Briefing

Pada bab briefing ini, praktikan akan mempelajari tentang sejarah awal FPGA, konsep dasar dan cara kerja FPGA, konsep dasar gerbang logika, serta software dan hardware yang akan digunakan dalam praktikum FPGA. Asisten praktikum atau praktikan dapat membaca tujuan dan persyaratan praktikum bab ini agar praktikum dapat berjalan sesuai prosedur.

Tujuan

Tujuan	Penjelasan
Memahami sejarah perkembangan FPGA	Praktikan diharapkan dapat memahami sejarah awal perkembangan FPGA dan bagaimana teknologi ini berevolusi hingga digunakan secara luas saat ini
Memahami konsep dasar dan cara kerja FPGA	Praktikan diharapkan dapat memahami komponen FPGA dan cara kerja pada FPGA
Memahami jenis-jenis software dan hardware FPGA	Praktikan diharapkan dapat memahami software dan hardware yang akan digunakan untuk desain dan simulasi FPGA
Memahami konsep dasar dari gerbang logika	Praktikan diharapkan dapat mengenali dan menjelaskan fungsi dasar gerbang logika seperti AND, OR, NOT,NAND,NOR,XOR,XNOR dan bagaimana gerbang-gerbang ini digunakan dalam desain FPGA.

Persyaratan

Disarankan praktikan menggunakan hardware dan software sesuai pada dokumentasi ini. Apabila terdapat versi hardware atau software yang cukup lama dari versi yang direkommendasikan maka sebaiknya bertanya kepada Asisten Mengajar Shift.

HARDWARE YANG DIBUTUHKAN PRAKTIKUM
PC / Laptop
FPGA Board Nexys A7 dan Nexys 4
Kabel Power USB

SOFTWARE YANG DIBUTUHKAN PRAKTIKUM
Vivado Design Suite

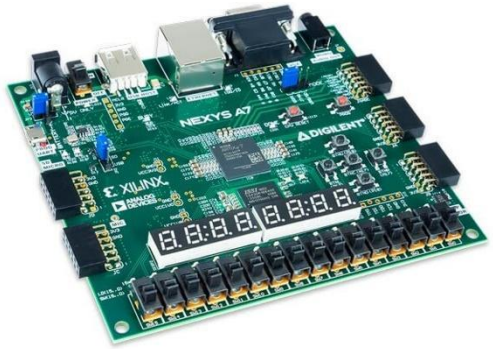
> ⚠ Diusahakan untuk memakai **Versi** dan **Aplikasi** yang sama agar tidak terjadinya kesalahan yang tidak diinginkan!

B.1. Sejarah FPGA

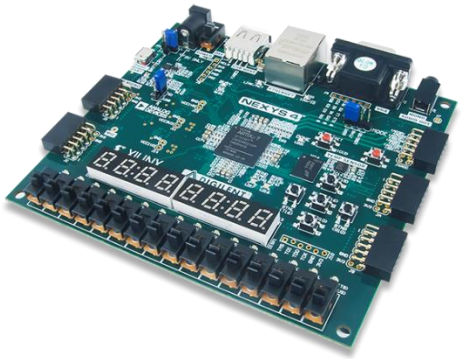
FPGA (*Field Programmable Gate Array*) merupakan sebuah IC digital yang digunakan untuk merangkai rangkaian digital. FPGA ini mulai diperkenalkan pada tahun 1980 dan dikembangkan sejak tahun 1985 oleh perusahaan Xilinx yang berbasis di San Jose, California yang menciptakan FPGA pertama adalah XC2064 pada tahun 1985. Perangkat ini sangat primitif, dengan hanya 800 gerbang, hanya sebagian kecil jika dibandingkan dengan jutaan operasi gerbang yang dapat dilakukan pada FPGA saat ini. Perangkat ini juga relatif mahal, yaitu $55, yang jika disesuaikan dengan inflasi akan menjadi sekitar $145 saat ini. Meskipun demikian, XC2064 telah memulai seluruh industri, dan Xilinx telah menjadi salah satu perusahaan dominan di pasar FPGA selama lebih dari 30 tahun. Selain Xilinx, terdapat juga beberapa perusahaan lain yang memproduksi FPGA seperti Altera, Lattice, dan Quicklogic.

B.1.1 Versi Produk FPGA Xilinx

1. Virtex Family
2. Kintex Family
3. Artix Family
4. Zynq Family
5. Spartan Family
6. Easy Path
7. Versal



Gambar B. 1 Nexys A7

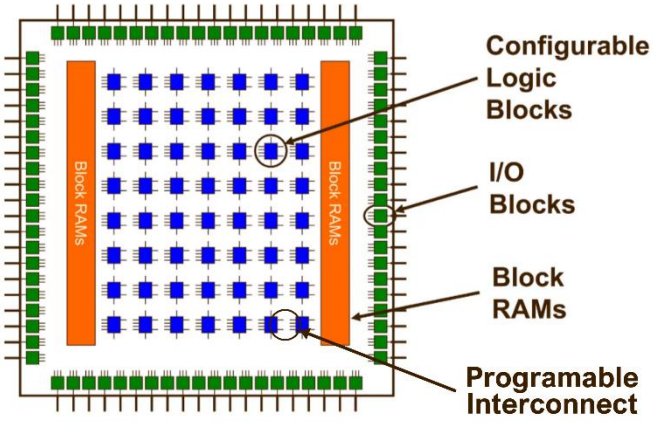


Gambar B. 2 Nexys 4

B.2. Komponen FPGA

Bila dilihat dari namanya, (*Field Programmable*) dapat diartikan bahwa FPGA ini bersifat dapat dirancang sesuai dengan keinginan dan kebutuhan user atau pemakai.

Sedangkan (*Gate Array*) artinya FPGA ini terdiri atas gerbang-gerbang digital dimana masing-masing dari gerbang tersebut dapat dikonfigurasikan antara satu sama lain. FPGA ini juga memiliki sifat *volatile*, yaitu sifat dimana apabila sumber dayanya dicabut maka program yang sudah ditanam sebelumnya akan hilang. Namun dibeberapa perangkat, sudah ada yang ditanamkan ROM untuk dapat menyimpan program yang sudah dibuat.



Gambar B. 3 Komponen FPGA

Dalam FPGA terdapat komponen-komponen yang memiliki fungsinya masing-masing yaitu :

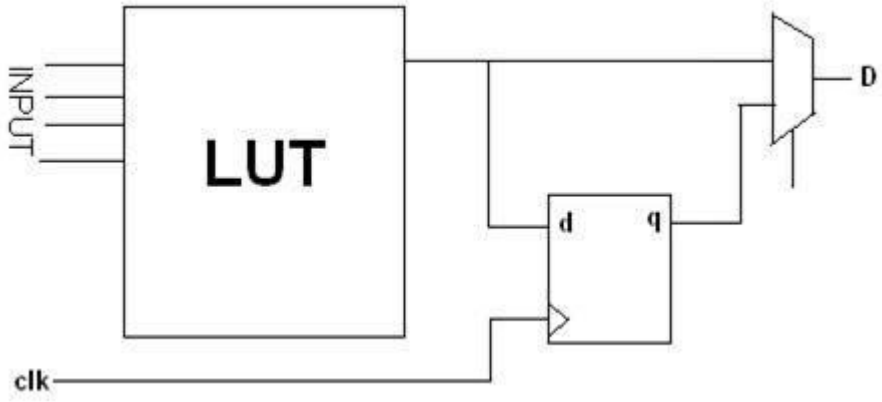
- *Configure Logic Blocks* (CLB), adalah komponen utama dari FPGA yang berfungsi memproses segala bentuk logika yang di blok user atau pengguna. CLB biasanya terdiri dari beberapa elemen logika (gerbang logika, LUT (*Look Up Tables*), Flip-Flop, dan multiplexer) dan digunakan untuk memproses rangkaian logika yang dibuat oleh pengguna.
- I/O Blocks, berfungsi sebagai *interface* yang dapat diprogram untuk *input* dan *output* transfer data dalam FPGA.
- Block RAM, blok memori yang berfungsi sebagai penyimpanan data dalam FPGA

Programmable Interconnect, bagian ini berfungsi sebagai saklar yang menghubungkan antara satu CLB dengan CLB lainnya.

B.3. Cara Kerja FPGA

FPGA tersusun dari *logic block* yang saling terhubung satu satu sama lain. Kumpulan-kumpulan dari *logic block* ini berjumlah ratusan bahkan ribuan sehingga membentuk suatu

fungsi yang kompleks. Sebuah *logic block* pada dasarnya terdiri dari dari sebuah LUT (*Lookup Table*), *Flip-Flop*, dan sebuah *multiplexer*.



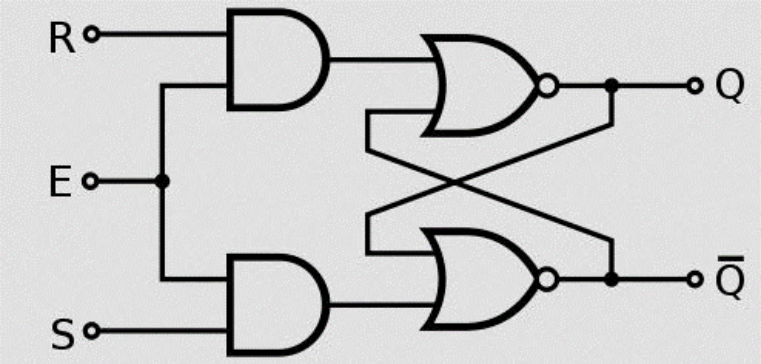
Gambar B. 4 Logic-cell

B.3.1. LUT (Lookup Table)

LUT atau *Lookup Table* merupakan sejenis RAM yang berkapasitas kecil. Di dalam FPGA, LUT ini memegang peran dalam proses implementasi fungsi-fungsi logika dan mempunyai ciri khas yaitu memiliki 4 masukan.

B.3.2. Flip-Flop

Flip-flop merupakan sebuah rangkaian yang memiliki dua keadaan stabil dan mewakili satu bit. *Flip-flop* adalah *resource* penyimpanan terkecil pada FPGA. Setiap *flip-flop* dalam sebuah CLB adalah register biner yang digunakan untuk menyimpan status logika antara siklus clock pada rangkaian FPGA.

