BAB 1

Entity & Architecture

Pada bab ini, praktikan akan mempelajari konsep entity & Architecture dalam VHDL, yang merupakan bagian penting dalam mendefinisikan antarmuka modul pada desain digital. Selain itu, praktikan juga akan mempelajari cara membuat entity sederhana, 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 pada bab ini agar praktikum dapat berjalan sesuai prosedur.

Tujuan

Tujuan	**Penjelasan**
Mengenal dan memahami tentang entity & Architecture dalam VHDL	Praktikan diharapkan dapat memahami konsep dasar entity & Architecture dalam VHDL, jenis-jenis architecture, bagaimana architecture dan entity bekerja bersama dalam mendesain rangkaian digital.
Membuat entity & Architecture sederhana	Praktikan diharapkan dapat membuat entity & Architecture 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 direkomendasikan 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!

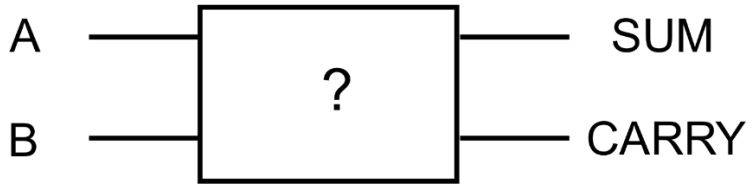
Teori

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 variabel 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*".

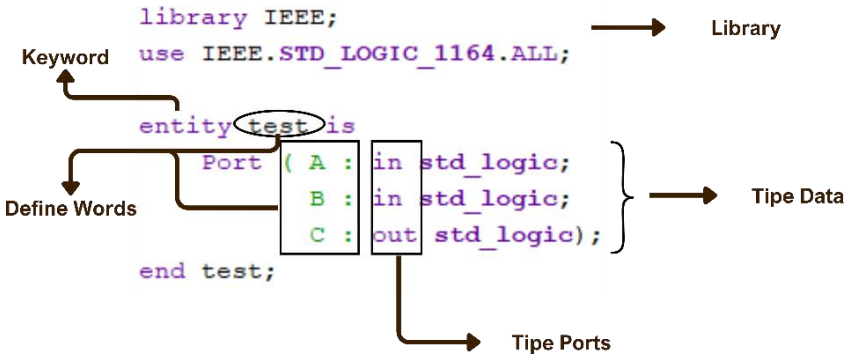
Entity memberikan arti tentang bagaimana sebuah bagian rancangan dideskripsikan di VHDL dalam hubungannya dengan model VHDL lain dan juga memberikan nama untuk model tersebut. Di dalam *entity* juga diperbolehkan untuk mendefinisikan beberapa parameter yang mengambil model menggunakan hierarki. Kerangka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

