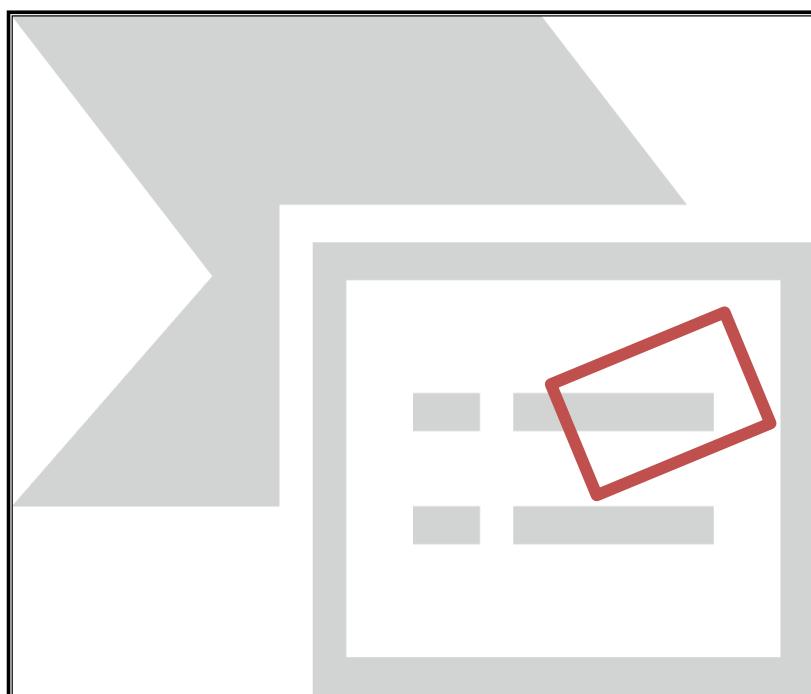


5. 1 Mengenal Push Button

Pada intinya tombol merupakan sebuah saklar pushbutton. Push Button merupakan sebuah device untuk menghubungkan dan memutuskan rangkaian listrik antara 2 titik. Penggunaan pushbutton dalam kehidupan sehari – hari hamper menyentuh semua bidang. Di bidang komputer dengan keyboard dan mouse, dibidang otomotif dengan panel – panel kontrolnya, bahkan peralatan rumah tangga sekalipun seperti kontrol peralatan listrik juga menggunakan push button. (S. Naskan, Yogyakarta)

5.2 Pengertian Push Button

Push Button adalah saklar yang beroperasi dengan cara ditekan, dan bisa melakukan dua fungsi yang berbeda, yakni menutup sirkuit bila ditekan, atau justru membuka sirkuit bila ditekan. Jika tekanan dilepaskan atau terjadi tekanan berikutnya maka akan menormalkan kembali tombol ke posisi semula dan sirkuit kembali ke status semula. (eprints.ung.ac.id)



Gambar 3.1 FPGA NEXYS 3 (PUSH BUTTON)



5.3 Fungsi dan Penggunaan Push Button

Alat ini befungsi sebagai pemberi sinyal masukan pada rangkaian listrik, ketika / selama bagian knopnya ditekan maka alat ini akan bekerja sehingga kontak-kontaknya akan terhubung untuk jenis normally open dan akan terlepas untuk jenis normally close, dan sebaliknya ketika knopnya dilepas kembali maka kebalikan dari sebelumnya, untuk membuktikannya pada terminalnya bisa digunakan alat ukur tester / ohm meter, pada umumnya pemakaian terminal jenis NO digunakan untuk menghidupkan rangkaian dan terminal jenis NC digunakan untuk mematikan rangkaian, namun semuanya tergantung dari kebutuhanmesin.

Seperti telah kita ketahui, alat ini sangat banyak digunakan, dalam sebuah operation panel bisa terdapat beberapa Push Button tergantung dari keperluan, alat ini juga memiliki kode warna pada bagian knopnya untuk membedakan fungsi dari masing-masing alat,seperti warna merah digunakan untuk tombol berhenti/stop, lalu warna hitam digunakan untuk tombol jalan/start kemudian warna kuning digunakan untuk tombol reset atau alarm stop, ada beberapa contoh penggunaan Push Button seperti untuk menjalankan motor/pompa, menjalankan conveyor, menghidupkan lampu, mereset alarm, menyalakan bell, menghidupkan cylinder dan masih banyak lagi.

5.4 Edge Detection

Penggunaan push button umumnya untuk melakukan sebuah aksi setelah ditekan beberapa kali. Untuk ini kita perlu menghitung berapa kali terjadinya sebuah perubahan logika dari HIGH ke LOW atau sebaliknya. Ini yang disebut sebagai Edge Detection.

Dalam push button berjenis NO, pada saat kondisi kebuka, pin terhubung ke ground (melalui resistor pull-down) dan kita membaca LOW. Sebaliknya jika ditutup (ditekan), kedua kakinya akan terhubung dan mengalirkan arus, sehingga kita membaca HIGH. IC yang kita gunakan harus mampu membaca sinyal input dari push button secara terus menerus agar dapat dibandingkan dengan kondisi inputan sebelumnya melalui sebuah perulangan. Jika kondisi nilainya tidak sama dengan sebelumnya, maka telah terjadi sebuah Edge.

Adapun Jenis-jenisnya:

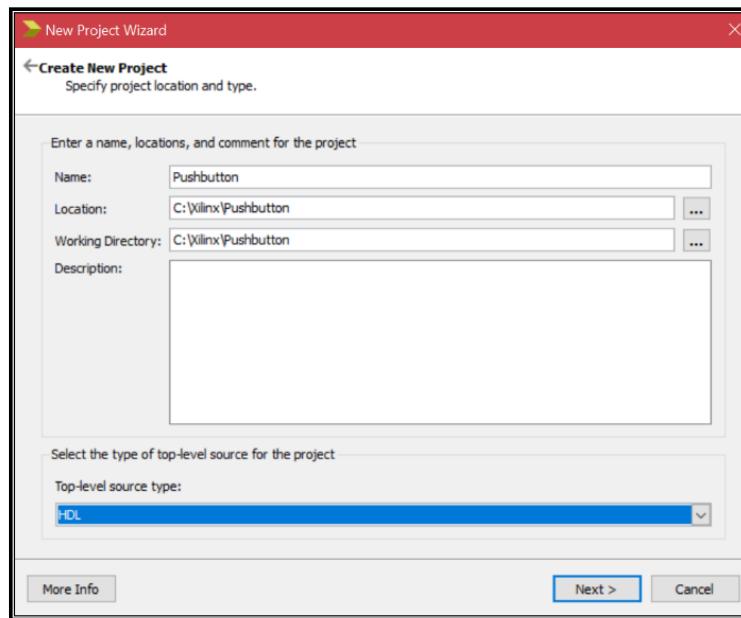
- Falling – Ketika terjadi perubahan input dari logika HIGH/1 menjadi LOW/0.
- Rising - Ketika terjadi perubahan input logika pada dari LOW/0 menjadi HIGH/1.



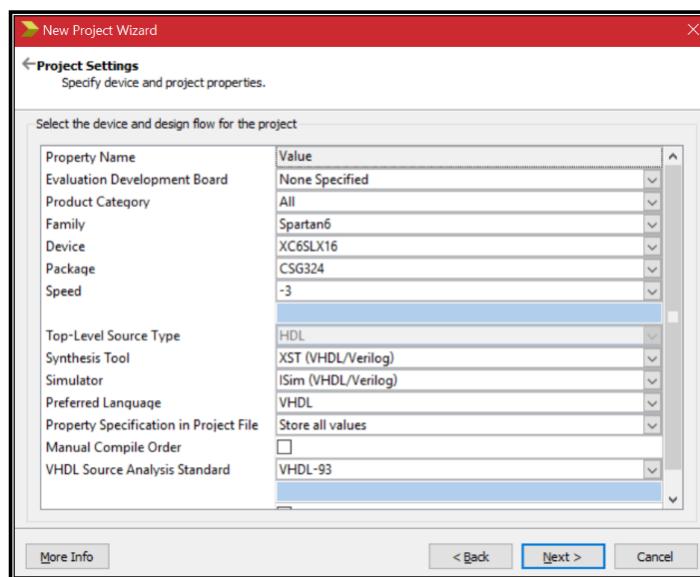
Percobaan 1 : Push Button With LED

Langkah – langkah :

21. Buka software Xilinx ISE 14.5
22. Pilih menu **File → New Project**
23. Lalu isikan nama project pada tampilan dengan nama bebas, misalnya : **Pushbutton**
Kemudian untuk *Top – level Source Type* isi dengan pilihan **HDL**. Setelah itu klik **Next**.

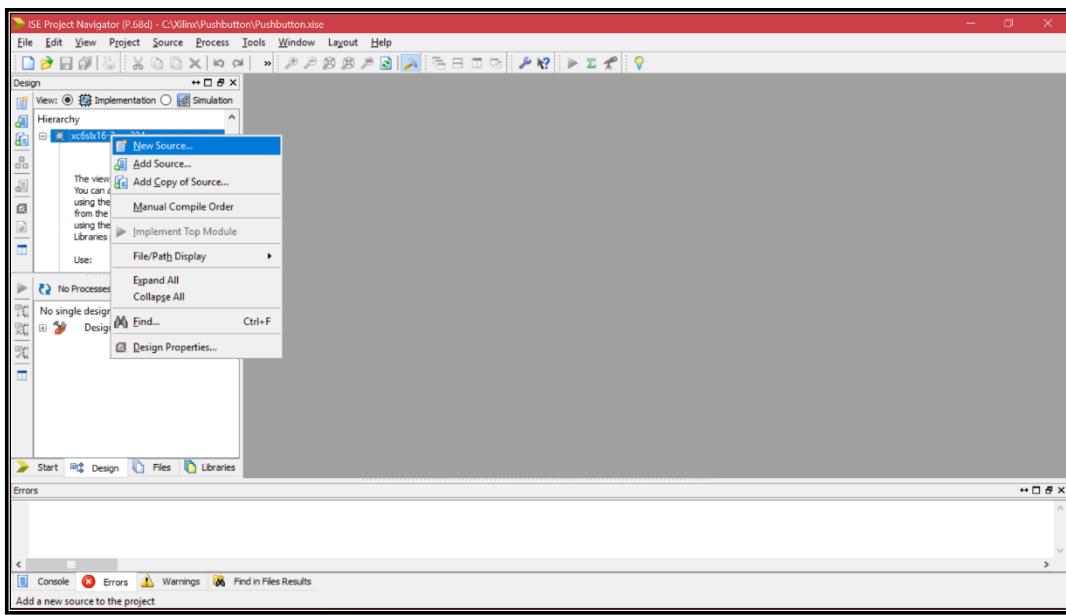


24. Atur *Device Properties* seperti gambar dibawah ini. Kemudian klik **Next → Next → Finish**

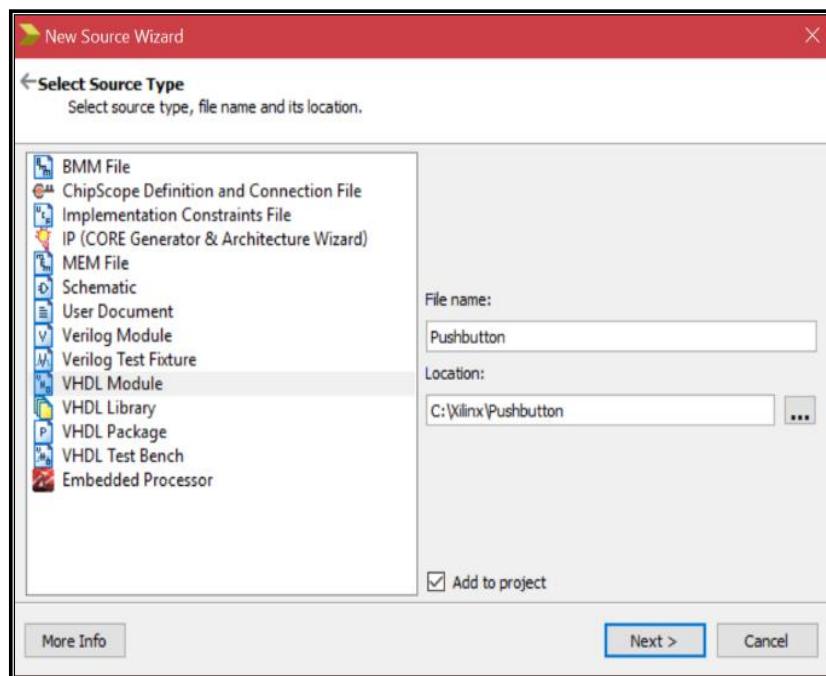


25. Lalu pada *Hierarchy*, klik kanan lalu **New Source**





26. Setelah itu, muncul tampilan seperti gambar dibawah. Tampilan tersebut untuk menentukan tipe asal (*source type*), nama file (*file name*), dan lokasi – nya (*its location*). *Source type* pilih **VHDL Module**, lalu *File name* isikan sesuai keinginan atau sesuai gambar, misalnya : **Pushbutton**, sedangkan *location* sudah **default** seperti pada gambar.



27. Kemudian klik **Next → Next → Finish**
 28. Lengkapi *Source Code* yang diberikan asisten dengan *Source Code* dibawah ini :



```

19 library IEEE;
20 use IEEE.STD_LOGIC_1164.ALL;
21 use IEEE.STD_LOGIC_UNSIGNED.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23
24
25 entity counter4bit is
26 port (clk : in std_logic;
27        reset : in std_logic;
28        pause : in std_logic;
29        count_up : in std_logic;
30        count_down : in std_logic;
31        count_out : out std_logic_vector(3 downto 0));
32
33 end counter4bit;
34
35 architecture Behavioral of counter4bit is
36 signal temp_count : std_logic_vector(3 downto 0) := x"0";
37 signal slow_clk : std_logic;
38 -- Clock divider can be changed to suit application.
39 -- Clock (clk) is normally 50 MHz, so each clock cycle
40 -- is 20 ns. A clock divider of 'n' bits will make 1
41 -- slow_clk cycle equal  $2^n$  clk cycles.
42 signal clk_divider : std_logic_vector(23 downto 0) := x"000000";
43
44 begin
45     clk_division : process (clk, clk_divider)
46     begin

```

```

44 begin
45     clk_division : process (clk, clk_divider)
46     begin
47         if (clk = '1' and clk'event) then
48             clk_divider <= clk_divider + 1;
49         end if;
50         slow_clk <= clk_divider(23);
51     end process;
52     counting : process(reset, pause, slow_clk, temp_count, count_up, count_down)
53 begin
54 if reset = '1' then
55     temp_count <= "0000";      -- Asynchronous reset.
56 elsif pause = '1' then
57     temp_count <= temp_count; -- Asynchronous count pause.
58 else
59     if slow_clk'event and slow_clk='1' then -- Counting state
60         if (count_up = '1' and count_down = '0') then
61             if temp_count < 15 then
62                 temp_count <= temp_count + 1; -- Counter increase
63             else
64                 temp_count <= "0000";      -- Rollover to zero
65             end if;
66         elsif (count_up = '0' and count_down = '1') then
67             if temp_count > 0 then
68                 temp_count <= temp_count - 1; -- Counter increase
69             else
70                 temp_count <= "1111";      -- Rollover to zero
71             end if;

```



```

62           temp_count <= temp_count + 1; -- Counter increase
63     else
64       temp_count <= "0000";      -- Rollover to zero
65     end if;
66   elsif (count_up = '0' and count_down = '1') then
67     if temp_count > 0 then
68       temp_count <= temp_count - 1; -- Counter increase
69     else
70       temp_count <= "1111";      -- Rollover to zero
71     end if;
72   end if;
73   end if;
74   count_out <= temp_count;      -- Output
75 end process;
76
77 end Behavioral;

```

29. Kemudian **Save**.

30. Langkah berikutnya, klik *expand* (+) pada **User Constraints**, pilih **I/O Pin Planning (PlanAhead) – Pre – Synthesis.**

31. Lalu akan muncul *Text Editor* dengan format *UCF* dan lengkapi Code yang diberikan Asisten.

```

3  NET "pause" LOC = "C4";
4  NET "count_up" LOC = "A8";
5  NET "count_down" LOC ="C9";
6  NET "count_out(0)" LOC = "U16";
7  NET "count_out(1)" LOC = "V16";
8  NET "count_out(2)" LOC = "U15";
9  NET "count_out(3)" LOC = "V15";

```

32. Berikutnya, kembali ke tab **Pushbutton.vhd**. Lalu pilih **Synthesize – XST**

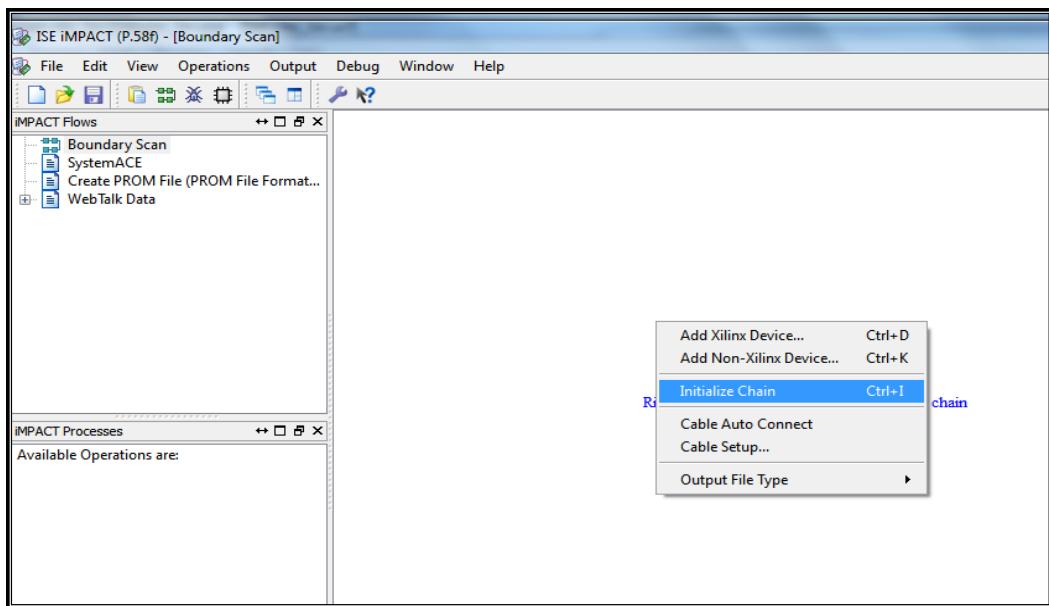
33. Klik **Implement Design**

34. Lalu, *double klik* pada **Generate Programming File**

35. Klik *expand* (+) pada **Configue Target Device** → **Generate Target PROM/ACE File**
→ klik kanan **Run**

36. Pada jendela ISE iMPACT, *double klik* pada **Boundary Scan**, lalu klik kanan pada *field Boundary Scan*, kemudian pilih **Initialize Chain**





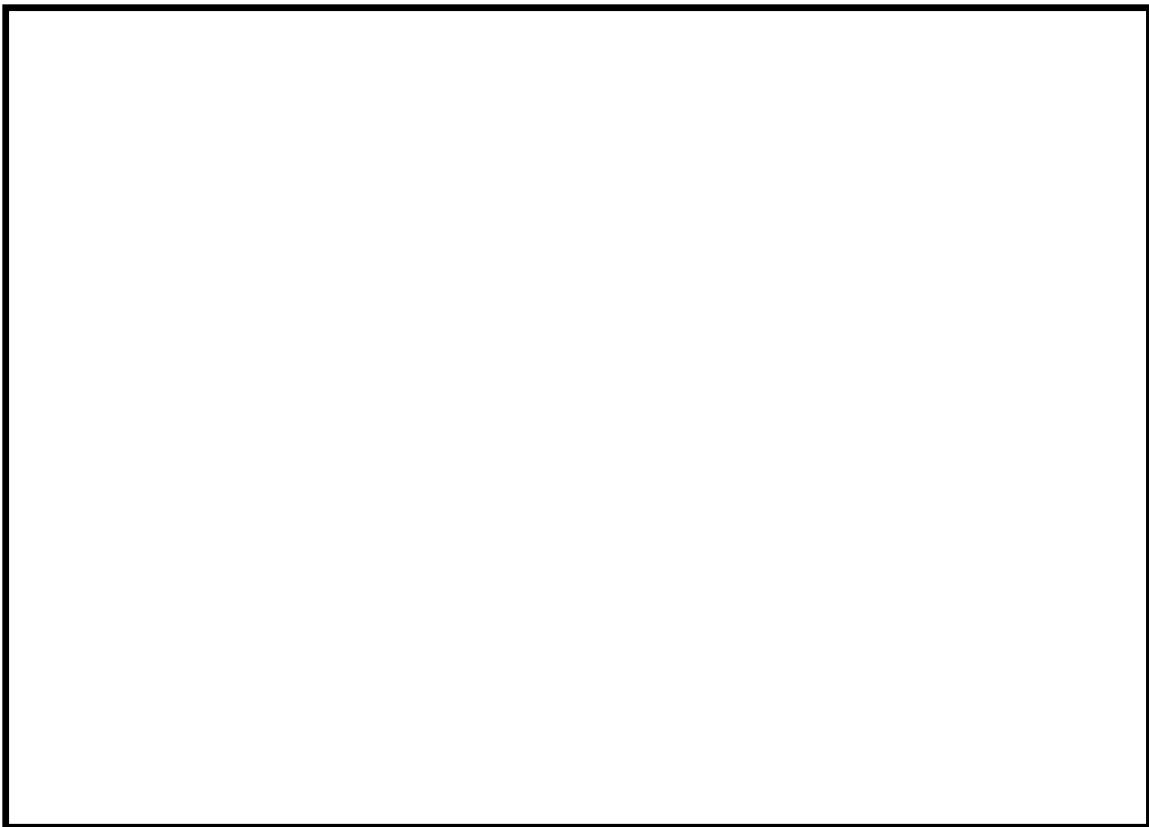
37. Pilih (**--- nama module---.bit**) → **Open** → bila ada *warning* klik **Ok**
38. Pada tab *Xilinx Web Talk Dialog*, klik **No** → **Ok**
39. Klik kanan pada gambar **IC Xilinx (warna hijau)** lalu pilih **Program...** seperti pada gambar



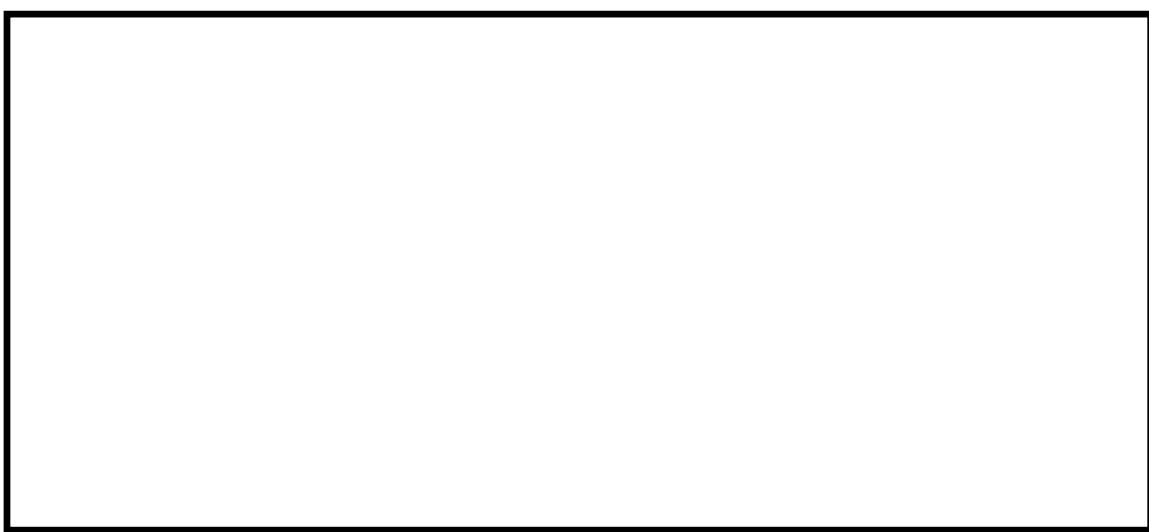
40. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**



SOURCE CODE :



OUTPUT & KESIMPULAN :



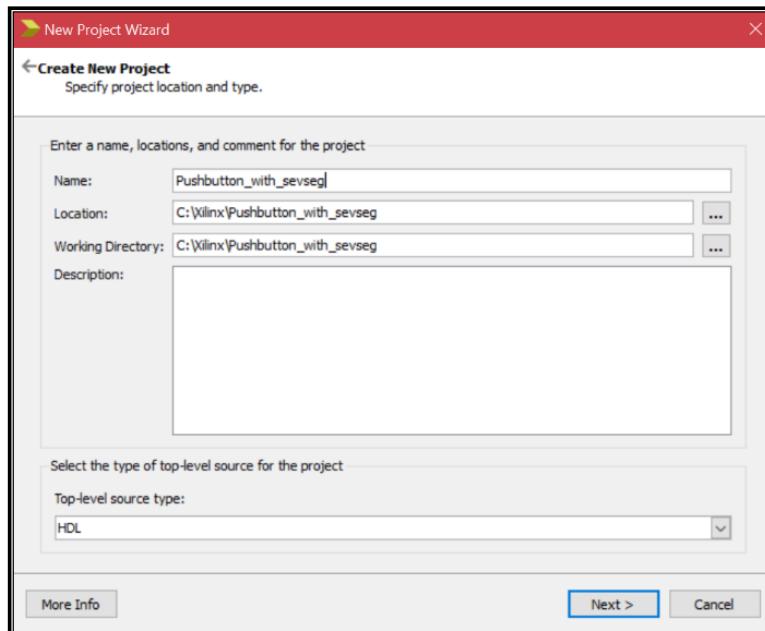
Percobaan Mandiri 1 :

Source Code UpDown Counter pada percobaan sebelumnya yaitu 4 Bit Counter, sekarang buatlah output menjadi UpDown Counter 8 Bit.

