BAB 7
ARITHMETIC LOGIC UNIT

Pada bab ini, praktikan akan mempelajari konsep dasar dari Arithmetic Logic Unit (ALU) serta implementasinya dalam sistem digital menggunakan FPGA. ALU merupakan komponen penting dalam pemrosesan data yang mampu melakukan operasi aritmatika seperti penjumlahan dan pengurangan, serta operasi logika. Asisten praktikum atau praktikan diharapkan membaca tujuan dan persyaratan bab ini agar praktikum dapat berjalan sesuai dengan prosedur.

Tujuan

Tujuan	Penjelasan
Memahami Konsep ALU (Arithmetic Logic Unit)	Praktikan diharapkan dapat memahami konsep dasar dari ALU, termasuk operasi aritmatika dan logika yang dapat dilakukan oleh unit ini dalam sebuah sistem digital.
Mengimplementasikan ALU 3-bit pada FPGA	Praktikan diharapkan mampu membuat dan memprogram ALU sederhana yang dapat melakukan operasi penjumlahan, pengurangan, dan logika dasar menggunakan FPGA.

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

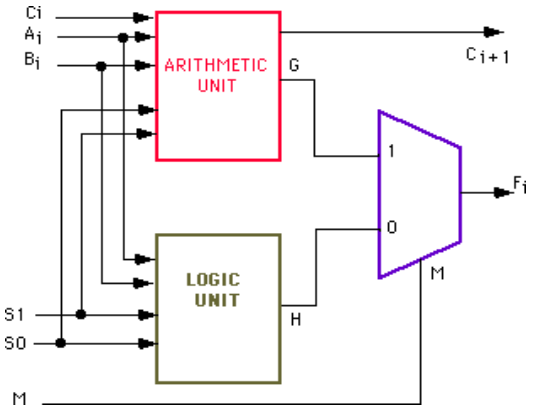
⚠ Diusahakan untuk memakai **Versi** dan **Aplikasi** yang sama agar tidak terjadinya kesalahan yang tidak diinginkan!

Teori

7.1. Pengertian ALU

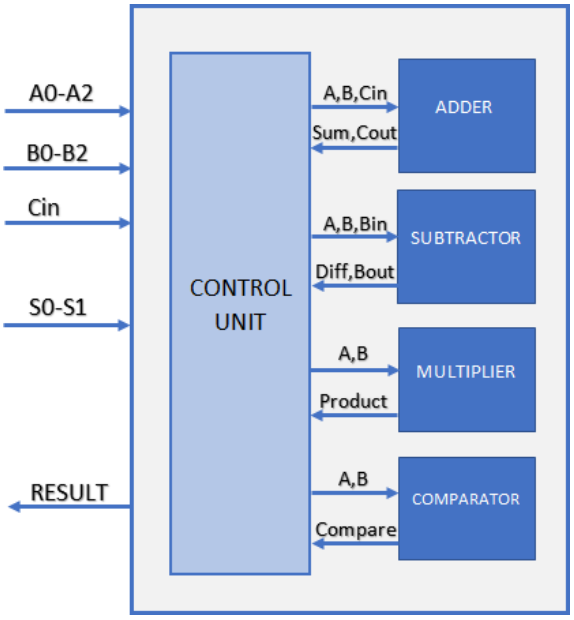
ALU (Arithmetic Logic Unit) adalah komponen utama dari Central Processing Unit atau CPU yang bertugas untuk melakukan operasi aritmatika dan operasi logika menggunakan nilai biner. ALU dapat melakukan operasi aritmatika dan logika berdasarkan input yang dimasukkan dan operasi apa yang akan dilakukan dengan memilih pada bagian operand atau selector seperti operasi pertambahan, pengurangan, perkalian dan lain lain. Unit pemrosesan pada ALU dibagi menjadi dua bagian yaitu AU (Arithmetic Unit) dan LU (Logic Unit).

- Arithmetic Unit : Bagian dari ALU yang melakukan operasi matematika yang termasuk dalam operasi sederhana yaitu pertambahan, pengurangan, perkalian,dan pembagian. Arithmetic Unit bekerja dengan menggunakan bilangan biner sebagai input ataupun sebagai outputnya,
- Logic Unit : Bagian dari ALU yang melakukan operasi logika berdasarkan gerbang-gerbang logika dasar seperti AND, OR, NOT dan gerbang logika lainnya, selain operasi logika bagian ini juga dapat melakukan operasi perbandingan dan operasi bitwise yang digunakan untuk menggeser dan memutar nilai bit.



Gambar 7. 1 Rangkaian ALU

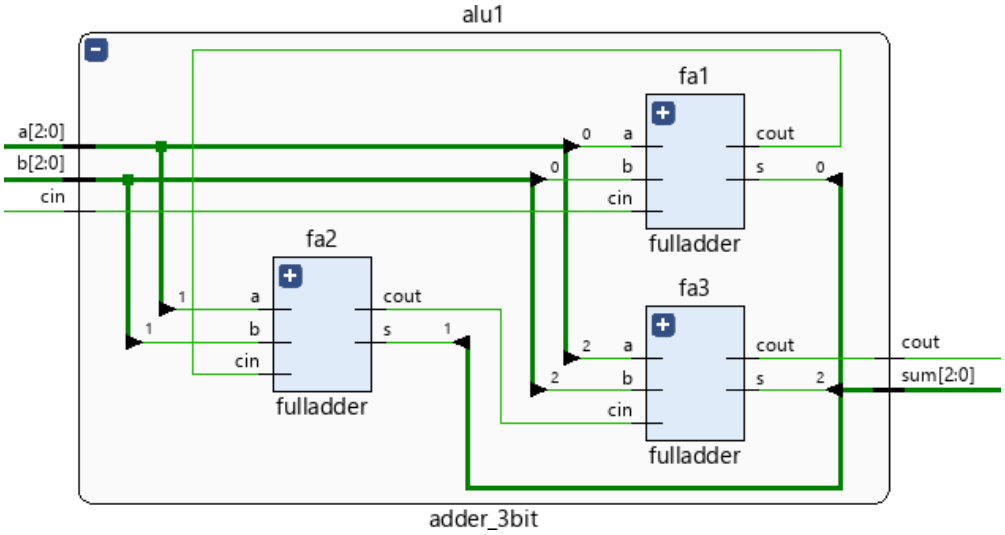
ALU secara internal selalu melakukan beberapa operasi seperti penjumlahan, pengurangan, pembagian, dan perkalian. Kita harus menentukan hasil mana yang ingin dihasilkan sebagai output dengan menggunakan operand atau selector. Dalam praktikum ini, kita akan mengimplementasikan ALU 3-bit yang akan melakukan operasi aritmatika menggunakan bilangan biner dan operasi yang akan dilakukan adalah Adder (penjumlahan), Subtractor (pengurangan), Multiplier (perkalian), dan Comparator (pembanding).