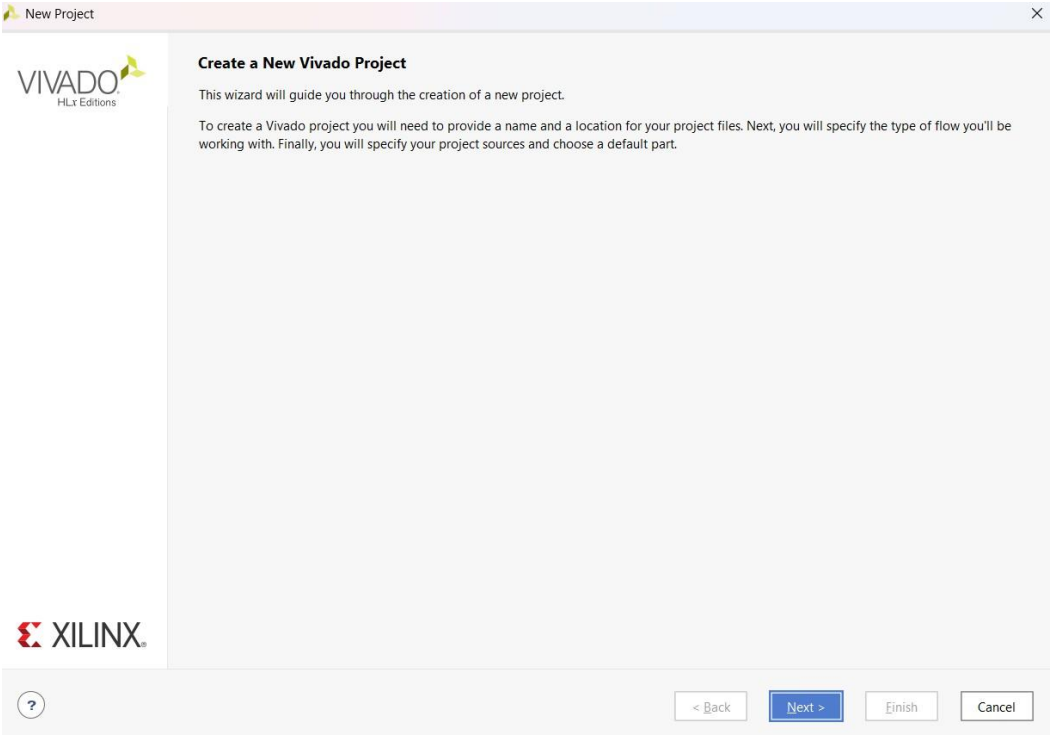
- *Entity*, digunakan untuk menyatakan *entity* pada VHDL.
- *Architecture*, digunakan untuk menyatakan *architecture* pada VHDL.
- *Process*, digunakan untuk tugas-tugas seperti membuat proses yang dikendalikan oleh *clock* dan operasi kondisional.
- *Signal* dan *Variable*, digunakan untuk membuat 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.