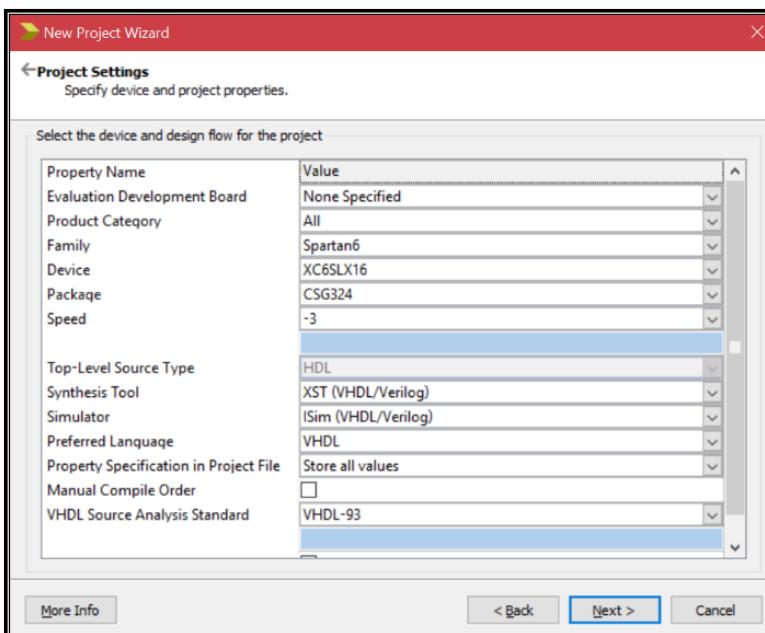
SOURCE CODE :**OUTPUT & KESIMPULAN :**

Percobaan 2 : Pushbutton With Seven Segment

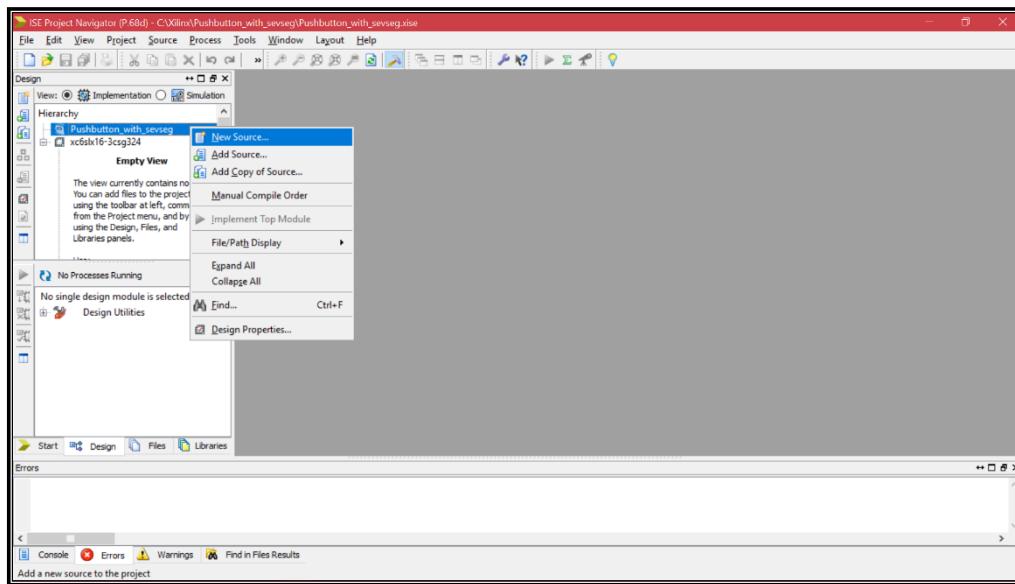
1. Buka software Xilinx ISE 14.5
2. Pilih menu **File → New Project**
3. Lalu isikan nama project pada tampilan dengan nama bebas, misalnya : **Pushbutton** Kemudian untuk *Top – level Source Type* isi dengan pilihan **HDL**. Setelah itu klik **Next**.



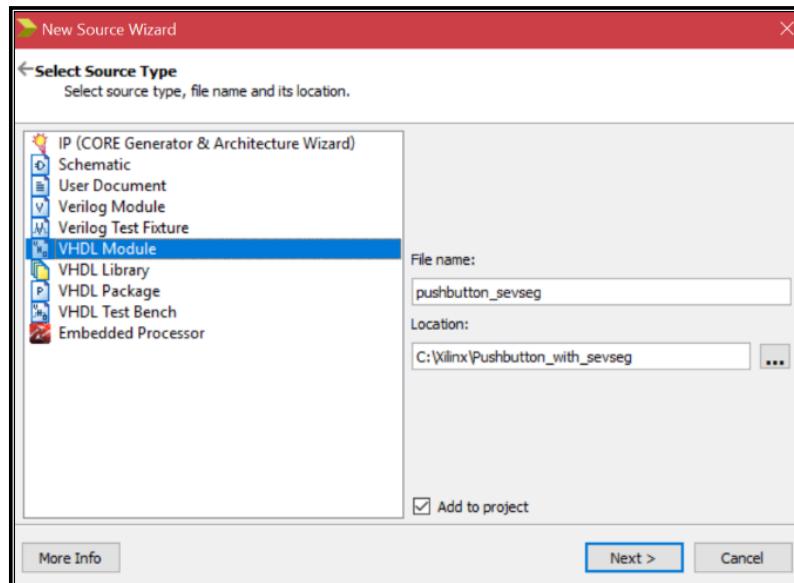
4. Atur *Device Properties* seperti gambar dibawah ini. Kemudian klik **Next** → **Next** → **Finish**



5. Lalu pada *Hierarchy*, klik kanan lalu **New Source**



6. Setelah itu, muncul tampilan seperti gambar dibawah. Tampilan tersebut untuk menentukan tipe asal (*source type*), nama file (*file name*), dan lokasi – nya (*its location*). *Source type* pilih **VHDL Module**, lalu *File name* isikan sesuai keinginan atau sesuai gambar, misalnya : **Pushbutton_sevseg**, sedangkan *location* sudah **default** seperti pada gambar.



7. Kemudian klik **Next → Next → Finish**

8. Lengkapi *Source Code* yang diberikan asisten dengan *Source Code* dibawah ini :



```

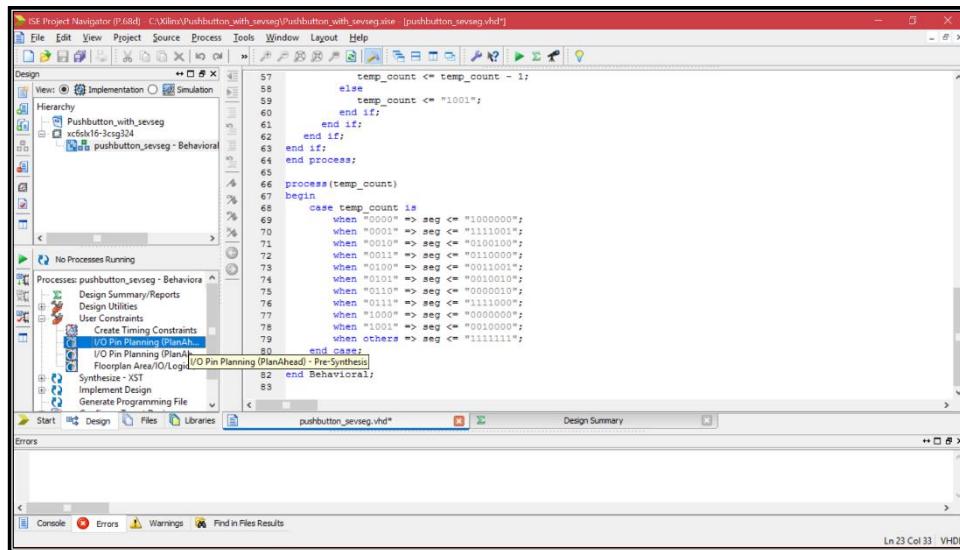
19 -----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_UNSIGNED.ALL;
23 use IEEE.STD_LOGIC_ARITH.ALL;
24 entity pushbutton_with_sevseg is
25 port(
26   clk : in std_logic;
27   count_up : in std_logic;
28   count_down : in std_logic;
29   seg : out std_logic_vector(6 downto 0));
30 end pushbutton_with_sevseg;
31 -----
32 -----
33 architecture Behavioral of pushbutton_with_sevseg is
34   signal temp_count : std_logic_vector(3 downto 0):="0000";
35   signal up_edge : std_logic_vector( 1 downto 0 );
36   signal down_edge: std_logic_vector( 1 downto 0 );
37   -----
38 begin
39 -----
40 begin
41 process(clk, temp_count)
42 begin
43 if rising_edge(clk) then
44   up_edge<= up_edge(0) & count_up;
45   if rising_edge(clk) then
46     up_edge<= up_edge(0) & count_up;
47     down_edge<= down_edge(0) & count_down;
48     if (up_edge = "01") then --rising
49       if temp_count <9 then
50         temp_count <= temp_count + 1;
51       else
52         temp_count <= "0000";
53       end if;
54     else
55       if (down_edge = "10") then --falling
56         if temp_count >0 then
57           temp_count <= temp_count - 1;
58         else
59           temp_count <= "1001";
60         end if;
61       end if;
62     end if;
63   end if;
64 end process;
65 process(temp_count)
66 begin
67 -----
68   case temp_count is
69     when "0000" => seg <= "1000000";
70     when "0001" => seg <= "1111001";
71     when "0010" => seg <= "0100100";
72     when "0011" => seg <= "0110000";
73     when "0100" => seg <= "0011001";
74     when "0101" => seg <= "0010010";
75     when others => seg <= "1111111";
76   end case;
77 end process;
78 end Behavioral;
79

```

9. Kemudian Save.

10. Langkah berikutnya, klik *expand* (+) pada **User Constraints**, pilih **I/O Pin Planning** (**PlanAhead**) – **Pre – Synthesis**.





11. Lalu akan muncul *Text Editor* dengan format *UCF* dan lengkapi Code yang diberikan Asisten.

```

1  NET "clk" LOC = "V10";
2  NET "count_up" LOC = "A8";
3  NET "count_down" LOC = "C9";
4
5  NET "seg(0)" LOC = "T17";
6  NET "seg(1)" LOC = "T18";
7  NET "seg(2)" LOC = "U17";
8  NET "seg(3)" LOC = "U18";
9  NET "seg(4)" LOC = "M14";
10 NET "seg(5)" LOC = "N14";
11 NET "seg(6)" LOC = "L14";

```

12. Berikutnya, kembali ke tab **Pushbutton.vhd**. Lalu pilih **Synthesize – XST**

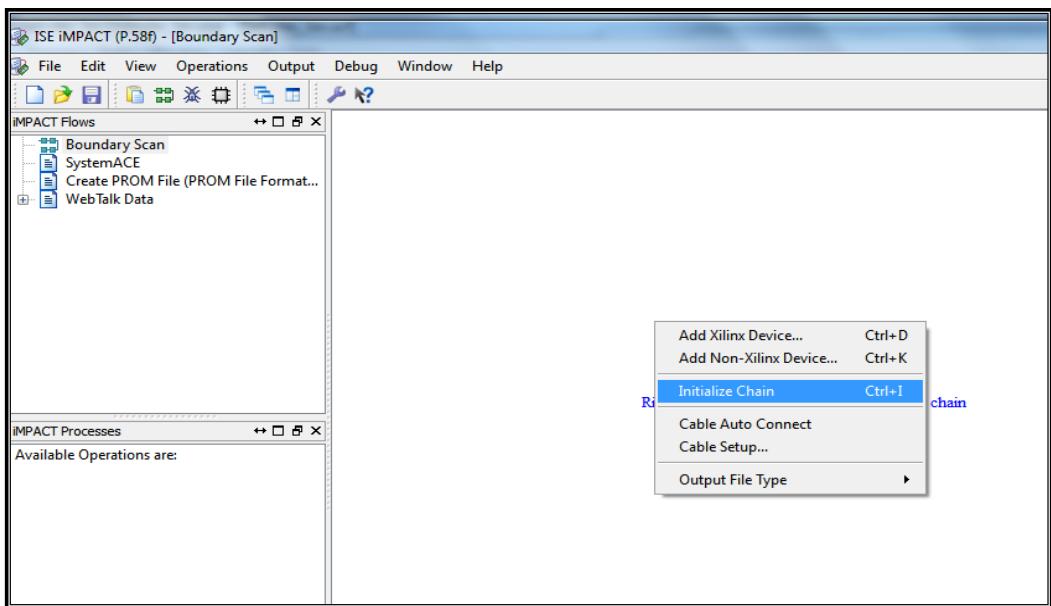
13. Klik **Implement Design**

14. Lalu, *double klik* pada **Generate Programming File**

15. Klik *expand (+)* pada **Configure Target Device** → **Generate Target PROM/ACE File** → klik kanan **Run**

16. Pada jendela ISE iMPACT, *double klik* pada **Boundary Scan**, lalu klik kanan pada *field Boundary Scan*, kemudian pilih **Initialize Chain**





17. Pilih **(--- nama module---).bit** → **Open** → bila ada *warning* klik **Ok**
18. Pada tab *Xilinx Web Talk Dialog*, klik **No** → **Ok**
19. Klik kanan pada gambar **IC Xilinx (warna hijau)** lalu pilih **Program...** seperti pada gambar



20. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**



SOURCECODE :



OUTPUT & KESIMPULAN :



Percobaan Mandiri 2

Source Code Pushbutton With Sevseg pada percobaan sebelumnya yaitu menghasilkan output 0 – 5 , sekarang buatlah output menjadi 0 - 9.

SOURCE CODE:

OUTPUT & KESIMPULAN:



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal dan Memahami Pemrograman Keyboard pada FPGA
- Praktikan Dapat Merancang Program Keyboard pada FPGA
- Praktikan Dapat memahami penggunaan Keyboard dalam FPGA

II. Dasar Teori

- Pengenalan Keyboard
- Pengenalan Perancangan Program VHDL
- Aplikasi perancangan program VHDL untuk Keyboard

III. Peralatan

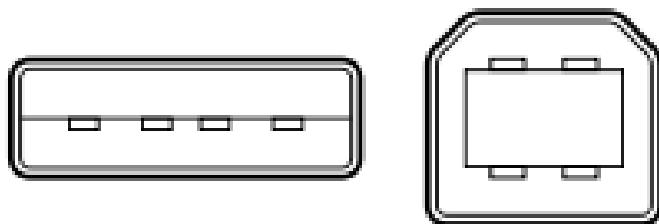
- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC
- Keyboard



6.1 Keyboard USB

Universal Serial Bus (USB) adalah standar bus serial untuk perangkat penghubung, biasanya kepada komputer namun juga digunakan di peralatan lainnya seperti konsol permainan, ponsel dan PDA (*Personal Digital Assistant*, yang berarti Pembantu Digital Pribadi seperti Handphone).

Sistem USB mempunyai desain yang asimetris, yang terdiri dari pengontrol *host* dan beberapa peralatan terhubung yang berbentuk "pohon" dengan menggunakan peralatan *hub* yang khusus.



Gambar 6.1 : Konektor USB Tipe A (Kiri) dan Konektor USB Tipe B (Kanan)

Desain USB ditujukan untuk menghilangkan perlunya penambahan *expansion card* ke ISA (*Industry Standard Architectur*) komputer atau bus PCI (*Peripheral Component Interconnect*), dan memperbaiki kemampuan plug-and-play (pasang-dan-mainkan) dengan memperbolehkan peralatan-peralatan ditukar atau ditambah ke sistem tanpa perlu mereboot komputer. Ketika USB dipasang, ia langsung dikenal sistem komputer dan memroses *device driver* yang diperlukan untuk menjalankannya.

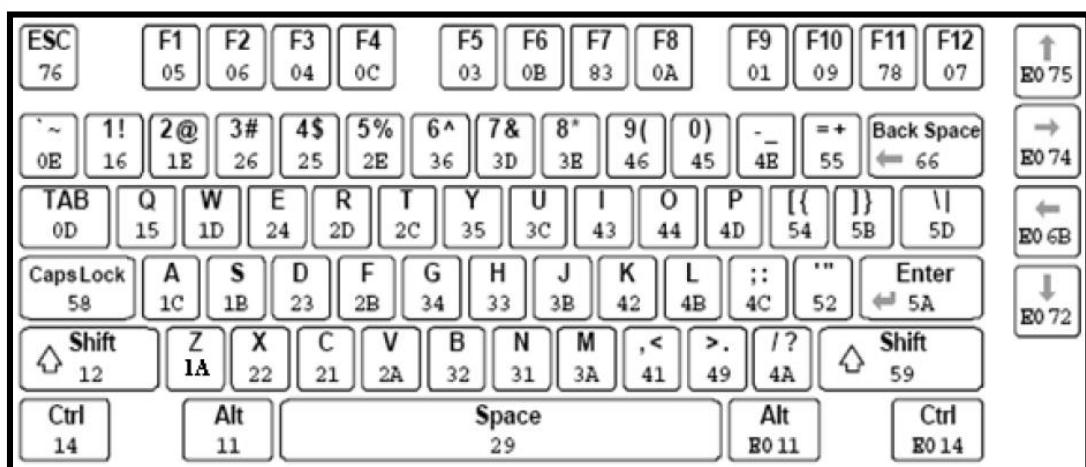
USB dapat menghubungkan peralatan tambahan komputer seperti mouse, keyboard, pemindai gambar, kamera digital, printer, hard disk, dan komponen *networking*. USB kini telah menjadi standar bagi peralatan multimedia seperti pemindai gambar dan kamera digital.

Keyboard USB merupakan salah satu yang contoh menggunakan konektor USB. Keyboard USB ini sudah banyak digunakan di komputer. Bahkan sekarang hampir setiap komputer sudah menggunakan Keyboard USB, dan mulai meninggalkan Keyboard PS/2, dikarenakan transfer data Keyboard USB lebih cepat di bandingkan dengan Keyboard PS/2.



6.2 Scan Code

Istilah-kan *host* yang berarti komputer, jika *keyboard* atau *mouse* dihubungkan dengan *PC (Komputer)* atau *host* dapat berupa mikrontroler bila *keyboard* atau *mouse* dihubungkan dengan mikrokontroler. Pada *keyboard* komputer, setiap kali salah satu tombol ditekan atau dilepas, *keyboard* akan mengirim kode ke *host*. Kode yang dikirimkan ke *host* tersebut dinamakan sebagai *scan code*. Sebagai contoh, bila *scan code* tombol ‘i’ adalah 43H (0100 0011). Ketika tombol ‘i’ ditekan *keyboard* akan mengirimkan 43H, jika tombol ‘i’ ditekan terus maka *keyboard* akan terus mengirimkan 43H terus menerus sampai tombol ‘i’ tadi dilepaskan atau ada tombol lain yang ditekan. *Keyboard* juga mengirimkan kode saat ada tombol yang dilepaskan, kodennya adalah F0H (1111 0000) kemudian diikuti dengan *scancode* tombol yang ditekan, jadi kalau tombol ‘i’ tadi dilepas *keyboard* akan mengirimkan kode F0H dan 43H. Kode-kode tersebut dikirim *keyboard* secara serial, artinya dikirimkan satu bit demi satu bit dimulai dari bit LSB dahulu, seperti pada gambar 2. Misalnya 43 dikirimkan dengan cara: mula-mula dikirim ‘1’, sesaat kemudian ‘1’ lagi dan menyusul ‘0’ setelah itu ‘0’ sampai akhirnya 8 bit yang berbentuk 0100 0011.



Gambar 6.2 : Scancode pada keyboard

6.3 Sistem Bilangan Biner

Sistem bilangan biner atau sistem bilangan basis dua adalah sebuah sistem penulisan angka dengan menggunakan dua simbol yaitu 0 dan 1. Sistem bilangan ini merupakan dasar dari semua sistem bilangan berbasis digital. Dari bilangan biner, kita dapat mengkonversinya ke dalam bilangan Oktal atau Hexadesimal. Sistem ini juga dapat kita sebut dengan istilah *bit* (*Binary Digit*). Pengelompokan biner dalam komputer selalu berjumlah 8, dengan istilah 1

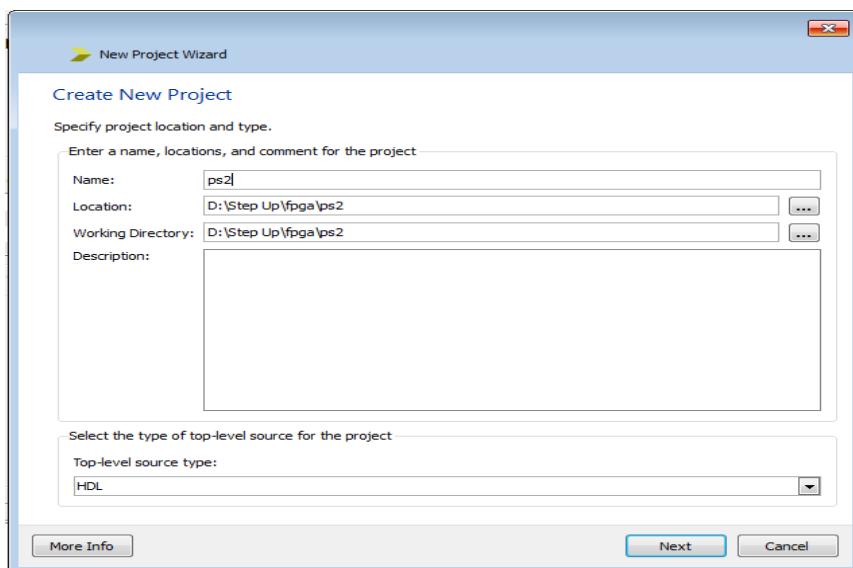


Byte/bita. Dalam istilah komputer, 1 Byte = 8 bit. Kode-kode rancang bangun komputer, seperti ASCII, (American Standard Code for Information Interchange) menggunakan sistem pengkodean 1 Byte.

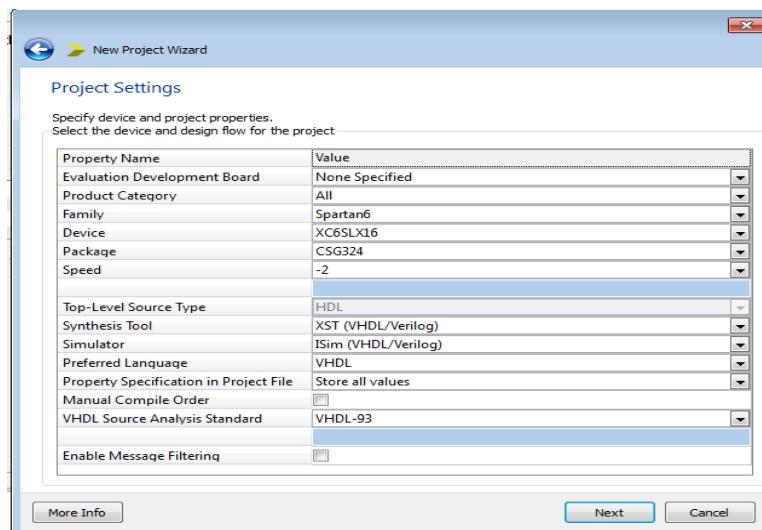
Percobaan 1 : Port USB

Langkah – langkah :

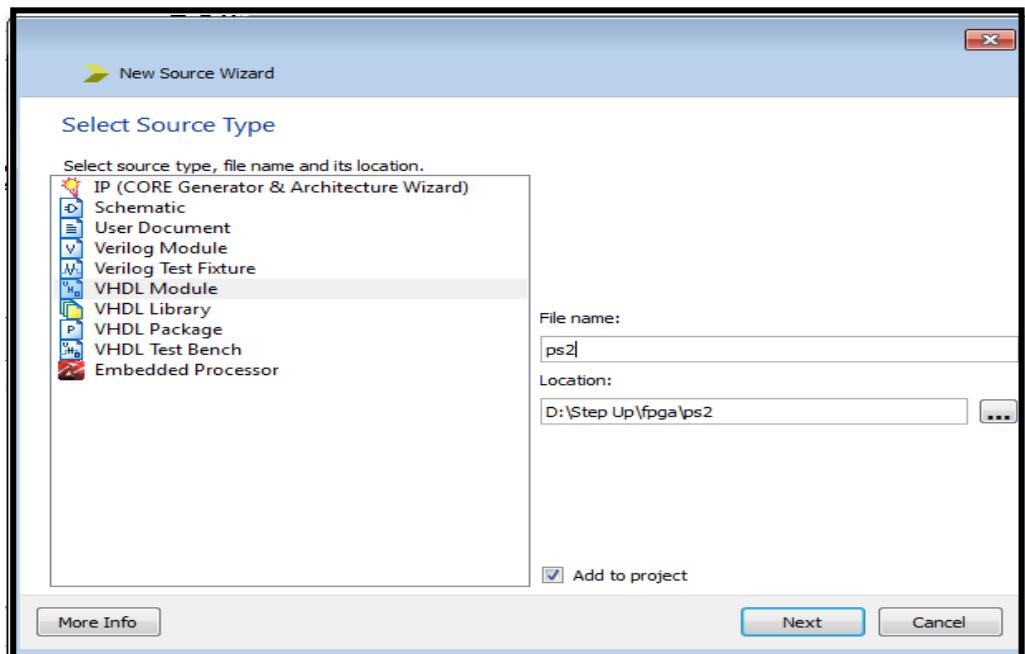
1. Bukalah software Xilinx ISE 14.5
2. pilih menu **File** -> kemudian pilih **New project**
3. Lalu isikan nama project pada tampilan dengan nama *USB*, kemudian untuk Top-level source type kita isi dengan pilihan **HDL**. Setelah itu klik Next.



4. Atur device properties seperti pada gambar di bawah ini. Kemudian klik **Next**.



5. Klik Next lalu klik Finish
6. Klik kanan pada hierarchy lalu pilih new source -> Pada tampilan sebelah kiri gambar pilih *VHDL Module*. Kemudian untuk file name, kita isi dengan nama yang sama pada saat kita membuat project baru yaitu **USB**. Lalu klik Next.