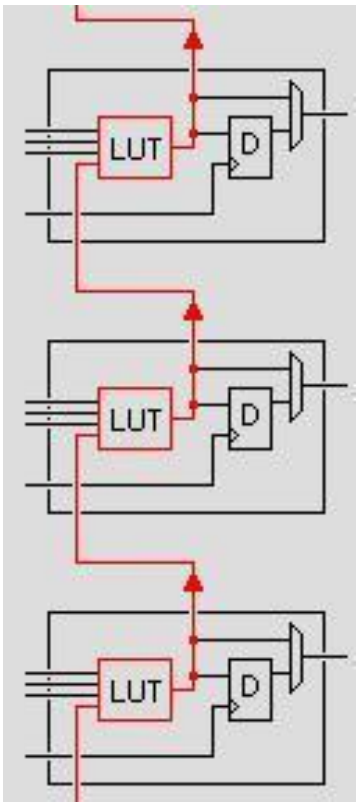


Gambar B. 5 Rangkaian Flip-Flop

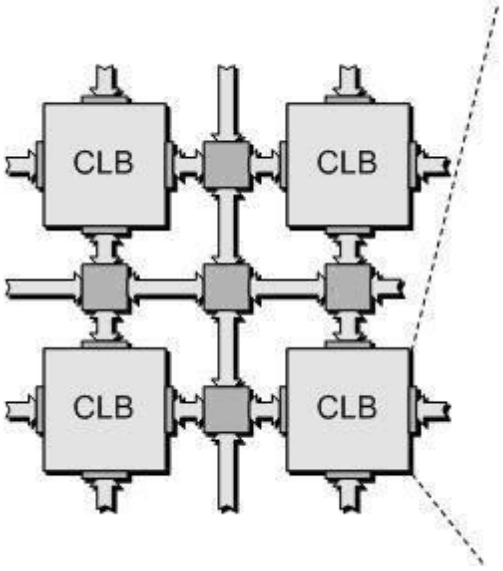
B.3.3. Multiplexer

Multiplexer bekerja sebagai *switch* (saklar) yang menghubungkan beberapa *input* menjadi satu *input*-an saja.

Setiap *logic block* dapat dihubungkan dengan *logic block* lainnya melalui jalur koneksi yang ada. Setiap *block* hanya mampu bekerja secara ringkas dan sederhana, namun apabila antar *block* saling terhubung satu sama lain, maka sebuah fungsi-fungsi logika yang kompleks dapat terbentuk.



Gambar B. 6 Kumpulan Logic-cell



Gambar B. 7 Kumpulan CLB

B.4. Software

Pada umumnya, perusahaan pembuat FPGA memberikan perangkat lunak secara gratis yang sudah termasuk kedalam paket pembelian produk mereka. Namun, software yang gratis ini hanya untuk jenis FPGA tingkat rendah-menengah saja, tetapi sudah cukup mendukung untuk digunakan dalam pembelajaran. Berikut beberapa software pendukung yang gratis :

1. Xilinx, terkenal dengan software miliknya yang bernama ISE Design Suite dan Vivado.

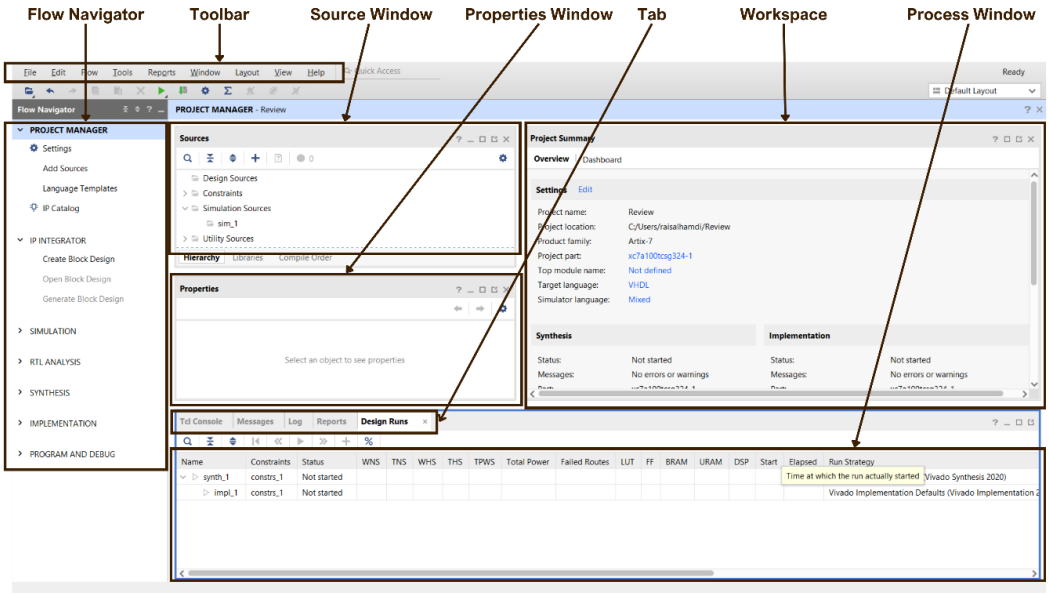
2. Altera, terkenal dengan software miliknya yang bernama Quartus.

Xilinx, Inc. adalah sebuah perusahaan teknologi asal Amerika yang terutama memasok perangkat logika terprogram. Perusahaan ini merupakan pencipta FPGA. Xilinx juga merupakan perusahaan semikonduktor yang menciptakan model manufaktur nirfabrikasi pertama. Didirikan bersama-sama oleh Ross Freeman, Bernard Vonderschmitt, dan James V Barnett II pada tahun 1984, perusahaan ini resmi melantai di NASDAQ pada tahun 1989. Pada bulan Oktober 2020, AMD mengumumkan akuisisi terhadap Xilinx. Akuisisi ini selesai pada tanggal 14 Februari 2022.

Gambar B. 8 Logo Vivado

Pembelajaran praktikum sebelumnya menggunakan *software* ISE Design Suite yang kini diganti menjadi Vivado. Vivado merupakan generasi penerus dari ISE Design Suite, menawarkan sejumlah keunggulan yang membuatnya lebih cocok untuk pembelajaran FPGA saat ini terlebih antarmuka pengguna yang lebih modern dan *user-friendly* dibandingkan dengan ISE Design Suite.

Vivado mendukung sepenuhnya kedua bahasa pemrograman HDL (*Hardware Description Language*) yang paling umum digunakan, yaitu VHDL dan Verilog. Ini memungkinkan pengguna untuk mempelajari dan mengimplementasikan desain FPGA menggunakan bahasa yang mereka kuasai atau yang paling sesuai dengan kebutuhan proyek. *Software* ini dilengkapi dengan alat pengecekan sintaks dan semantik yang kuat. Ini membantu pengguna mengidentifikasi kesalahan dalam kode sebelum proses sintesis, sehingga mempercepat proses debugging dan meningkatkan produktivitas. Hierarki desain dari Vivado membuat pengguna untuk mengorganisasi kode menjadi modul-modul yang lebih kecil dan lebih mudah dikelola.



Gambar B. 9 Tampilan Awal Software Vivado

B.4.1 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, pengguna 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.

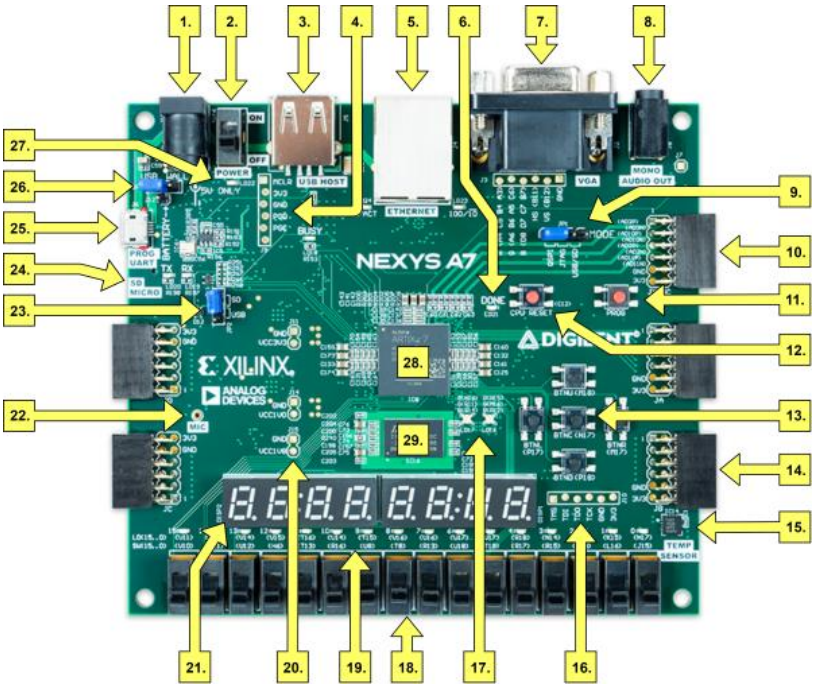
- Error Reported 

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

- Out-Of-Date 

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

B.5. Hardware



Gambar B. 10 Pin Callout Nexys A7

Tabel B. 1 Pin Callout dan Keterangannya

Callout	Component Description	Callout	Component Description
1	Power jack	16	JTAG port for (optional) external cable
2	Power switch	17	Tri-color (RGB) LEDs
3	USB host connector	18	Slide switches
4	PIC24 programming port (factory use)	19	LEDs
5	Ethernet connector	20	Power supply test point(s)
6	FPGA programming done LED	21	Eight digit 7-seg display
7	VGA connector	22	Microphone
8	Audio connector	23	External configuration jumper (SD / USB)
9	Programming mode jumper	24	MicroSD card slot
10	Analog signal Pmod port (XADC)	25	Shared UART/ JTAG USB port
11	FPGA configuration reset button	26	Power select jumper and battery header
12	CPU reset button (for soft cores)	27	Power-good LED

13	Five pushbuttons	28	Xilinx Artix-7 FPGA
14	Pmod port(s)	29	DDR2 memory
15	Temperature sensor		

Artix-7

Artix-7 merupakan salah satu seri board FPGA (Field Programmable Gate Array) yang diproduksi oleh Xilinx. Artix-7 dirancang khusus untuk memberikan performa yang optimal dengan biaya yang terjangkau. Seri ini sangat populer digunakan dalam berbagai aplikasi karena memiliki fitur yang lengkap dan memiliki konsumsi daya yang rendah, serta mudah untuk digunakan.

Nexys A7

Nexys™A7, yang berbasis pada FPGA Artix-7, memiliki performa dan desain yang berfokus untuk digunakan oleh pelajar. Dengan FPGA berkapasitas besar, memori eksternal yang luas, dan berbagai *port* seperti USB, Ethernet, dan lainnya, Nexys A7 dapat digunakan untuk mendesain rangkaian digital dasar hingga proses yang kompleks.

Tabel B. 2 Spesifikasi Nexys A-7

FPGA	**XC7A100T-1CSG324C**
I/O Interfaces	• USB-UART • One 10/100 Ethernet • USB OTG 2.0 • USB-UART bridge • 12-bit VGA • 3-axis accelerometer • PWM audio output • Temperature sensor • PDM microphone • USB HID
Memory	• 128 Mbyte DDR2 • 128 Mbit Serial Flash • Micro SD card connector
Display	2 4-digit 7-Segment display
Switch and LED	• 16 Slide switches • 16 LEDs

	• 2 tri-color LEDs • 5 Push-buttons
Clock	100 MHz crystal oscillator
Expansion Ports	• Pmod XADC signals • 4 Pmod ports

B.6. HDL (Hardware Description Language)

Salah satu metode untuk perancangan rangkaian adalah menggunakan HDL (*Hardware Description Language*). Nantinya, tiap-tiap komponen serta jalur yang menghubungkannya akan dideskripsikan lewat tulisan atau kode tertentu. 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*.

B.6.1. Verilog

Verilog adalah bahasa HDL yang dikembangkan oleh IEEE (*Institute of Electrical and Electronic Engineering*) dan formatnya berbentuk teks untuk mendeskripsikan rangkaian dan sistem elektronik. Dalam desain elektronik, Verilog digunakan dalam verifikasi melalui simulasi, analisis waktu, analisis uji (penilaian kesalahan), dan sintesis logika.