Gambar 7. 2 3-Bit ALU

7.1.1. 3-Bit Adder

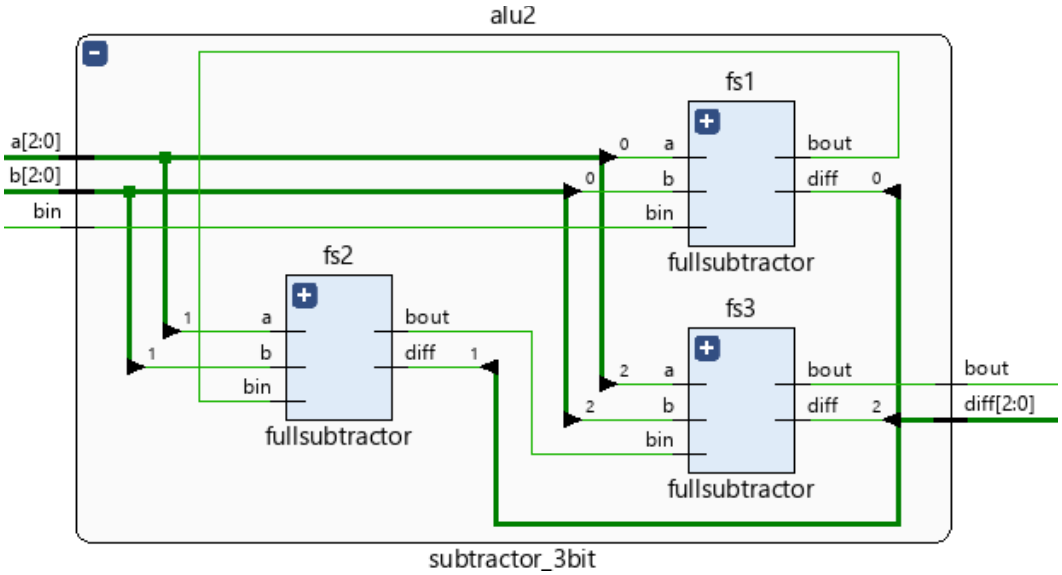
Rangkaian 3-Bit Adder melakukan penjumlahan biner dengan input yang memiliki nilai 3-bit, rangkaian 3-Bit adder dapat dibuat dengan cara menghubungkan 3 rangkaian full adder menjadi satu rangkaian. Setiap full adder menjumlahkan satu bit dari setiap bilangan biner dan carry dari full adder sebelumnya. Output dari 3-bit adder adalah bilangan biner 4-bit, yang merupakan hasil dari penjumlahan dari dua bilangan biner 3-bit.



Gambar 7. 3 Skematik 3-Bit Adder

7.1.2. 3-Bit Subtractor

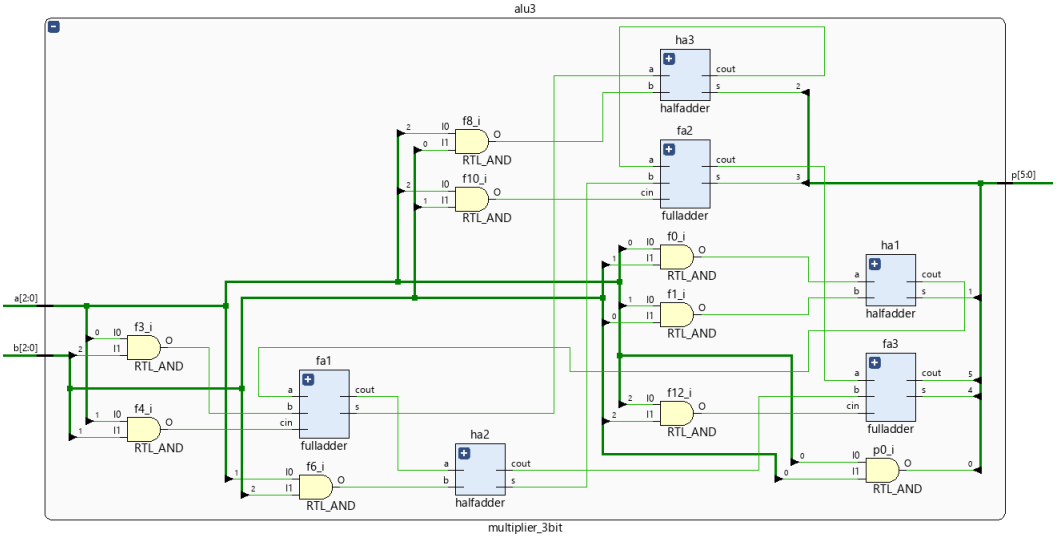
Rangkaian 3-Bit Subtractor melakukan pengurangan biner dengan input yang memiliki nilai 3-bit, rangkaian 3-Bit Subtractor dapat dibuat dengan cara menghubungkan 3 rangkaian full subtractor menjadi satu rangkaian. Setiap full subtractor mengurangkan satu bit dari bilangan biner pertama dengan satu bit dari bilangan biner kedua dan borrow dari full subtractor sebelumnya. Output dari 3-bit subtractor adalah bilangan biner 3-bit yang merupakan selisih antara dua bilangan biner 3-bit dan sebuah borrow output.



Gambar 7. 4 Skematik 3-Bit Subtractor

7.1.3. 3-Bit Multiplier

Rangkaian 3-Bit Multiplier adalah rangkaian yang digunakan untuk melakukan operasi perkalian dua bilangan biner 3-bit. Rangkaian ini menghasilkan output 6 bit yang merupakan hasil dari perkalian dari kedua bilangan.