7. Kemudian klik *Next* lagi hingga *Finish* -> lalu pilih yes
8. Setelah itu klik *Next* lagi -> kemudian *Next* kembali -> kemudian *Finish*
9. Lengkapi source code yang diberikan oleh asisten dengan source code berikut ini:

```

72   process(clk, kd, m)
73   begin
74     if clk' event and clk  = '1' then
75       case m(7 downto 0) is
76         when "00010110" => NUM <= "00001";--1
77         when "00011110" => NUM <= "00010";--2
78         when "00100110" => NUM <= "00011";--3
79         when "00100101" => NUM <= "00100";--4
80         when "00101110" => NUM <= "00101";--5
81         when "00110110" => NUM <= "00110";--6
82
83         when others => NUM <= "11111";
84       end case;
85     end if;
86   end process;
87   led <= num;
88 end Behavioral;

```

Kemudian klik save



10. Langkah berikutnya adalah kita pilih pilihan **Synthesize – XST**. Dengan cara klik dua kali.
11. Klik tanda + pada User Constraints pilih **I/O Pin Planning (PlanAhead) – Pre-Synthesis** dengan klik kanan pilih **Run**.
12. Selanjutnya klik tanda + pada project gerbangAnd di Hierarchy -> double klik ps2.ucf -> pilih yes. Lalu isikan Pin pda bagian Loc seperti pada gambar di bawah ini. Setelah itu save dan klik Ok.

```

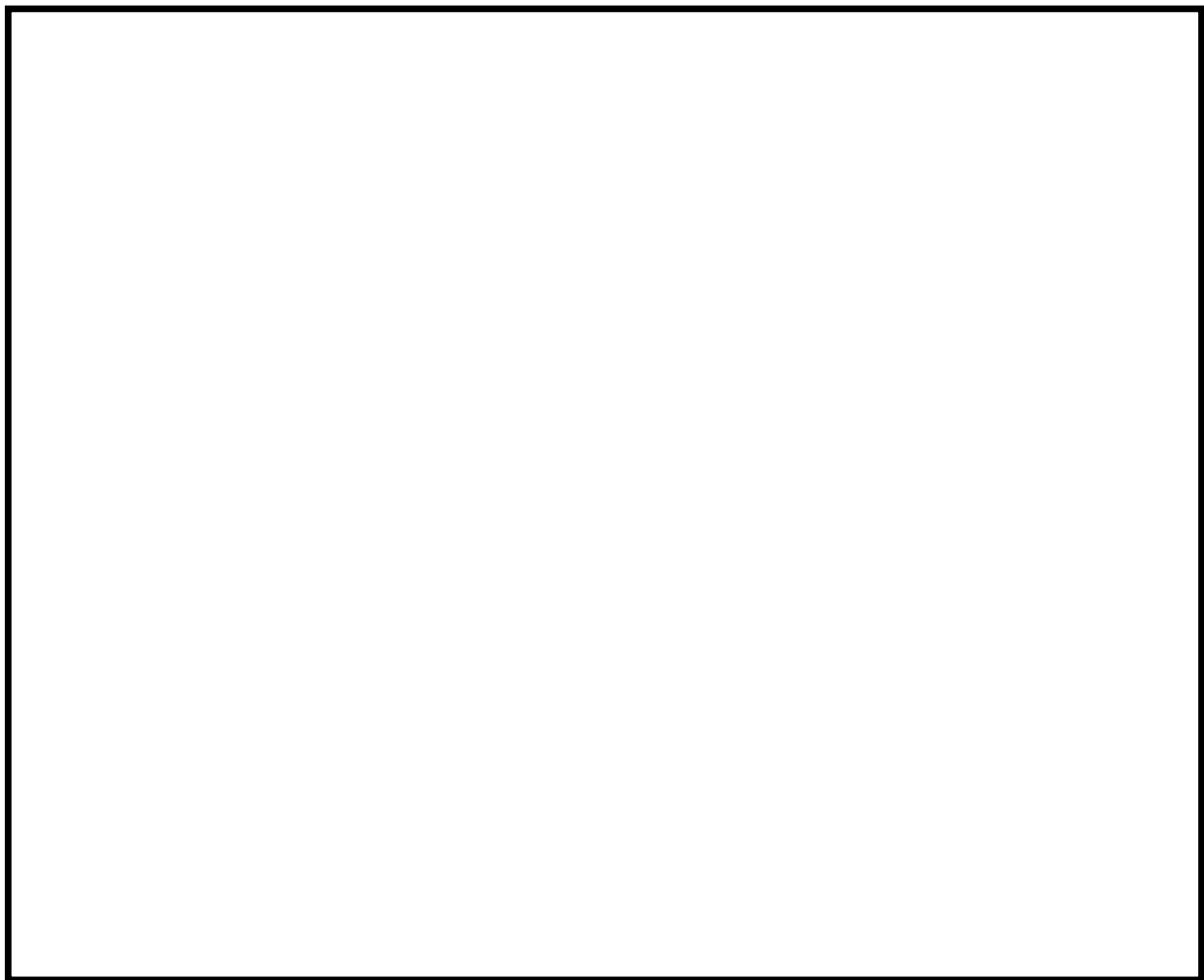
1  NET clkin  LOC = V10 ;
2
3  ## Pic USB-HID interface
4  Net "kd" LOC = J13 | IOSTANDARD = LVCMOS33 | PULLUP;
5  Net "kc" LOC = L12 | IOSTANDARD = LVCMOS33 | PULLUP;
6  #PACE: Start of PACE Area Constraints
7  Net "Led<0>"      LOC = "U16" | IOSTANDARD = "LVCMOS33";
8  Net "Led<1>"      LOC = "V16" | IOSTANDARD = "LVCMOS33";
9  Net "Led<2>"      LOC = "U15" | IOSTANDARD = "LVCMOS33";
10 Net "Led<3>"      LOC = "V15" | IOSTANDARD = "LVCMOS33";

```

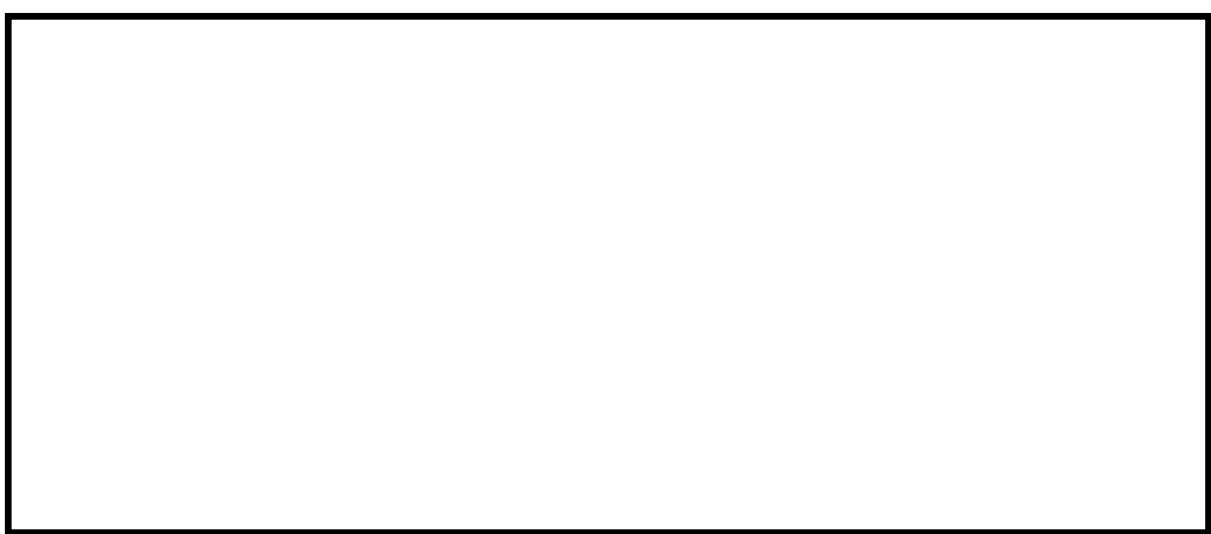
9. Lalu klik dua kali pada bagian **Implement Design**
10. Klik tanda + pada **Generate Programming File** -> pilih **Configure Device (IMPACT)** klik kanan lalu Run
11. Klik 2 kali **Configure Target Device** ->Pilih **Boundary Scan** ->Klik kanan pilih **Initialize Chain**
12. Tunggu beberapa detik lalu **Pilih (- - - nama module- - -)**.bit project yang di buat klik **Open** -> Bila ada tanda warning klik **no**
13. klik **OK** -> Identify Succeeded
14. Klik kanan pada gambar IC Xilinx sebelah kiri lalu pilih **program** seperti gambar
15. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**



OUTPUT & SOURCE CODE:



KESIMPULAN :



Program Mandiri 1:

Dengan menggunakan source program diatas, buatlah sebuah project baru. Dimana project tersebut menghasilkan output biner dari 1 sampai dengan huruf f:

SOURCE CODE:**OUTPUT & KESIMPULAN:**

Program Mandiri 2:

Dengan menggunakan source program diatas, buatlah sebuah project baru. Dimana project tersebut menghasilkan output sebagai berikut :

1. Apabila tombol angka 1 pada keyboard ditekan maka 4 led sebelah kiri mati dan 4 led sebelah kanan menyala.
2. Apabila tombol angka 2 pada keyboard ditekan maka 4 led sebelah kiri menyala dan 4 led sebelah kanan mati.
3. Apabila tombol angka 3 pada keyboard ditekan maka 2 led menyala dan mati.
4. Apabila tombol angka 4 pada keyboard ditekan maka 2 led sebelah kiri dan kanan menyala dan 4 led di tengah mati.
5. Apabila tombol angka 5 pada keyboard ditekan maka 1 led mati 1 led menyala dst.
6. Apabila tombol angka 6 seluruh led menyala

SOURCE CODE:

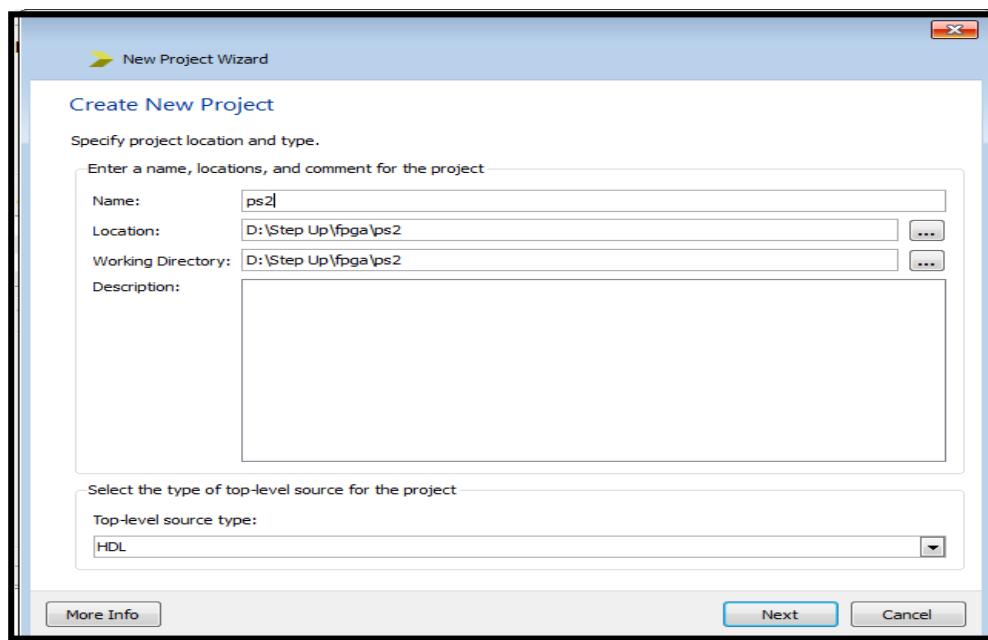
OUTPUT & KESIMPULAN:



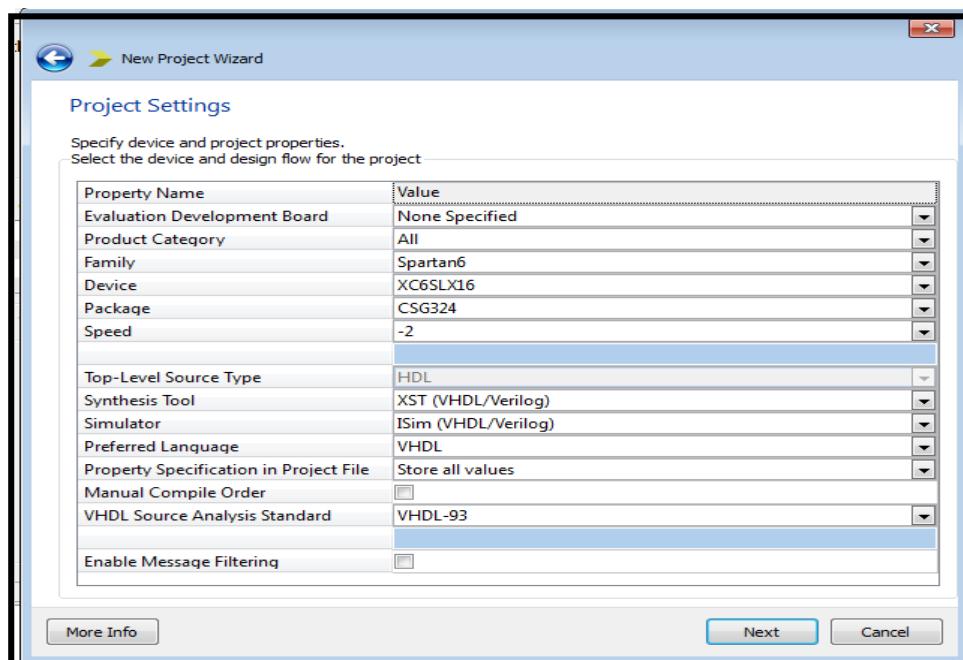
Percobaan 2

Langkah – langkah :

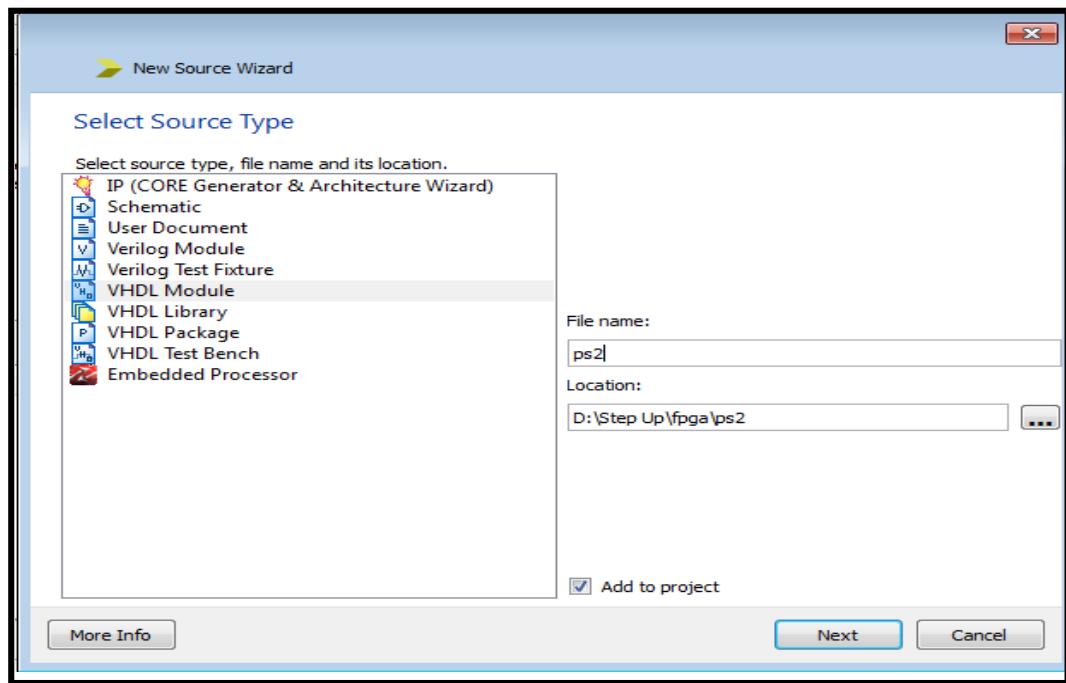
1. Bukalah software Xilinx ISE 14.5
2. pilih menu **File** -> kemudian pilih **New project**
3. Lalu isikan nama project pada tampilan dengan nama *ps2*, kemudian untuk Top-level source type kita isi dengan pilihan **HDL**. Setelah itu klik Next.



4. Atur device properties seperti pada gambar di bawah ini. Kemudian klik Next.



- Klik Next lalu klik Finish
- Klik kanan pada hierarchy lalu pilih new source -> Pada tampilan sebelah kiri gambar pilih *VHDL Module*. Kemudian untuk file name, kita isi dengan nama yang sama pada saat kita membuat project baru yaitu *ps2*. Lalu klik Next.



- Kemudian klik *Next* lagi hingga *Finish* -> lalu pilih *yes*
- Setelah itu klik *Next* lagi -> kemudian *Next* kembali -> kemudian *Finish*
- Lengkapi source code yang diberikan oleh asisten dengan source code berikut ini:

```

process(clk,kd,m)
begin
if clk' event and clk  = '1' then
  case m(7 downto 0) is
    when "00010110" => num <= "11111001";--1
    when "00011110" => num <= "10100100";--2
    when "00100110" => num <= "10110000";--3
    when "00100101" => num <= "10011001";--4
    when "00101110" => num <= "10010010";--5
    when "00110110" => num <= "10000010";--6
    when others => num <= "11111111";
  end case;
end if;
end process;
seg <= num;
end Behavioral;

```

Kemudian klik save



10. Langkah berikutnya adalah kita pilih pilihan **Synthesize – XST**. Dengan cara klik dua kali.
11. Klik tanda + pada User Constraints pilih **I/O Pin Planning (PlanAhead) – Pre-Synthesis** dengan klik kanan pilih **Run**.
12. Selanjutnya klik tanda + pada project gerbangAnd di Hierarchy -> double klik ps2.ucf -> pilih yes. Lalu isikan Pin pda bagian Loc seperti pada gambar di bawah ini. Setelah itu save dan klik Ok.

```

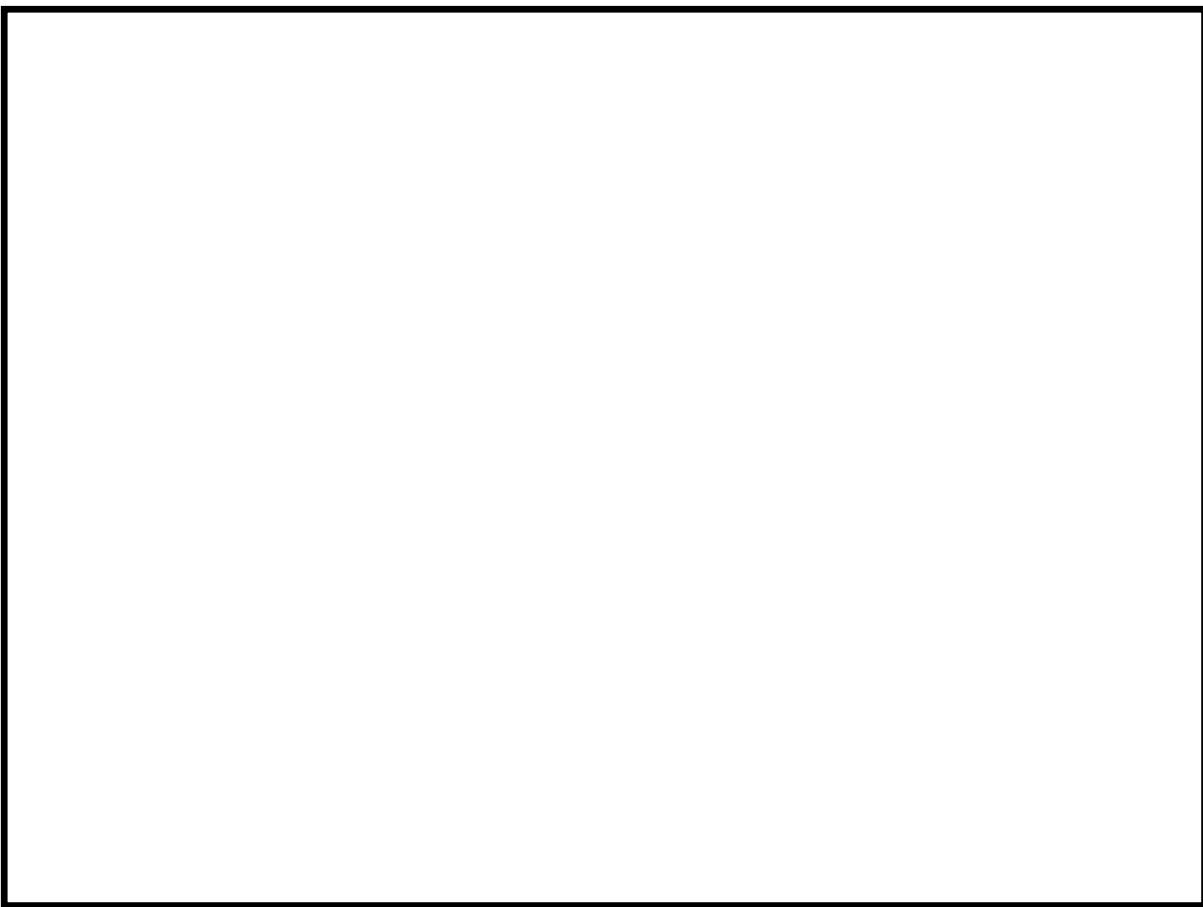
1  NET clkin LOC = V10 ;
2
3  ## Pic USB-HID interface
4  Net "kd" LOC = J13 | IOSTANDARD = LVCMOS33 | PULLUP;
5  Net "kc" LOC = L12 | IOSTANDARD = LVCMOS33 | PULLUP;
6  #PACE: Start of PACE Area Constraints
7  NET seg<0> LOC= T17;
8  NET seg<1> LOC= T18;
9  NET seg<2> LOC= U17;
10 NET seg<3> LOC= U18;
11 NET seg<4> LOC= M14;
12 NET seg<5> LOC= N14;
13 NET seg<6> LOC= L14;
14 NET seg<7> LOC= M13;

```

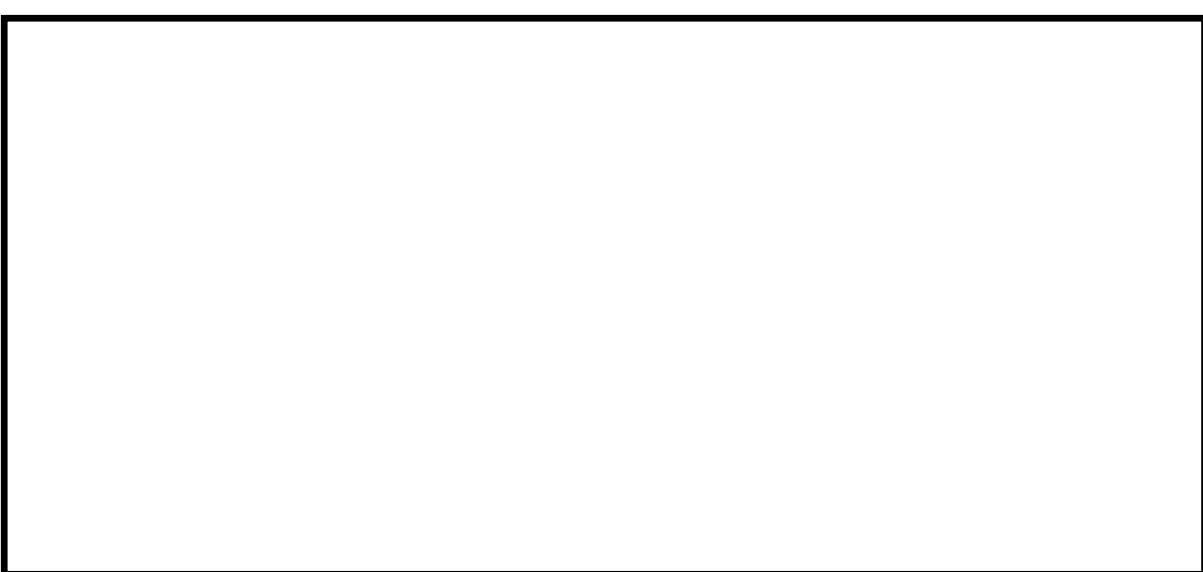
13. Lalu klik dua kali pada bagian **Implement Design**
14. Klik tanda + pada **Generate Programming File** -> pilih **Configure Device (iMPACT)** klik kanan lalu Run
15. Klik 2 kali **Configure Target Device** ->Pilih **Boundary Scan** ->Klik kanan pilih **Initialize Chain**
16. Tunggu beberapa detik lalu **Pilih (- - - nama module- - -).bit** project yang di buat klik **Open** -> Bila ada tanda warning klik **no**
17. klik **OK** -> Identify Succeeded
18. Klik kanan pada gambar IC Xilinx sebelah kiri lalu pilih **program** seperti gambar
19. Lalu pilih **OK** sampai muncul tulisan **Program Succeeded**



SOURCE CODE:



OUTPUT & KESIMPULAN:



Percobaan Mandiri 2 :

Dari source code pada percobaan diatas, buatlah program segment lainnya. Dimana program tersebut menghasilkan output angka 1 sampai dengan huruf f

SOURCE CODE:

OUTPUT & KESIMPULAN:



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal dan Memahami Program VHDL pada FPGA
- Praktikan Dapat Merancang Program VHDL pada Pemrograman FPGA menggunakan Port VGA
- Praktikan Dapat Memahami Penggunaan port VGA pada FPGA

II. Dasar Teori

- Pengenalan Port VGA
- Merancang Desain Program Menggunakan VHDL
- Aplikasi Perancangan Program VHDL Untuk Port VGA

III. Peralatan

- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC
- VGA Cable
- Monitor

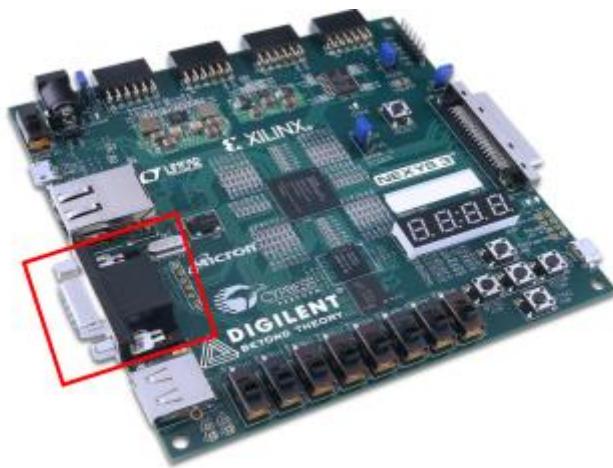


7.1 VGA

VGA adalah standar tampilan komputer analog yang dipasarkan pertama kali oleh IBM pada tahun 1987. Walaupun standar VGA sudah tidak lagi digunakan karena sudah diganti oleh standar yang lebih baru, VGA masih diimplementasikan pada Pocket PC. VGA merupakan standar grafis terakhir yang diikuti oleh mayoritas pabrik pembuat kartu grafis komputer. Tampilan Windows sampai sekarang masih menggunakan modus VGA karena didukung oleh banyak produsen monitor dan kartu grafis.