B.6.2. VHDL

Pada praktikum ini, kita akan akan menggunakan VHDL untuk membuat desain program kedalam FPGA tersebut. VHDL (*VHISC Hardwere Description Language*); VHSIC (*Very High Speed Integrated Circuit*) adalah sebuah bahasa pemrograman yang dikembangkan oleh IEEE (*Institute of Electrical and Electronic Engineering*). VHDL juga digunakan sebagai bahasa pemrograman untuk simulasi rangkaian dari komponen-komponen digital. Pada VHDL, konsep serta sintaks 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. Untuk mempelajari VHDL, terdapat 5 komponen penting dalam menulis desain programnya, yaitu :

1. Entity
2. Architecture
3. Configuration
4. Package Declaration

5. Package Body

Perbedaan bahasa Verilog dan VHDL

Verilog dan VHDL adalah bahasa HDL yang digunakan untuk merancang sirkuit digital. Keduanya memiliki tujuan yang sama, namun memiliki pendekatan yang sedikit berbeda. Verilog cenderung lebih intuitif dan mirip dengan bahasa pemrograman tingkat tinggi, sehingga lebih disukai oleh para *engineer* yang memiliki latar belakang *software*. Bahasa ini sering digunakan untuk merancang sirkuit pada tingkat yang lebih rendah. Di sisi lain, VHDL memiliki struktur yang lebih formal dan matematis, membuatnya lebih cocok untuk verifikasi dan simulasi yang kompleks. Bahasa ini sering digunakan dalam proyek-proyek besar dan kompleks yang membutuhkan tingkat abstraksi yang lebih tinggi.

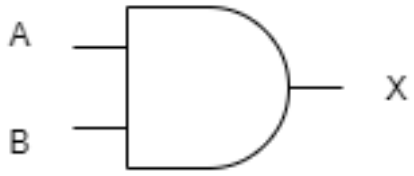
Tabel B. 3 Perbedaan Verilog dan VHDL

Verilog	VHDL
Bahasa HDL yang digunakan untuk memodelkan sistem elektronik.	Bahasa HDL yang digunakan untuk mendeskripsikan sistem digital dan campuran sinyal.
Berdasarkan bahasa C	Berdasarkan bahasa Pascal
Case Sensitive	Tidak *Case Sensitive*
Tidak mendukung Array multidimensi	Mendukung Array multidimensi
Verilog tidak mengizinkan pemanggilan tugas secara bersamaan	VHDL mengizinkan pemanggilan prosedur secara bersamaan.
Memiliki tipe data yang sederhana	Memiliki tipe data yang kompleks
Tidak memiliki *Library*	Terdapat *Library*

B.7. Jenis-jenis Gerbang Logika

a. Gerbang AND

Gerbang AND adalah gerbang logika dasar yang melakukan perkalian logika berdasarkan dari input yang diterapkan padanya. Gerbang ini menghasilkan output HIGH atau bernilai 1, hanya ketika semua input atau masukan bernilai 1. Jika tidak, output gerbang AND akan memiliki nilai LOW atau 0.



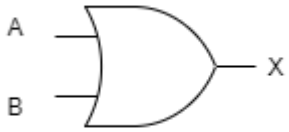
Gambar B. 11 Gerbang Logika AND

Tabel B. 4 Gerbang Logika AND

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

b. Gerbang OR

Gerbang OR adalah gerbang logika dasar yang melakukan penjumlahan logika berdasarkan nilai dari input yang diterapkan pada gerbang. Gerbang ini akan menghasilkan nilai 1 apabila salah satu atau semua input memiliki nilai 1. Apabila tidak terdapat nilai 1 dari input, maka output gerbang OR akan bernilai 0.



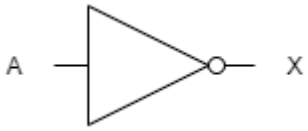
Gambar B. 12 Gerbang Logika OR

Tabel B. 5 Gerbang Logika OR

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

c. Gerbang NOT

Gerbang NOT atau disebut juga sebagai inverter, akan menghasilkan output yang komplemen dengan hasil inputnya. Jika inputnya bernilai 1 maka output akan bernilai 0 dan sebaliknya.



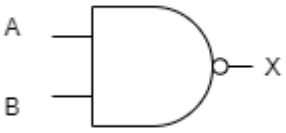
Gambar B. 13 Gerbang Logika NOT

Tabel B. 6 Gerbang Logika NOT

A	Out
1	0
0	1

d. Gerbang NAND

Gerbang NAND adalah gabungan dari gerbang logika AND dan juga gerbang logika NOT. Hasil dari gerbang logika NAND adalah kebalikan dari gerbang AND.



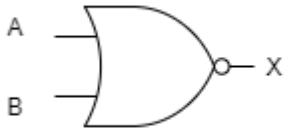
Gambar B. 14 Gerbang Logika NAND

Tabel B. 7 Gerbang Logika NAND

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

e. Gerbang NOR

Gerbang NOR adalah gabungan dari gerbang logika OR dan juga gerbang logika NOT. Hasil dari gerbang logika NOR adalah kebalikan dari gerbang OR.



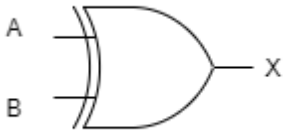
Gambar B. 15 Gerbang Logika NOR

Tabel B. 8 Gerbang Logika NOR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

f. Gerbang XOR

Gerbang XOR (*Exclusive* OR) adalah gerbang logika paritas ganjil yang akan menghasilkan nilai 1 apabila kedua inputnya memiliki nilai yang berbeda.



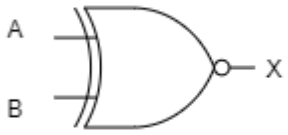
Gambar B. 16 Gerbang Logika XOR

Tabel B. 9 Gerbang Logika XOR

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

g. Gerbang XNOR

Gerbang XNOR (*Exclusive* NOR) adalah gerbang logika paritas genap yang akan menghasilkan nilai 1 apabila kedua inputnya memiliki nilai yang sama.



Gambar B. 17 Gerbang Logika XNOR

Tabel B. 10 Gerbang Logika XNOR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

BAB 1

Entity

Pada bab ini, praktikan akan mempelajari konsep entity dalam VHDL, yang merupakan bagian penting dalam mendefinisikan antarmuka modul pada desain digital. Selain itu, praktikan juga akan mempelajari cara membuat entity sederhana, serta memahami berbagai jenis library yang mendukung desain. Asisten praktikum atau praktikan diharapkan membaca tujuan dan persyaratan pada bab ini agar praktikum dapat berjalan sesuai prosedur.

Tujuan

Tujuan	Penjelasan
Mengenal dan memahami tentang entity dalam VHDL	Praktikan diharapkan dapat memahami konsep dasar entity dalam VHDL, bagaimana entity berperan dalam mendeskripsikan antarmuka modul, serta bagaimana entity digunakan untuk mendefinisikan input dan output pada desain.
Membuat entity sederhana	Praktikan diharapkan dapat membuat entity sederhana menggunakan VHDL, sehingga dapat memahami struktur penulisan entity serta komponen yang harus ada dalam mendesain suatu rangkaian digital.
Mengenal jenis-jenis library	Praktikan diharapkan dapat mengenal berbagai jenis library yang sering digunakan dalam VHDL, memahami fungsi dari setiap library, serta bagaimana library ini mendukung proses pengembangan desain digital.

Persyaratan

Disarankan praktikan menggunakan hardware dan software sesuai pada dokumentasi ini. Apabila terdapat versi hardware atau software yang cukup lama dari versi yang direkommendasikan maka sebaiknya bertanya kepada Asisten Mengajar Shift.

HARDWARE YANG DIBUTUHKAN PRAKTIKUM
PC / Laptop
FPGA Board Nexys A7 dan Nexys 4
Kabel Power USB

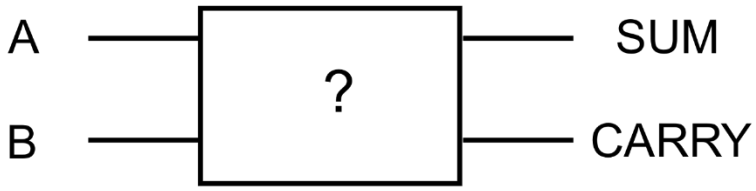
SOFTWARE YANG DIBUTUHKAN PRAKTIKUM
Vivado Design Suite

⚠ Diusahakan untuk memakai **Versi** dan **Aplikasi** yang sama agar tidak terjadinya kesalahan yang tidak diinginkan!

1.1. Pengertian

Entity adalah nama dari sebuah desain yang biasanya didalamnya sudah mendefinisikan *input* dan *output* dari suatu desain program. Selain itu, *entity* ini juga dapat mengatur jenis atau tipe *port* apa yang akan dipakai. Di dalam *entity*, diperlukan sebuah nama atau variable untuk menentukan *port* masukan dan keluaran. Penentuan *port* mengandung nama, mode, dan tipe data. Mode *port* terdiri dari 3 jenis, yaitu :

1. IN, merupakan *port* yang digunakan untuk mendeklarasikan masukan atau *input* pada desain *entity*.
2. OUT, merupakan *port* yang digunakan untuk mendeklarasikan keluaran atau *output* pada desain *entity*.
3. INOUT (*bidirectional*), merupakan *port* yang dapat digunakan sebagai masukan sekaligus keluaran pada desain *entity*.



Gambar 1. 1 Contoh Desain Entity

Dari gambar tersebut, terdapat sebuah desain yang didalamnya masing-masing terdapat dua *input* dan dua *output*. Diketahui bahwa rangkaian diatas merupakan rangkaian *half adder* yang memiliki dua *output* yaitu *Sum* dan *Carry*. Seperti yang sudah dijelaskan diawal, *entity* adalah nama dari sebuah desain yang akan dirancang serta mendefinisikan *input* dan *output*, maka dari itu *entity* dari sebuah desain diatas dapat diberi nama "*Half Adder*".

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 :

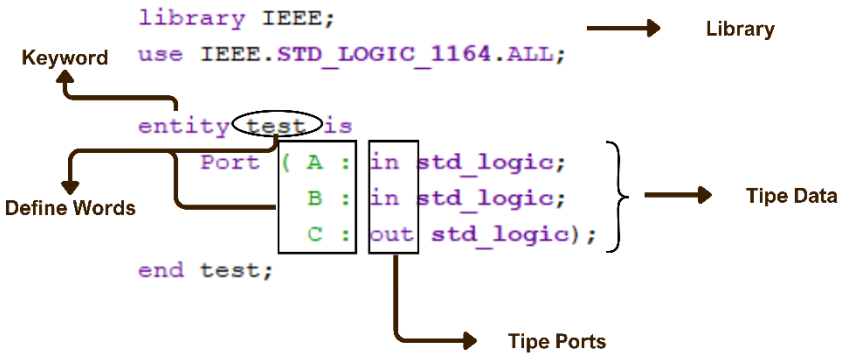
```
entity <entity_name> is
    generic (
       <generic_name> : <type>
    );
    port (
        <input_name>    : in <type>;
        <output_name>   : out <type);
        <bidir_name>    : inout <type>
    );
end entity <entity_name>;
```

Dalam konstruksi ini, <entity_name> akan menjadi nama komponen yang sedang dirancang. Kemudian menggunakan port dalam deklarasi *entity* VHDL untuk mendefinisikan *input* dan *output* dari komponen yang dirancang. Oleh karena itu, *port* sama seperti dengan pin pada komponen elektronik pada umumnya.

