BAB 3

RTL ANALYSIS, SYNTHESIZE, AND IMPLEMENTATION

Pada bab ini, praktikan akan mempelajari konsep dasar RTL Analysis, Synthesize, dan Implementation. RTL (Register Transfer Level) menggambarkan aliran data dan kontrol dalam sirkuit digital, sementara proses synthesis mengubah deskripsi RTL menjadi gerbang logika yang siap diimplementasikan ke FPGA. Praktikan juga akan memahami cara analisis RTL dan implementasi desain secara fisik pada FPGA. Asisten praktikum atau praktikan diharapkan membaca tujuan dan persyaratan bab ini agar praktikum dapat berjalan sesuai dengan prosedur.

Tujuan

Tujuan	Penjelasan
Memahami Konsep RTL	Praktikan diharapkan dapat memahami fungsi RTL Analysis dalam proses desain sirkuit digital, termasuk bagaimana RTL menggambarkan aliran data dan kontrol di dalam sirkuit secara hierarkis.
Melakukan Proses Synthesize	Praktikan diharapkan mampu melakukan proses synthesis dari deskripsi RTL menjadi netlist, yaitu mengonversi desain logika dalam HDL (Hardware Description Language) menjadi gerbang logika yang lebih terstruktur dan siap untuk diimplementasikan.
Melakukan Proses Implementasi	Praktikan diharapkan mampu memahami dan menjalankan proses implementasi desain digital dari netlist ke dalam perangkat keras FPGA.
Menganalisis Hasil RTL dan Implementasi	Praktikan diharapkan dapat menganalisis hasil implementasi RTL, melakukan debugging, dan memahami bagaimana desain HDL direalisasikan secara fisik di 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

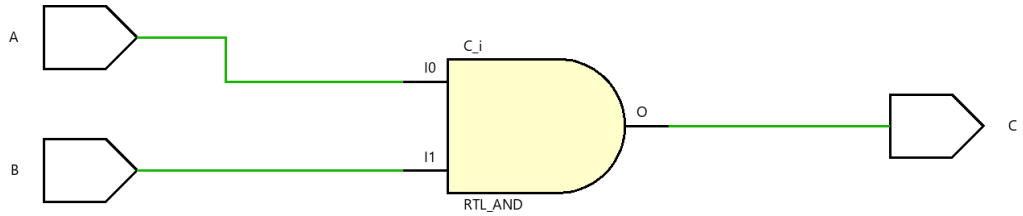
Vivado Design Suite

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

Teori

RTL Analysis

RTL atau *Register Transfer Level* adalah level abstraksi (penyederhanaan sistem) yang penting dalam pengembangan FPGA. RTL memungkinkan desainer untuk menspesifikasikan perilaku sirkuit digital yang berfokus pada aliran data dan operasi logika. Pada *flow manager* yang ada di aplikasi Vivado terdapat RTL Analysis yang berfungsi untuk mengubah kode VHDL yang telah dibuat menjadi desain *Elaborated*, atau penggambaran skematik berdasarkan dari kode program yang dibuat. Selain itu, RTL Analysis juga digunakan untuk memeriksa apakah kode yang dibuat sudah benar sebelum masuk ke proses *Synthesis*.

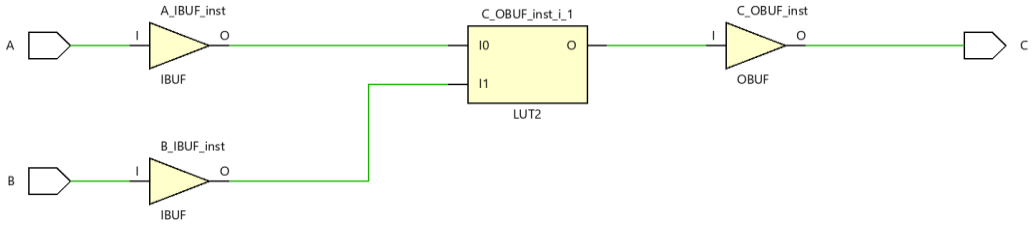


Gambar 5. 1 Hasil Skematik RTL Analysis

Synthesis

Synthesis adalah proses di mana kode RTL yang mendeskripsikan desain *hardware* diterjemahkan menjadi *netlist* gerbang logika dan dioptimalkan untuk diimplementasikan pada *board* FPGA. *Netlist* sendiri adalah koneksi dari suatu sirkuit elektronik, dan terdiri dari daftar komponen elektronik dalam sirkuit dan daftar node dimana komponen dalam rangkaian terhubung.