Praktek

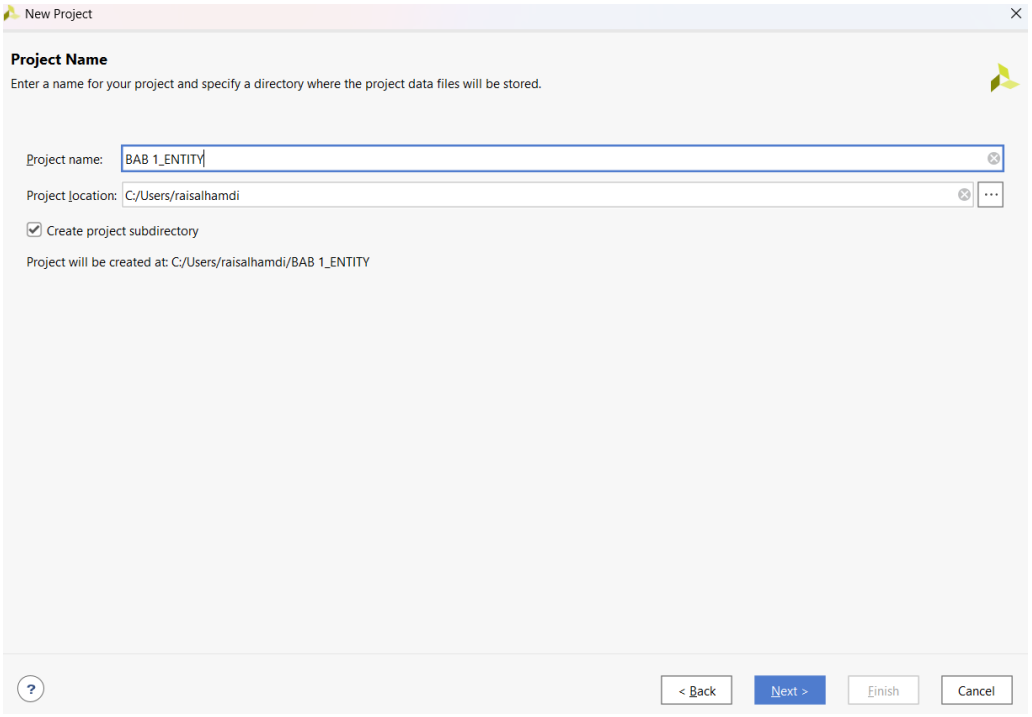
Membuat Entity & Architecture Sederhana

1. Buka aplikasi Vivado 2020

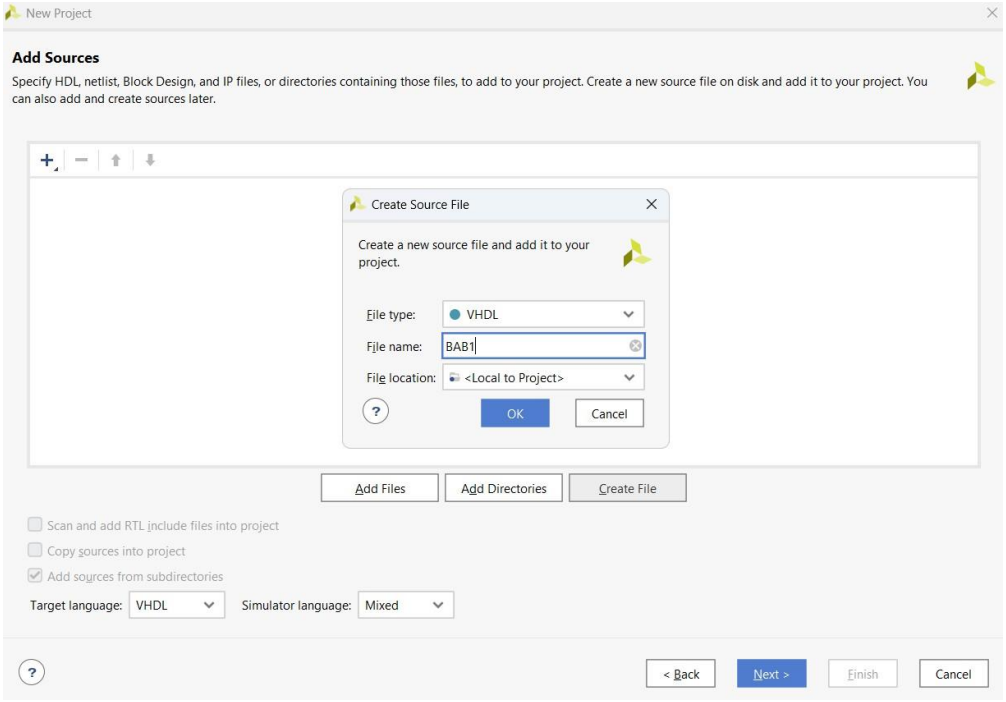
2. Buka menu File > Project > New Project