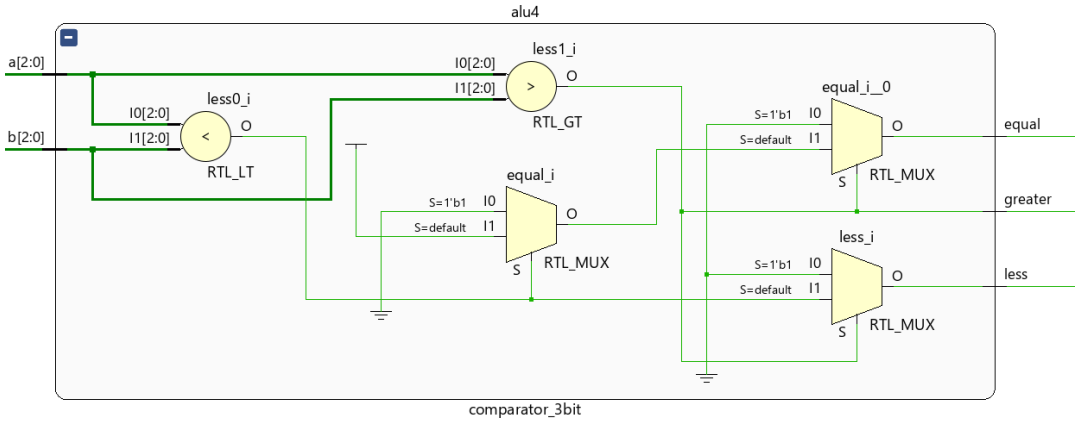


Gambar 7. 5 Skematik 3-Bit Multiplier

7.1.4. 3-Bit Comparator

Comparator adalah rangkaian yang digunakan untuk membandingkan nilai berdasarkan input yang diterimanya. Comparator 3-bit adalah rangkaian yang membandingkan antara kedua input yang memiliki nilai 3-bit dan menghasilkan output berdasarkan perbandingan nilai biner kedua input tersebut. Hasil dari perbandingan dua bilangan biner 3-bit dapat dinyatakan dalam tiga kemungkinan kondisi yaitu :

- A < B: Jika nilai A lebih kecil dari B. (less)
- A = B: Jika nilai A sama dengan B. (equal)
- A > B: Jika nilai A lebih besar dari B. (greater)