Secara sederhana, Synthesis mengubah deskripsi perilaku sirkuit menjadi representasi fisik yang dapat di *mapping* ke *resource* yang ada pada *board* FPGA.

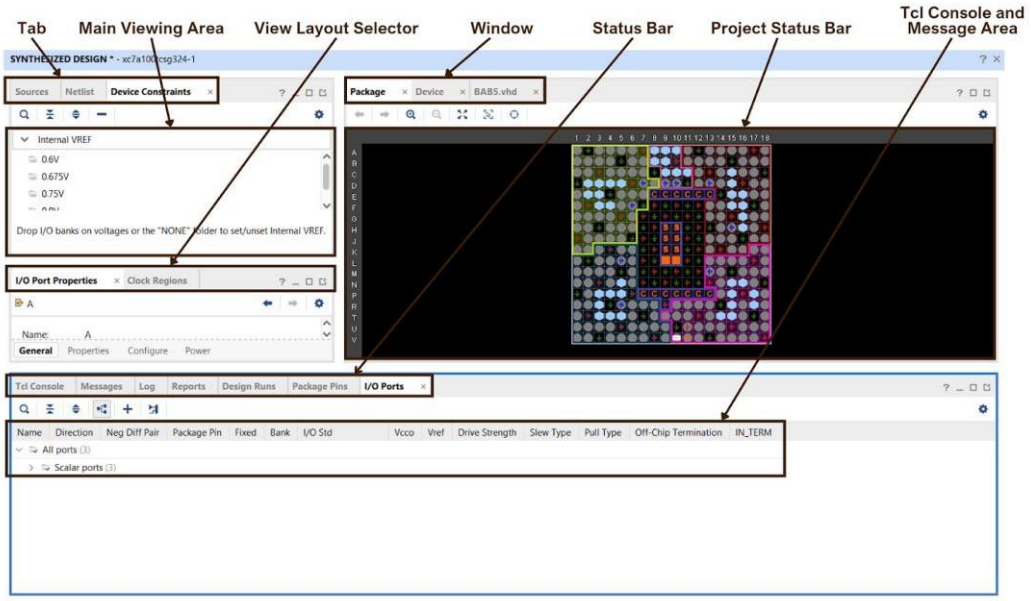


Gambar 5. 2 Hasil Skematik RTL Synthesis

Implementation

Implementation adalah tahapan dimana setelah *netlist* yang dihasilkan dari proses Synthesis dipetakan ke sumber daya fisik pada FPGA dan dirutekan. Tujuannya adalah menghasilkan *bitstream* yang dapat diprogram ke FPGA untuk mengimplementasikan desain atau rancangan yang telah dibuat. Pada bagian *Implementation* pengguna juga mendefinisikan *user-constraint* yang berisikan *port* dan ketentuan lainnya yang akan diimplementasikan pada *board* FPGA.

Melakukan *Synthesize* dan *Implementation* Program VHDL



Gambar 5. 2 Tampilan Awal I/O Planning

Keterangan :

- Tab, mengorganisasi dan menampilkan berbagai aspek dari desain yang akan digunakan pada board FPGA.
- Main Viewing Area, menampilkan ringkasan proyek dan tampilan grafis dari desain.
- View Layout Selector, menyediakan akses ke konfigurasi tata letak tampilan yang ditentukan.
- Window, berisi program yang sedang dijalankan.
- Status Bar, menampilkan informasi tentang projek yang sedang dibuka dan objek apa pun yang saat ini berada dibawah kursor.
- Main Menu, berisi perintah yang tersedia.
- Project Status Bar, menampilkan status projek dan perintah yang aktif berjalan.
- Tcl Console and Message Area, menunjukkan status perintah, pesan aplikasi, hasil kompilasi dan laporan.