3. Kemudian akan muncul tampilan diatas dan kemudian pilih next.

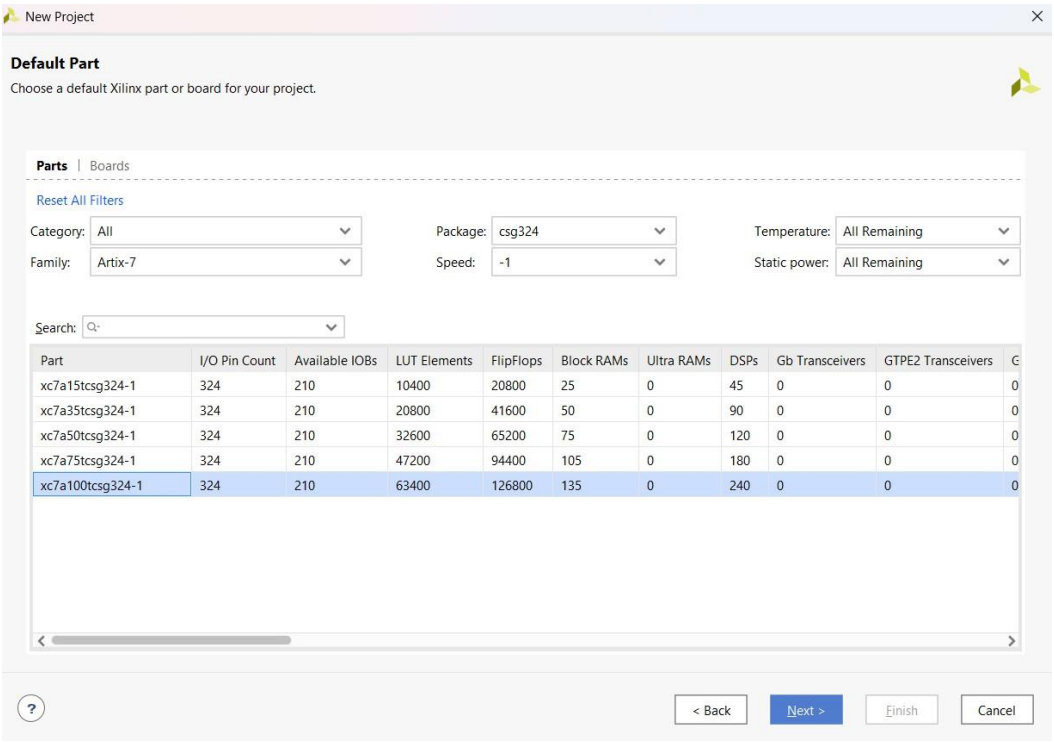


4. Buatlah nama project seperti diatas kemudian pilih next.

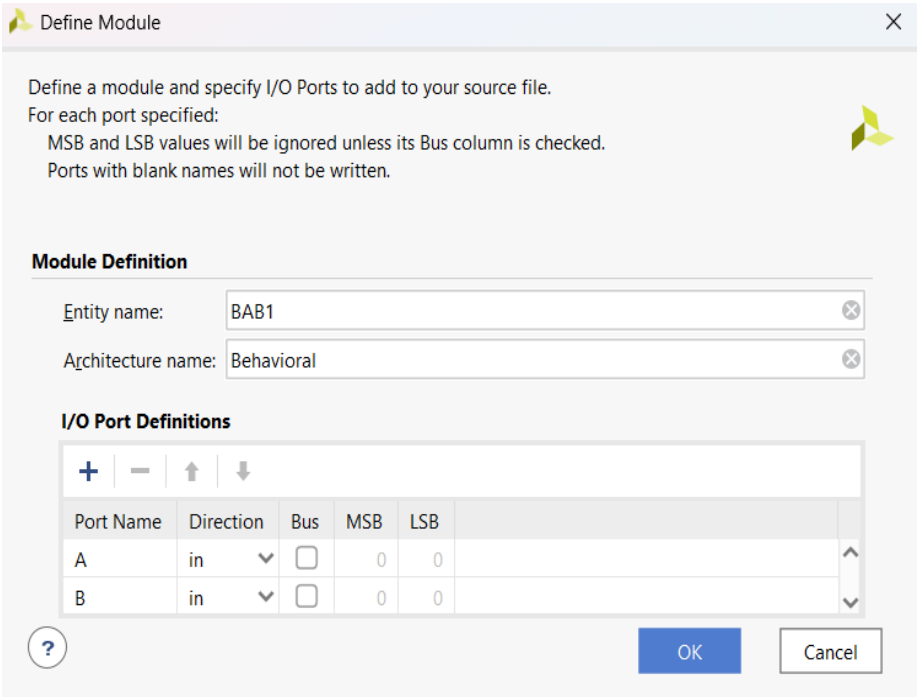


5. Kemudian muncul tampilan menu baru yaitu *add source* lalu pastikan target *language* yang digunakan adalah VHDL, lalu pilih *create file*.