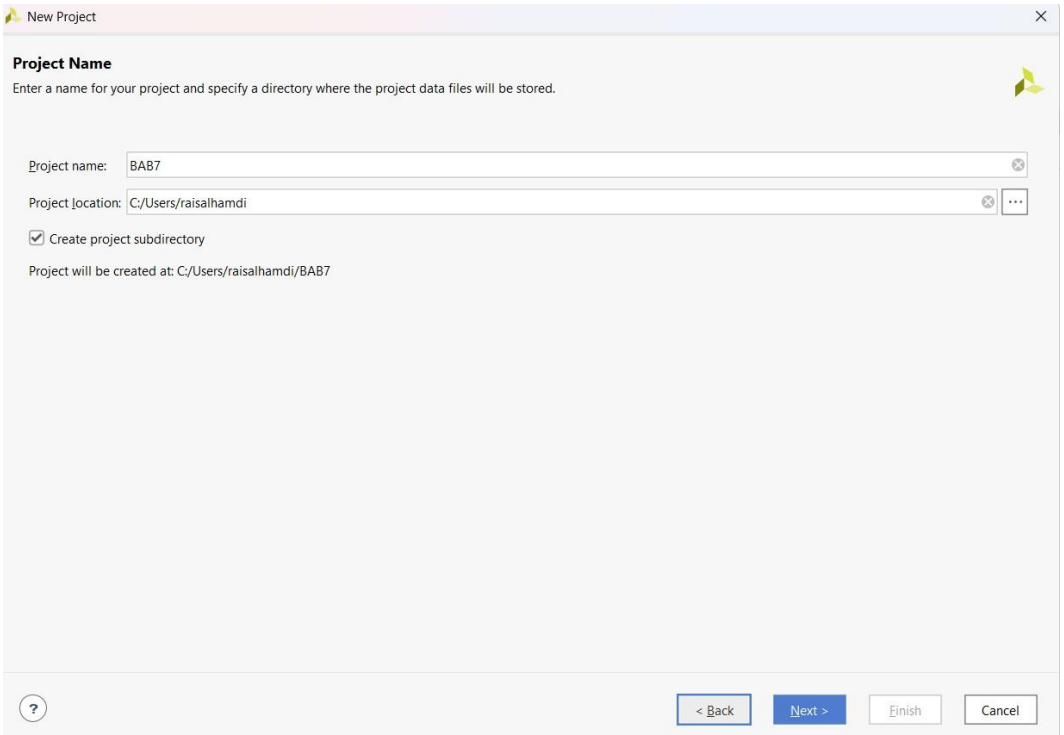


Gambar 7. 6 Skematik 3-Bit Comparator

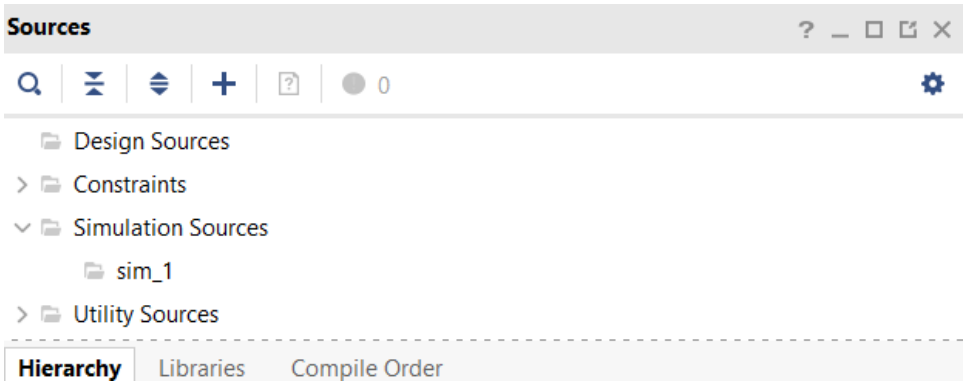
Praktek

- **Membuat Program ALU**

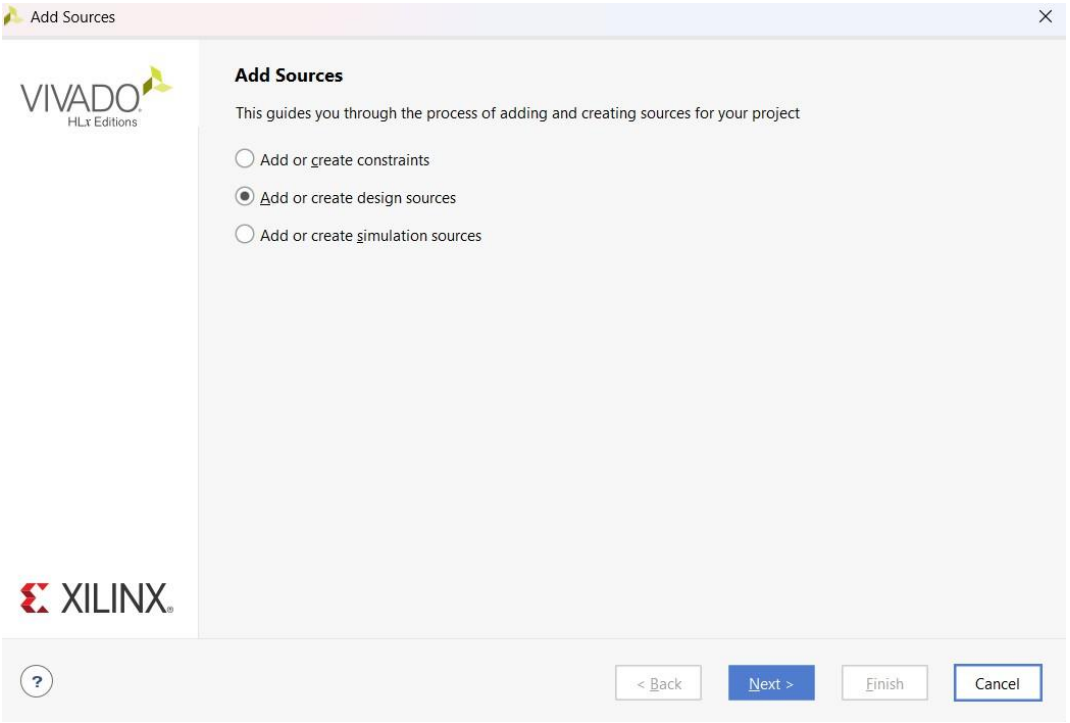
1. Membuka aplikasi Vivado 2020.2
2. Membuat Project baru
3. Buat project dengan nama BAB7_Nama



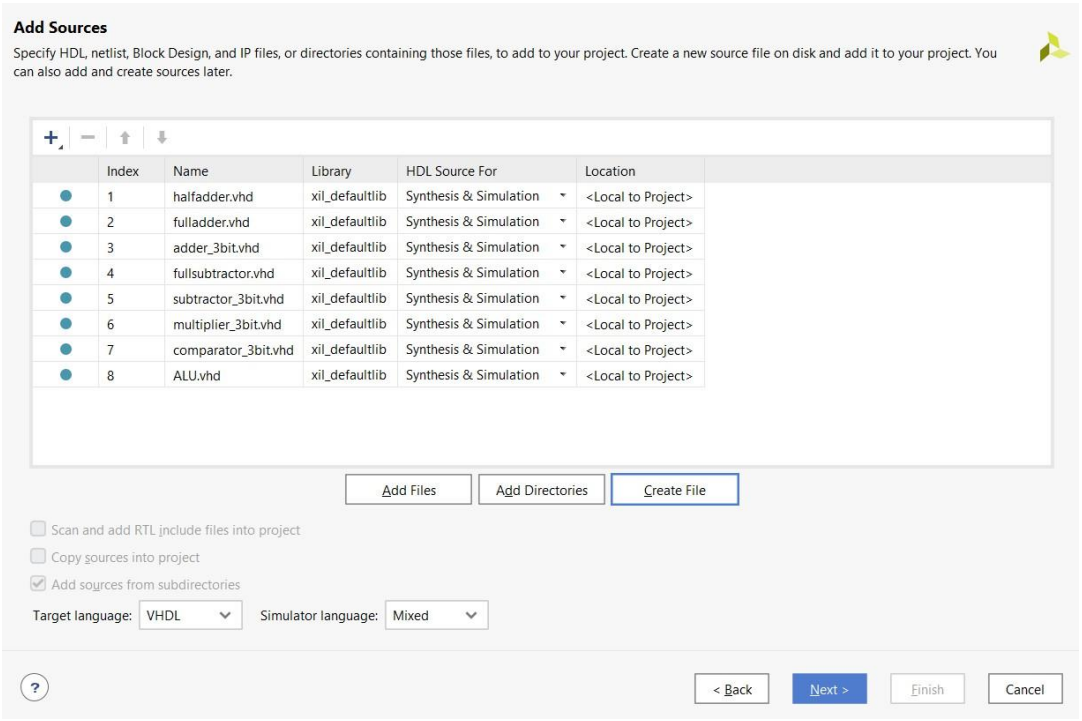
4. Kemudian pada bagian source pilih tanda + untuk membuat source baru.



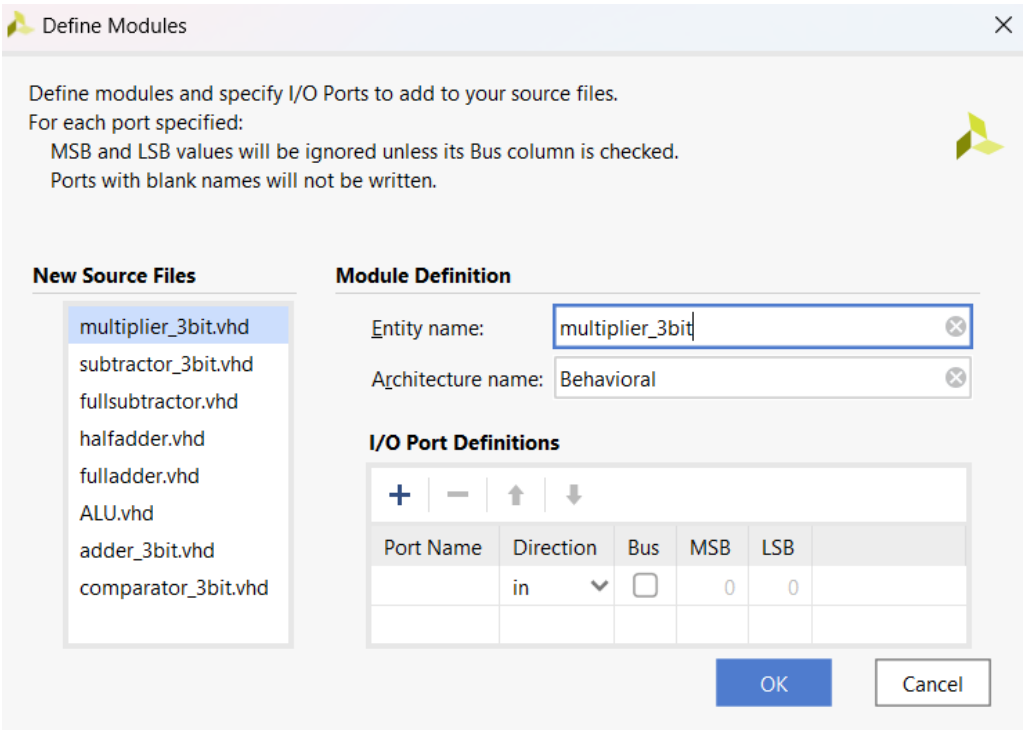
5. Lalu, akan muncul tampilan window baru, pilih add or create design source