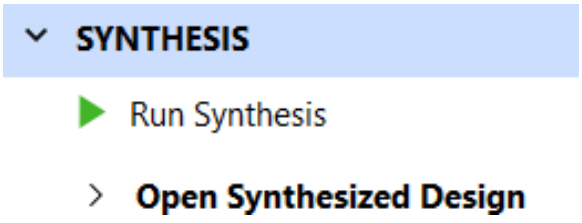
Praktek

- **Melakukan Synthesize dan Implementation**

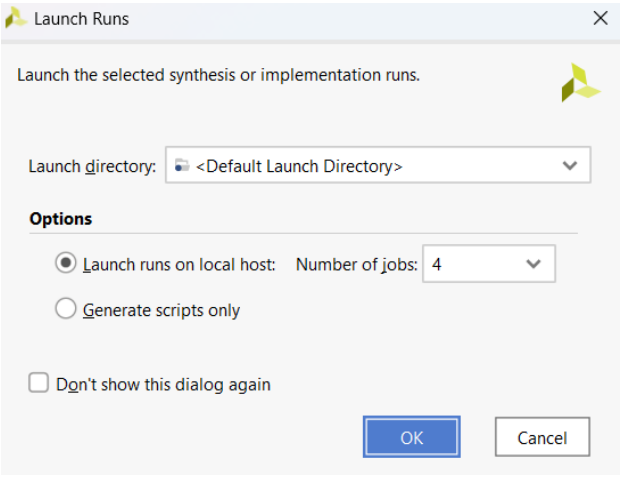
1. Buka aplikasi Vivado 2020.2
2. Buka menu File > New Project dan lakukan langkahnya sama seperti pada bab sebelumnya
3. Buat Entity dengan nama "BAB5" kemudian masukan kode program seperti berikut.

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

entity BAB5 is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : out STD_LOGIC);
end BAB5;

architecture dataflow of BAB5 is

begin
C <= A and B;