Istilah VGA juga sering digunakan untuk mengacu kepada resolusi layar berkuran layar 640x480, apa pun pembuat perangkat keras kartu grafisnya. Kartu VGA berguna untuk menerjemahkan keluaran komputer ke monitor. Untuk proses desain grafis atau bermain permainan video, diperlukan kartu grafis yang berdaya tinggi. Produsen kartu grafis yang terkenal antara ATI dan nVidia.

Selain itu, VGA juga dapat mengacu kepada konektor VGA 15-pin yang masih digunakan secara luas untuk mengantarkan sinyal video analog ke monitor. Standar VGA secara resmi digantikan oleh standar XGA dari IBM, tetapi nyatanya VGA justru digantikan oleh Super VGA.



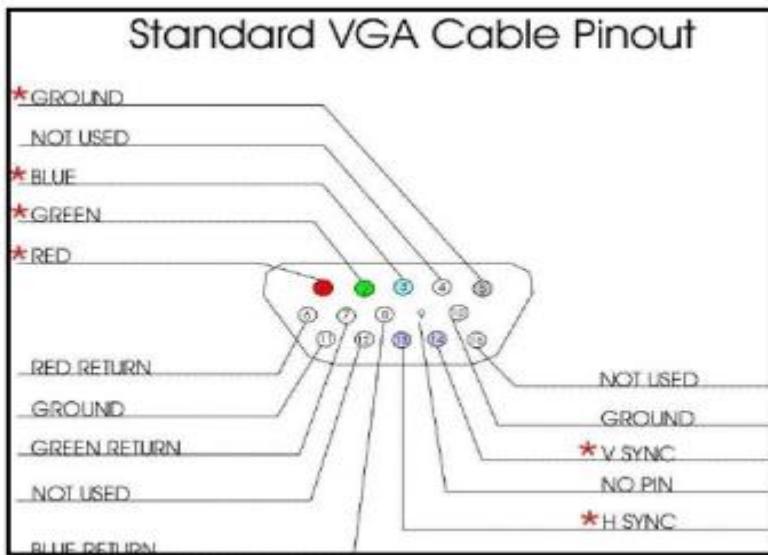
Gambar 7.1 Port VGA pada FPGA

Port pada VGA terdiri dari 15-pin yang fungsinya masing-masing. Di bawah ini gambar dari port VGA beserta fungsi dari setiap pin-pinnya.





Gambar 7.2 Port VGA 15-pin



Gambar 7.3 Fungsi Pin pada Port VGA

7.2 Resolusi VGA

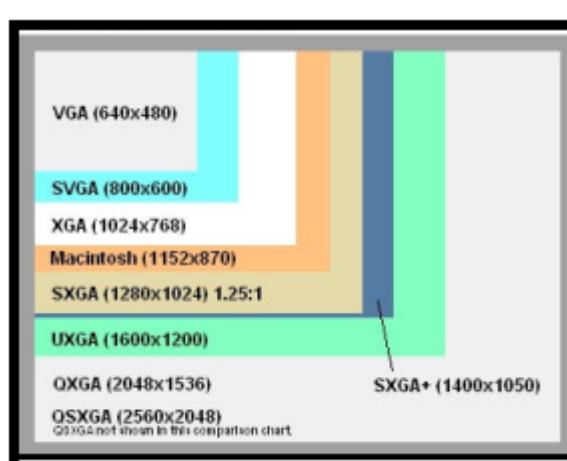
Berikut ini ialah beberapa graphics array lain yang teknologinya diturunkan dari VGA, namun mendukung resolusi yang lebih atau berbeda dari VGA :

- WVGA, singkatan dari Wide Video Graphic Array, ialah resolusi display yang mendukung ukuran :
 - 800 x 480 pixel
 - 848 x 480 pixel
 - 854 x 480 pixel
- SVGA dan WSVGA adalah perkembangan lebih lanjut dari VGA, resolusi SVGA adalah :
 - 800 x 600 pixel
- WSVGA adalah jenis dari SVGA untuk tampilan widescreen dengan perbandingan 16:9, resolusi SVGA adalah :



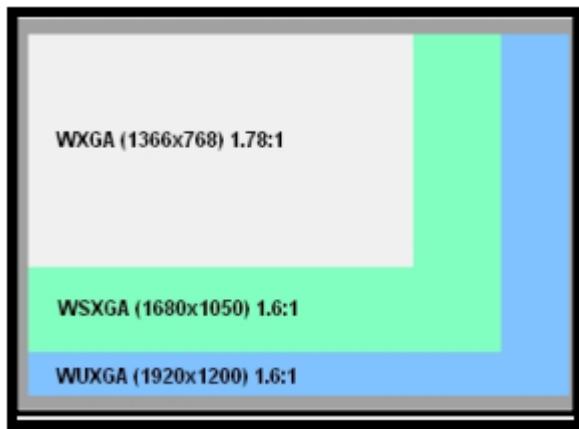
- 1024 x 576 / 600 pixel
- XGA adalah singkatan dari Extended Graphics Array. Singkatan ini menggunakan X sebab XGA merujuk pada adapter video lama dengan teknologi berbeda. Resolusi XGA adalah :
 - 800 x 600 pixel
 - 1024 x 768 pixel
- WXGA adalah singkatan dari Wide Extended Graphics Array. Ini adalah varian resolusi dari XGA yang dikhususkan untuk tampilan bermodus widescreen. Ada beberapa resolusi yang bisa dikelaskan sebagai WXGA :
 - 1280 x 720 pixel
 - 1280 x 768 pixel
 - 1280 x 800 pixel
 - 1360 x 768 pixel
 - 1366 x 768 pixel
- WSXGA adalah singkatan dari Wide Super Extended Graphics Array, tingkat resolusi ini diatas dari WXGA, dan resolusinya sendiri adalah :
 - 1680 x 1050 pixel
- WUXGA adalah singkatan dari Wide Ultra Extended Graphic Array, WUXGA mempunyai resolusi :
 - 1920 x 1200 pixel (2K)

Dan gambar di bawah ini adalah standar resolusi yang digunakan pada layar monitor



Gambar 7.3 Standard Screen Resolution





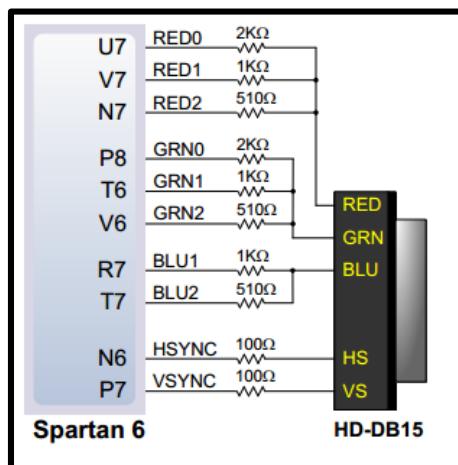
Gambar 7.4 Wide Screen Resolution

7.3 DAC (Digital Analog Converter)

DAC adalah perangkat yang digunakan untuk mengkonversi sinyal masukan dalam bentuk digital menjadi sinyal keluaran dalam bentuk analog (tegangan). Tegangan keluaran yang dihasilkan **DAC** sebanding dengan nilai digital yang masuk ke dalam **DAC**.

- **RAMDAC (Random Access Memory Digital – Analog Converter)**

Fungsinya mengubah gambar digital menjadi sinyal analog agar dapat dibaca di monitor. Dimana RAM akan menyimpan sementara informasi gambar yang akan ditampilkan di monitor dalam data digital. Untuk menampilkannya ke dalam monitor analog, RAMDAC akan membaca isi video memory, mengubah data digital menjadi sinyal analog, dan mengirimkannya melalui kabel video ke monitor. Komponen inilah yang menentukan refresh rate dari kartu video.



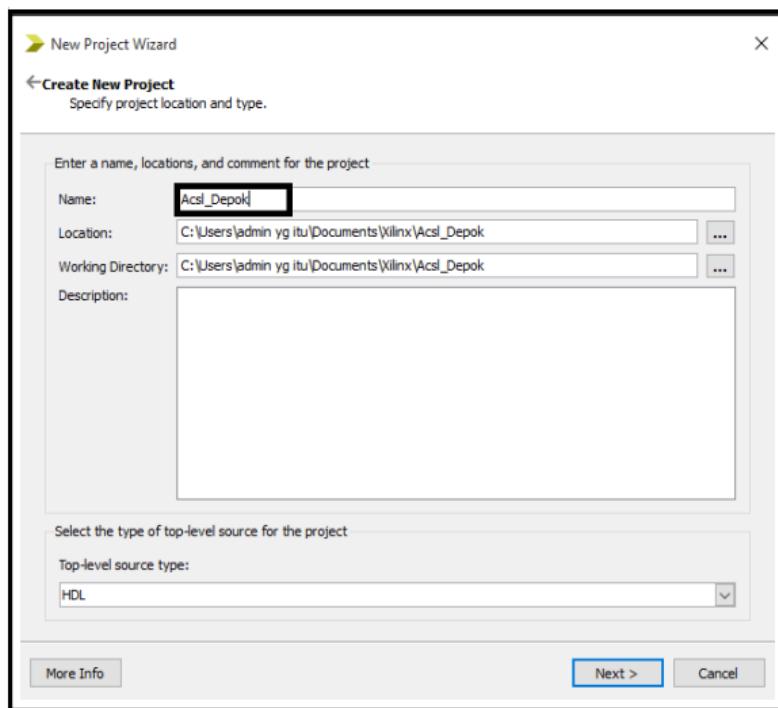
Gambar 7.5 Rangkain DAC pada VGA



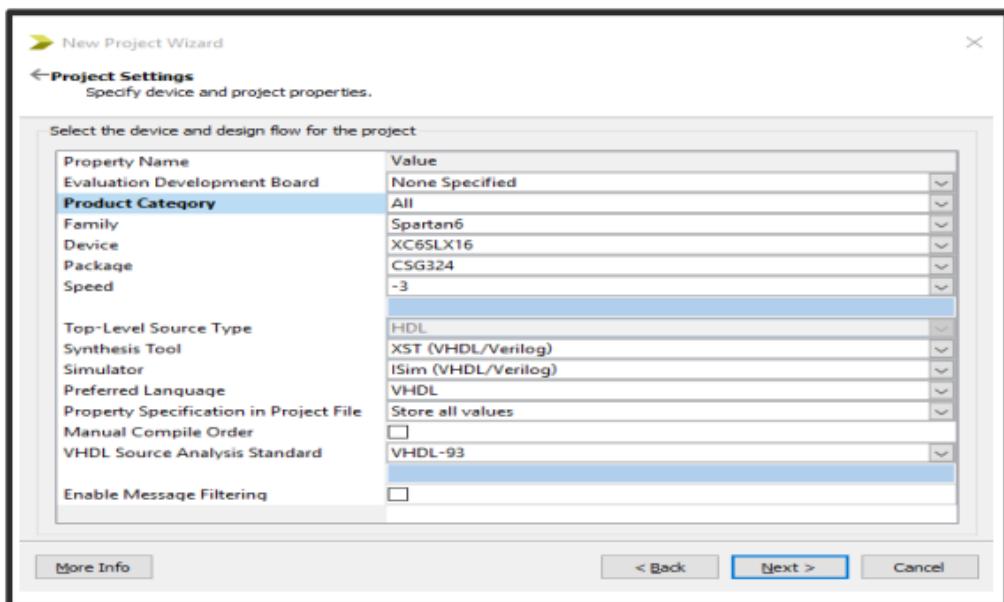
Percobaan 1 : Membuat tampilan warna blink pada layar monitor

Langkah – langkah :

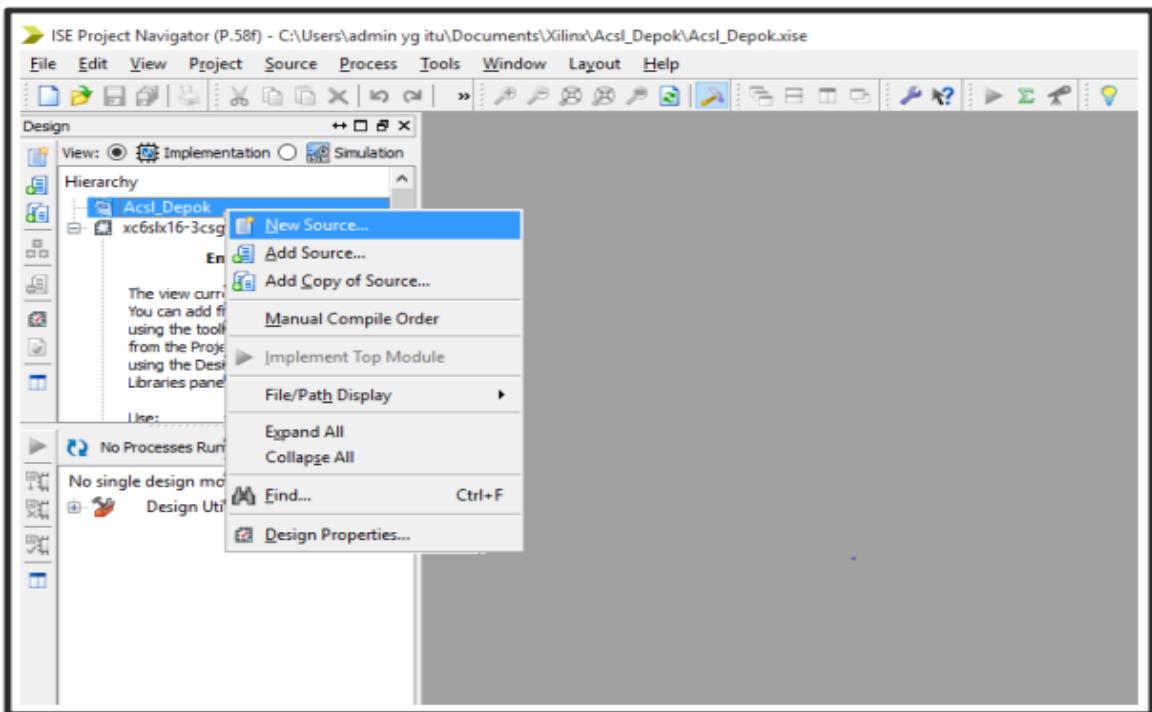
1. Buka aplikasi **ISE Design Suite 14.5**
2. Buka Menu **File > New Project**
3. Buat nama project lalu klik next seperti gambar di bawah ini :



4. Setelah itu akan muncul project settings dan ikuti aturannya seperti gambar di bawah ini :

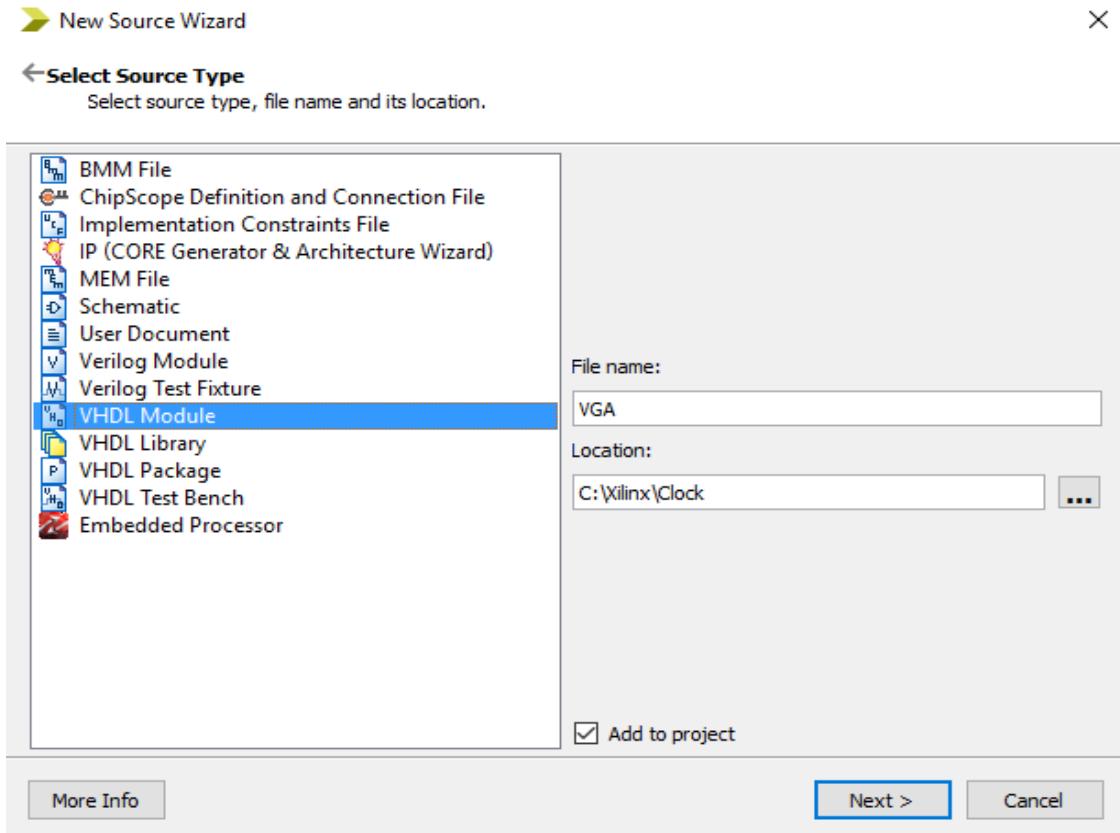


5. Klik **Next** > lalu klik **Finish**
6. Lalu akan muncul halaman form untuk project, setelah itu **klik kanan** pada nama **Acsl_Depok** > **Pilih New Source** seperti gambar di bawah ini :

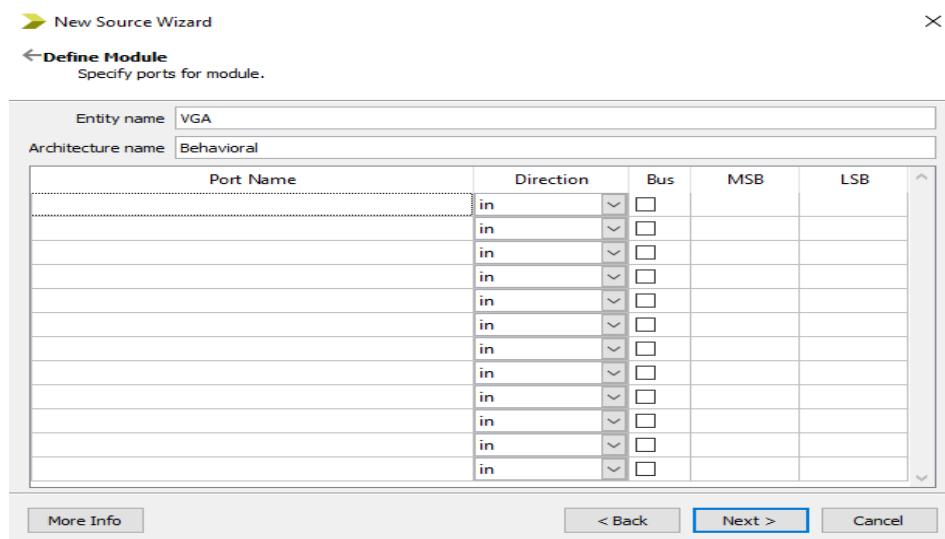


7. Pilih bagian **VHDL Module** lalu isikan nama project dengan **VGA**, lalu klik **Next** seperti gambar di bawah ini :





8. Selanjutnya untuk bagian I/O tidak usah diisi > selanjutnya klik **Next** dan **Finish**.



9. Sampai tahap ini pastikan port VGA yang ada pada FPGA terhubung dengan port VGA pada layar monitor
10. Masukan source code seperti di bawah ini dan **Save** :



```

1 _____;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity _____ is
7     port(clk100 : in std_logic;
8             red_out : out std_logic := '0';
9             green_out : out std_logic := '0';
10            blue_out : out std_logic := '0';
11            hs_out   : out std_logic;
12            vs_out   : out std_logic);
13 end _____;
14
15 architecture Behavioral of _____ is
16     signal clk25      : std_logic := '0';
17     signal clk50      : std_logic := '0';
18     signal color       : std_logic_vector (2 downto 0);
19     signal horizontal_counter : std_logic_vector (9 downto 0);
20     signal vertical_counter    : std_logic_vector (_ downto _);
21
22 begin
23 process (clk100)
24 variable cnt : integer := 0;
25 begin
26     if clk100'event and clk100 = '1' then
27         clk50 <= not clk50;
28         if cnt = 0 then
29             clk25 <= not clk25;
30             cnt := 1;
31         else
32             cnt := 0;
33         end if;
34     end if;
35 end process;
36
37 process (clk25)
38 variable clr : integer := 0;
39 variable tbl : integer := 0;
40 begin
41     clr := clr + 1;
42
43     if clr = 500000 then
44         case tbl is
45             _____
46             0 =>           color <= "111";
47             when 1 =>       color <= "000";
48             when 2 =>       color <= "010";
49             when others => color <= "100";
50
51         end case;
52
53     clr := 0;
54 end if;
55
56

```



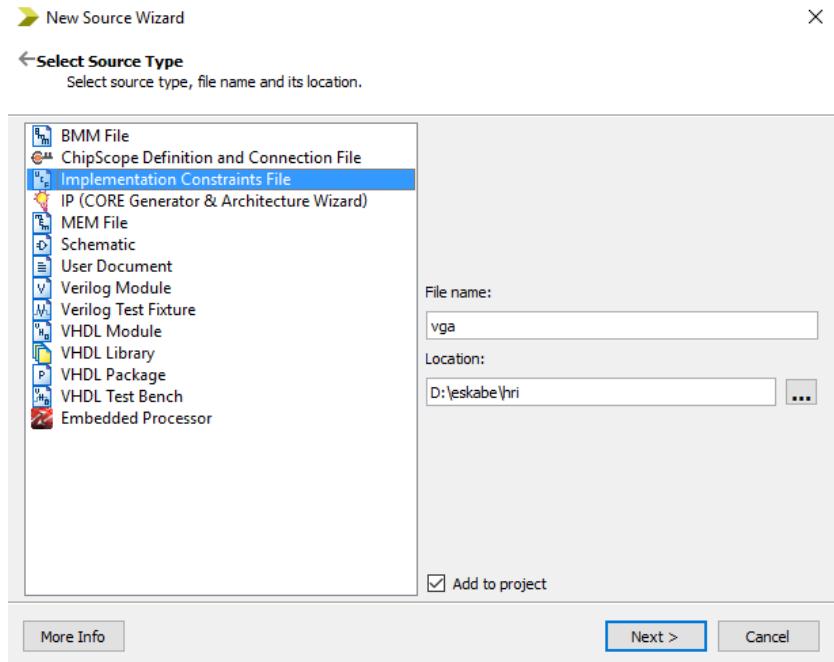
```

58      if clk25'event and clk25 = '1' then
59          if (horizontal_counter >= "0010010000" ) -- 144
60              and (horizontal_counter < "1100010000" ) -- 784
61              and (vertical_counter >= "0000100111" ) -- 39
62              and (vertical_counter < "1000000111" ) -- 519
63          then
64              _____out <= _____(0);
65              _____out <= color(1);
66              _____out <= color(2);
67          else
68              red_out <= '0';
69              green_out <= '0';
70              blue_out <= '0';
71          end if;
72
73          if (horizontal_counter > "0000000000" )
74              and (horizontal_counter < "0001100001" ) -- 96+1
75          then
76              hs_out <= '_';
77          else
78              hs_out <= '1';
79          end if;
80
81          if (vertical_counter > "0000000000" )
82              and (vertical_counter < "0000000011" ) -- 2+1
83          then
84              vs_out <= '_';
85          else
86              vs_out <= '1';
87          end if;
88
89          horizontal_counter <= horizontal_counter+"0000000001";
90
91          if (horizontal_counter="1100100000") then
92              vertical_counter <= vertical_counter+"0000000001";
93              tbl := (tbl + 1) mod 4;
94              horizontal_counter <= "0000000000";
95          end if;
96
97          if (vertical_counter="1000001001") then
98              vertical_counter <= "0000000000";
99          end if;
00      end if;
01  end process;
02
03 end _____;

```

11. Halaman form untuk project, setelah itu **klik kanan** pada nama **Acsl_Depok > Pilih New Source dan Pilih Implementation Constrain File** seperti gambar di bawah ini :





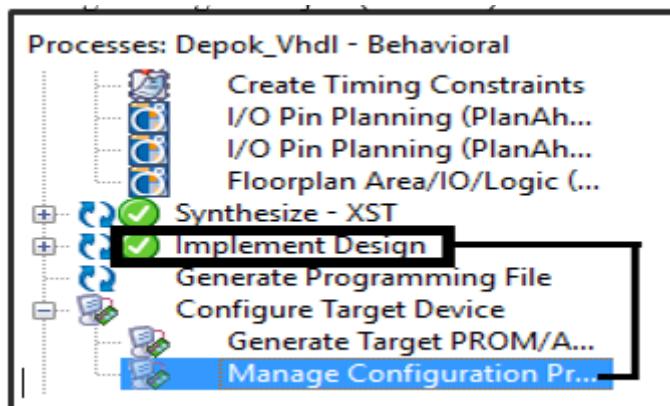
12. Pilih bagian file ucf isikan dengan source code seperti dibawah ini :

```

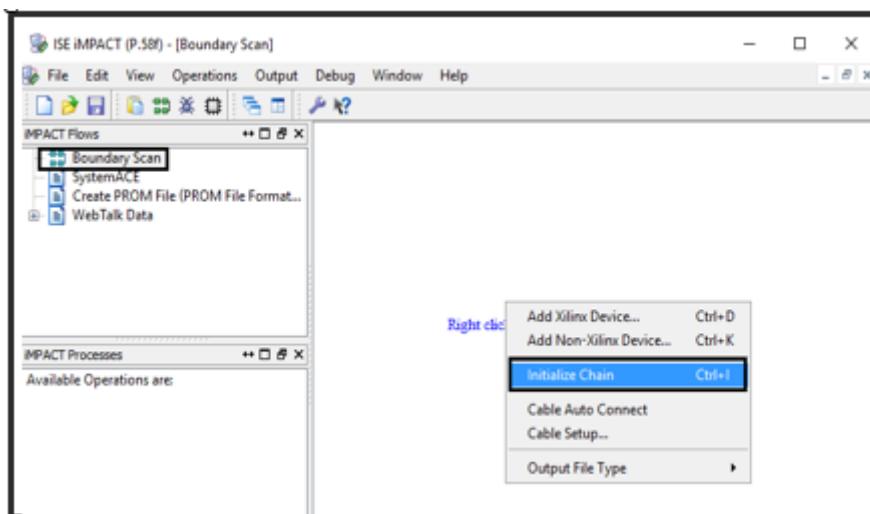
1 Net "clk100" LOC=V10 | IOSTANDARD=LVCMS33;
2 Net "clk100" TNM_NET = sys_clk_pin;
3 TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz;
4 NET "red_out" LOC = N7 | IOSTANDARD = LVCMS33; # Bank = 2, pin name = IO_L44P, Sch name = RED2
5 NET "green_out" LOC = V6 | IOSTANDARD = LVCMS33; # Bank = 2, pin name = IO_L45N, Sch name = GRN2
6 NET "blue_out" LOC = T7 | IOSTANDARD = LVCMS33; # Bank = 2, pin name = IO_L46N, Sch name = BLU2
7 NET "hs_out" LOC = N6 | IOSTANDARD = LVCMS33; # Bank = 2, pin name = IO_L47P, Sch name = HSYNC
8 NET "vs_out" LOC = P7 | IOSTANDARD = LVCMS33; # Bank = 2, pin name = IO_L47N, Sch name = VSYNC

```

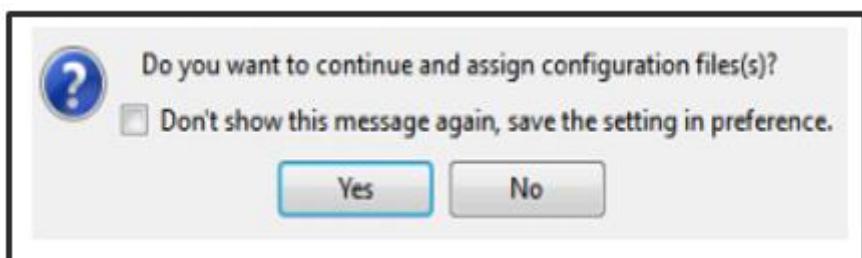
13. Selanjutnya pilih **Implement Desain** lalu klik tanda + pada **Configure Target Device** dan Pilih Manage Configure Project (iMPACT).