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



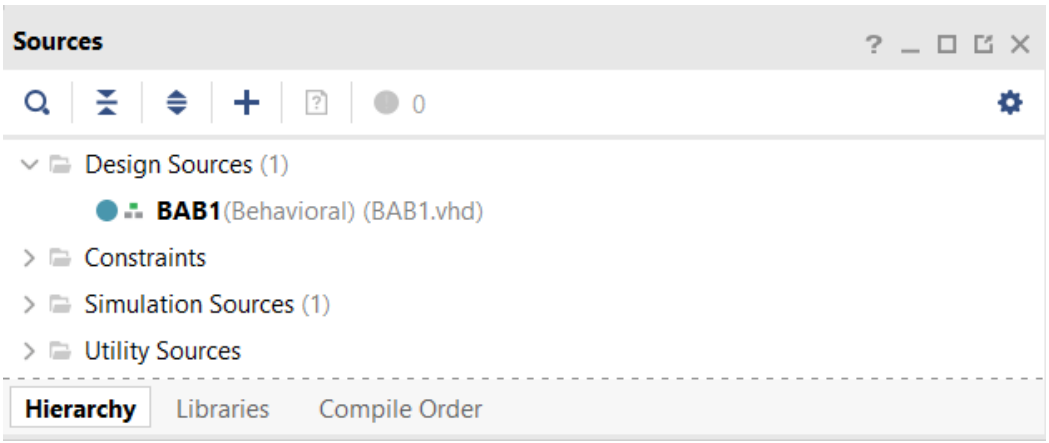
7. Lalu pada tampilan default part pastikan *family, package* dan *speed* yang digunakan seperti gambar diatas lalu pilih *board* yang akan digunakan yaitu **xc7a100tcsg324-1**.



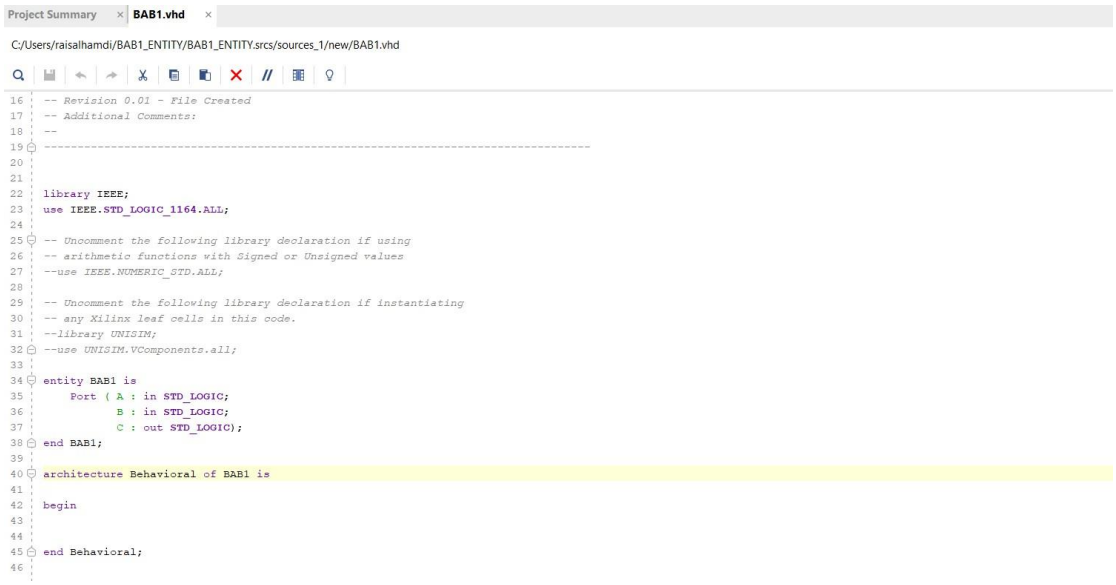
8. Setelah itu akan muncul menu baru yaitu *define module* yang digunakan untuk mendefinisikan port-port yang akan digunakan.

9. Buatlah *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.



12. Berikutnya akan muncul tampilan program seperti gambar diatas.

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

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*,* dan lain-lain yang berfungsi untuk menggunakan komponen dalam sebuah desain dengan *library,* untuk menentukan pustaka yang akan dicari dan pernyataan *use* untuk setiap paket yang akan digunakan. 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;
```

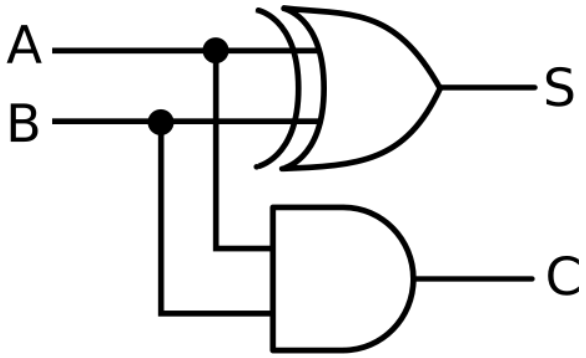
Gambar 1. 4 Jenis-jenis Library

- 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. Jenis-jenis Architecture

1.5.1. 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;
```