end dataflow;
```

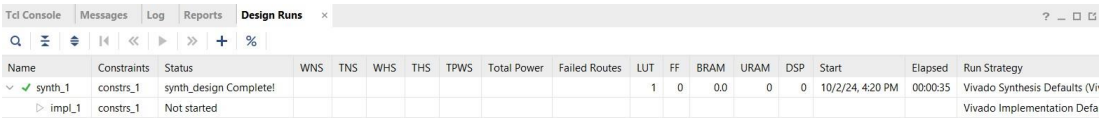
4. Lalu, pada *Flow Manager* pada bagian *Synthesis* pilih *run Synthesis*,



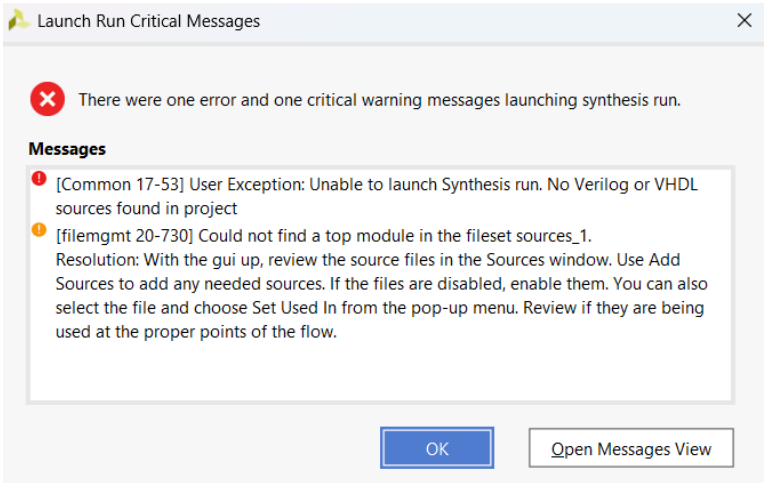
5. Akan muncul menu seperti dibawah ini, lalu klik OK.



6. Pada tab *Design Runs* akan muncul bahwa program sedang menjalankan *Synthesis* dan tunggu sampai proses menujukan tanda centang hijau seperti gambar dibawah.

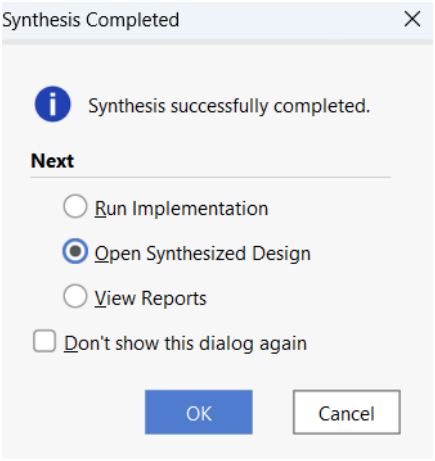


7. Jika proses *Synthesis* gagal, maka akan muncul peringatan seperti berikut.

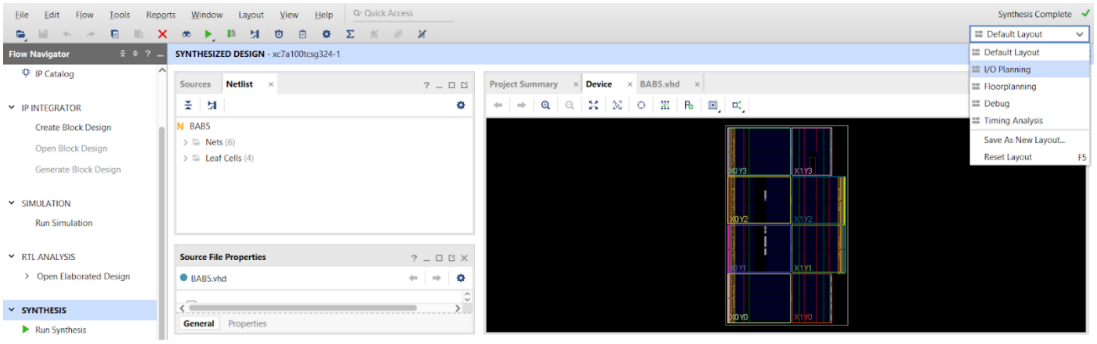


Dan, terdapat juga peringatan *Error* dan *Critical Warning* pada tab *messages*.

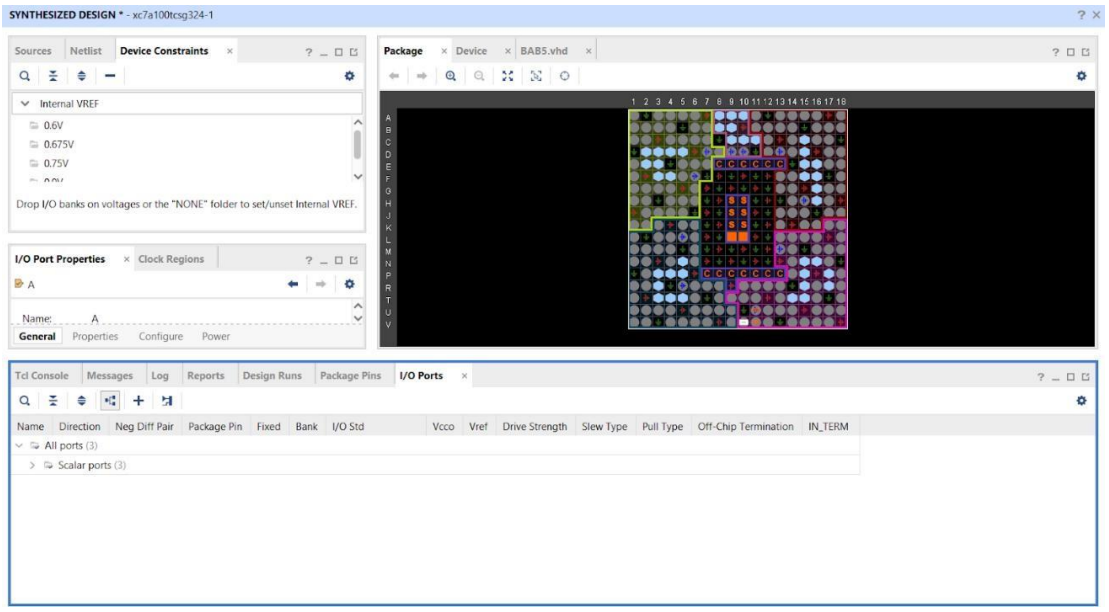
8. Jika proses Synthesis berhasil akan muncul tampilan seperti gambar dibawah, lalu pilih *Open Synthesized Design*, dan klik OK



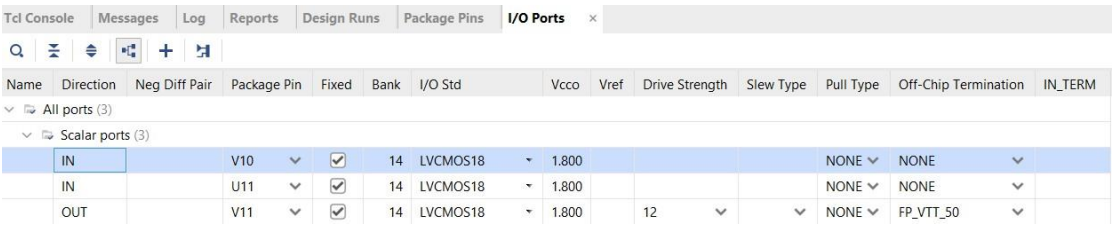
9. Langkah selanjutnya yaitu menentukan pin *input* dan *output* yang akan digunakan pada *board*. Pada pojok kanan atas, pilih I/O Planning.



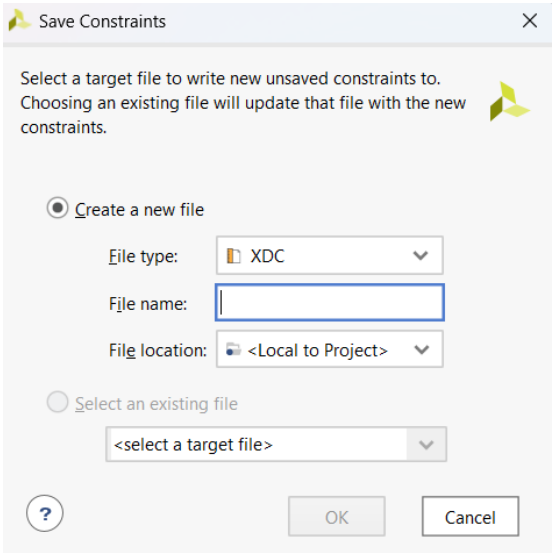
I/O Planning digunakan untuk mengatur komponen *input* dan *output* sesuai dengan yang kita inginkan dengan memasukan pin yang ada pada *board* FPGA. Berikut adalah tampilan pada *window* I/O Planning.



10. Langkah selanjutnya adalah mengkonfigurasi pin *input* dan *output* yang akan digunakan dengan mengklik *expand* pada "*Scalar Ports*", dan isikan seperti dibawah ini.

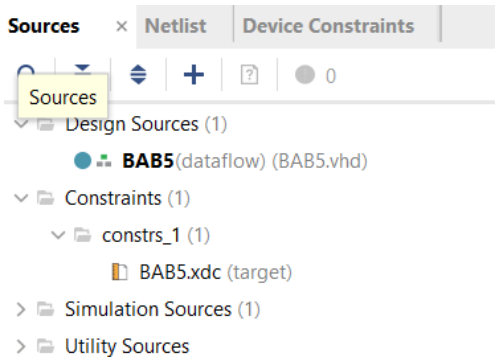


11. Setelah dikonfigurasi, kemudian simpan dengan cara klik tombol "CTRL + S" atau klik ikon *save* di kiri atas dan akan muncul *window* baru seperti gambar dibawah.