Port dapat didefinisikan sebagai *in, out,* atau *inout*, yang masing-masing berhubungan dengan *input, output,* dan *port bidirectional*. Biasanya berbagai jenis *port* ini disebut sebagai mode. Selain itu, mendeklarasikan jenis data yang digunakan oleh *port* dalam bagian <signal_type>.

Misalkan sebuah *entity* diberi nama "test", maka kerangka *entity* tersebut akan menjadi :

```
entity test is

end test;
```



Gambar 1. 2 Bagan Program

1.2. Port

Sebuah cara atau metode untuk menghubungkan *entity* secara bersama adalah menggunakan *port*. Berikut adalah cara mendefinisikan *port* atau struktur dari *port* sebagai berikut :

```
entity test is
    Port ( input : in <type> ;
           output : out <type>;
           bidir : inout <type>);
end test;
```

Penulisan *port* adalah dengan memberikan nama pada *port* yang akan digunakan kemudian masukan mode *port* dan dilanjutkan dengan tipe data.

Contoh :

```
entity test is
    Port ( A : in  std_logic;
           B : out std_logic;
           C : inout std_logic);
end test;
```

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

1.3. Keywords dan Define Words

Dalam penulisan *entity*, dikenal juga istilah "*Keywords*" dan "*Define Words*". *Keywords* merupakan sintaks yang digunakan untuk menulis program VHDL, sedangkan *Define Words* adalah deskripsi dari sebuah desain yang ingin kita buat.

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

entity test is
    Port ( A : in std_logic;
           B : in std_logic;
           C : out std_logic);
end test;
```

Dari gambar tersebut, tulisan yang berwarna ungu merupakan *keywords*. Penulisan program VHDL ini harus mengikuti sintaks yang sudah ada, selanjutnya diikuti dengan *define words. Define words* sebenarnya hanyalah sebuah variabel, kita dapat menuliskan apa saja dengan catatan kita mengetahui *define words* tersebut akan kita gunakan untuk apa. Berikut merupakan jenis-jenis keyword pada VHDL :

FIGURE 1-15

abs	disconnect	label	package	sla
access	downto	library	port	sll
after	else	linkage	postponed	sra
alias	elsif	literal	procedure	srl
all	end	loop	process	subtype
and	entity	map	protected	then
architecture	exit	mod	pure	to
array	file	nand	range	transport
assert	for	new	record	type
attribute	function	next	register	unaffected
begin	generate	nor	reject	units
block	generic	not	rem	until
body	group	null	report	use
buffer	guarded	of	return	variable
bus	if	on	rol	wait
case	impure	open	ror	when
component	in	or	select	while
configuration	inertial	others	severity	with
constant	inout	out	shared	xnor
	is		signal	xor

Gambar 1. 3 Jenis-jenis Keyword pada VHDL

- *Entity*, digunakan untuk menyatakan *entity* pada VHDL.
- *Architecture*, digunakan untuk menyatakan *architecture* pada VHDL.
- *Process*, digunakan untuk tugas-tugas seperti membuat proses yang dikendalikan oleh *clock* dan operasi kondisional.
- *Signal* dan *Variable*, digunakan untuk membuat objek data dalam VHDL dan keduanya berbeda dalam hal karakteristik penugasan dan waktu.
- *If, then,* dan *elseif*, digunakan sebagai bagian dari pernyataan kondisional yang digunakan dalam VHDL untuk mendefinisikan percabangan di dalam proses atau blok konkuren.
- *Logic Gate*, digunakan untuk mendeklarasikan gerbang logika yang akan digunakan pada VHDL.

1.4. Library

Pada pemrograman dikenal pula istilah *library* atau pustaka yang biasanya terdapat pada bahasa pemrograman yang lain seperti C atau *header* pada Pascal. Sebuah *library* adalah sebuah direktori, dan setiap package adalah file di dalam direktori tersebut. File *package* merupakan basis data yang berisi informasi tentang komponen-komponen dalam paket tersebut (*input* komponen*, output,* tipe*,* dll). Untuk menggunakan komponen dalam sebuah desain dengan *library* untuk menentukan pustaka yang akan dicari dan pernyataan *use* untuk setiap paket yang akan gunakan. Di dalam *library* tersebut terdapat *sub-tree* yang disebut sebagai *package*, diantaranya :

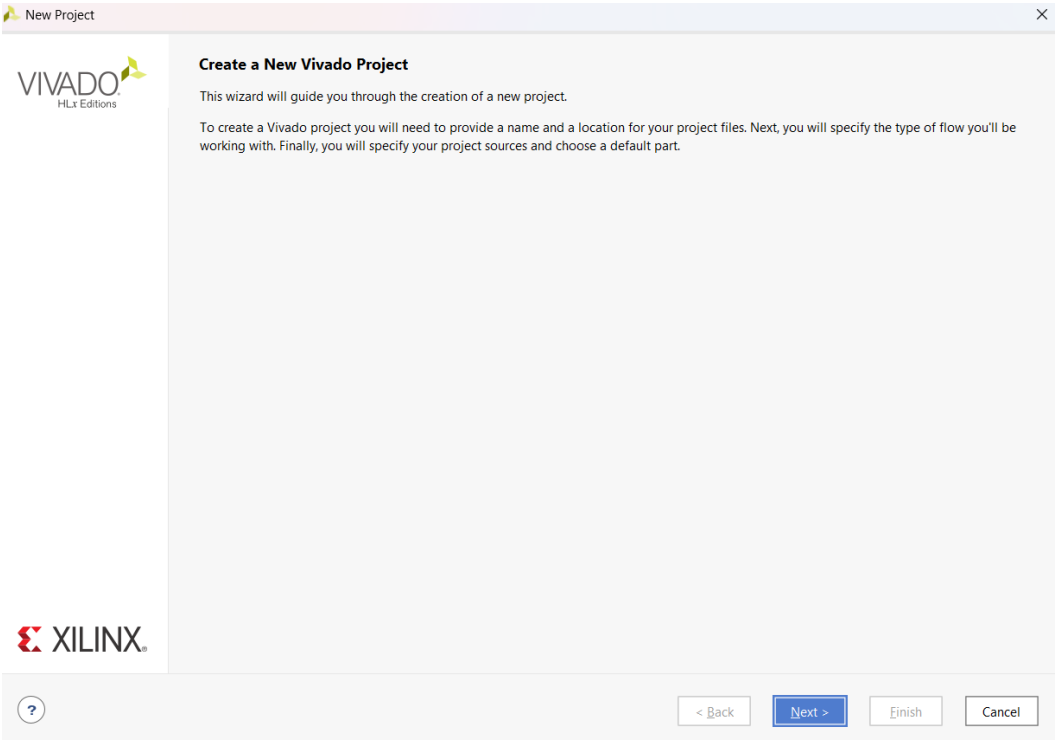
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_arith.all;

use IEEE.numeric_bit.all;
use IEEE.numeric_std.all;


use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use IEEE.math_complex.all;

library STD;
use STD.standard;
use STD.textio;
```

Gambar 1. 4 Jenis-jenis Library

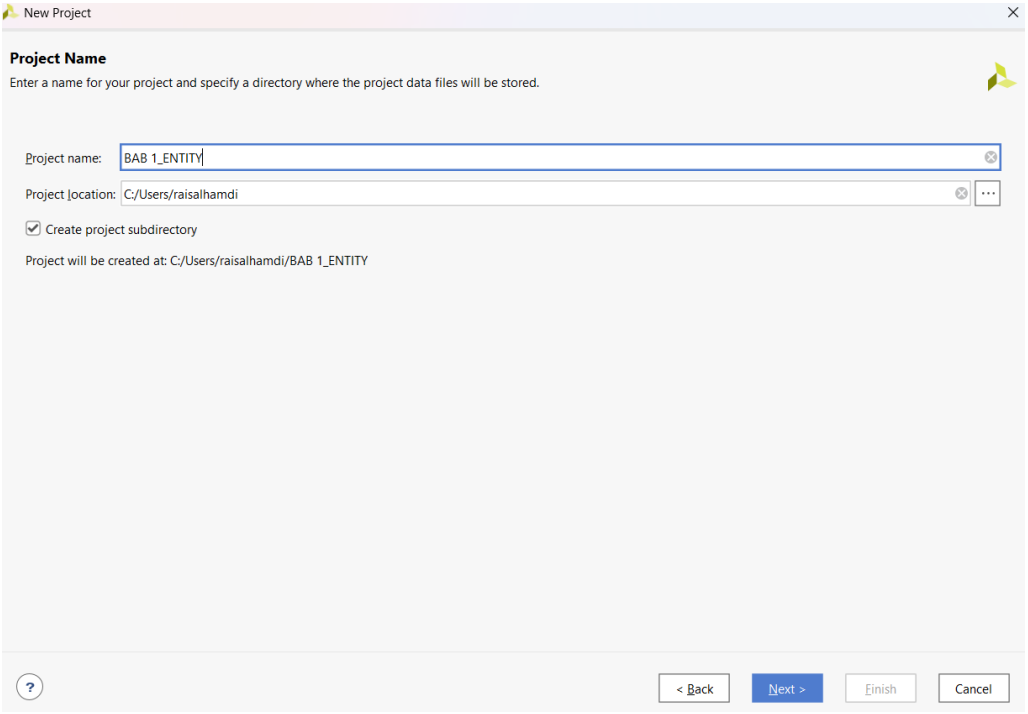
- IEEE.STD_LOGIC_1164 : *Library* ini menyediakan tipe data standar untuk sinyal digital, seperti '0', '1', 'X' (*unknown*), 'Z' (high impedance), dan lain-lain. Juga menyediakan operasi logika dasar (AND, OR, NOT, dll.).
- IEEE.NUMERIC_STD : *Library* ini menyediakan tipe data numerik yang digunakan untuk operasi aritmetika, seperti *integer, unsigned,* dan *signed.*
- IEEE.STD_LOGIC_ARITH : *Library* ini menyediakan operasi aritmetika pada tipe data std_logic_vector.