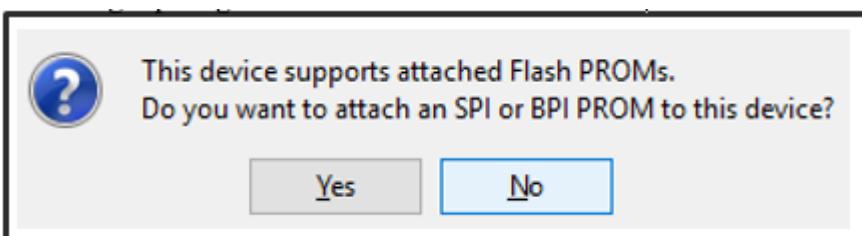
14. Tampilan selanjutnya akan muncul pilih **Boundary Scan** > lilih klik kanan pada form “Right click to Add Device or Initialize JTAG chain” > pilih **Initialize Chain** seperti gambar di bawah ini:



15. Pilih Yes pada **Auto Assign Configuration Files Query Dialog**

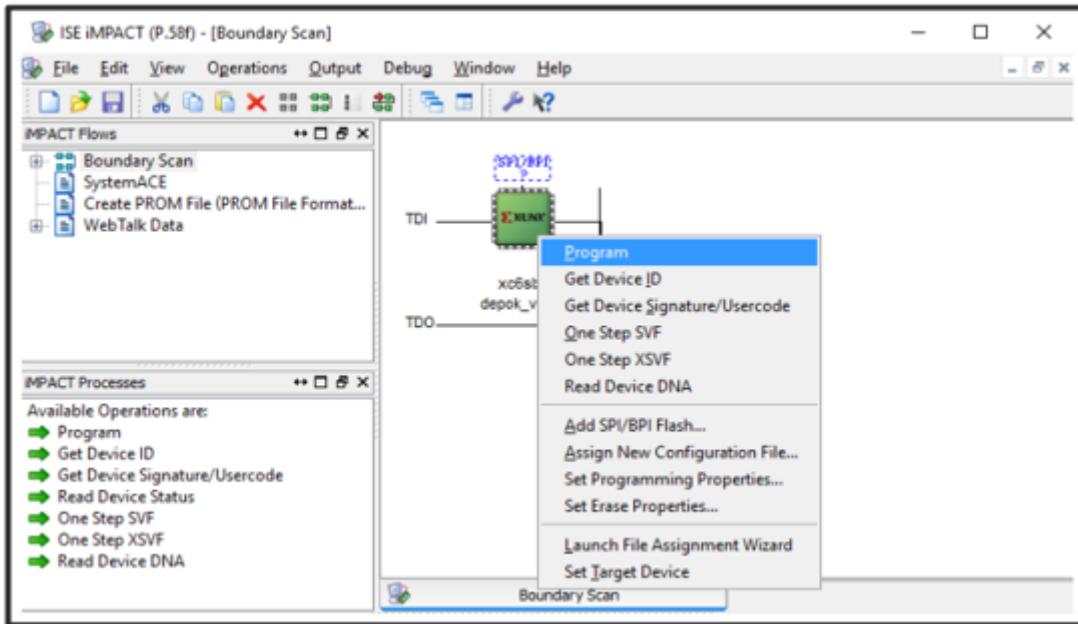


16. Jika muncul tampilan **Assign New Configuration File**, back satu kali lalu pilih file yang ber extensi .bit sesuai dengan nama project yg dibuat > klik **Open**
17. Pilih **No** pada dialog seperti gambar di bawah ini > dan Pilih **OK** :

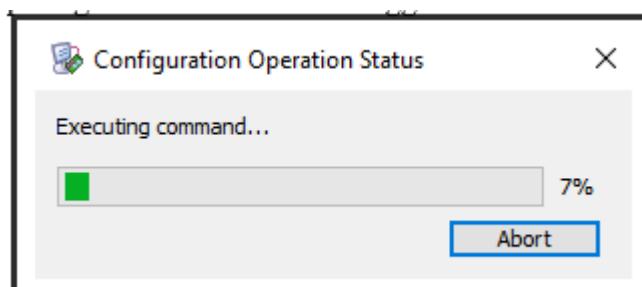


18. Klik kanan di bagian gambar IC FPGA dan Pilih **Program**.





19. Proses seperti gambar di bawah ini hingga selesai :



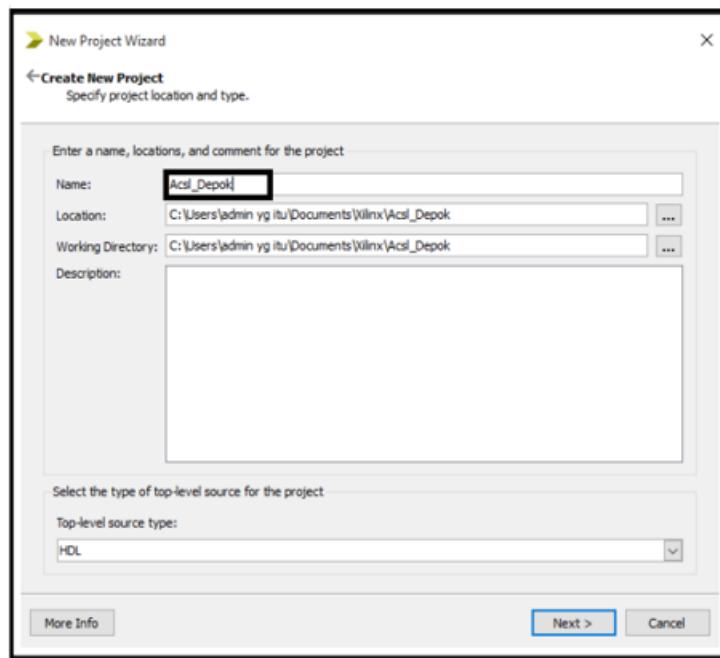
20. Setelah **100%** lihat hasilnya di FPGA kalian



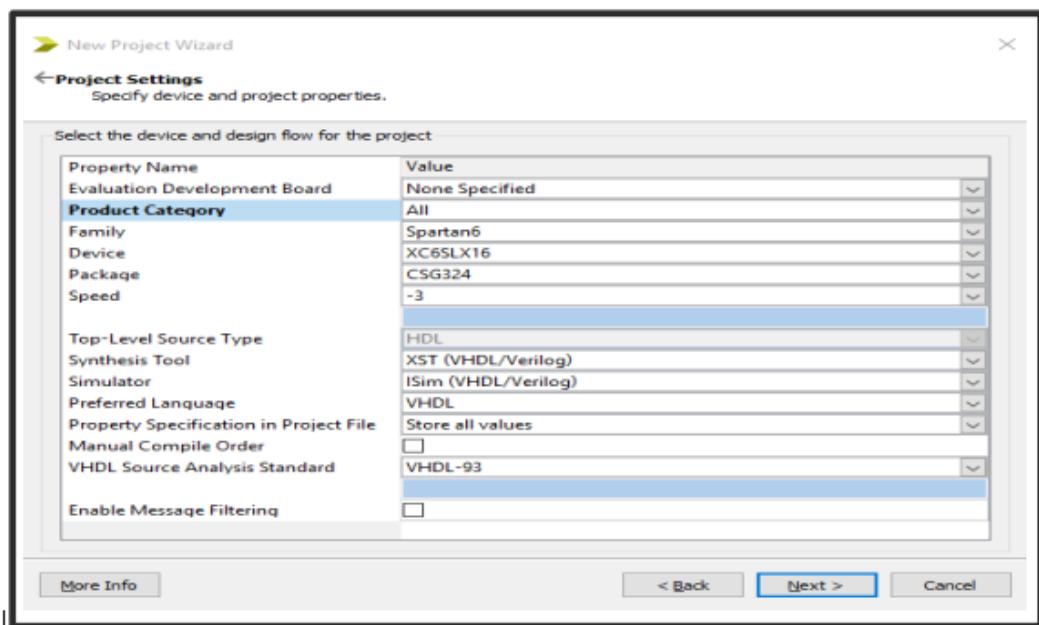
Percobaan 2 : Membuat tampilan warna pada layar monitor

Langkah – langkah :

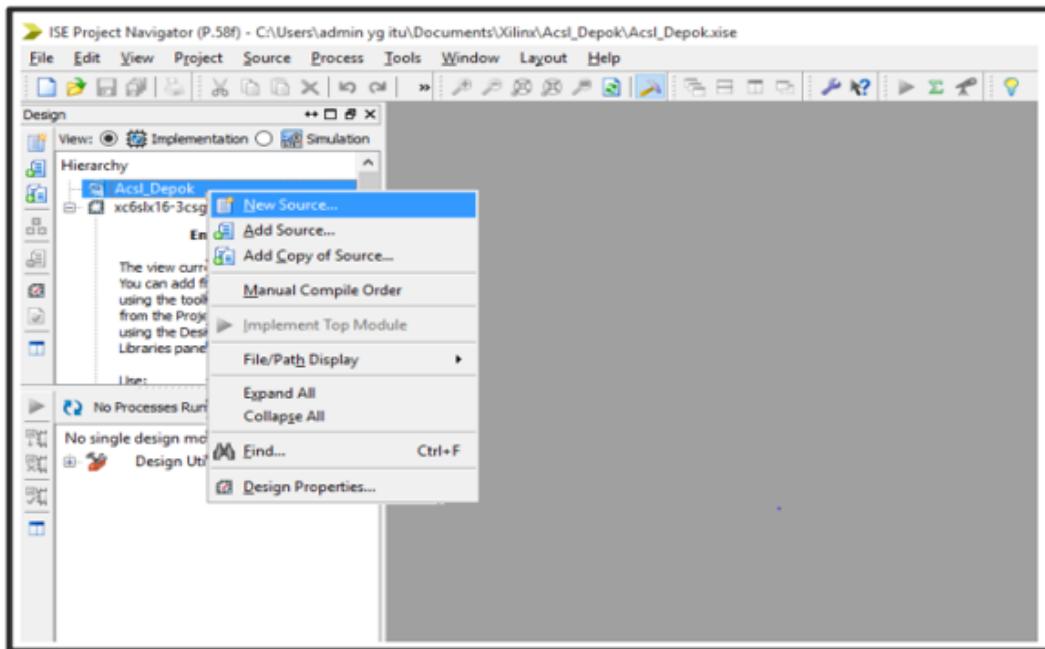
1. Buka aplikasi ISE Design Suite 14.5
2. Buka Menu File > New Project
3. Buat nama project lalu klik next seperti gambar di bawah ini :



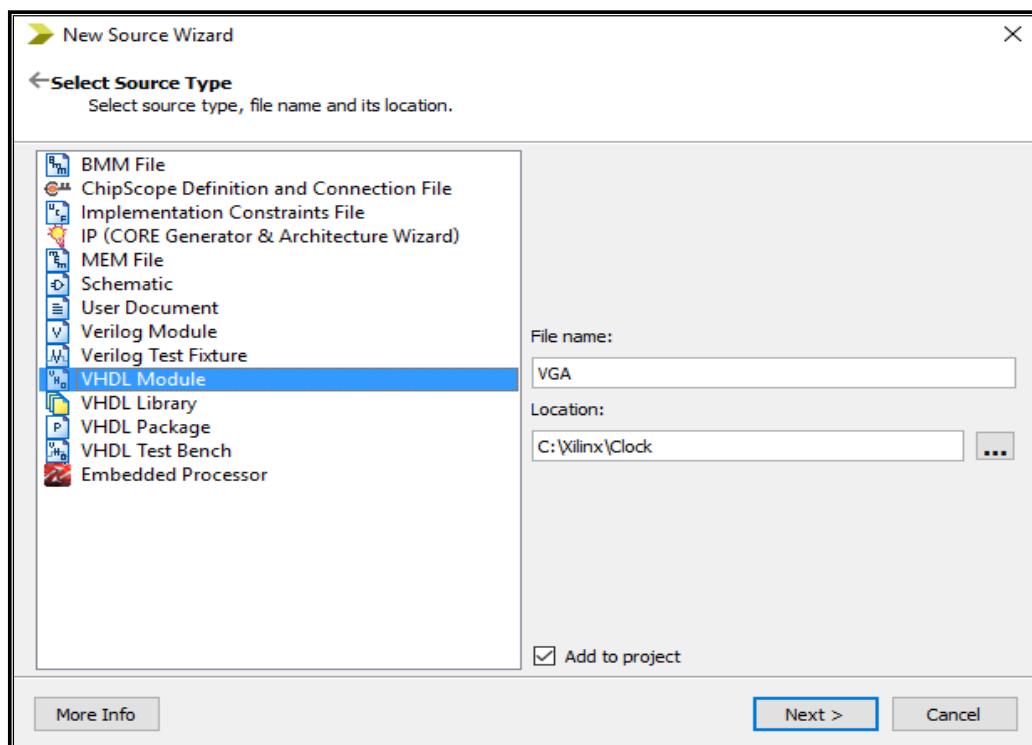
4. Setelah itu akan muncul project settings dan ikuti aturannya seperti gambar di bawah ini :



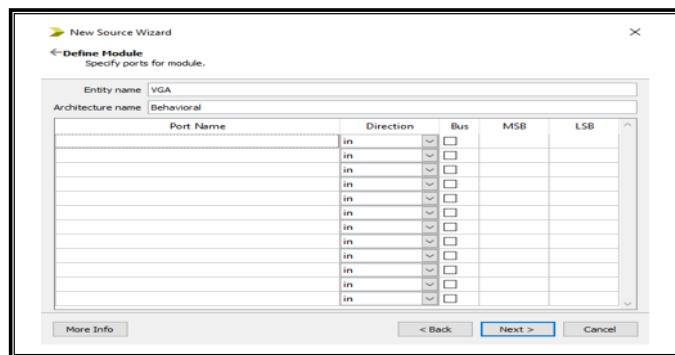
5. Klik **Next >** lalu klik **Finish**
6. Lalu akan muncul halam form untuk project, setelah itu **klik kanan** pada nama **Acs1_Depok > Pilih New Source** seperti gambar di bawah ini :



7. Pilih bagian **VHDL Module** lalu isikan nama project dengan **VGA**, lalu klik **Next** seperti gambar di bawah ini :



8. Selanjutnya untuk bagian **I/O** tidak usah diisi > selanjutnya klik **Next** dan **Finish**.



9. Sampai tahap ini pastikan port VGA yang ada pada FPGA terhubung dengan port VGA pada layar monitor
10. Masukan source code seperti di bawah ini dan **Save** :

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity __ is
7     port(clk100 : in std_logic;
8           red_out  : out std_logic_vector( 2 downto 0);
9           green_out : out std_logic_vector( 2 downto 0);
10          blue_out  : out std_logic_vector( 1 downto 0) ;
11          hs_out    : out std_logic;
12          vs_out    : out std_logic);
13 end __;
14
15 architecture Behavioral of __ is
16
17 signal clk25          : std_logic;
18 signal clk50          : std_logic;
19 signal horizontal_counter : std_logic_vector (9 downto 0);
20 signal vertical_counter  : std_logic_vector (9 downto 0);
21
22 begin
23
24 -- generate a 25Mhz clock
25 process (clk100)
26 variable cnt : integer := 0;
27 begin
28     if clk100'event and clk100 = '1' then
29         clk50 <= not clk50;
30         if cnt = 0 then
31             clk25 <= not clk25;
32             cnt := 1;
33         else
34             cnt := 0;
35         end if;
36     end if;
37 end process;

```

ty



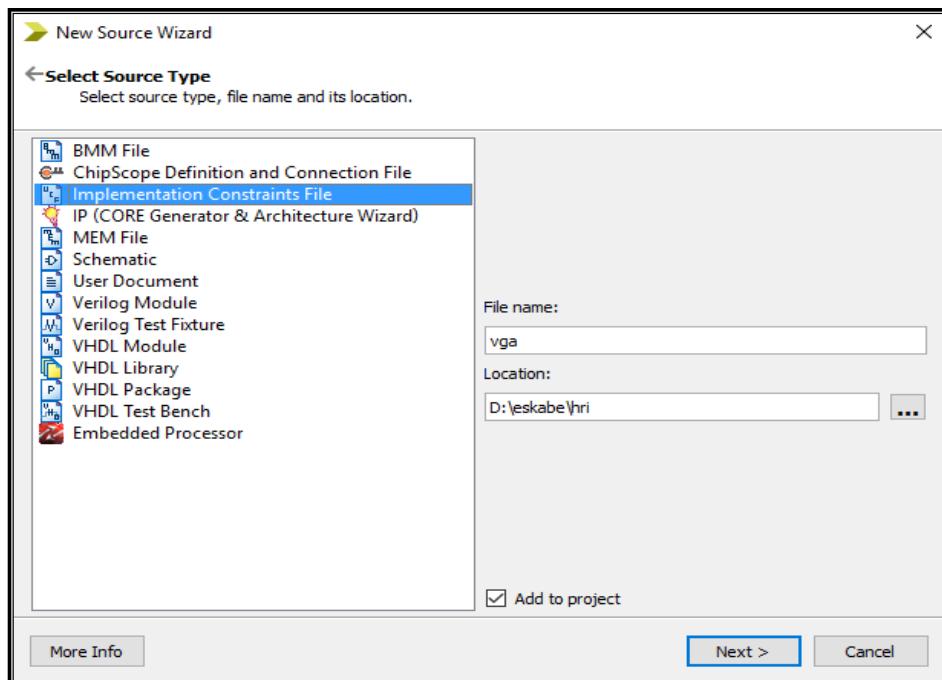
```