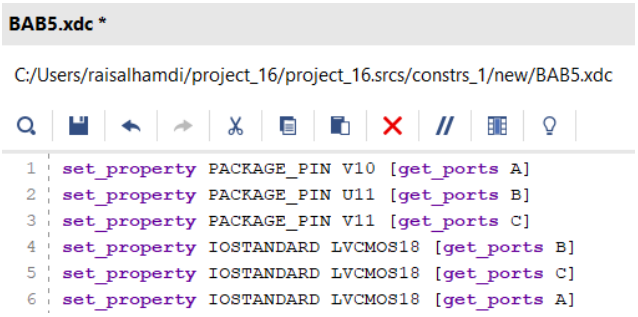


12. Beri nama file BAB5, lalu klik OK

13. Lihat pada bagian *Sources*, lalu *expand* (klik panah bawah) pada *Constraints*.



Pada *expand Constraits* diatas, terdapat file baru dengan format .xdc yang berisikan hasil konfigurasi yang telah dibuat di I/O Planning.



Dapat dilihat, file ini berisikan konfigurasi pin yang telah kita buat sebelumnya. Maksud dari file yaitu :

a. Pin A berada dilokasi V10.

b. Pin B berada dilokasi U11.

c. Pin C berada dilokasi V11.

Lokasi ketiga pin tersebut dapat dilihat juga pada *board* FPGA. Selanjutnya, terdapat IOSTANDARD yang merupakan sebuah deskripsi *attribute*, sedangkan LVCMOS (*Low Voltage Complementary Metal Oxide Semiconductor*) adalah sebuah teknologi tegangan rendah yang digunakan pada sirkuit FPGA untuk menjalankan komponen yang akan digunakan. Angka "18" sendiri menunjukkan berapa besar voltase yang digunakan. Pada kasus ini, angka "18" menunjukkan tegangan sebesar 1,8V. Berikut ini merupakan tabel dari LVCMOS tersebut.

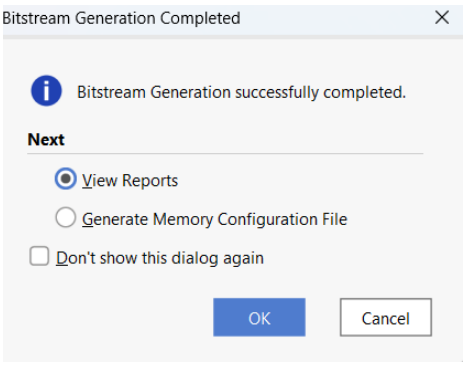
Tabel 5. 1 Tabel LVCMOS

Logic Volts	Tolerance Volts	Tolerance Percent
5.0V	+/-0.5V	+/-10.0%
3.3V	+/-0.3V	+/-9.09%
2.5V	+/-0.2V	+/-8.00%
1.8V	+/-0.15V	+/-8.33%
1.5V	+/-0.1V	+/-8.33%
1.2V	+/-0.1V	+/-8.33%
1.0V	+/-0.1V	+/-8.33%
0.9V	+/-0.045V	+/-5.00%
0.8V	+/-0.04V	+/-5.00%
0.7V	+/-0.05V	+/-7.14%

14. Setelah membuat file .xdc untuk konfigurasi *input dan output*, langkah selanjutnya yaitu melakukan *Implementation*. Jalankan hal tersebut dengan cara klik *run implementation*, lalu tunggu hingga proses *running* selesai dan statusnya menjadi centang berwarna hijau.