1.5. Membuat entity sederhana

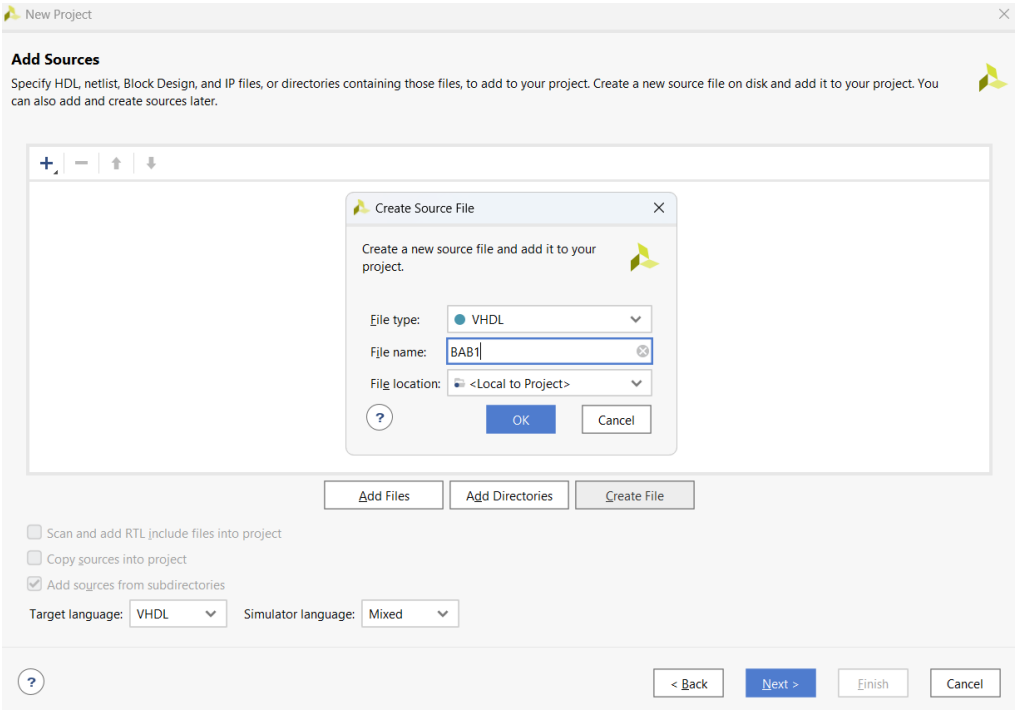
1. Buka aplikasi Vivado 2020
2. Buka menu File > Project > New Project



3. Kemudian akan muncul tampilan diatas dan kemudian pilih next

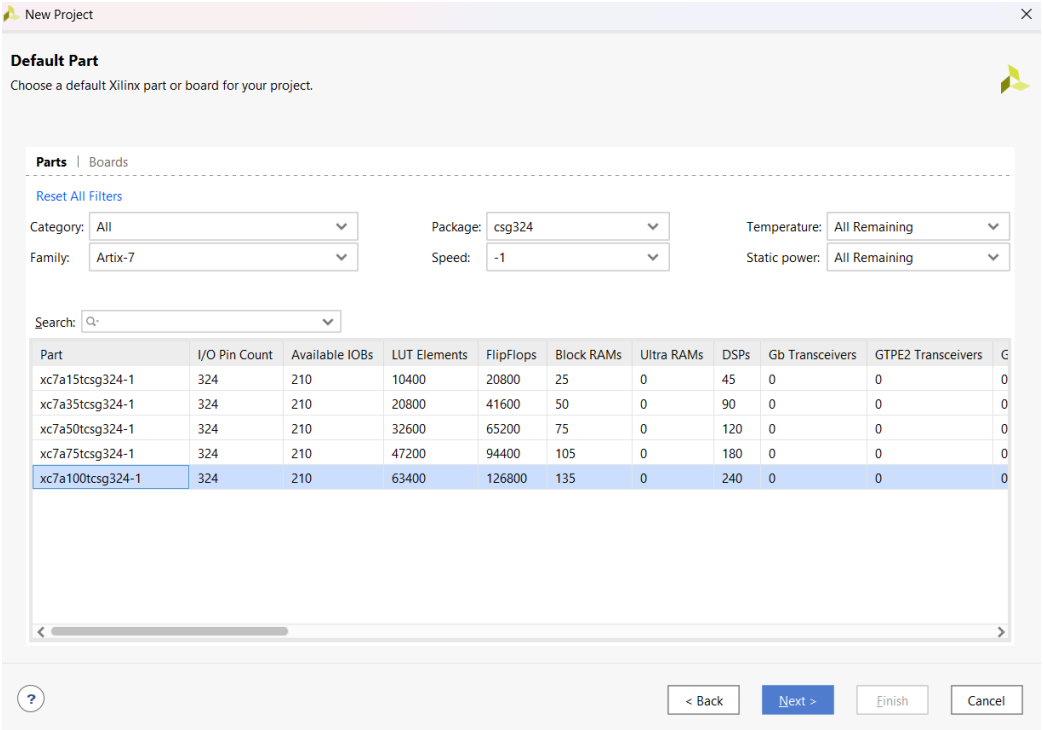


4. Buatlah nama project seperti diatas kemudian pilih next.

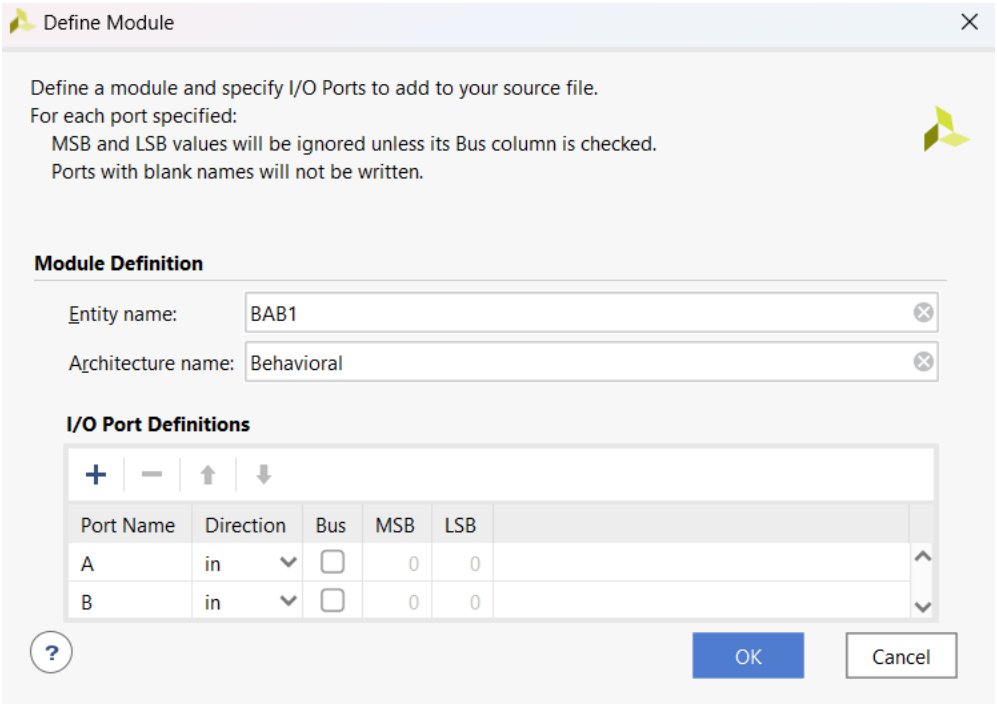


5. Kemudian muncul tampilan menu baru yaitu *add source* lalu pastikan target *language* yang digunakan adalah VHDL, lalu pilih *create file*.

6. Maka tampilan *create source file* akan muncul dan berikan nama BAB 1 lalu pilih ok.



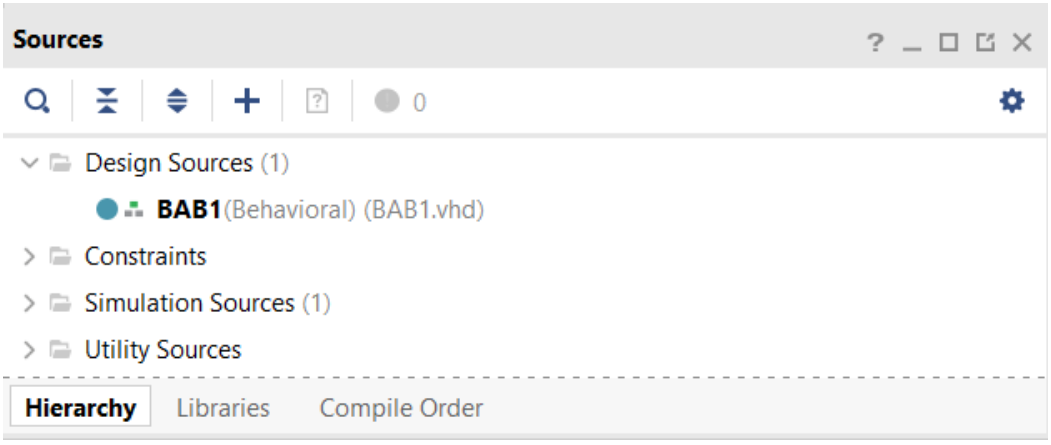
7. Lalu pada tampilan default part pastikan *family, package* dan *speed* yang digunakan seperti gambar diatas lalu pilih *board* yang akan digunakan yaitu **xc7a100tcsg324-1**.



8. Setelah itu akan muncul menu baru yaitu *define module* yang digunakan untuk mendefinisikan port port yang akan digunakan.

9. Buat lah *port name* A, B dan C.

10. Ubah *port direction* A dan B menjadi *in* dan *port direction* C menjadi *out*, lalu pilih ok.



11. Kemudian pada bagian *sources* > *design sources* akan muncul *entity* yang telah dibuat sebelumnya, lalu klik 2 kali pada *entity* yang telah dibuat sebelumnya yaitu BAB 1.

Project Summary | BAB1.vhd

C:/Users/raisalhamdi/BAB1_ENTITY/BAB1_ENTITY.srcs/sources_1/new/BAB1.vhd

```
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20
21
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity BAB1 is
35      Port ( A : in STD_LOGIC;
36             B : in STD_LOGIC;
37             C : out STD_LOGIC);
38  end BAB1;
39
40  architecture Behavioral of BAB1 is
41
42  begin
43
44
45  end Behavioral;
46
```

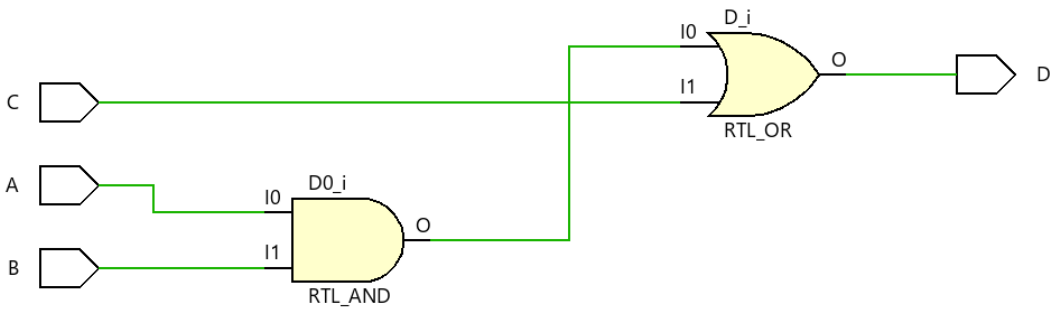
12. Berikutnya akan muncul tampilan program seperti gambar diatas.

Inilah salah satu cara untuk membuat sebuah *entity*. Sebenarnya kita juga dapat membuat manual dengan cara melewati langkah 8 dan mengisi program itu sesuai dengan apa yang ingin kita buat.