1.5.2. Structural

Architecture jenis ini memerlukan definisi dari setiap komponen yang dipakai. Setiap komponen, harus didefinisikan 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;
```

1.5.3. 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

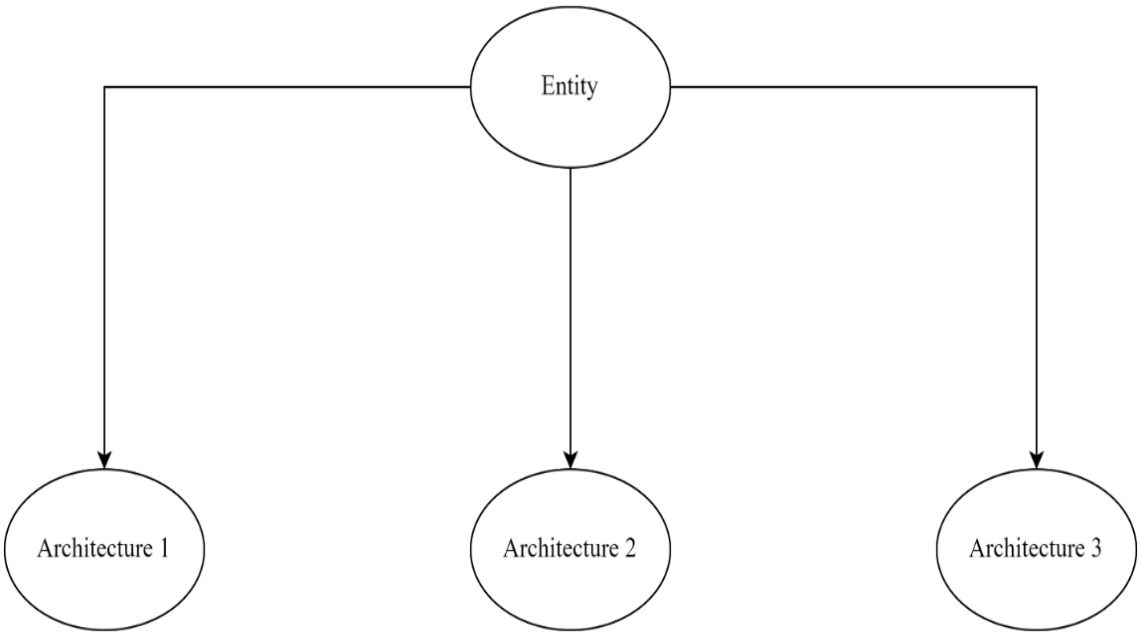
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

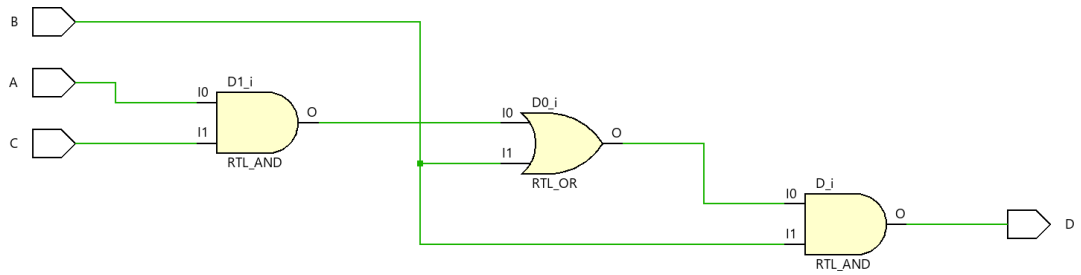
didefinisikan semua untuk mendapatkan hasil yang diinginkan yaitu rangkaian *half adder*. Dalam suatu desain program VHDL, sangat memungkinkan untuk menggunakan lebih dari satu architecture tergantung dari