6. Buatlah 8 source baru dan beri nama masing-masing source halfadder, fulladder, adder_3bit, fullsubtractor, subtractor_3bit, multiplier_3bit, comparator_3bit dan ALU.



Setelah semua source telah dibuat klik next, lalu akan muncul tampilan define modules pilih OK.



7. Pada bagian source, klik 2 kali pada source halfadder.vhd kemudian masukkan kode berikut.

- **Halfadder**

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

entity halfadder is
Port(   a,b : in std_logic;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end halfadder;

architecture Behavioral of halfadder is
begin
s <= a xor b;
cout <= a and b;
end Behavioral;
```

Setelah kode berhasil ditulis pada source tekan ctrl+s untuk menyimpan source dan kode yang telah ditulis. Ulangi langkah di atas dan masukkan masing- masing kode berikut sesuai dengan nama dari sourcenya.

- **Fulladder**

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

entity fulladder is
port(   a : in STD_LOGIC;
        b : in STD_LOGIC;
        cin : in STD_LOGIC;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end fulladder;

architecture Behavioral of fulladder is
component halfadder
Port(   a,b : in std_logic;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end component;
signal s1,c1,c2:std_logic;

begin
ha1: halfadder port map(a,b,s1,c1);
ha2: halfadder port map(s1,cin,s,c2);
cout <= c1 or c2;
end Behavioral;
```

- **Adder_3bit**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity adder_3bit is
Port (  a : in STD_LOGIC_VECTOR (2 downto 0);
        b : in STD_LOGIC_VECTOR (2 downto 0);
        cin : in STD_LOGIC;
        sum : out STD_LOGIC_VECTOR (2 downto 0);
        cout : out STD_LOGIC);
end adder_3bit;

architecture Behavioral of adder_3bit is

component fulladder
port(   a : in STD_LOGIC;
        b : in STD_LOGIC;
        cin : in STD_LOGIC;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end component;

signal c1,c2:std_logic;
begin
fa1:fulladder port map(a(0),b(0),cin,sum(0),c1);
fa2:fulladder port map(a(1),b(1),c1,sum(1),c2);
fa3:fulladder port map(a(2),b(2),c2,sum(2),cout);
end Behavioral;
```

- **Fullsubtractor**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fullsubtractor is
Port (  a : in STD_LOGIC;
        b : in STD_LOGIC;
        bin : in STD_LOGIC;
        diff : out STD_LOGIC;
        bout : out STD_LOGIC);
end fullsubtractor;
architecture Behavioral of fullsubtractor is
begin
diff <= a xor b xor bin;
bout <= ((not a)and b)or((not a)and bin)or(b and bin);
end Behavioral;
```

- **Subtractor_3bit**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity multiplier_3bit is
Port(   a : in STD_LOGIC_VECTOR (2 downto 0);
        b : in STD_LOGIC_VECTOR (2 downto 0);
        p : out STD_LOGIC_VECTOR (5 downto 0));
end multiplier_3bit;

architecture Behavioral of multiplier_3bit is
component fulladder is
Port(   a : in STD_LOGIC;
        b : in STD_LOGIC;
        cin : in STD_LOGIC;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end component;
component halfadder is
port(   a,b : in std_logic;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end component;
signal br0,br1: std_logic;
begin
fs1: fullsubtractor port map(a(0),b(0),bin,diff(0),br0);
fs2: fullsubtractor port map(a(1),b(1),br0,diff(1),br1);
fs3: fullsubtractor port map(a(2),b(2),br1,diff(2),bout);
end Behavioral;
```

- **Multiplier_3bit**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity multiplier_3bit is
Port(   a : in STD_LOGIC_VECTOR (2 downto 0);
        b : in STD_LOGIC_VECTOR (2 downto 0);
        p : out STD_LOGIC_VECTOR (5 downto 0));
end multiplier_3bit;

architecture Behavioral of multiplier_3bit is
component fulladder is
Port(   a : in STD_LOGIC;
        b : in STD_LOGIC;
        cin : in STD_LOGIC;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end component;
component halfadder is
port(   a,b : in std_logic;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end component;
```

```
signal f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,s1,s2: std_logic;
begin
ha1: halfadder port map(f0,f1,p(1),f2);
fa1: fulladder port map(f2,f3,f4,s1,f5);
ha2: halfadder port map (f5,f6,s2,f7);
ha3: halfadder port map (s1,f8,p(2),f9);
fa2: fulladder port map (f9,s2,f10,p(3),f11);
fa3: fulladder port map (f11,f7,f12,p(4),p(5));
f0 <= a(0) and b(1);
f1 <= a(1) and b(0);
f3 <= a(0) and b(2);
f4 <= a(1) and b(1);
f6 <= a(1) and b(2);
f8 <= a(2) and b(0);
f10<= a(2) and b(1);
f12<= a(2) and b(2);
P(0)<=a(0) and b(0);
end Behavioral;
```