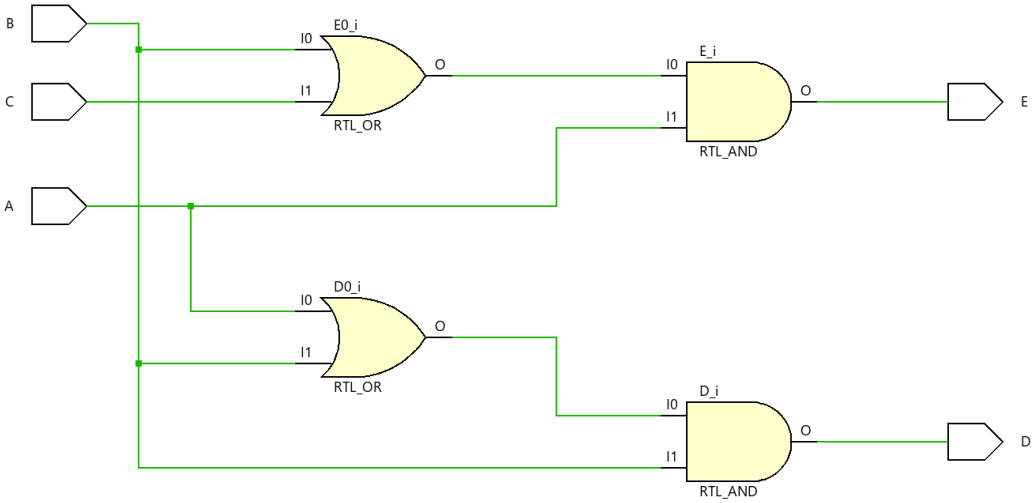
1.6. Latihan Soal

Buatlah program VHDL untuk menampilkan rangkaian berikut!

1.



2.


B
C
A
E0_i
I0
I1
O
RTL_OR
E_i
I0
I1
O
RTL_AND
E
D0_i
I0
I1
O
RTL_OR
D_i
I0
I1
O
RTL_AND
D

BAB 2

Architecture

Pada bab ini, praktikan akan mempelajari konsep dasar architecture dalam VHDL, yang digunakan untuk mendeskripsikan perilaku internal dari sebuah entity. Praktikan akan mempelajari cara membuat architecture sederhana dan bagaimana architecture berfungsi dalam merancang desain yang lebih besar dan kompleks. Asisten praktikum atau praktikan diharapkan membaca tujuan dan persyaratan bab ini agar praktikum dapat berjalan sesuai prosedur.

Tujuan

Tujuan	Penjelasan
Mengenal dan memahami tentang architecture dalam VHDL	Praktikan diharapkan dapat memahami konsep dasar architecture dalam VHDL, jenis-jenis architecture, bagaimana architecture dan entity bekerja bersama dalam mendesain rangkaian digital.
Membuat architecture sederhana	Praktikan diharapkan dapat membuat architecture sederhana yang menggambarkan perilaku sebuah entity, sehingga dapat memahami bagaimana sebuah desain digital dijelaskan secara detail dalam VHDL.
Menggunakan architecture untuk desain program yang lebih besar	Praktikan diharapkan dapat menggunakan architecture dalam merancang program yang lebih kompleks.

Persyaratan

Disarankan praktikan menggunakan hardware dan software sesuai pada dokumentasi ini. Apabila terdapat versi hardware atau software yang cukup lama dari versi yang direkommendasikan maka sebaiknya bertanya kepada Asisten Mengajar Shift.

HARDWARE YANG DIBUTUHKAN PRAKTIKUM
PC / Laptop
FPGA Board Nexys A7 dan Nexys 4
Kabel Power USB

SOFTWARE YANG DIBUTUHKAN PRAKTIKUM
Vivado Design Suite

⚠ Diusahakan untuk memakai **Versi** dan **Aplikasi** yang sama agar tidak terjadinya kesalahan yang tidak diinginkan!

2.1. Pengertian

Architecture dalam VHDL merupakan sebuah cara bagaimana sebuah desain dapat bekerja. Cara tersebut tentunya berkaitan dengan *entity* yang sudah dibuat sebelumnya dan *Architecture* berada diantara antara *input* dan *output.* Disinilah kita merangkai sebuah program sesuai dengan yang kita inginkan. Maka dari itu *architecture* dapat dikatakan :

- *Behavior*, yaitu bagaimana sebuah desain bekerja dan apa yang akan dilakukan desain tersebut
- *Function*, bagaimana hubungan antara input dan output bekerja dalam sebuah entity.

Sintaks atau struktur penulisan dari *Architecture* adalah sebagai berikut :

```
Architecture architecture_name of entity_name is
declarations
begin
(concurrent_statements)
end architecture_name
```

Contoh :

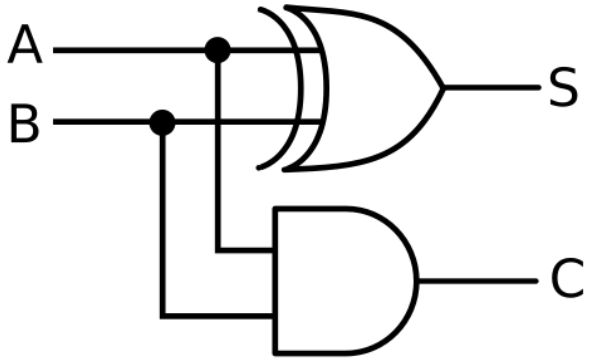
```
Architecture Behavioral of Switch_LED is

begin
led0 <= sw0 ;
led1 <= sw1 ;
end Behavioral;
```

Architecture tersebut berfungsi untuk menyalakan LED melalui *switch*. *Switch* pada dasarnya memiliki dua kondisi, yaitu kondisi 0 (LOW) dan 1 (HIGH). Ketika *switch* berubah kondisi dari LOW ke HIGH, maka akan ada tegangan yang dialirkan ke LED sehingga LED dapat menyala.

2.2. Jenis-jenis Architecture

2.2.1. Dataflow

Dataflow menggambarkan suatu sistem berdasarkan bagaimana data mengalir melalui sistem. Deskripsi *dataflow* secara langsung mengimplikasikan implementasi pada level gerbang yang sesuai dengan deskripsi gerbang logika yang digunakan. *Dataflow* terdiri dari satu atau lebih pernyataan penugasan sinyal yang berjalan secara bersamaan dan biasanya digunakan pada desain program yang lebih kecil atau sederhana.



Gambar 2. 1 Contoh Architecture Dataflow

Gambar tersebut merupakan rangkaian *half adder* yang terdapat dua *input* (A, B) dan dua *output* (S, C). *Half adder* merupakan rangkaian elektronika yang bekerja melakukan perhitungan penjumalahan dari 2 buah bilangan biner yang masing-masing terdiri dari 1 bit.

S = *sum*

C = *carry*

Berikut adalah kode program untuk membuat rangkaiannya.

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

entity half_adder is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : out STD_LOGIC;
           S : out STD_LOGIC);
end half_adder;

architecture dataflow of half_adder is

begin
S <= A xor B;
C <= A and B;

end dataflow;
```

2.2.2. Structural

Architecture jenis ini memerlukan definisi dari setiap komponen yang dipakai. Setiap komponen, harus di definisikan satu persatu untuk dapat menjadi sebuah rangkaian yang utuh. Structural menggambarkan struktur internal suatu rangkaian dengan mendeskripsikan komponen-komponen yang lebih kecil (*entity*).

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

entity BAB1_Structural is
    Port ( A : in bit;
           B : in bit;
           C : out bit;
           S : out bit);
end BAB1_Structural;

architecture structural of BAB1_Structural is

    component XOR1
    port(P,Q : in bit; R : out bit);
    end component;

    component AND1
    port(X,Y : in bit; Z : out bit);
    end component;

begin
X1 : XOR1 port map (A,B,S);
A1 : AND1 port map (A,B,C);

end structural;
```

2.2.3. Behavioral

Architecture behavioral menggambarkan perilaku suatu rangkaian secara keseluruhan tanpa mendefinisikan struktur internalnya. *Behavioral* didefinisikan menggunakan kode prosedural yang dieksekusi secara berurutan dan mekanisme utama yang digunakan adalah pernyataan proses. Jenis *architecture* ini biasanya digunakan untuk program yang lebih rumit dan kompleks.

Berikut adalah *truth table* dari rangkaian *half adder* :

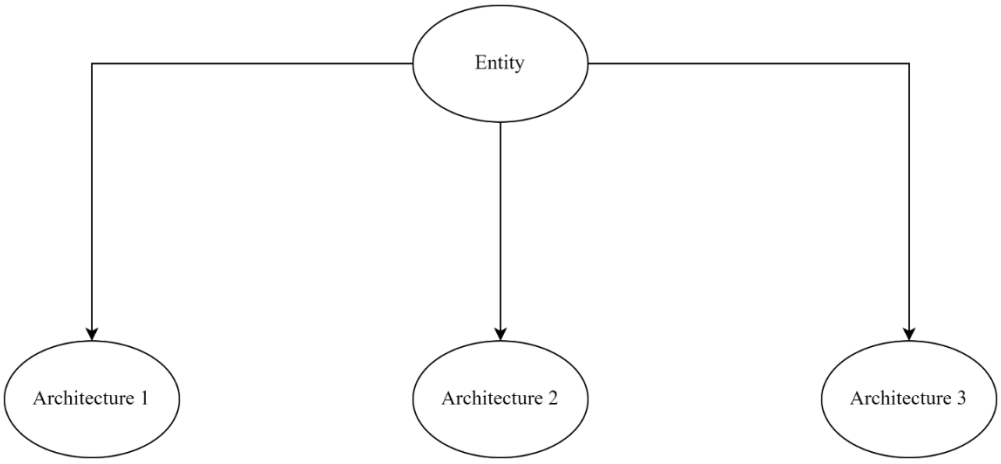
Tabel 2. 1 Truth Table Half Adder

Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Jika hasil dari table tersebut diimplementasikan menggunakan architecture Behavioral pada bahasa VHDL maka programnya :

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

entity BAB1_Behavioral is
    Port ( A : in bit;
           B : in bit;
           C : out bit;
           S : out bit);
end BAB1_Behavioral;

architecture Behavioral of BAB1_Behavioral is

begin
    process (a, b)
    begin
      if A&B = "00" then
      S <= '0';
      C <= '0';

      elsif A&B = "10" or A&B = "01" then
      S <= '1';
      C <= '0';

      else
      S <= '0';
      C <= '1';

      end if;
    end process;