39 process (clk25)
40 begin
41   if clk25'event and clk25 = '1' then
42     if (horizontal_counter >= "0010010000" ) -- 144
43     and (horizontal_counter < "1100010000" ) -- 784
44     and (vertical_counter >= "0000100111" ) -- 39
45     and (vertical_counter < "1000000111" ) -- 519
46     then
47       |
48       --here you paint!!
49       red_out <= "____";
50       green_out <= "____";
51       blue_out <= "____";
52     |
53   else
54     red_out <= "____";
55     green_out <= "____";
56     blue_out <= "____";
57   end if;
58   if (horizontal_counter > "0000000000" )
59   and (horizontal_counter < "0001100001" ) -- 96+1
60   then
61     hs_out <= '____';
62   else
63     hs_out <= '____';
64   end if;
65   |
66   if (vertical_counter > "0000000000" )
67   and (vertical_counter < "0000000011" ) -- 2+1
68   then
69     vs_out <= '____';
70   else
71     vs_out <= '____';
72   end if;
73   horizontal_counter <= horizontal_counter+"0000000001";
74   if (horizontal_counter="1100100000") then
75     vertical_counter <= vertical_counter+"0000000001";
76     horizontal_counter <= "0000000000";
77   end if;
78   if (vertical_counter="1000001001") then
79     vertical_counter <= "0000000000";
80   end if;
81 end ____;
82
83 end ____;

```

11. Halaman form untuk project, setelah itu **klik kanan** pada nama **AcsL_Depok > Pilih New Source dan Pilih Implementation Constrain File** seperti gambar di bawah ini :



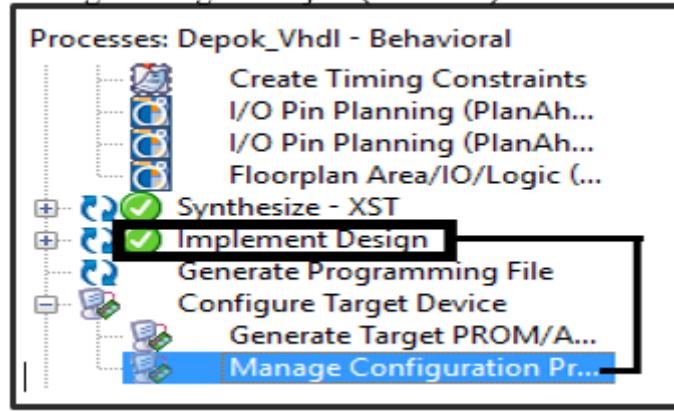


12. Pilih bagian file ucf isikan dengan source code seperti dibawah ini :

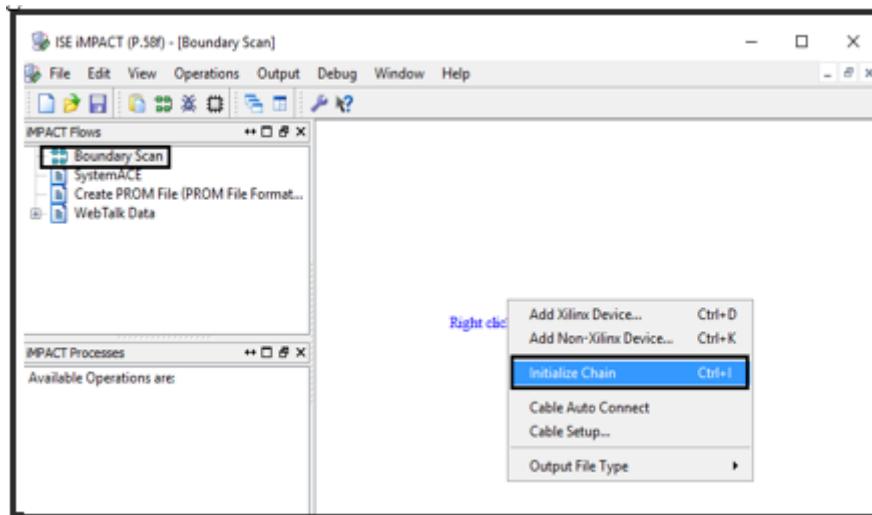
```
1 NET "clk100" LOC = V10;
2 NET "clk100" TNM_NET = "sys_clk_pin";
3 TIMESPEC TS_sys_clk_pin = PERIOD "sys_clk_pin" 100000 KHz;
4   NET "hs_out" LOC = N6 ;
5   NET "vs_out" LOC = P7 ;
6
7 # PlanAhead Generated physical constraints
8
9 NET "red_out[0]" LOC = U7;
10 NET "red_out[1]" LOC = V7;
11 NET "red_out[2]" LOC = N7;
12 NET "green_out[0]" LOC = P8;
13 NET "green_out[1]" LOC = T6;
14 NET "green_out[2]" LOC = V6;
15 NET "blue_out[0]" LOC = R7;
16 NET "blue_out[1]" LOC = T7;
```

13. Selanjutnya pilih **Implement Desain** lalu klik tanda + pada **Configure Target Device** dan Pilih Manage Configure Project (iMPACT).

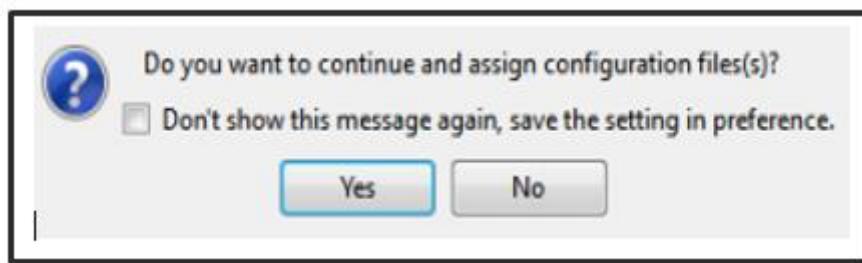




14. Tampilan selanjutnya akan muncul pilih **Boundary Scan** > lilih klik kanan pada form “Right click to Add Device or Initialize JTAG chain” > pilih **Initialize Chain** seperti gambar di bawah ini:

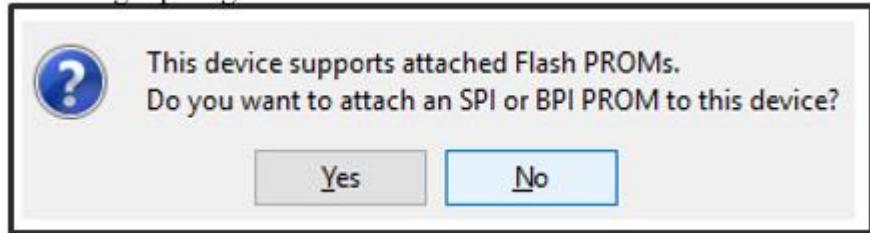


15. Pilih **Yes** pada **Auto Assign Configuration Files Query Dialog**

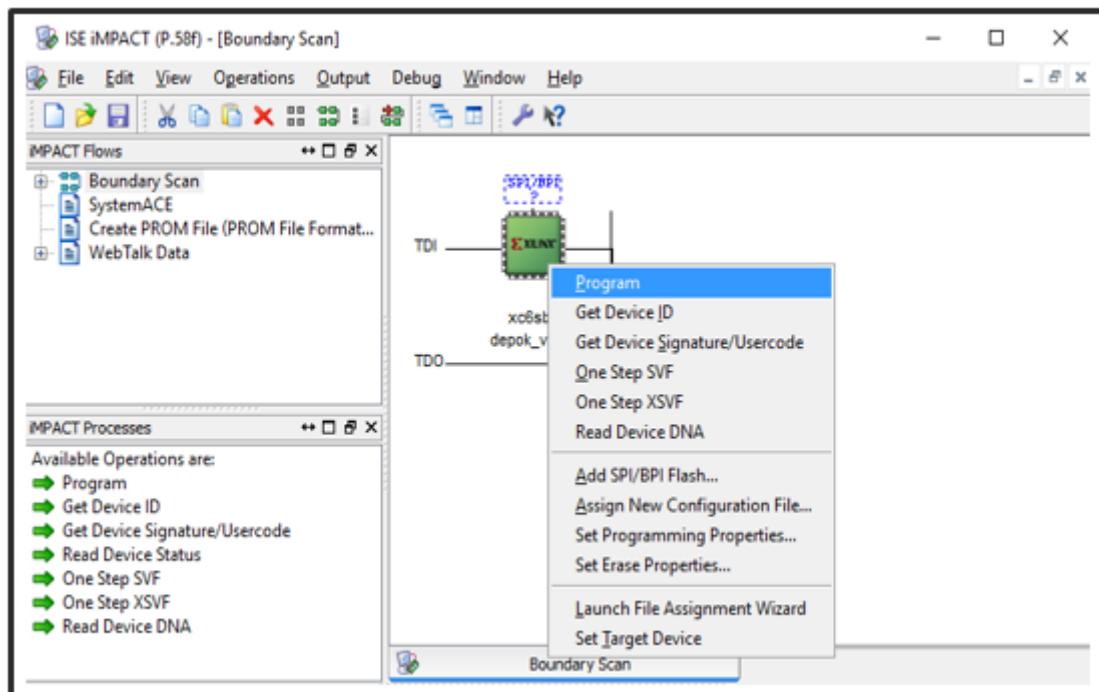


16. Jika muncul tampilan **Assign New Configuration File**, back satu kali lalu pilih file yang ber extensi .bit sesuai dengan nama project yg dibuat > klik **Open**
17. Pilih **No** pada dialog seperti gambar di bawah ini > dan Pilih **OK** :

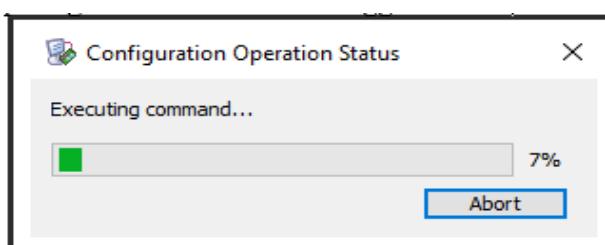




18. Klik kanan di bagian gambar IC FPGA dan Pilih **Program**.



19. Tunggu proses seperti gambar di bawah ini hingga selesai :



20. Setelah **100%** lihat hasilnya di FPGA kalian



SOURCE CODE:



OUTPUT & KESIMPULAN:



I. Tujuan Praktikum :

- Praktikan Dapat Mengenal dan Memahami Komunikasi Serial dan UART pada FPGA
- Praktikan Dapat Merancang Program VHDL pada Pemrograman FPGA menggunakan UART
- Praktikan Dapat Memahami Penggunaan UART pada FPGA

II. Dasar Teori

- Pengenalan UART
- Merancang Desain Program Menggunakan VHDL
- Pengenalan Putty
- Aplikasi Perancangan Program VHDL Untuk Port UART

III. Peralatan

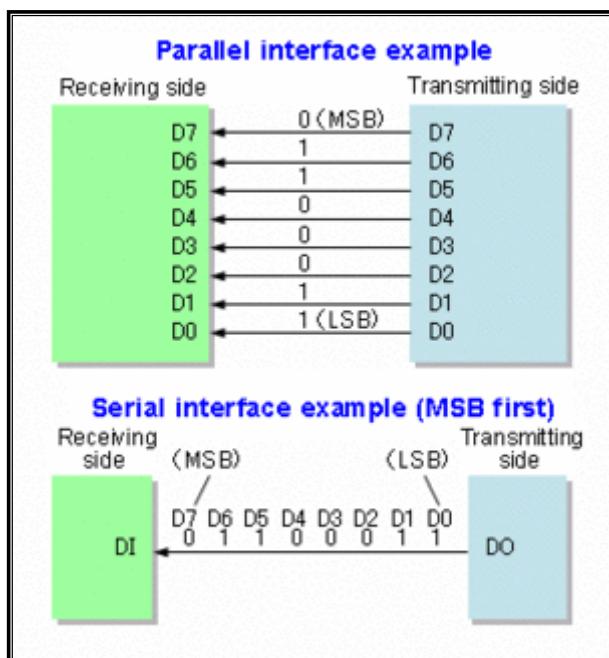
- FPGA XILINX SPARTAN 6
- Adaptor 5 Volt
- 1 buah PC
- Kabel micro USB



8.1 Komunikasi Serial

Komunikasi serial adalah proses pengiriman bit data satu per satu, secara berurutan, melalui sebuah saluran komunikasi(bus). Hal ini berbeda dengan komunikasi paralel, di mana beberapa bit dikirim secara keseluruhan, link dengan beberapa saluran paralel.

Metode komunikasi serial sering disebut sebagai TTL serial (transistor-transistor logic). Komunikasi serial pada tingkat TTL akan selalu tetap antara batas 0V dan Vcc, yang sering 5V atau 3.3V. Sebuah logika tinggi ('1') diwakili oleh Vcc, sedangkan logika rendah ('0') adalah 0V.



Gambar 8.1 Komunikasi parallel dan serial

Pada saat berkomunikasi dengan semakin tingginya frekuensi pengiriman data, semakin tinggi juga gangguan elektromagnetik. Setiap kabel dapat diperlakukan sebagai antenna, menangkap noise yang ada di sekitarnya, dan mengganggu data yang sedang ditransmisikan. Dalam komunikasi parallel, karena banyaknya kabel yang digunakan, masalah gangguan elektromagnetik menjadi lebih serius. Komunikasi serial dibutuhkan jumlah kabel yang lebih sedikit, bisa hanya menggunakan tiga kabel, yaitu saluran Transmit Data(Tx),



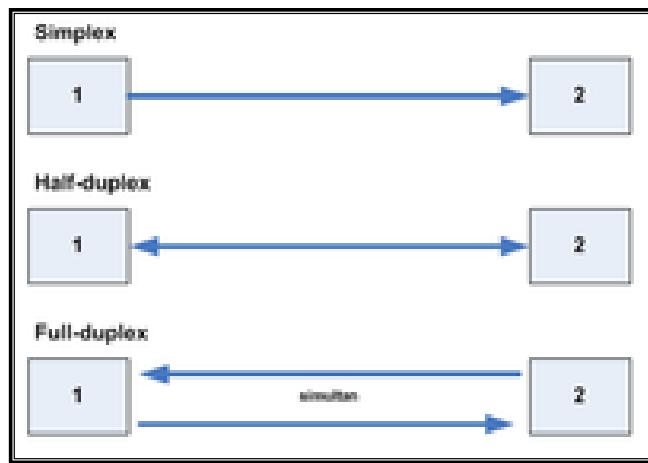
saluran Receive Data(Rx), dan saluran Ground (GND). Oleh karena itu, kabel untuk komunikasi serial juga bisa lebih panjang.

Komunikasi parallel walau datanya dikirim secara bersamaan, data yang dibaca belum tentu bersamaan. Secara teori komunikasi penggunaan komunikasi serial akan lebih lambat dibandingkan dengan parallel karena data harus dikirim satu persatu. Tetapi ini tidak sepenuhnya benar. Kecepatan komunikasi serial dapat saja dipercepat dengan mudah tanpa menangkap banyak noise yang menyebabkan data error.

8.1.1 Jenis-Jenis Komunikasi Serial

- **Synchronous (sinkron)** adalah kondisi pengiriman data serial yang disertai dengan pengiriman detak (clock) sebagai pengatur waktu untuk mengindikasi bahwa ada bit siap untuk dibaca. Contoh: I2C, USART, SPI, PS/2, dll.
- **Asynchronous (asinkron)** adalah kondisi dimana detak tidak dikirim bersamaan dengan data serial sehingga masing-masing perangkat keras yang berkomunikasi harus menciptakan detaknya sendiri yang sama. Contoh: UART, RS-232, USB, dll.
- **Full duplex** adalah jenis komunikasi serial yang menyatakan hubungan antara dua perangkat keras, A dan B. Jika A sedang melakukan pengiriman data, pada saat yang sama, A dapat menerima data dari B, dan sebaliknya. Kondisi ini dinamakan full duplex atau komunikasi dua arah. Contohnya, telepon.
- **Half duplex** merupakan kondisi ketika proses pengiriman dan penerimaan data tidak dapat dilakukan secara bersamaan seperti pada full duplex namun dilakukan secara bergantian. Contohnya, pesawat intercom dan walkie talkie.
- **Simplex** jenis merupakan jenis komunikasi dimana pengiriman hanya terjadi satu arah saja kepada penerima. Contohnya seperti membroadcast sebuah pesan.





Gambar 8.2 Jenis komunikasi berdasarkan jalur

Baud rate secara definisi merupakan kecepatan terjadinya perubahan sinyal dalam satu detik. Ini dapat diartikan bahwa baud rate adalah kecepatan pengiriman data dalam berkomunikasi. Dalam sistem digital menggunakan kode biner, 1 baud = 1bit/s. Akan tetapi dalam sistem analog, 1 baud merupakan terjadinya perubahan data yang dapat bervariasi pada jangkauan data tersebut.

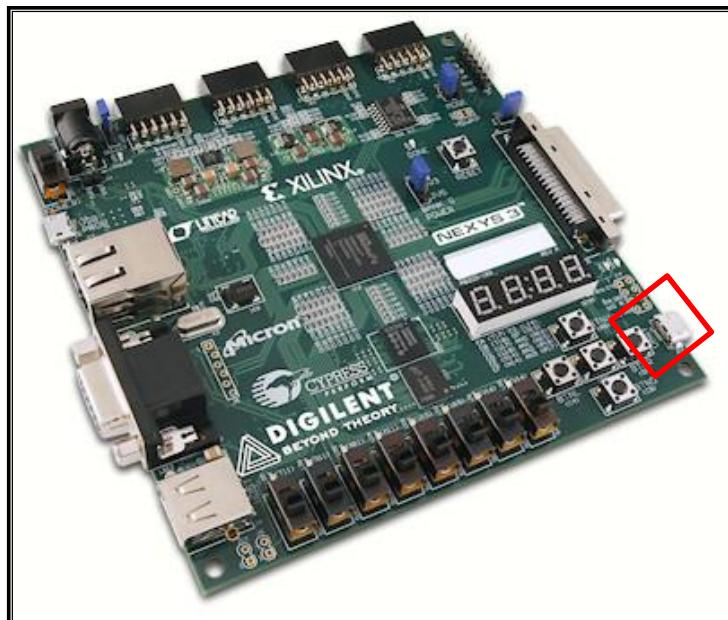
Pada komunikasi serial terdapat baud rate yang sudah distandarisasi seperti 2400, 4800, 9600, 14400, 19200, dst. Akan tetapi pengaturan baud rate tidak dapat sembarang diubah dikarenakan besarnya error rate yang mungkin terjadi. Pengiriman data error dapat saja terjadi dikarenakan clock yang tidak sesuai pada perangkat dengan baud rate yang diinginkan.

$$\%Error = \frac{BAUDactual - BAUDawal}{BAUDawal} \cdot 100\%$$

8.2 UART

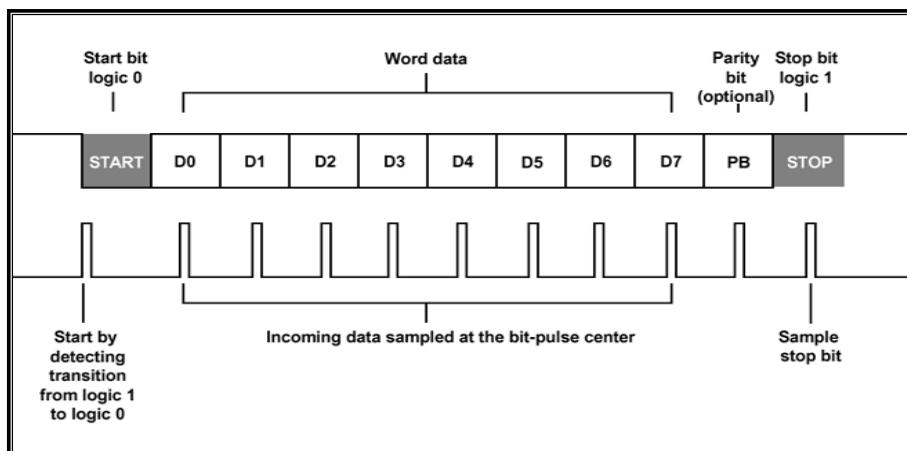
UART atau Universal Asynchronous Receiver Transmitter adalah protokol komunikasi serial yang umum digunakan dalam pengiriman data serial antara device satu dengan yang lainnya. Sebagai contoh komunikasi antara sesama mikrokontroler atau mikrokontroler ke PC.





Gambar 8.3 Port Uart pada FPGA Board Xilinx Spartan 6

Dalam pengiriman data menggunakan UART, baud rate antara pengirim dan penerima harus sama karena paket data dikirim tiap bit mengandalkan clock tersebut. Inilah salah satu keuntungan model asynchronous dalam pengiriman data karena dengan hanya satu kabel transmisi maka data dapat dikirimkan. Berbeda dengan model synchronous yang terdapat pada protokol SPI dan I2C karena protokol membutuhkan minimal dua kabel dalam transmisi data, yaitu transmisi clock dan data. Namun kelemahan model asynchronous adalah dalam hal kecepatannya dan jarak transmisi. Karena semakin cepat dan jauhnya jarak transmisi membuat paket-paket bit data menjadi terdistorsi sehingga data yang dikirim atau diterima bisa mengalami error.



Gambar 8.4 Format data komunikasi serial



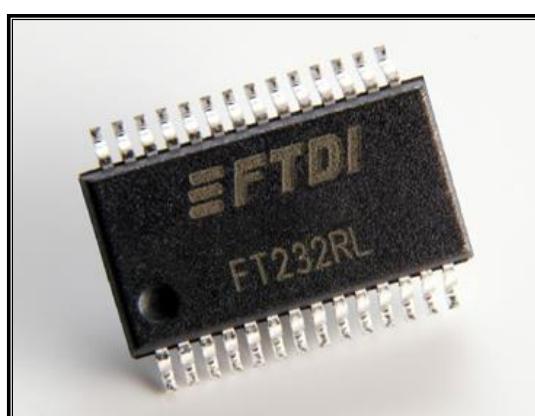
8.3 Cara Kerja UART

Sama seperti komunikasi serial lainnya, digunakan bit Start yang bernilai logic 0 sebagai tanda kepada penerima bahwa ada data yang akan dikirim. Bit individu dari data yang dikirim dimulai dari yang terkecil pertama(LSB). Setiap bit dalam transmisi ditransmisikan serupa dengan jumlah bit lainnya, dan penerima mendeteksi jalur di sekitar pertengahan periode setiap bit untuk menentukan apakah bit adalah 1 atau 0. Misalnya, jika dibutuhkan dua detik untuk mengirim setiap bit, penerima akan memeriksa sinyal untuk menentukan apakah itu adalah 1 atau 0 setelah satu detik telah berlalu, maka akan menunggu dua detik dan kemudian memeriksa nilai bit berikutnya , dan seterusnya.

UART memiliki tugas mengubah data yang diterima dari komputer melewati sirkuit paralel menjadi bit stream serial untuk dikirimkan ke perangkat keras, dan sebaliknya. UART juga berfungsi menambahkan bit parity untuk melindungi data dari kesalahan, menambahkan start bit dan stop bit pada waktu pengiriman data, serta menangani interrupt dari perangkat keras.

8.4 FTDI Chip

Komunikasi UART tidak bisa diterapkan melalui jalur USB secara langsung tanpa menggunakan sebuah modul FTDI. Modul ini mengelola transaksi data dengan menerapkan protokol USB dari data yang ditransfer melalui serial asinkron. Tanggung jawab perangkat ini adalah untuk memberitahu PC anda perangkat yang digunakan dengan menambahkan beberapa informasi identifikasi, sehingga komputer Anda dapat memuat driver yang tepat.



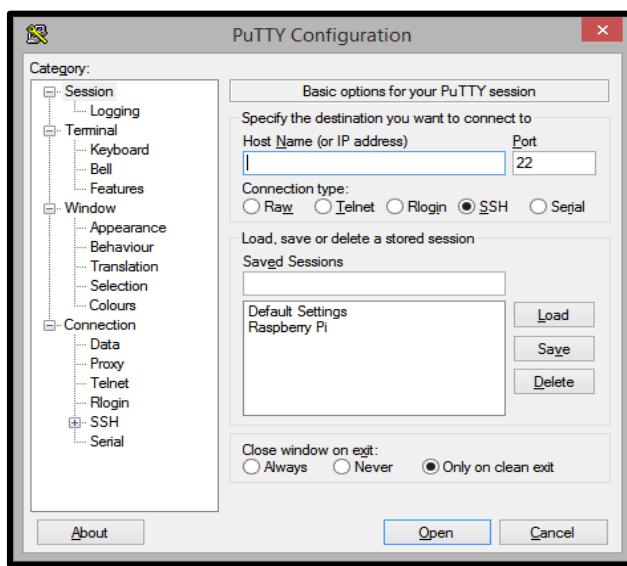
Gambar 8.5 Chip FTDI FT232RL



8.5 PuTTY

PuTTY adalah sebuah program open source yang dapat Anda gunakan untuk melakukan protokol jaringan SSH, Telnet, Serial dan Rlogin. Protokol ini dapat digunakan untuk menjalankan sesi remote pada sebuah komputer melalui sebuah jaringan, baik itu LAN, maupun internet. PuTTY pada awalnya dibuat untuk Microsoft Windows, tetapi telah mendukung berbagai sistem operasi lainnya.

Aplikasi PuTTY digunakan ketika anda ingin mentransfer sebuah data dari komputer ke sebuah perangkat lain dan fungsi bagi penggunaanya dapat menerima data. Program ini banyak digunakan oleh para pengguna komputer tingkat menengah ke atas, yang biasanya digunakan untuk menyambungkan, mensimulasi, atau mencoba berbagai hal yang terkait dengan jaringan. PuTTY juga dapat Anda gunakan sebagai tunnel di suatu jaringan.



Gambar 8.6 Tampilan Awal PuTTY

8.6 Kode ASCII

Kode ASCII (American Standard Codes for International Interchange) adalah kumpulan kode – kode yang digunakan untuk mempermudah interaksi antara user dan komputer. ‘Interaksi’ yang dimaksud adalah sarana untuk menyelesaikan permasalahan hubungan antara komputer yang mengenal angka, sedangkan manusia tidak mungkin harus menghafal angka yang cukup banyak tersebut dan menggunakan keyboard sebagai masukan antar perintah yang diinginkannya. Kode ASCII sebenarnya memiliki komposisi bilangan biner sebanyak 8 bit, dimulai dari 00000000 hingga 1111111. Total kombinasi yang dihasilkan sebanyak 256, dimulai dari kode 0 hingga 255 dalam sistem bilangan decimal.



Pada dasarnya kode ASCII merepresentasikan kode-kode untuk :

1. Angka (0,1,2,3,4,5,6,7,8,9)
2. Huruf (a – z , A - Z)
3. Simbol (&, ^, %, \$,)
4. Tombol (Enter, Esc, Tab,)
5. Karakter Grafis (kode ASCII Standar nomor 128 s/d 255)
6. Kode Komunikasi (ETX, STX, ENQ,...)

8.5.1 Kode Standard ASCII

Kode ini merepresentasikan angka, huruf serta tombol standar, Enter, Escape, Backspace dan Space. Selain itu juga terdapat karakter-karakter yang tidak terdapat pada keyboard, yang dapat diaktifkan dengan melakukan penekanan tombol kombinasi “Alt” dan angka yang dimaksud, sebagai contoh tombol kombinasi “Alt” dan angka “127” akan menghasilkan karakter grafis.

Karakter dasar lain juga digunakan untuk komunikasi, seperti yang Anda ketahui bersama, karakter tersebut adalah “ACK” dan “ENQ”. Pada saat akan dilakukan komunikasi pada jaringan dengan protokol Ethernet, maka bentuk komunikasi yang terjadi adalah komputer akan mengirimkan “ACK” (Acknowledge) pada komputer lain yang akan berkomunikasi, jika komputer lain merespon, maka komputer tersebut akan membalasnya dengan mengirim “ENQ” (Enquiry).

8.5.2 Kode Extended ASCII

Kode ASCII Extended akan bertindak sebagai kode perluasan (extended) dari kode ASCII yang ada, karena tidak semuanya mampu tertampung dalam kode ASCII standard. Kode ASCII jenis ini lebih banyak bertindak sebagai kode-kode tombol khusus, seperti kode untuk tombol F1 s/d F12. Sebagai contoh adalah kode ASCII extended untuk F12 adalah “123”. Belum lagi dengan tombol kombinasi, misalnya “Alt” dan “F1”, “Ctrl” dan “F1”, atau tombol-tombol yang biasa kita lakukan “Alt” + “F” untuk membuka menu file, “Ctrl” dan “O” untuk membuka dokumen dsb.



ASCII control characters		ASCII printable characters				Extended ASCII characters			
00	NULL (Null character)	32	space	64	@	96	`	128	Ç
01	SOH (Start of Header)	33	!	65	A	97	a	129	Ü
02	STX (Start of Text)	34	"	66	B	98	b	130	é
03	ETX (End of Text)	35	#	67	C	99	c	131	â
04	EOT (End of Trans.)	36	\$	68	D	100	d	132	ä
05	ENQ (Enquiry)	37	%	69	E	101	e	133	à
06	ACK (Acknowledgement)	38	&	70	F	102	f	134	à
07	BEL (Bell)	39	.	71	G	103	g	135	ç
08	BS (Backspace)	40	(72	H	104	h	136	ê
09	HT (Horizontal Tab)	41)	73	I	105	i	137	ë
10	LF (Line feed)	42	*	74	J	106	j	138	è
11	VT (Vertical Tab)	43	+	75	K	107	k	139	í
12	FF (Form feed)	44	,	76	L	108	l	140	î
13	CR (Carriage return)	45	-	77	M	109	m	141	í
14	SO (Shift Out)	46	.	78	N	110	n	142	Ä
15	SI (Shift In)	47	/	79	O	111	o	143	À
16	DLE (Data link escape)	48	0	80	P	112	p	144	É
17	DC1 (Device control 1)	49	1	81	Q	113	q	145	æ
18	DC2 (Device control 2)	50	2	82	R	114	r	146	Æ
19	DC3 (Device control 3)	51	3	83	S	115	s	147	ô
20	DC4 (Device control 4)	52	4	84	T	116	t	148	ö
21	NAK (Negative acknowl.)	53	5	85	U	117	u	149	ó
22	SYN (Synchronous idle)	54	6	86	V	118	v	150	û
23	ETB (End of trans. block)	55	7	87	W	119	w	151	ù
24	CAN (Cancel)	56	8	88	X	120	x	152	ÿ
25	EM (End of medium)	57	9	89	Y	121	y	153	Ö
26	SUB (Substitute)	58	:	90	Z	122	z	154	Ü
27	ESC (Escape)	59	;	91	[123	{	155	ø
28	FS (File separator)	60	<	92	\	124		156	£
29	GS (Group separator)	61	=	93]	125	}	157	Ø
30	RS (Record separator)	62	>	94	^	126	~	158	×
31	US (Unit separator)	63	?	95	_			159	f
127	DEL (Delete)							191	ñ

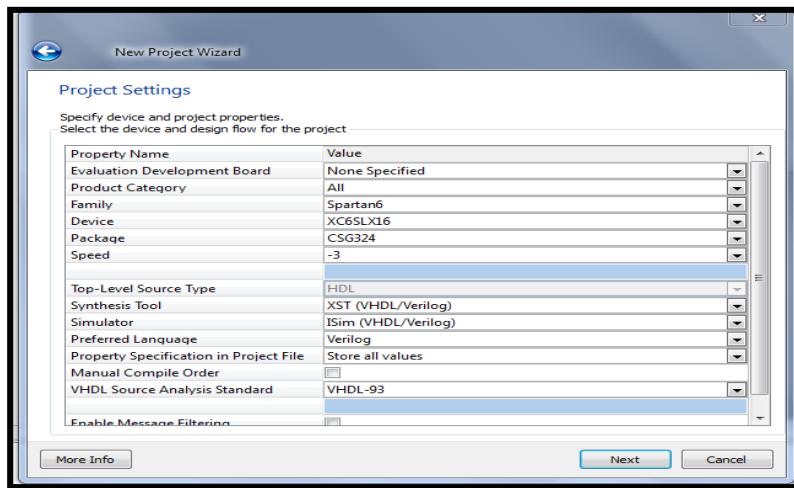
Gambar 8.5 Tabel Kode ASCII



Percobaan 1 : Transmitter

Langkah-Langkah:

1. Buka Xilinx ISE 14.5
2. Klik **File -> New Project**
3. Berilah nama pada project kalian lalu klik **next**
4. Atur Device Properties seperti gambar dibawah ini :



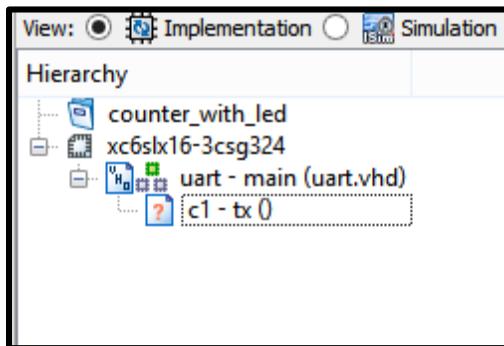
- 5 Klik **Next -> Next -> Finish**
- 6 Klik kanan pada project kalian yang ada pada tab hierarchy, pilih **new source** -> ketik nama file namanya **UART** pilih **VHDL Module** -> klik **next**
- 7 Copy source code yang diberikan Asisten
- 8 Lengkapi source code seperti dibawah ini lalu **Save**:

```
0 component tx port(
1   clk:in std_logic;
2   start: in std_logic;
3   busy: out std_logic;
4   data : in std_logic_vector(7 downto 0);
5   tx_line: out std_logic
6 );
7 end component tx;
8
9 begin
10
11 c1: tx port map(clk,tx_start,tx_busy, tx_data,uart_tx);
12
13
14 process(clk)
15 begin
16   if(clk'event and clk='1') then
17     send_edge<=send_edge(0) & send;
18     if(send_edge="01" and tx_busy='0') then
19       tx_data<=sw(7 downto 0);
20       tx_start<='1';
21     else
22       tx_start<='0';
23     end if;
24   end if;
25 end process;
26
27 end main;
```

- 9 Lalu **Save**



- 10 Pada tab Hierarchy sekali lagi pilih **new source** -> ketik nama file namanya tx pilih **VHDL Module** -> klik next



- 11 Pada file .vhd yang baru lengkapi source code seperti dibawah ini lalu **Save**:

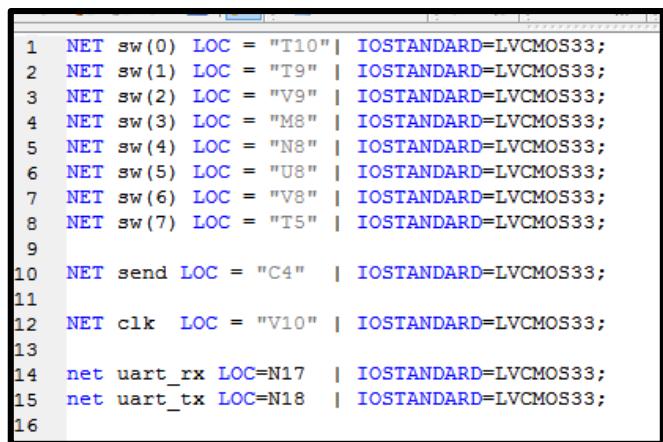
```

22 process(clk) begin
23 if(clk'event and clk='1') then
24   if(tx_flg='0' and start ='1') then
25     tx_flg<='1';
26     busy<='1';
27     datafill(9 downto 0) <= '1' & data & '0'; --stop bit & 8bit data & start bit
28   end if;
29   if(tx_flg='1') then
30     if(prscl<5207) then
31       prscl<=prscl+1;
32     else
33       prscl<=0;
34     end if;
35     if (prscl=2607) then
36       tx_line<=datafill(index);
37       if(index<9) then
38         index<=index+1;
39       else
40         tx_flg<='0';
41         busy<='0';
42         index<=0;
43       end if;
44     end if;
45   end if;
46 end if;

```

- 12 Pada bagian tab processes pilih **Synthesize -XST** klik tanda +klik kanan **CheckSyntax** pilih run atau cukup klik 2x pada **Synthesize -XST**
- 13 Maka selanjutnya klink kanan pada subname yang diberi nama **UART**-> pilih newsource, ketik nama file namanya **Counter** pilih **Implementation Contrains File** -> klik next. (**Menggunakan UCF**)



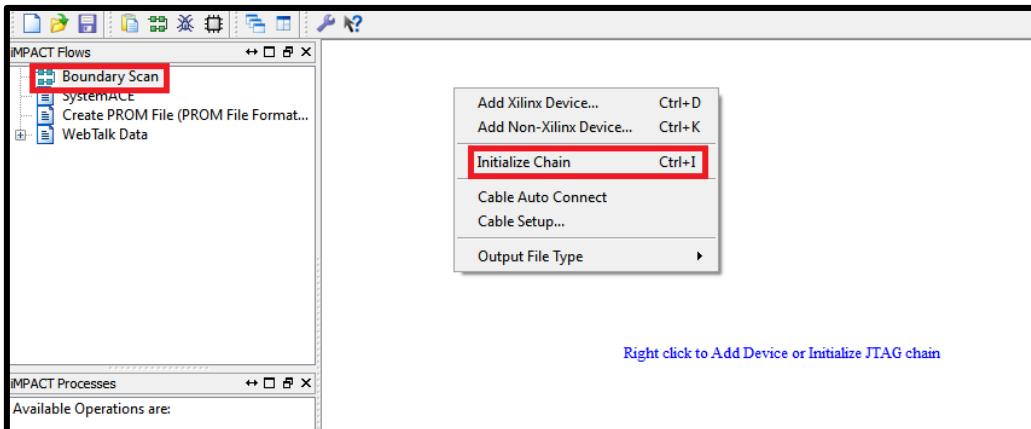


```

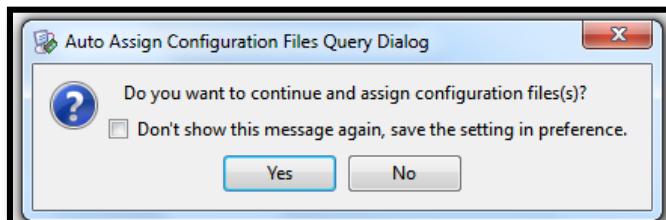
1  NET sw(0) LOC = "T10" | IOSTANDARD=LVCMOS33;
2  NET sw(1) LOC = "T9" | IOSTANDARD=LVCMOS33;
3  NET sw(2) LOC = "V9" | IOSTANDARD=LVCMOS33;
4  NET sw(3) LOC = "M8" | IOSTANDARD=LVCMOS33;
5  NET sw(4) LOC = "N8" | IOSTANDARD=LVCMOS33;
6  NET sw(5) LOC = "U8" | IOSTANDARD=LVCMOS33;
7  NET sw(6) LOC = "V8" | IOSTANDARD=LVCMOS33;
8  NET sw(7) LOC = "T5" | IOSTANDARD=LVCMOS33;
9
10 NET send LOC = "C4" | IOSTANDARD=LVCMOS33;
11
12 NET clk LOC = "V10" | IOSTANDARD=LVCMOS33;
13
14 net uart_rx LOC=N17 | IOSTANDARD=LVCMOS33;
15 net uart_tx LOC=N18 | IOSTANDARD=LVCMOS33;
16

```

- 14 Sebelum masuk text editor, klik tanda + pada project gerbangAnd, di hierarchy -> double klik **Counter.ucf**. Pada text editor ketiklah pin FPGA seperti dibawah ini : **(Setelah selesai jangan lupa untuk save)**
- 15 Selanjutnya klik 2x **Implement Desain** (**Sebelum klik 2x Implement Desain, jangan lupa untuk menghubungkan dengan FPGA terlebih dahulu**).
- 16 Lalu klik tanda + pada **Configure Target Device**, Pilih **Manage Configuration Project**
- 17 Pilih boundary scan, lalu klik kanan pada area “right click to add device or initialize JTAG chain”, kemudian pilih **Initialize Chain** seperti gambar di bawah ini :



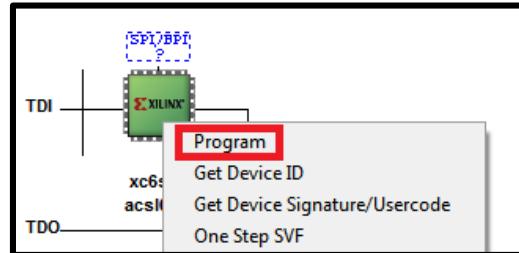
- 18 Pilih Yes pada **Auto Assign Configuration Files Query Dialog**



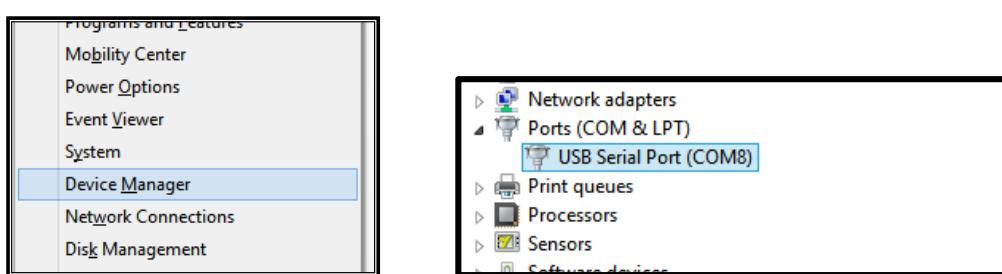
- 19 Pilih file **UART.bit** pada **Assign New Configuration** file lalu pilih open > **UART.bit**



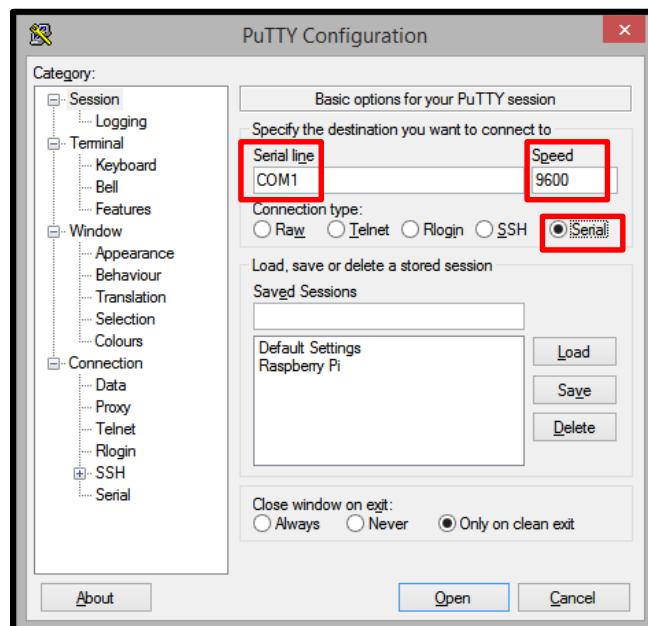
- 20 Pada tampilan pop up selanjutnya pilih Cancel -> kemudian pilih OK, maka akan muncul tampilan seperti gambar dibawah ini, setelah itu klik kanan pada gambar IC -> klik **Program**



- 21 Hubungkan port UART pada FPGA kalian menggunakan sebuah kabel micro USB.
- 22 Buka **Device Manager** untuk memastikan jika sudah terhubung. Tekan **Windows + x** -> klik **Device Manager**
- 23 Check port komunikasi FPGA kalian pada **Ports (COM & LPT)**



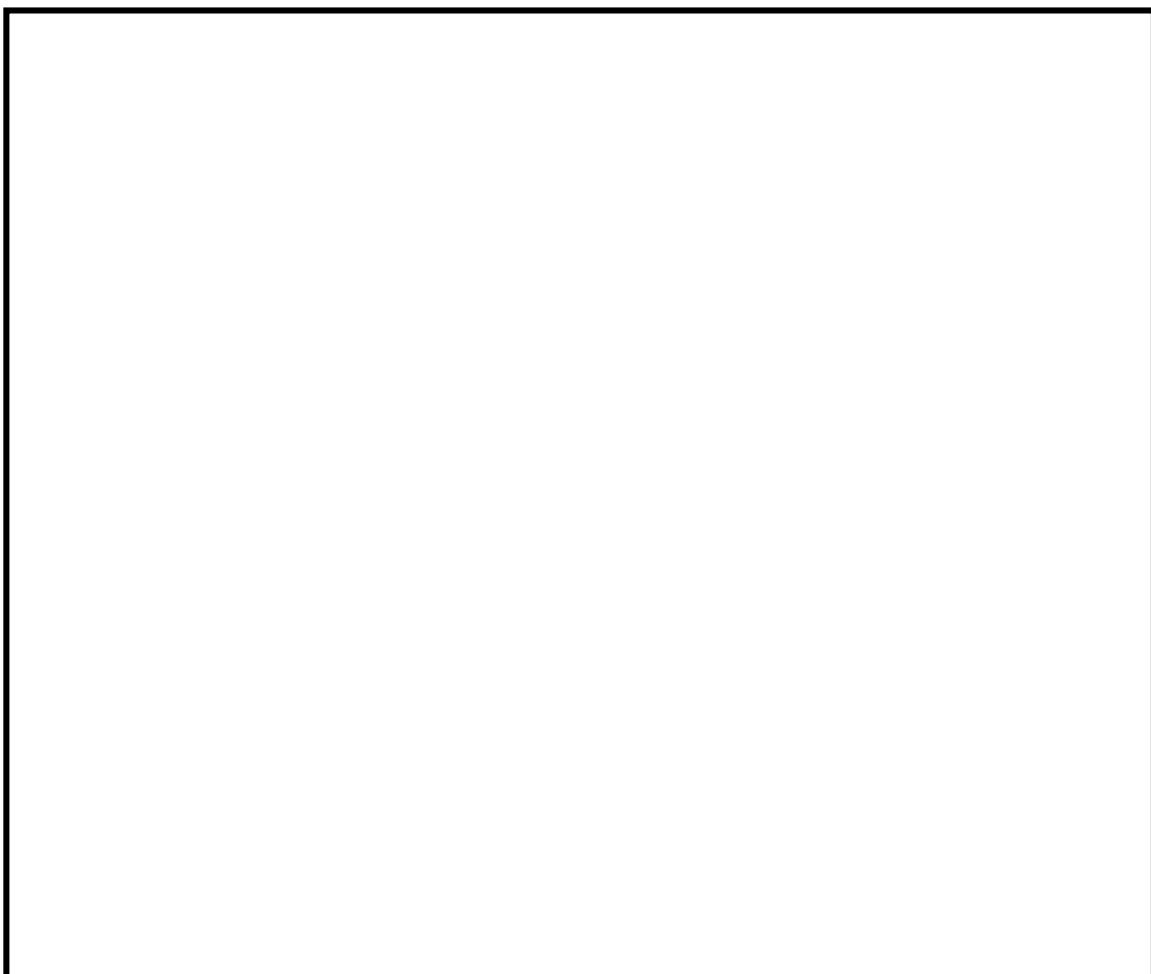
- 24 Buka aplikasi **PuTTY** -> klik **Serial**



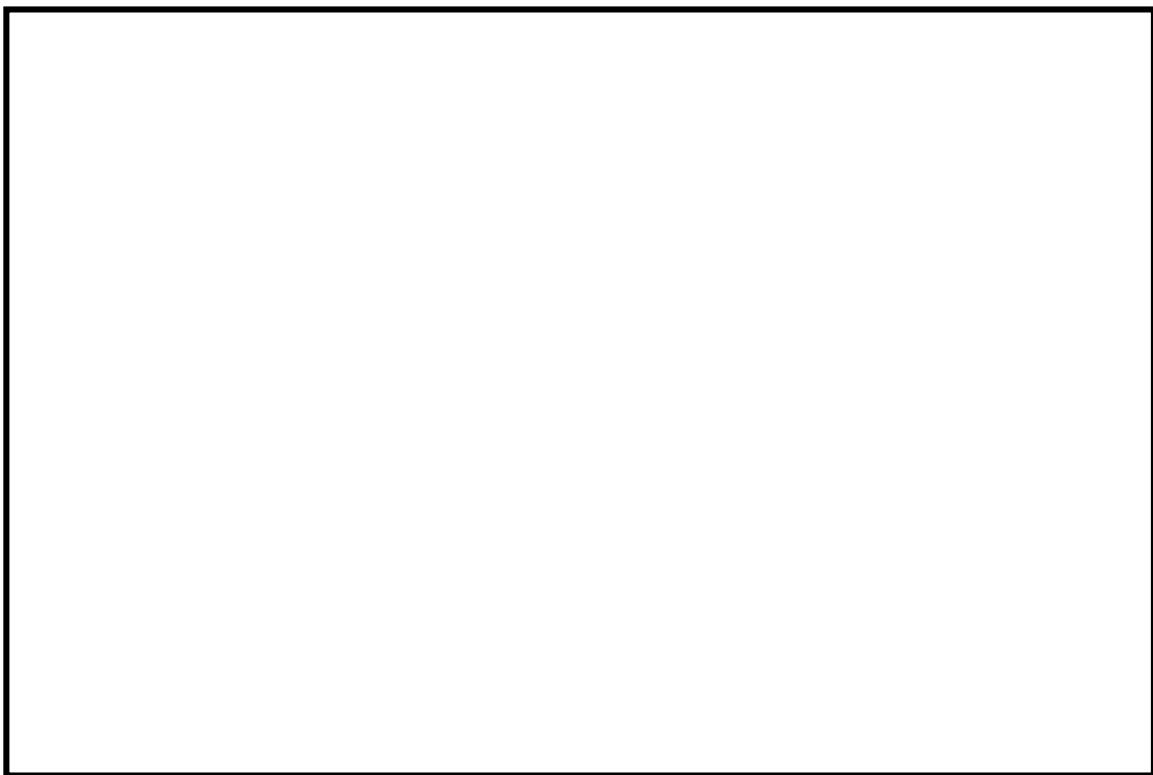
- 25 Ubah Serial Line sesuai pada port komunikasi dari Device Manager dan Speed sesuai Baud Rate -> klik **Open**.



SOURCE CODE:



OUTPUT & KESIMPULAN:



Percobaan Mandiri 1 :

Hitunglah prescaler terdekat untuk mengubah Buad Rate menjadi 9600 bps dan ubahlah pada program.

SOURCE CODE:

HASIL OUTPUT & KESIMPULAN:



Percobaan 2 : Menambahkan receiver untuk komunikasi duplex

Langkah-Langkah:

1. Ubahlah nilai prescaler seperti sebelumnya.
2. Pada source code yang kalian buat tambahkan source code seperti dibawah ini lalu **Save**:



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity uart is PORT(
    clk: in std_logic;
    sw: in std_logic_vector(7 downto 0);
    send: in std_logic;
    led: out std_logic_vector(7 downto 0);
    uart_tx: out std_logic; --N18
    uart_rx: in std_logic --N17
);
end uart;
architecture main of uart is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            send<=send_edge(0) & send;
            if(send_edge="01" and tx_busy='0') then
                tx_data<=sw(7 downto 0);
                tx_start<='1';
            else
                tx_start<='0';
            end if;
        end if;
    end process;
end main;

```

HDL Hardware Description Language file

```

begin
    c1: tx port map(clk, tx_start, tx_busy, tx_data,uart_tx);
    c2: rx port map(clk, uart_rx, rx_data, rx_busy);

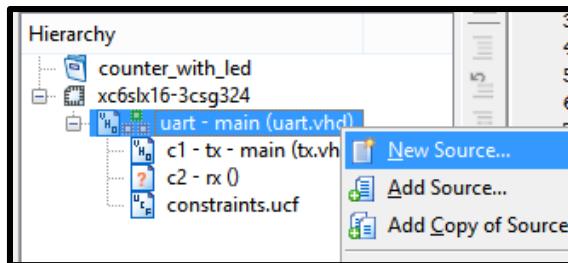
    process(rx_busy)
    begin
        if(rx_busy'event and rx_busy='0') then
            led<=rx_data;
        end if;
    end process;

    process(clk)
    begin
        if(clk'event and clk='1') then
            send_edge<=send_edge(0) & send;
            if(send_edge="01" and tx_busy='0') then
                tx_data<=sw(7 downto 0);
                tx_start<='1';
            else
                tx_start<='0';
            end if;
        end if;
    end process;
end main;

```

3. Pada tab Hierarchy sekali lagi pilih **new source** -> ketik nama file namanya **rx** pilih **VHDL Module** -> klik **next**





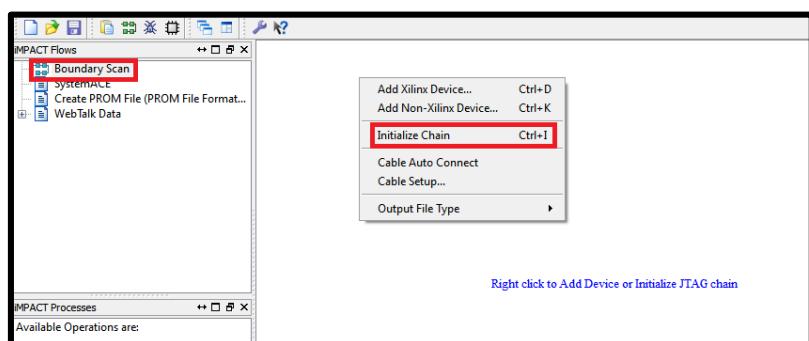
4. Copy source code untuk file rx.vhd yang diberikan oleh asisten.
5. Pada bagian tab processes pilih **Synthesize –XST** klik tanda +klik kanan **CheckSyntax** pilih run atau cukup klik 2x pada **Synthesize –XST**
6. Buka file **Implementation Contrains File** yang telah kalian buat dan tambahkan pin untuk led 8 bit.

```

1  NET led(0) LOC = "U16" | IOSTANDARD=LVC MOS33;
2  NET led(1) LOC = "V16" | IOSTANDARD=LVC MOS33;
3  NET led(2) LOC = "U15" | IOSTANDARD=LVC MOS33;
4  NET led(3) LOC = "V15" | IOSTANDARD=LVC MOS33;
5  NET led(4) LOC = "M11" | IOSTANDARD=LVC MOS33;
6  NET led(5) LOC = "N11" | IOSTANDARD=LVC MOS33;
7  NET led(6) LOC = "R11" | IOSTANDARD=LVC MOS33;
8  NET led(7) LOC = "T11" | IOSTANDARD=LVC MOS33;
9

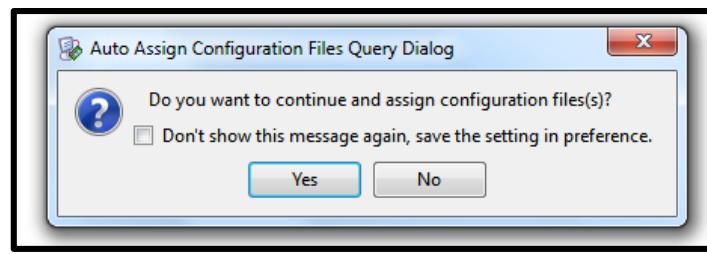
```

7. Selanjutnya klik 2x **Implement Desain** (Sebelum klik 2x Implement Desain, jangan lupa untuk menghubungkan dengan FPGA terlebih dahulu).
8. Lalu klik tanda + pada **Configure Target Device**, Pilih **Manage Configuration Project**
9. Pilih boundary scan, lalu klik kanan pada area “right click to add device or initialize JTAG chain”, kemudian pilih **Initialize Chain** seperti gambar di bawah ini :

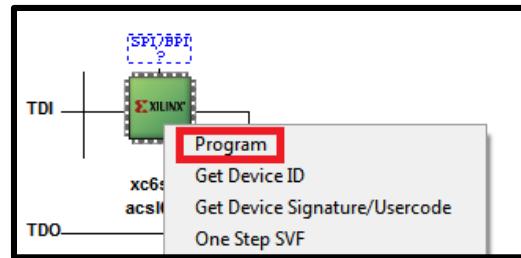


11. Pilih Yes pada Auto Assign Configuration Files Query Dialog

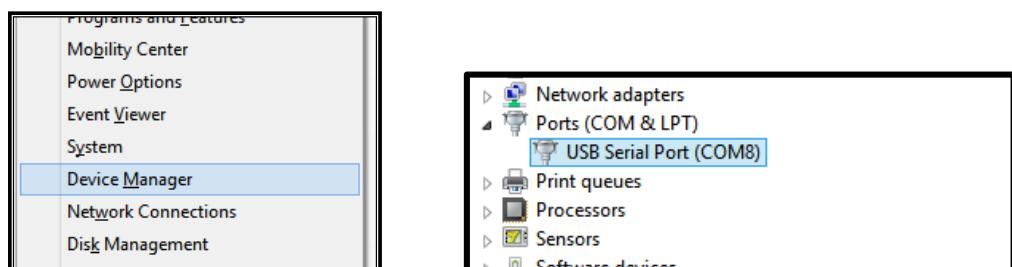




12. Pilih file **UART.bit** pada **Assign New Configuration** file lalu pilih open > **UART.bit**
13. Pada tampilan pop up selanjutnya pilih Cancel -> kemudian pilih OK, maka akan muncul tampilan seperti gambar dibawah ini, setelah itu klik kanan pada gambar IC - > klik **Program**

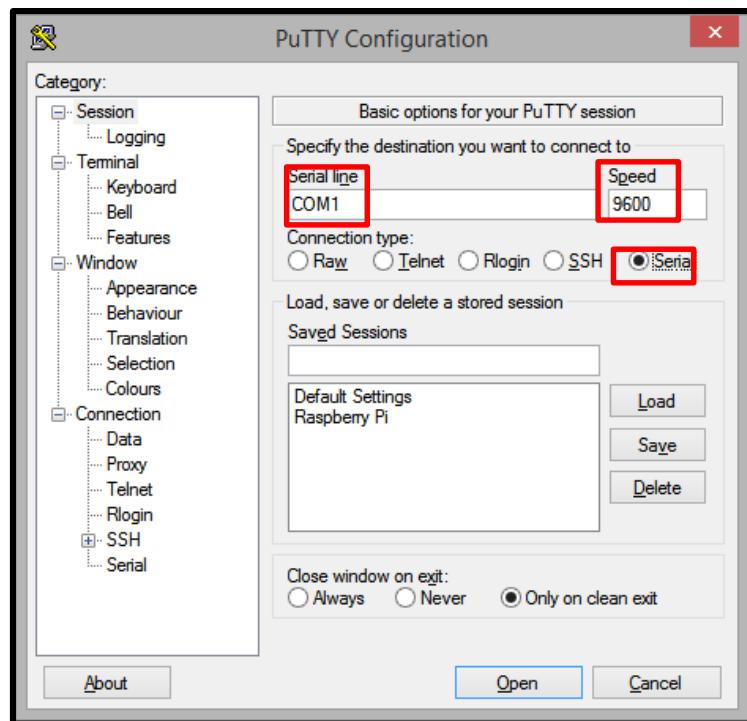


14. Hubungkan port UART pada FPGA dengan PC kalian menggunakan sebuah kabel micro USB.
15. Buka **Device Manager** untuk memastikan jika sudah terhubung. Tekan **Windows + x** -> klik **Device Manager**
16. Check port komunikasi FPGA kalian pada **Ports (COM & LPT)**