- **Comparator_3bit**

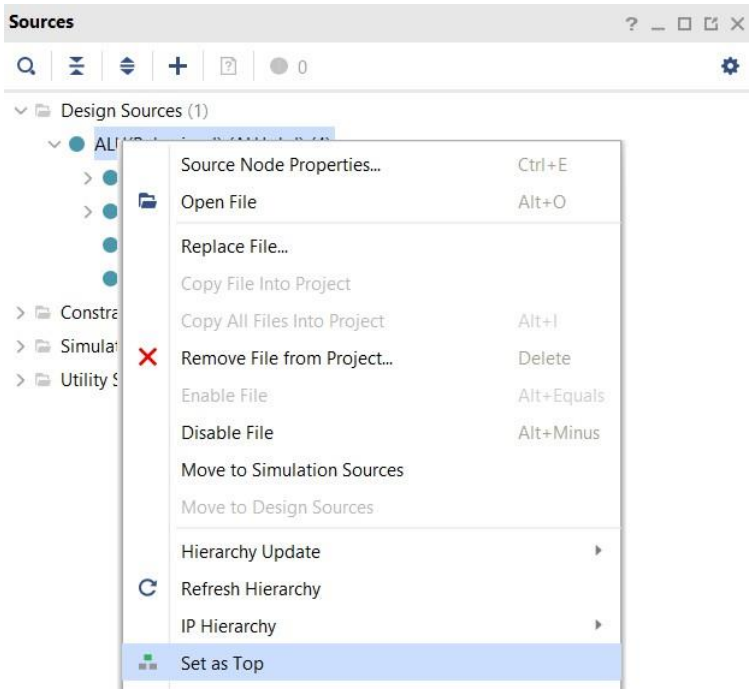
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity comparator_3bit is
Port (  a : in std_logic_vector(2 downto 0);
        b : in std_logic_vector(2 downto 0);
        less : out std_logic;
        equal : out std_logic;
        greater : out std_logic);
end comparator_3bit;
architecture Behavioral of comparator_3bit is
begin
process(a,b)
begin
if (a > b ) then --condition a is greater than b
less <= '0';
equal <= '0';
greater <= '1';
elsif (a < b) then --condition a is less than b
less <= '1';
equal <= '0';
greater <= '0';
else --condition a is equals b
less <= '0';
equal <= '1';
greater <= '0';
end if;
end process;
end Behavioral;
```

ALU

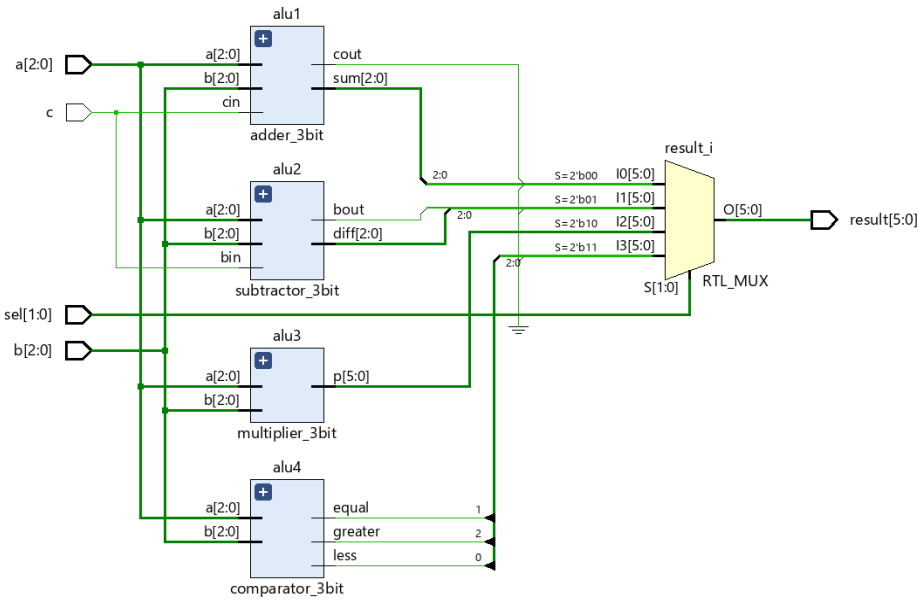
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ALU is
Port (  a : in STD_LOGIC_VECTOR (2 downto 0);
        b : in STD_LOGIC_VECTOR (2 downto 0);
        c: in std_logic;
        sel : in STD_LOGIC_VECTOR (1 downto 0);
        result : out STD_LOGIC_VECTOR (5 downto 0));
end ALU;

architecture Behavioral of ALU is

component comparator_3bit
port(   a : in std_logic_vector(2 downto 0);
        b : in std_logic_vector(2 downto 0);
        less : out std_logic;
        equal : out std_logic;
        greater : out std_logic);
end component;

component multiplier_3bit
Port (  a : in STD_LOGIC_VECTOR (2 downto 0);
        b : in STD_LOGIC_VECTOR (2 downto 0);
        p : out STD_LOGIC_VECTOR (5 downto 0));
end component;

   component subtractor_3bit
   Port (  a : in STD_LOGIC_VECTOR (2 downto 0);
           b : in STD_LOGIC_VECTOR (2 downto 0);
           bin : in STD_LOGIC;
           diff : out STD_LOGIC_VECTOR (2 downto 0);
           bout : out STD_LOGIC);
   end component;

   component adder_3bit is
   Port (  a : in STD_LOGIC_VECTOR (2 downto 0);
           b : in STD_LOGIC_VECTOR (2 downto 0);
           cin : in STD_LOGIC;
           sum : out STD_LOGIC_VECTOR (2 downto 0);
           cout : out STD_LOGIC);
   end component;
   signal t_comp, t_multi, t_sub, t_adder : std_logic_vector(5 downto 0) := (others => '0');

   begin
   alu1: adder_3bit port map(a=>a,b=>b,cin=>c,sum=>t_adder(2 downto 0),cout=>t_adder(3));
   alu2: subtractor_3bit port map(a=>a,b=>b,bin=>c,diff=>t_sub(2 downto 0),bout=>t_sub(3));
   alu3: multiplier_3bit port map(a=>a,b=>b,p=>t_multi);
   alu4: comparator_3bit port map(a=>a,b=>b,less=>t_comp(0),equal=>t_comp(1),greater=>t_comp(2));
      process(sel,t_comp,t_multi,t_sub,t_adder)
      begin
      case sel is
      when "00"=>
           result<=t_adder; when "01"=>
           result<=t_sub; when "10"=>
           result<=t_multi; when "11"=>
           result<=t_comp; when others =>
           result<="000000";
      end case;
      end process;
      end Behavioral;
```

Setelah semua kode untuk masing-masing source berhasil ditulis dan tidak terdapat error, pastikan semua source telah di save.

8. Kemudian pastikan source ALU adalah top module. jika bukan source ALU yang menjadi top module, klik kanan ALU.vhd pada bagian source kemudian pilih set as top.