end Behavioral;
```

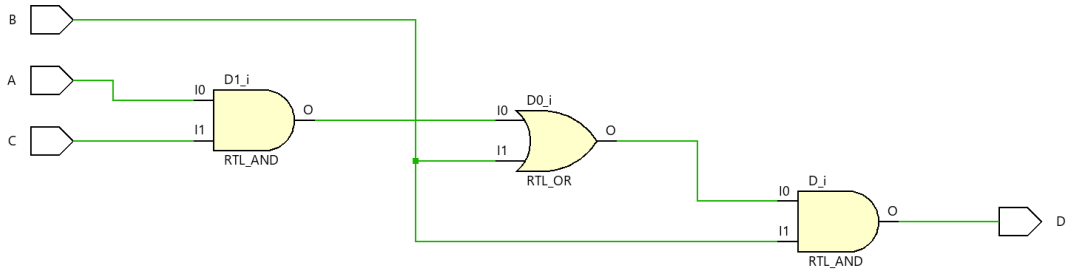
Dari program tersebut, terlihat jelas bahwa jenis ini lebih mengacu pada perilaku dari masing-masing komponen (XOR & AND *gate*). Perilaku tersebut harus didefinisikan semua untuk mendapatkan hasil yang diingkan yaitu rangkaian *half adder*. Dalam suatu desain program VHDL, sangat memungkinkan untuk menggunakan lebih dari satu architecture tergantung dari kebutuhan. Namun beberapa asrchitecture tersebut tetap harus memiliki sebuah entity **"One Entity Multiple Architecture"**.



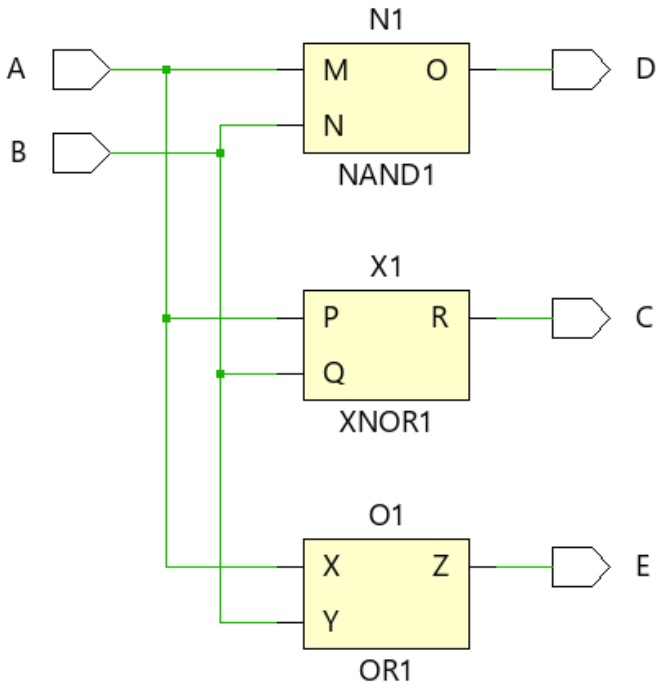
Gambar 2. 2 One Entity Multiple Architecture

2.3. Contoh Soal

1.

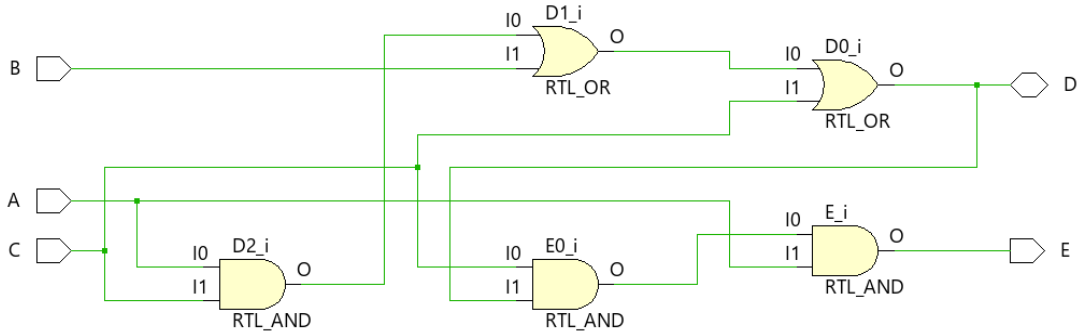


2.

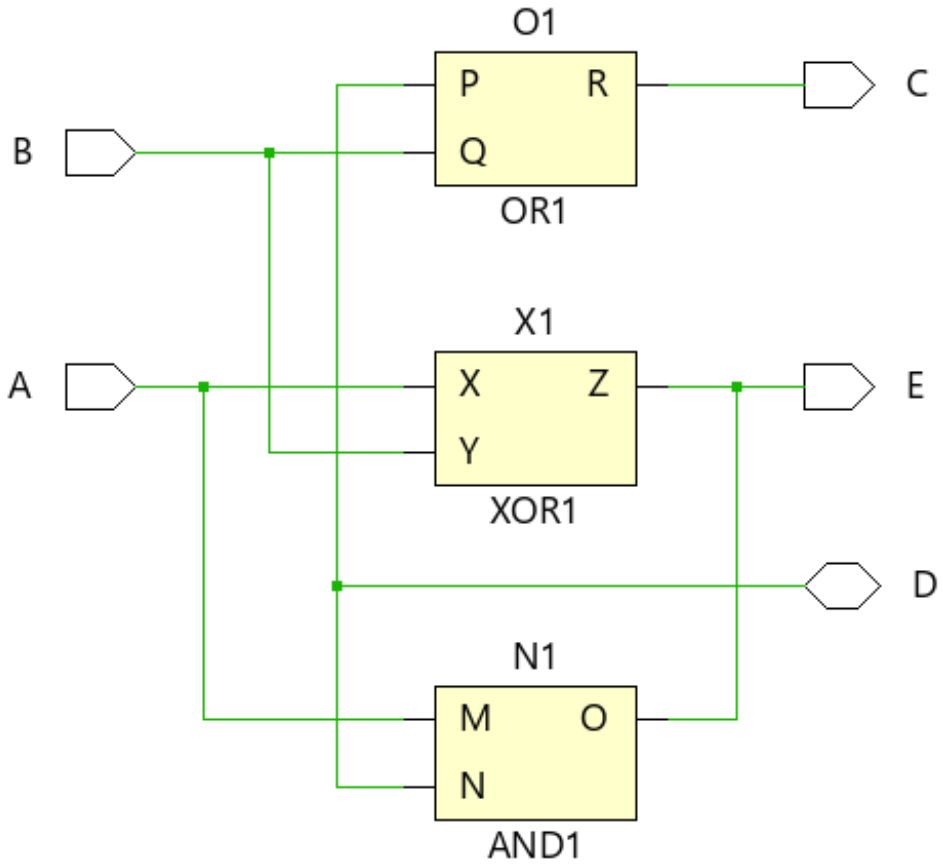


2.4. Latihan Soal

1.



2.



3. Buatlah program dengan *architecture behavioral* Full Adder menggunakan *truth table* di bawah!

Inputs			Outputs	
A	**B**	**C_{in}**	**Sum**	**Carry**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

BAB 3

Tipe Data

Pada bab ini, praktikan akan mempelajari konsep dasar architecture dalam VHDL, yang digunakan untuk mendeskripsikan perilaku internal dari sebuah entity. Praktikan akan mempelajari cara membuat architecture sederhana dan bagaimana architecture berfungsi dalam merancang desain yang lebih besar dan kompleks. Asisten praktikum atau praktikan diharapkan membaca tujuan dan persyaratan bab ini agar praktikum dapat berjalan sesuai prosedur

Tujuan

Tujuan	Penjelasan
Mengenal tipe data pada VHDL	Praktikan diharapkan dapat memahami berbagai tipe data yang tersedia dalam VHDL, seperti std_logic, bit, integer, dan boolean, serta bagaimana setiap tipe data berfungsi dalam mendesain rangkaian digital.
Membedakan penggunaan tipe data	Praktikan diharapkan dapat membedakan penggunaan berbagai tipe data sesuai dengan kebutuhan desain, memahami kapan menggunakan tipe data tertentu untuk memodelkan perilaku dan karakteristik rangkaian yang berbeda.
Menggunakan tipe data array untuk membuat architeture	Praktikan diharapkan dapat menggunakan tipe data array dalam VHDL untuk membuat architecture yang lebih kompleks.

Persyaratan

Disarankan praktikan menggunakan hardware dan software sesuai pada dokumentasi ini. Apabila terdapat versi hardware atau software yang cukup lama dari versi yang direkommendasikan maka sebaiknya bertanya kepada Asisten Mengajar Shift.

HARDWARE YANG DIBUTUHKAN PRAKTIKUM
PC / Laptop
FPGA Board Nexys A7 dan Nexys 4
Kabel Power USB

SOFTWARE YANG DIBUTUHKAN PRAKTIKUM
Vivado Design Suite

⚠ Diusahakan untuk memakai **Versi** dan **Aplikasi** yang sama agar tidak terjadinya kesalahan yang tidak diinginkan!

3.1. Pengertian

Tipe data adalah atribut yang dilampirkan ke data untuk menyusun VHDL dalam memahami cara memperlakukan data tersebut. Sederhananya, ini merupakan perintah khusus yang memberi tahu *compiler* pada VHDL apa yang harus dilakukan pada data tersebut dan bagaimana mengolahnya. Pada VHDL, kita mendefinisikan tipe data pada saat menginisialisasi sinyal, variabel, konstanta, dan generik. Untuk menuliskan kode VHDL secara efisien, sangatlah penting untuk mengetahui tipe-tipe data yang diperbolehkan, bagaimana, serta kapan penggunaannya.

3.2. Tipe Data VHDL

VHDL memiliki seperangkat tipe data standar (*predefined*/*built-in*) dan Juga terdapat tipe data dan subtipe yang didefinisikan pengguna (*user defined*). Artinya, dalam VHDL tipe data dapat diklasifikasikan menjadi berikut.

3.2.1. Tipe Data Predefined

VHDL memiliki beberapa tipe data standar (*predefined*) yang sudah didefinisikan sebelumnya oleh bahasa pemrograman. Tipe data ini menyediakan dasar yang kuat untuk membangun berbagai jenis rangkaian digital. Berikut adalah beberapa tipe data standar yang umum digunakan dalam VHDL :

3.2.1.1. Bit

Bit merupakan tipe data paling sederhana dan terpenting untuk sistem digital dan memiliki nilai '0' dan '1'. Setiap objek data dapat dideklarasikan sebagai tipe bit sebelum digunakan sebagai Sinyal, Variabel, atau Konstanta.

Variable temp : BIT := 0;

Signal CLK : BIT := 1;

3.2.1.2. Boolean

Boolean hanya memiliki satu nilai yaitu “TRUE” atau “FALSE”. Operasi yang dapat dilakukan pada variable Boolean dengan gerbang logika adalah gerbang “AND, OR, NOT, NAND, NOR, dan XOR”.

Variable DONE : BOOLEAN := FALSE;

Signal enable : BOOLEAN := TRUE;

3.2.1.3. Numeric

Sesuai dengan namanya, tipe data ini hanya dapat menampung nilai-nilai numerik. Tipe data numerik terdiri dari :

- *Integer*

Tipe data ini dapat menampung nilai dari -2.147.483.647 s/d +2.147.483.647. Selain itu, integer juga memiliki dua buah sub tipe yang sudah di definisikan pada library, yaitu :

a. Natural, memiliki range nilai dari 0 s/d +2.147.483.647.

b. Positif, memiliki range 1 s/d +2.147.483.647.

Variable VALUE : natural := 2;

Variable VALUE : positive := 2;

- *Real*

Tipe data ini dapat menampung floating point mulai dari -1.0E38 hingga+1.0E38. Selain itu, tipe data ini juga dapat membaca bilangan pi (π) untuk beberapa perhitungan. Bilangan real dapat menyimpan hingga 6 angka dibelakangkoma. Contoh :

Constant Pi : real := 3.14159;

3.2.1.4. Character

Tipe data ini dapat menampung karakter baik itu yang biasa maupun karakter khusus.

Variable VAL : character := '$';

Semua karakter yang termasuk dalam tipe data ini adalah sebagai berikut :