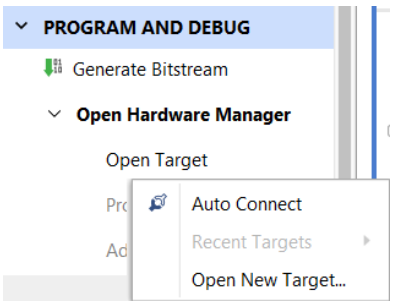
Proses ini dapat diartikan bahwa *input* dan *output* yang sudah dikonfigurasikan sebelumnya telah diimplementasikan atau digabungkan dengan *source code* yang sudah dibuat sebelumnya.

15. Setelah proses *Implementation* selesai, selanjutnya menjalankan proses *Generate Bitstream* dengan klik *Generate Bitstream* lalu tunggu hingga proses selesai. Setelah proses berhasil, maka akan muncul tampilan *window* baru seperti dibawah ini, lalu klik OK.



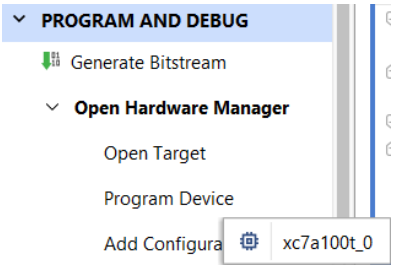
Bitstream adalah sebuah file yang berisi informasi dari konfigurasi yang telah dibuat sebelumnya dan memiliki ekstensi .bit. Pada proses kali ini, Vivado akan men-*streaming* file bit tersebut ke *port* yang ada pada *board* FPGA, dan kita dapat memprogram FPGA sesuai dengan konfigurasi yang diinginkan.

16. Setelah *Bitstream* berhasil dibuat, selanjutnya adalah mengunggah *Bitstream* tersebut ke *board* FPGA yang sudah terhubung dengan komputer menggunakan kabel micro USB. Pastikan board FPGA sudah menyala dan benar-benar terhubung.

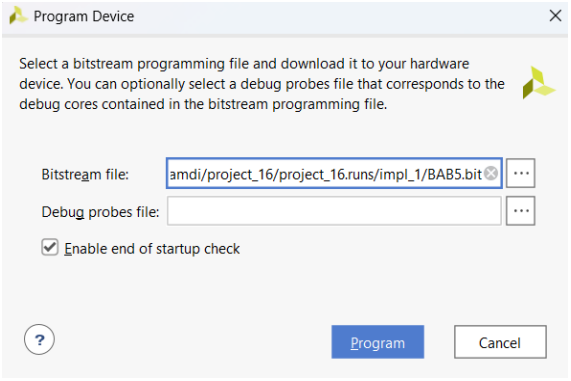


Expand pada bagian *Open Hardware Manager*, pilih *Open Target*, kemudian klik *Auto Connect* dan tunggu hingga prosesnya selesai.

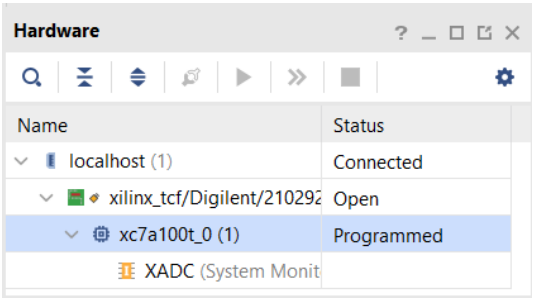
17. Setelah proses selesai, masih pada bagian *Hardware Manager*. Klik *Program Device* dan otomatis akan muncul jenis *board* yang telah terhubung melalui kabel USB dan pilih "Xc7a100t_0"