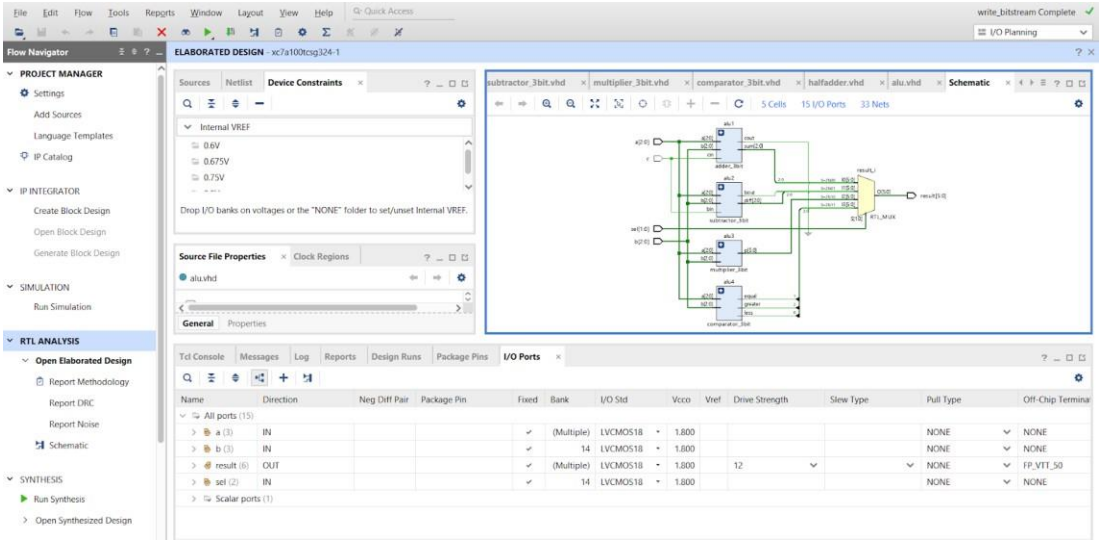


9. Selanjutnya pada bagian RTL analysis, pilih schematic untuk menampilkan rangkaian ALU yang telah dibuat.



Gambar diatas adalah tampilan ALU yang telah berhasil dibuat menggunakan kode vhdl dan dijadikan schematic pada aplikasi Vivado

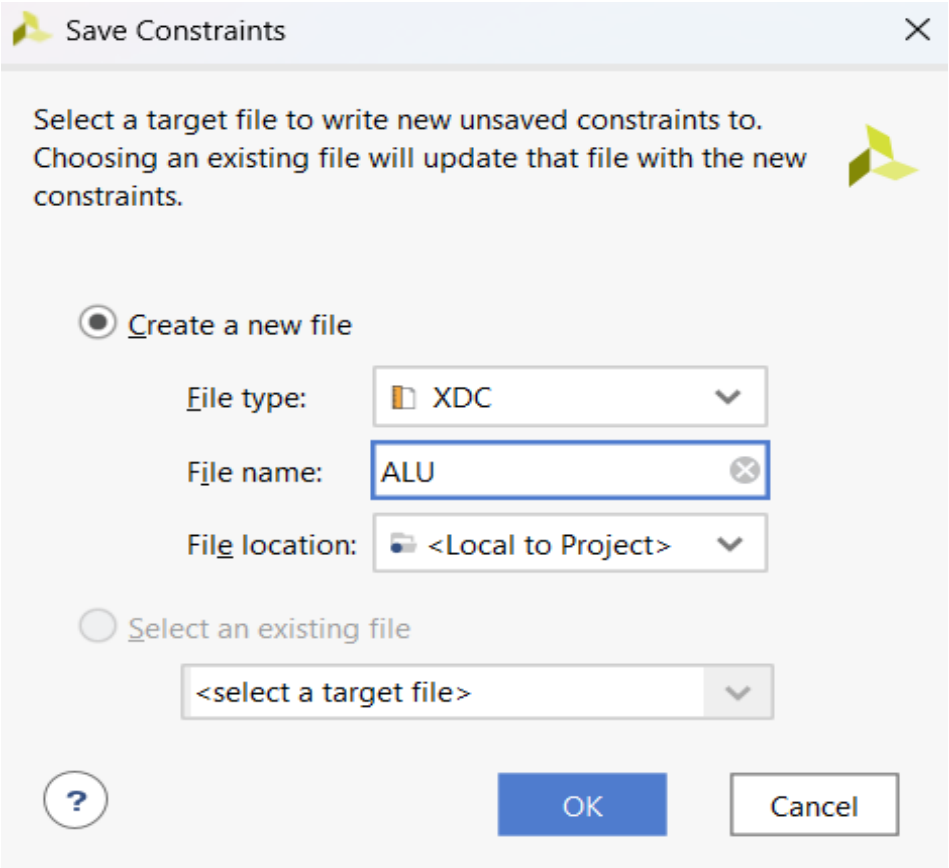
10. Kemudian pada pojok kanan atas pilih I/O Planning dan pilih I/O ports untuk memasukkan port yang akan digunakan untuk mengimplementasikan ALU pada board FPGA.



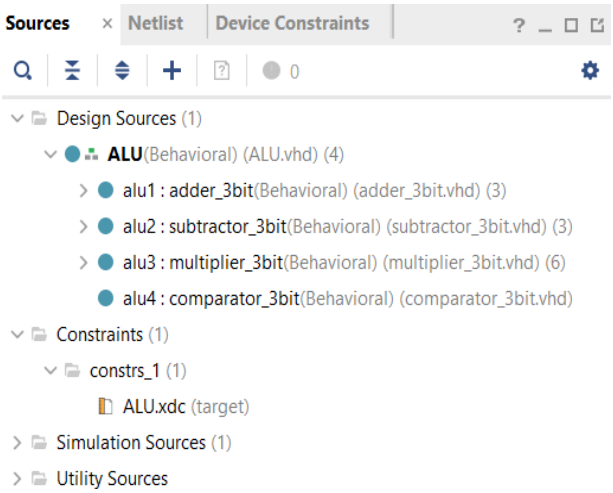
Berikut adalah port yang digunakan pada I/O ports