17. Buka aplikasi **PuTTY** -> klik **Serial**

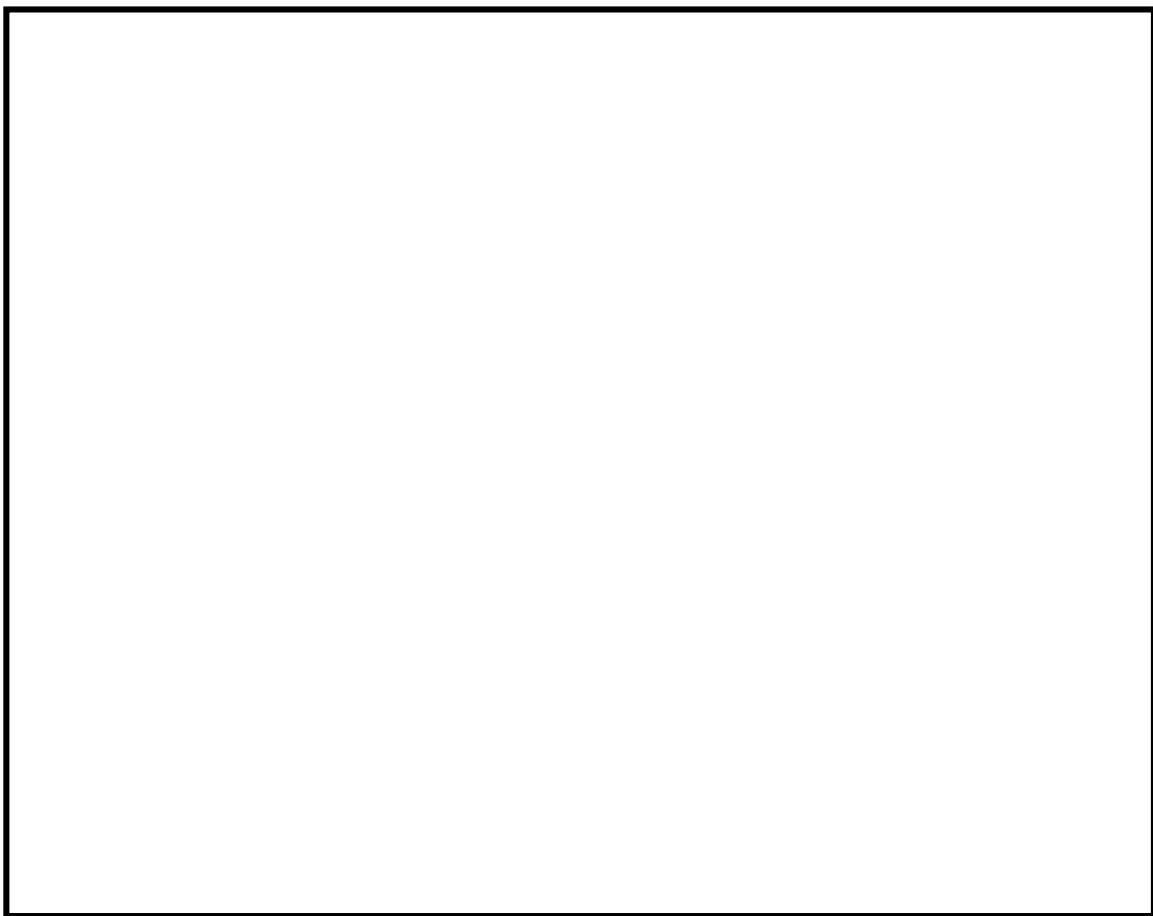




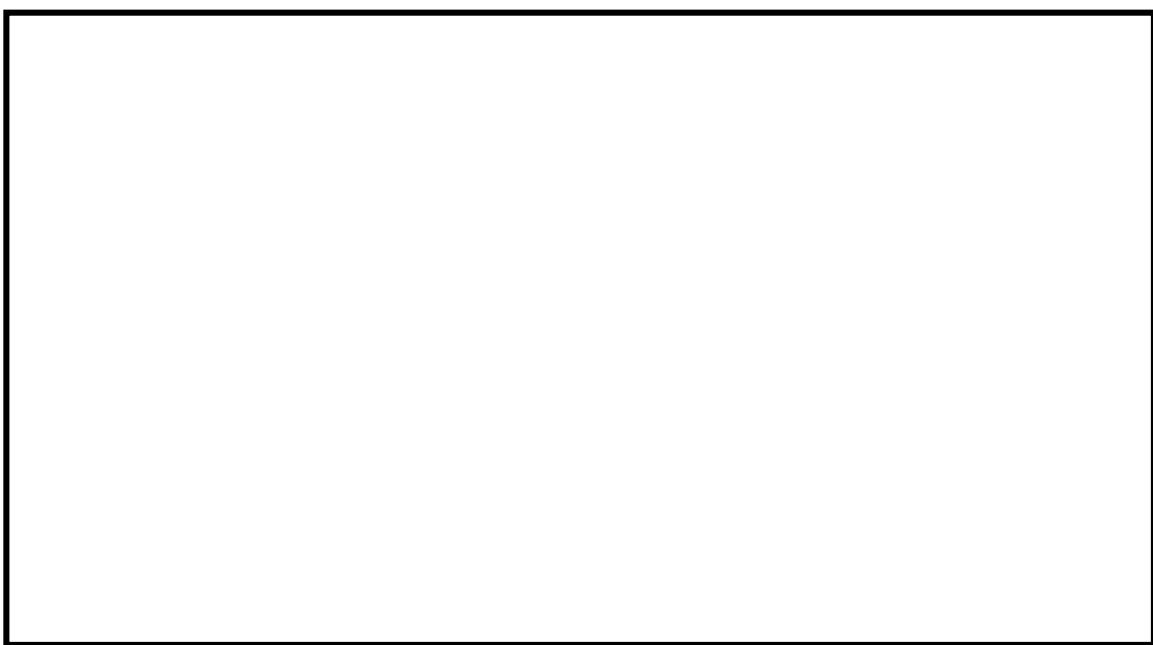
18. Ubah Serial Line sesuai pada port komunikasi dari Device Manager dan Speed sesuai Baud Rate -> Klik **Open**.



SOURCE CODE:



HASIL OUTPUT & KESIMPULAN:



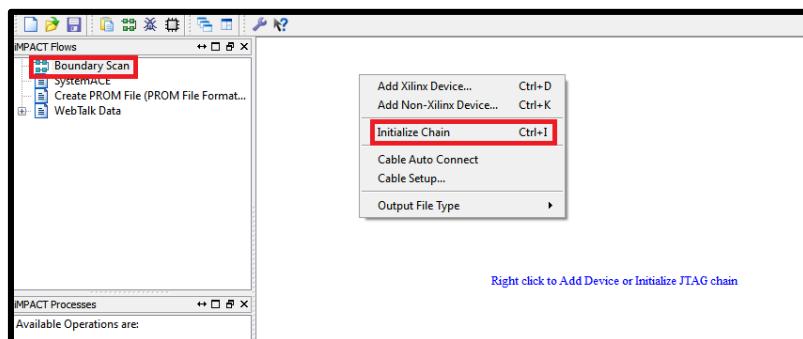
Percobaan 3 : Membuat interface menggunakan VB .NET

Langkah-Langkah:

1. Gunakan project pada percobaan sebelumnya dan ubah source code **uart.vhd** untuk menampilkan data receive seperti dibawah ini:

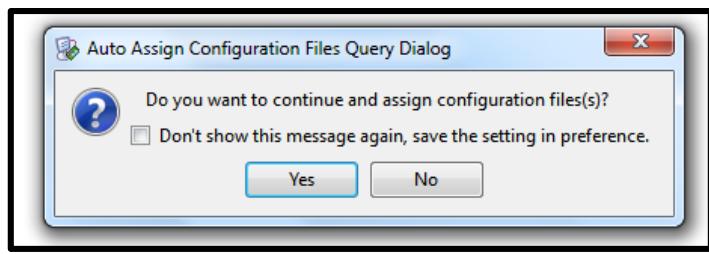
```
41
42  c1: tx port map(clk, tx_start, tx_busy, tx_data,uart_tx);
43  c2: rx port map(clk, uart_rx, rx_data, rx_busy);
44
45  process(rx_busy)
46  begin
47    if(rx_busy'event and rx_busy='0') then
48      case rx_data is
49        when x"31" => led(0)<='1';
50        when x"32" => led(1)<='1';
51        when x"33" => led(2)<='1';
52        when x"34" => led(3)<='1';
53        when x"35" => led(0)<='0';
54        when x"36" => led(1)<='0';
55        when x"37" => led(2)<='0';
56        when x"38" => led(3)<='0';
57        when others => led <= x"00";
58      end case;
59    end if;
60  end process;
61
```

2. Selanjutnya klik 2x **Implement Desain** (Sebelum klik 2x Implement Desain, jangan lupa untuk menghubungkan dengan FPGA terlebih dahulu).
3. Lalu klik tanda + pada **Configure Target Device**, Pilih **Manage Configuration Project**
4. Pilih boundary scan, lalu klik kanan pada area “right click to add device or initialize JTAG chain”, kemudian pilih **Initialize Chain** seperti gambar di bawah ini :

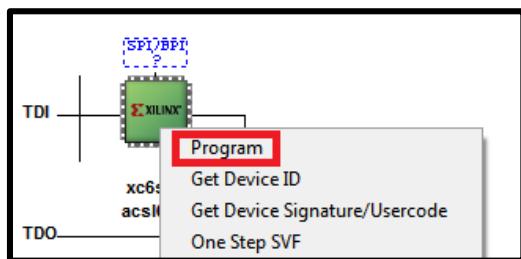


5. Pilih Yes pada **Auto Assign Configuration Files Query Dialog**

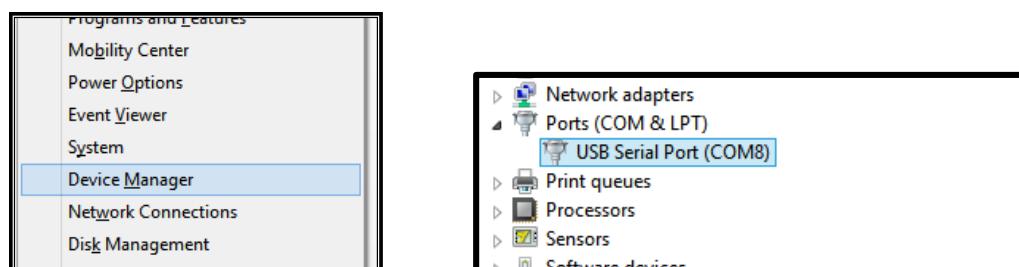




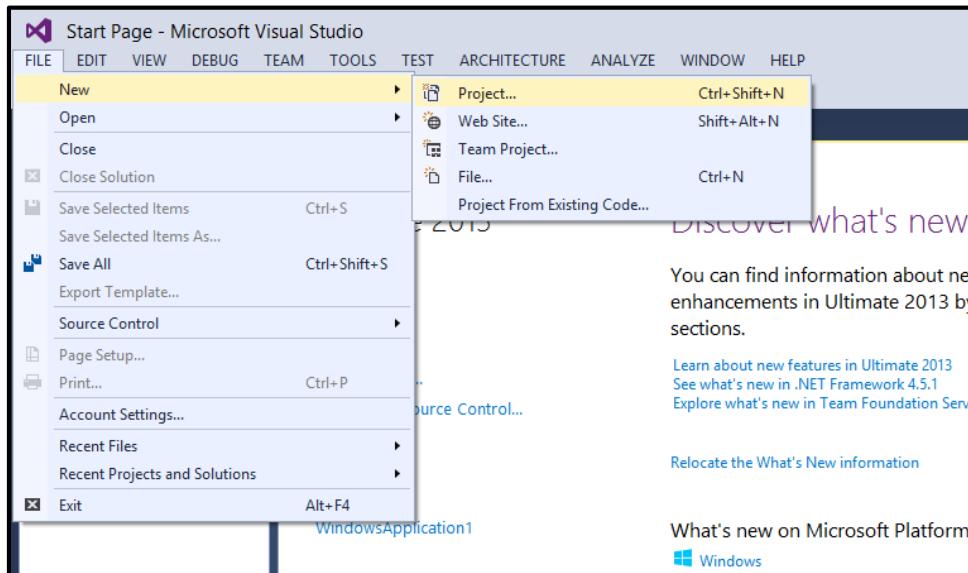
6. Pilih file **UART.bit** pada **Assign New Configuration** file lalu pilih open > **UART.bit**
7. Pada tampilan pop up selanjutnya pilih Cancel -> kemudian pilih OK, maka akan muncul tampilan seperti gambar dibawah ini, setelah itu klik kanan pada gambar IC - > klik **Program**



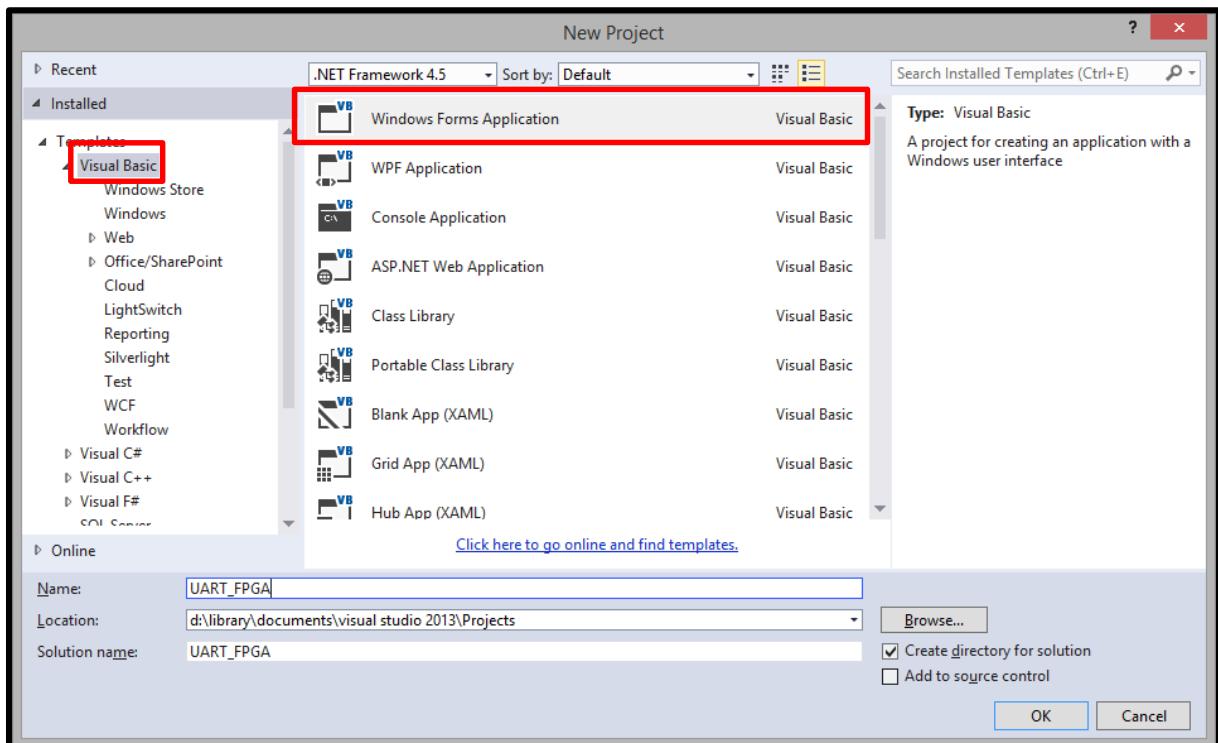
8. Hubungkan port UART pada FPGA dengan PC kalian menggunakan sebuah kabel micro USB.
9. Buka **Device Manager** untuk memastikan jika sudah terhubung. Tekan **Windows + x** -> klik **Device Manager**
10. Check port komunikasi FPGA kalian pada **Ports (COM & LPT)**



11. Buka Aplikasi **Visual Studio** -> pilih menu **File** pada toolbar -> **New -> Project...**



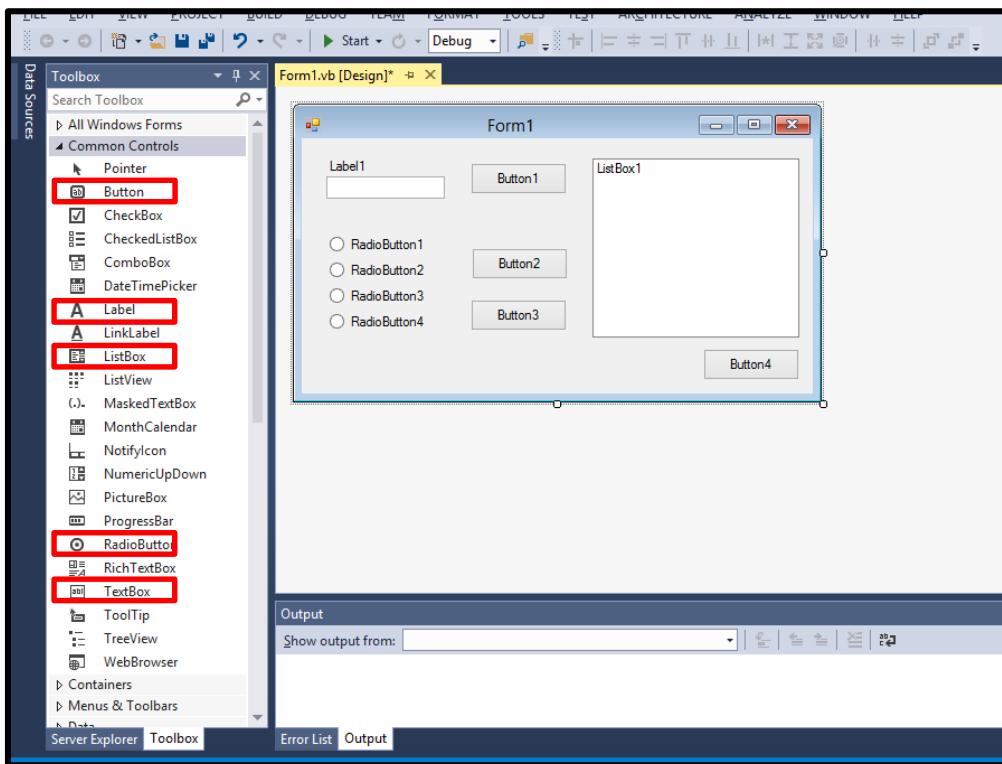
12. Pilih **Visual Basic** pada pilihan templates -> pilih **Windows Forms Application** -> lalu berilah nama untuk project kalian -> klik **OK**.



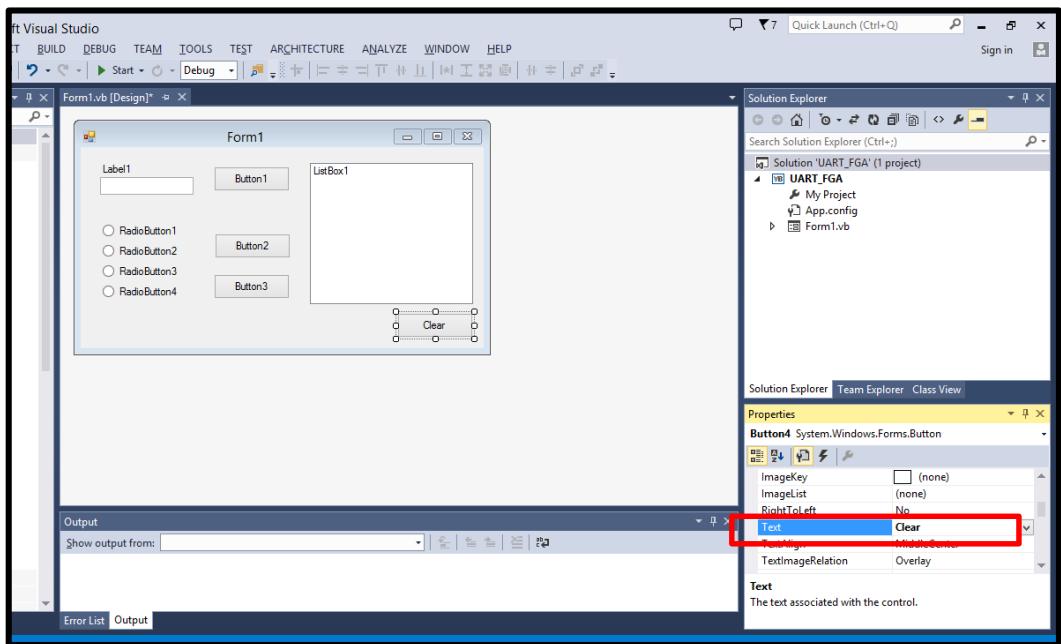
13. Pada tab **Toolbox**, klik > pada **Common Controls**.



14. Gunakan **Button**, **Label**, **TextBox**, **ListBox**, dan **RadioButton** untuk membuat tampilan seperti dibawah ini:

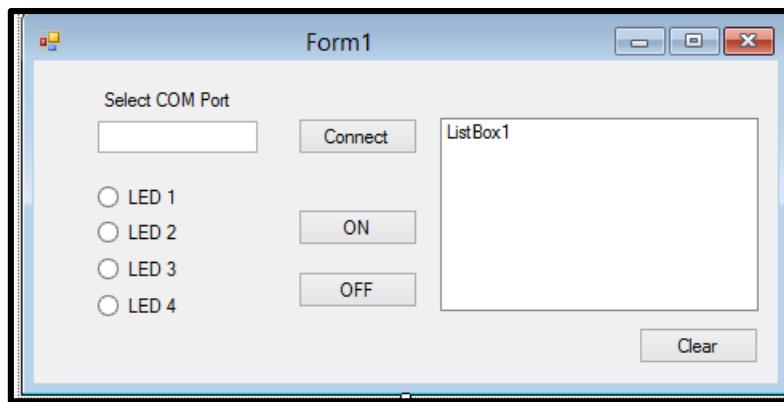


15. Klik pada **Button** yang sudah ada pada form -> Ubah **Text** pada tab **Properties**

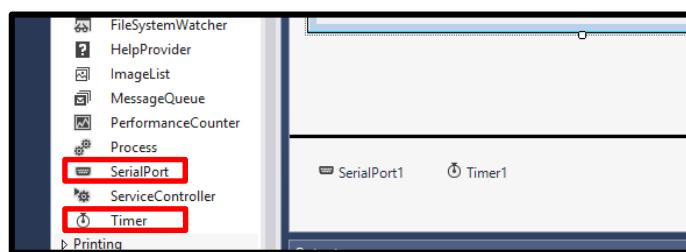


16. Ubah text untuk tool lainnya agar tampak seperti gambar dibawah ini:

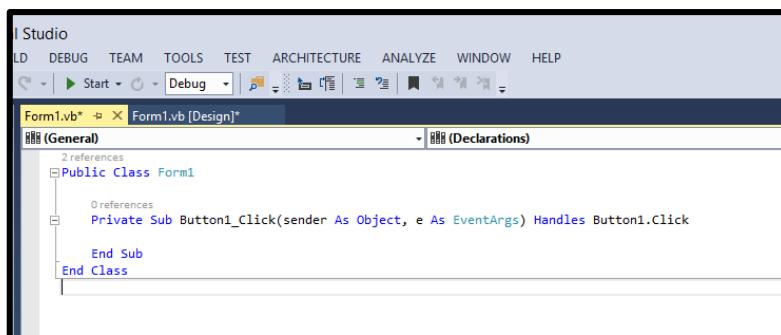




17. Pada tab **Toolbox**, klik > pada **Components**.
18. Tambahkan **SerialPort** dan **Timer** pada Form1.



19. Klik pada **Timer1** yang telah dibuat -> Ubah **Interval** menjadi 100 pada tab **Properties**
20. Selanjutnya klik 2x tool pada Form1 untuk menyisipkan program kedalamnya.



21. Lengkapi program seperti berikut:

Button CLEAR

```

0 references
Private Sub Button4_Click(sender As Object, e As EventArgs) Handles Button4.Click
    ListBox1.Items.Clear()
End Sub

```



Timer 1

```
0 references
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
    Dim Incoming = SerialPort1.ReadExisting()
    If Incoming <> Nothing Then
        ListBox1.Items.Add("Incoming Data= " + Incoming)
    End If
End Sub
```

Button Connect

```
0 references
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If (Button1.Text = "Connect") Then
        If (TextBox1.Text <> "") Then
            ListBox1.Items.Add("Connecting...")
            SerialPort1.Close()
            SerialPort1.PortName = TextBox1.Text
            SerialPort1.BaudRate = 19200
            SerialPort1.DataBits = 8
            SerialPort1.ReadTimeout = 2000

            SerialPort1.Open()
            Timer1.Start()
            ListBox1.Items.Add("Connected Successfully")
            Button1.Text = "Disconnect"
        Else
            MsgBox("Select a COM port first")
        End If
    Else
        Timer1.Stop()
        SerialPort1.Close()
        ListBox1.Items.Add("Disconnected..")
        Button1.Text = "Connect"
    End If

```

Button ON

```
0 references
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    If RadioButton1.Checked = True Then
        SerialPort1.Write("1")
        ListBox1.Items.Add("LED 1 turned ON")
    End If
    If RadioButton2.Checked = True Then
        SerialPort1.Write("2")
        ListBox1.Items.Add("LED 2 turned ON")
    End If
    If RadioButton3.Checked = True Then
        SerialPort1.Write("3")
        ListBox1.Items.Add("LED 3 turned ON")
    End If
    If RadioButton4.Checked = True Then
        SerialPort1.Write("4")
        ListBox1.Items.Add("LED 4 turned ON")
    End If
End Sub
```



Button OFF

```
0 references
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    If RadioButton1.Checked = True Then
        SerialPort1.Write("5")
        ListBox1.Items.Add("LED 1 turned OFF")
    End If
    If RadioButton2.Checked = True Then
        SerialPort1.Write("6")
        ListBox1.Items.Add("LED 2 turned OFF")
    End If
    If RadioButton3.Checked = True Then
        SerialPort1.Write("7")
        ListBox1.Items.Add("LED 3 turned OFF")
    End If
    If RadioButton4.Checked = True Then
        SerialPort1.Write("8")
        ListBox1.Items.Add("LED 4 turned OFF")
    End If
End Sub
```

22. Jika sudah selesai klik **Start** pada **Toolbar**

23. Jika program idak ada error, akan muncul form aplikasi yang telah kalian buat.