18. Akan muncul *window* baru seperti gambar dibawah, lalu klik Program.



19. Ketika berhasil pada bagian *Hardware*, akan muncul *Programmed* yang menandakan bahwa file *Bitstream* yang telah dibuat berhasil diunggah pada *board* FPGA.



Proses-proses tersebut merupakan langkah akhir untuk menjadikan file program yang dapat digunggah ke *board* FPGA. Nantinya, file inilah yang dapat dipakai berulang-

ulang sesuai dengan kebutuhan dalam menggunakan FPGA. Karena seperti yang kita tahu, FPGA memiliki sifat *volatile*, yaitu sifat dimana apabila sumber dayanya dicabut maka program yang sudah ditanam sebelumnya akan hilang. Namun, jika sudah memiliki file program ini, kita hanya perlu memasukannya kembali atau melakukan *flash program* kedalam FPGA tersebut.

Basic Arithmetic Implementation FPGA

FPGA dapat melakukan perhitungan sederhana seperti operasi penjumlahan (*adder*) dan pengurangan (*subtractor*) dalam VHDL. Blok logika aritmatika ini dideskripsikan secara spesifik, kemudian diimplementasikan menjadi konfigurasi hardware yang dapat dijalankan oleh FPGA.

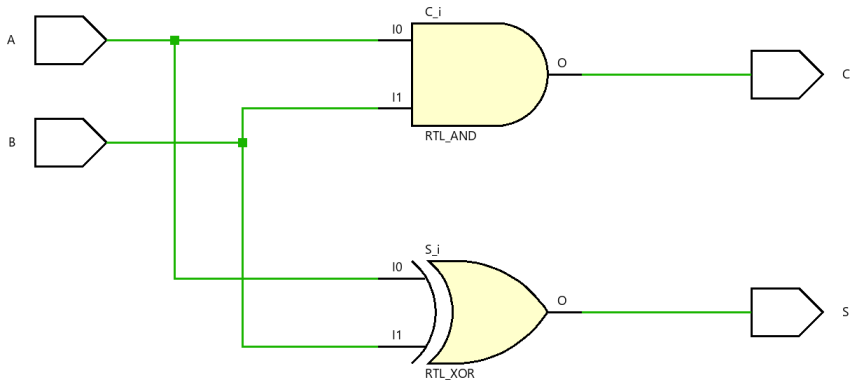
Half Adder

Half Adder adalah rangkaian logika kombinasional yang melakukan penjumlahan dua bit biner. Rangkaian ini memiliki dua *input* yaitu A dan B yang masing-masing *input* merepresentasikan satu bit, dan dua *output* yaitu Sum (S) dan Carry (C). *Output* Sum merupakan hasil penjumlahan dari kedua *input*, sedangkan *output* Carry menunjukkan apakah terjadi nilai lebih dari hasil penjumlahan dari kedua input. Berikut adalah *Truth Table* dari rangkaian *Half Adder*.