```
NUL, SOH, STX, ETX, EOT, ENQ, ACK,
BEL, BS, HT, LF, VT, FF, CR,
SO, SI, DLE, DCI, DC2, DC3, DC4,
NAK, SYN, ETB, CAN, EM, SUB, ESC,
FSP, GSP, RSP, USP,
' ' , ' ! ', ' " ', ' # ', ' $ ', ' % ', ' & ',
' ' ', ' (' , ' ) ', ' * ', ' + ', ' , ', ' - ',
' . ', ' / ', ' 0 ', ' 1 ', ' 2 ', ' 3 ', ' 4 ',
' 5 ', ' 6 ', ' 7 ', ' 8 ', ' 9', ' : ', ' ; ',
' < ', ' = ', ' > ', ' ? ',
' @ ', ' A ', ' B ', ' C ', ' D ', ' E ', ' F ',
' G ', ' H ', ' I ', ' J ', ' K ', ' L ', ' M ',
' N ', ' O ', ' P ', ' Q ', ' R ', ' S ', ' T ',
' U ', ' V ', ' W ', ' X ', ' Y ', ' Z ', ' [ ',
' \ ', ' ] ', ' ^ ', ' _ ',
' ` ', ' a ', ' b ', ' c ', ' d ', ' e ', ' f ',
' g ', ' h ', ' i ', ' j ', ' k ', ' l ', ' m ',
' n ', ' o ', ' p ', ' q ', ' r ', ' s ', ' t ',
' u ', ' v ', ' w ', ' x ', ' y ', ' z ', ' { ',
' | ', ' } ', ' ~ ', DEL
```

3.2.2. Tipe Data User Defined

Tipe data *useru defined* dalam VHDL memungkinkan pengguna untuk mendefinisikan sendiri tipe data khusus yang sesuai dengan rangkaian yang akan dibuat. Ini memberikan fleksibilitas yang lebih besar dalam memodelkan dan mengelola data dalam desain rangkaian digital**.** Tipe data *user defined* dibagi menjadi 4 tipe yaitu :

3.2.2.1. Tipe Scalar

Tipe scalar adalah jenis data yang hanya dapat menyimpan satu nilai pada satu waktu, maksudnya adalah bahwa sebuah variabel atau sinyal yang dideklarasikan dengan tipe data skalar hanya bisa menyimpan satu nilai tunggal pada saat tertentu, bukan beberapa nilai sekaligus.

- Enumerated, sangat fleksibel dan dapat digunakan untuk merepresentasikan berbagai jenis data yang memiliki jumlah nilai yang terbatas dan terdefinisi. Tipe data ini dapat berupa Bit, Boolean, Numeric dan character.

  signal bit_val : std_logic;

  bit_val <= '1';

- Physical, berisi nilai-nilai yang mewakili pengukuran suatu kuantitas fisik, seperti waktu, panjang, tegangan, dan arus. Nilai-nilai dari tipe ini dinyatakan sebagai kelipatan bilangan bulat dari satuan dasar.

  type CURRENT is range 0 to 1 E-9

  units

  nA; -- (base unit) nano-ampere

  uA = 1000 nA; -- micro-ampere

  mA = 1000 µA; --milli-ampere

  Amp = 1000 mA; -- ampere

  end units;

  dan

  t*ype TIMEis range 0 to -1 E18 to 1 E18*

  units

  pS; -- (base unit) Pico Sec

  nS = 1000pS;

  µS = 1000nS;

  mS = 1000µS;

  Sec= 1000mS

Min= 60Sec

end units;

3.2.2.2. Tipe Composites

Ini adalah tipe data yang digunakan untuk menggabungkan beberapa nilai, selain tipe data standar. memungkin representasi struktur data yang lebih kompleks dalam desain rangkaian digital. tipe data ini memiliki dua macam yaitu :

- Array, digunakan untuk merepresentasikan sebuah objek yang berisi lebih dari satu nilai dengan tipe yang sama. Misalnya, sebuah register delapan bit dapat diwakili sebagai sebuah objek dari tipe array yang terdiri dari nilai-nilai delapan bit. Jadi, delapan nilai dari tipe bit dikelompokkan menjadi satu objek dari tipe array.

  typeADDRESS_WORD is array (0 to 7) of BIT

- *Record*, memungkinkan pengelompokan beberapa elemen yang berbeda jenis menjadi satu entitas tunggal. Dengan kata lain, record memungkinkan kita untuk menggabungkan berbagai tipe data (seperti integer, boolean, std_logic, dll.) dalam satu struktur, yang memudahkan organisasi dan pengelolaan data yang kompleks dalam desain perangkat keras.

  type MODULE is

  record

  SIZE: INTEGER range 20 to 200;

  CRITICAL_DLY: TIME;

  NO_INPUTS: PIN_TYPE:

  NO_OUTPUTS: PIN_TYPE;

  end record;

  type DATEis

  record

  DAY: integer_range 1 to 31;

  MONTH:month_name;

  YEAR:integer_range0 to 4000;

  end record;

- Tipe Access

  Tipe *Access* digunakan untuk mendeklarasikan *pointer*, yaitu referensi ke objek lain dalam memori. Tipe *data access* memungkinkan akses dinamis ke objek dan data.

  variable p : int_ptr;

p := new integer;

p.all := 5; -- Mengisi nilai 5 ke memori yang ditunjuk oleh p

report integer'image(p.all); -- Menampilkan nilai yang dirujuk oleh p, yaitu 5

- Tipe File

  Tipe File digunakan untuk membaca dan menulis data dari dan ke file eksternal selama proses simulasi. Ini memungkinkan pengujian, debugging, dan pengumpulan data dengan cara yang lebih fleksibel dan terstruktur.

  file my_file : text is in "input_data.txt";

3.2.3. Operator

Operator dalam VHDL terdiri dari berbagai berbagai macam operator untuk melakukan operasi matematika, logika, dan perbandingan. Berikut adalah beberapa jenis operator yang umum digunakan dalam VHDL.

3.2.3.1. Operator Aritmatika

- Penjumlahan : +
- Pengurangan : -
- Perkalian : *
- Pembagian : /
- Modulus : mod (menghasilkan sisa pembagian)

3.2.3.2. Operator Logika

- AND : perlu semua operand bernilai TRUE untuk menghasilkan TRUE.
- OR : perlu setidaknya satu operand bernilai TRUE untuk menghasilkan TRUE.
- NOT : membalikkan nilai boolean.

3.2.3.3. Operator Perbandingan

- Sama dengan : =
- Tidak sama dengan : /=
- Lebih besar : >
- Lebih kecil : <
- Lebih besar atau sama dengan : >=
- Lebih kecil atau sama dengan : <=

3.2.3.4. Operator Khusus

- Negasi : - (untuk bilangan)
- Konkatenasi : & (untuk menggabungkan string)
- Akses elemen array : () (untuk mengakses elemen tertentu dalam array)

3.3. Array

Array adalah struktur data yang terdiri dari kumpulan elemen yang memiliki tipe data yang sama, disusun dalam urutan tertentu, dan diakses menggunakan indeks. Dalam konteks FPGA, array digunakan untuk menyimpan dan mengelola data secara efisien.

3.3.1. Jenis Array

Terdapat 2 jenis array yang ditentukan dalam standar VHDL, yaitu:

- String, merupakan karakter yang digunakan untuk menyimpan teks, seperti pesan atau data komunikasi.

  Variable MESSAGE : STRING (1 to 10) := "Acsl";

- Bit Vector, merupakan sebuah array dari tipe data BIT yang dikelompokkan. Biasanya digunakan untuk mendefinisikan banyak *input* pin. Jika kita ingin membuat sebuah rangkaian dengan 4 *input*, daripada mendefinisikannya satu persatu, kita dapat membuatnya lebih sederhana dengan menggunakan BIT vector ini. Contoh :

  Signal input : BIT_VECTOR (3 downto 0);

  Pernyataan tersebut mendefinisikan *input* 4 bit. Untuk mengaturnya, kita dapat menggunakan input (0) untuk mengakses bit pertama dan (1) untuk bit kedua, begitu seterusnya. Contoh :

  Input <= "0101";

3.3.2. Karakteristik Array

- Semua elemen dalam array memiliki tipe data yang sama.
- Setiap elemen dalam array diakses menggunakan indeks, yang biasanya dimulai dari 0.
- Ukuran array biasanya ditentukan saat deklarasi dan tidak dapat diubah selama *runtime* atau saat sedang dijalankan.

3.3.3. Contoh Program

1. Contoh dibawah ini adalah BIT vector sederhana yang meringkas 2 *input* A.

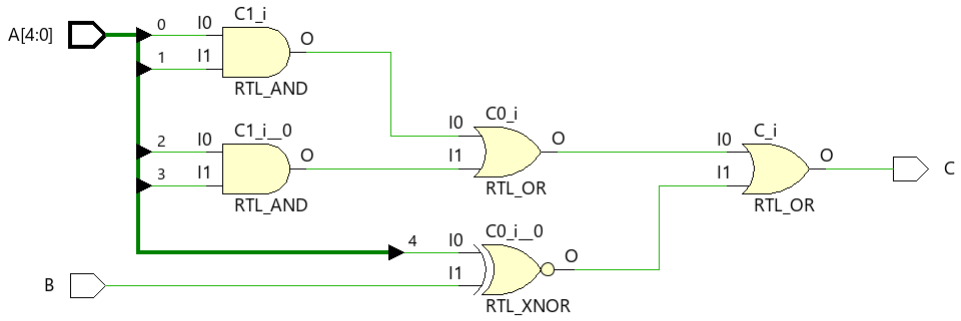
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bab3array is
    Port ( A : in std_logic_vector (1 downto 0);
           B : in std_logic;
           C : out std_logic);
end bab3array;

architecture Behavioral of bab3array is

begin
C <= (A(0) and A(1))or B;

end Behavioral;
```

2. Contoh dibawah ini memiliki *output* setiap array vector merupakan kebaliki dari dari *input*-nya.

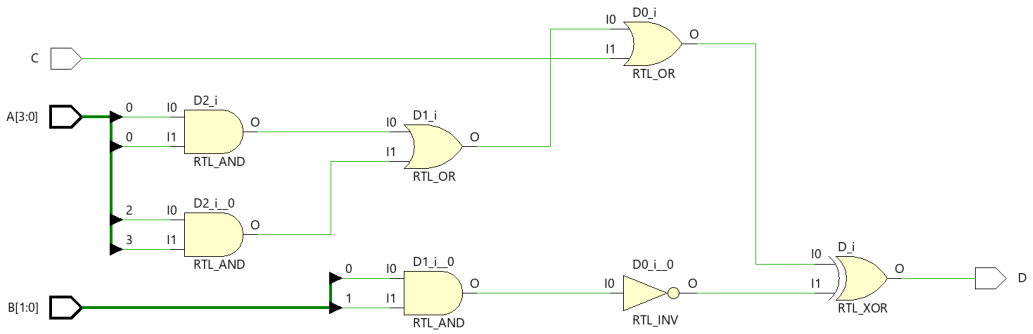
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bit_vector_example is
    Port (
        input_vector  : in  bit_vector(3 downto 0);
        output_vector : out bit_vector(3 downto 0)
    );
end bit_vector_example;

architecture Behavioral of bit_vector_example is
begin

    process(input_vector)
    begin
        output_vector <= not input_vector;
    end process;

end Behavioral;
```

3.4. Latihan Soal

1.



2.


C
A[3:0]
B[1:0]
D2_i
RTL_AND
D2_i_0
RTL_AND
D1_i
RTL_OR
D1_i_0
RTL_AND
D0_i
RTL_OR
D0_i_0
RTL_INV
D_i
RTL_XOR
D
I0
I1
O