kebutuhan. Namun beberapa architecture tersebut tetap harus memiliki sebuah entity **"One Entity Multiple Architecture"**.



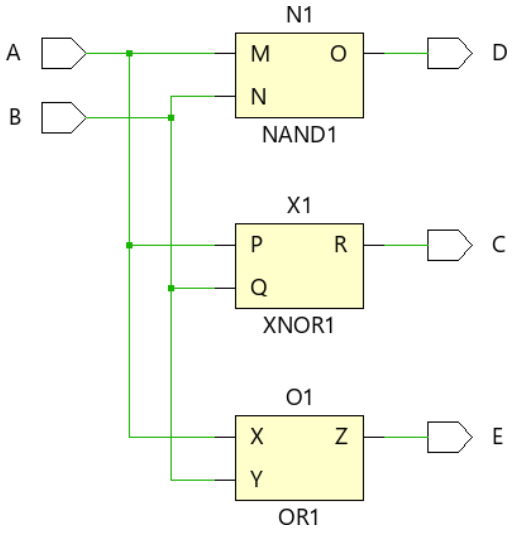
Gambar 2. 2 One Entity Multiple Architecture

Soal

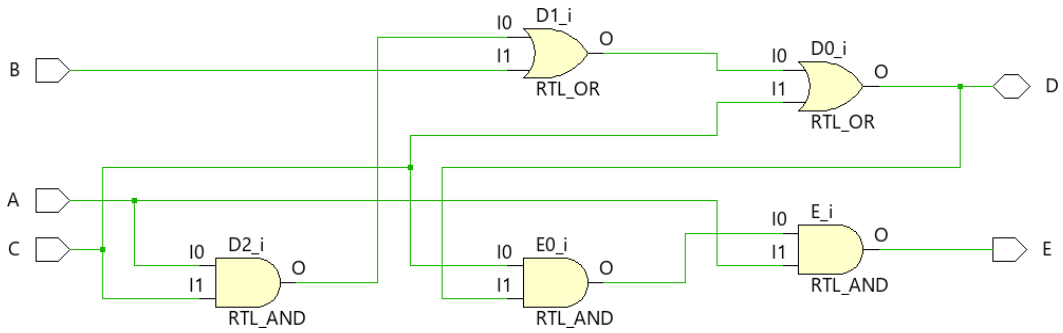
1.


B
A
C
D1_i
I0
I1
O
RTL_AND
D0_i
I0
I1
O
RTL_OR
D_i
I0
I1
O
RTL_AND
D

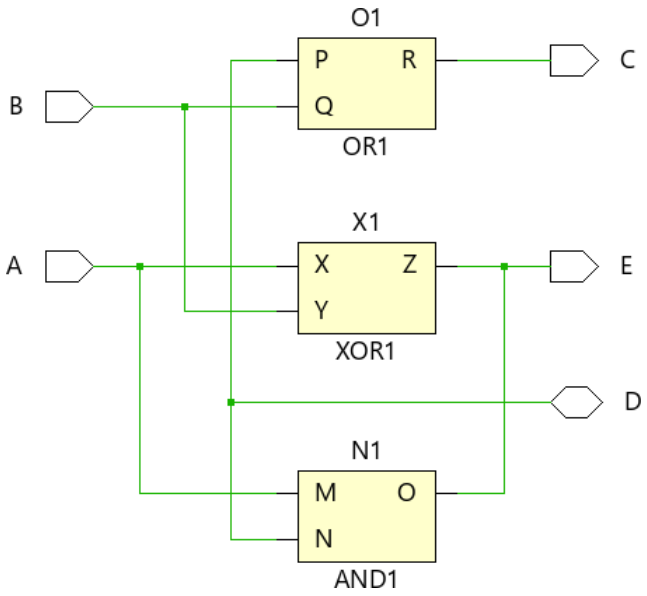
2.


N1
A
B
M
N
O
NAND1
D
X1
P
Q
R
XNOR1
C
O1
X
Y
Z
OR1
E

3.


B
A
C
D1_i
I0
I1
O
RTL_OR
D0_i
I0
I1
O
RTL_OR
D
D2_i
I0
I1
O
RTL_AND
E0_i
I0
I1
O
RTL_AND
E_i
I0
I1
O
RTL_AND
E

4.



5. 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