All ports (15)												
a (3)	IN			✓	(Multiple)	LVCMOS18	1.800			NONE	NONE
a[2]	IN		M13	✓	14	LVCMOS18	1.800			NONE	NONE
a[1]	IN		L16	✓	14	LVCMOS18	1.800			NONE	NONE
a[0]	IN		J15	✓	15	LVCMOS18	1.800			NONE	NONE
b (3)	IN			✓	14	LVCMOS18	1.800			NONE	NONE
b[2]	IN		T18	✓	14	LVCMOS18	1.800			NONE	NONE
b[1]	IN		R17	✓	14	LVCMOS18	1.800			NONE	NONE
b[0]	IN		R15	✓	14	LVCMOS18	1.800			NONE	NONE
result (6)	OUT			✓	(Multiple)	LVCMOS18	1.800	12		NONE	FP_VTT_50
result[5]	OUT		V17	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
result[4]	OUT		R18	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
result[3]	OUT		N14	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
result[2]	OUT		J13	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
result[1]	OUT		K15	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
result[0]	OUT		H17	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
sel (2)	IN			✓	14	LVCMOS18	1.800			NONE	NONE
sel[1]	IN		V10	✓	14	LVCMOS18	1.800			NONE	NONE
sel[0]	IN		U11	✓	14	LVCMOS18	1.800			NONE	NONE
Scalar ports (1)											
c	IN		U18	✓	14	LVCMOS18	1.800			NONE	NONE

Tekan ctrl + S untuk menyimpan konfigurasi yang telah dibuat, kemudian akan muncul tampilan save constraints. Beri nama ALU lalu pilih OK.

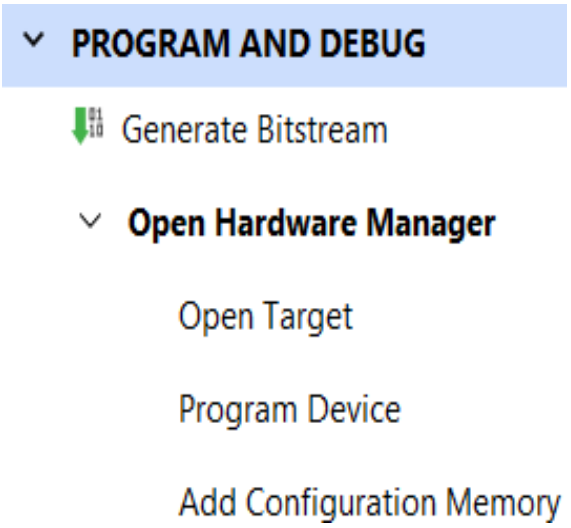


Lalu pada bagian baru akan muncul file constraint yang telah dibuat, klik 2 kali untuk menampilkan isi dari file ALU.xdc yang berisi konfigurasi port yang telah dibuat.

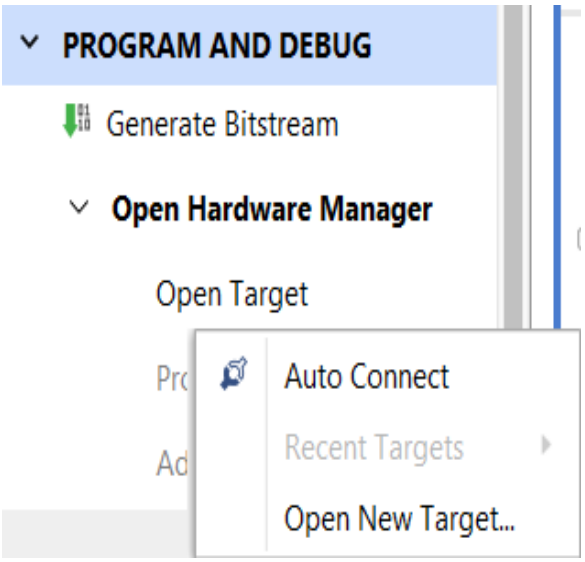



```
set_property IOSTANDARD LVCMOS18 [get_ports {a[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {a[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {a[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {sel[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {sel[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {a[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {a[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {a[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {result[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {sel[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {sel[0]}]
set_property PACKAGE_PIN J15 [get_ports {a[0]}]
set_property PACKAGE_PIN L16 [get_ports {a[1]}]
set_property PACKAGE_PIN M13 [get_ports {a[2]}]
set_property PACKAGE_PIN R15 [get_ports {b[0]}]
set_property PACKAGE_PIN R17 [get_ports {b[1]}]
set_property PACKAGE_PIN T18 [get_ports {b[2]}]
set_property PACKAGE_PIN H17 [get_ports {result[0]}]
set_property PACKAGE_PIN K15 [get_ports {result[1]}]
set_property PACKAGE_PIN J13 [get_ports {result[2]}]
set_property PACKAGE_PIN N14 [get_ports {result[3]}]
set_property PACKAGE_PIN R18 [get_ports {result[4]}]
set_property PACKAGE_PIN V17 [get_ports {result[5]}]
set_property PACKAGE_PIN V10 [get_ports {sel[0]}]
set_property PACKAGE_PIN U11 [get_ports {sel[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports c]
set property PACKAGE PIN U18 [get ports c]
```

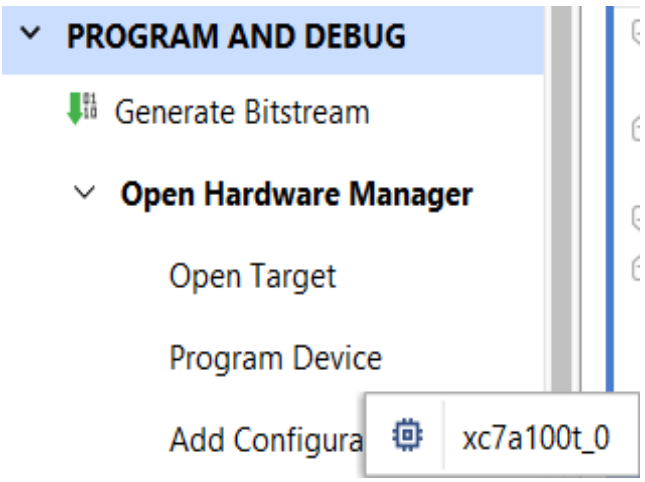
Selanjutnya, hubungkan board nexys dengan komputer menggunakan micro usb dan pada bagian Program and Debug pilih generate bitstream untuk membuat file .bit yang akan diupload pada board



11. Setelah proses generate bitstream selesai, klik open target dan pilih auto connect



12. Setelah board berhasil terhubung dengan komputer klik program dan pilih xc7a100t_0



13. Ketika program berhasil diupload pada board maka pada bagian hardware akan terllihat status Programmed.