Tabel 5. 2 Truth Table Half Adder

INPUT		**OUTPUT**	
A	**B**	**S**	**C**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Dari *Truth Table* diatas dapat dilihat bahwa *output* Sum akan bernilai 1 jika salah satu dari *input* (A atau B) bernilai 1. *Output* Carry bernilai 1 hanya jika kedua *input* (A dan B) bernilai 1. Rangkaian *Half Adder* dapat diimplementasikan menggunakan gerbang logika XOR untuk *output* Sum dan gerbang AND untuk *output* Carry.



Gambar 5. 3 Rangkaian Half Adder

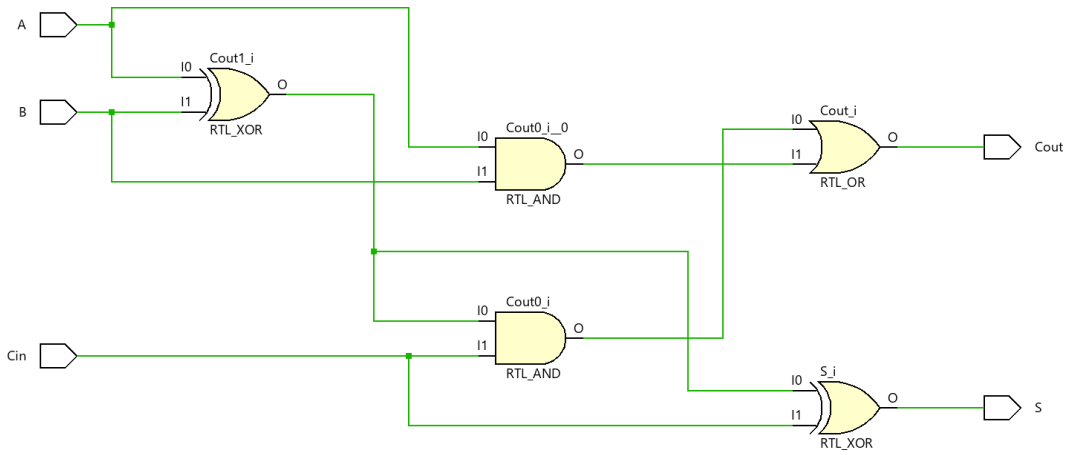
Full Adder

Full Adder juga merupakan rangkaian logika kombinasional yang melakukan penjumlahan. Namun, berbeda dengan *Half Adder, Full Adder* dapat menjumlahkan tiga bit biner. *Input* tambahan pada *Full Adder* adalah Carry-in (Cin), yang merupakan *carry* dari penjumlahan sebelumnya. *Full Adder* memiliki dua *output* yang sama seperti dengan *Half Adder*, yaitu Sum (S) dan Carry-*out* (Cout). Berikut adalah *Truth Table* dari rangkaian *Full Adder.*

Tabel 5. 3 Truth Table Full Adder

INPUT			**OUTPUT**	
A	**B**	**Cin**	**S**	**Cout**
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

Dari *Truth Table* diatas, *Full Adder* memiliki tiga *input* yaitu A, B, Cin dan dua *output* yaitu S dan Cout. *Output* S menunjukkan hasil penjumlahan dari ketiga *input*, sedangkan *output* Cout menunjukkan apakah ada lebih (Carry) dari penjumlahan tersebut. Rangkaian *Full Adder* dapat diimplementasikan menggunakan dua gerbang *Half Adder* dan satu gerbang OR.



Gambar 5. 4 Rangkaian Full Adder

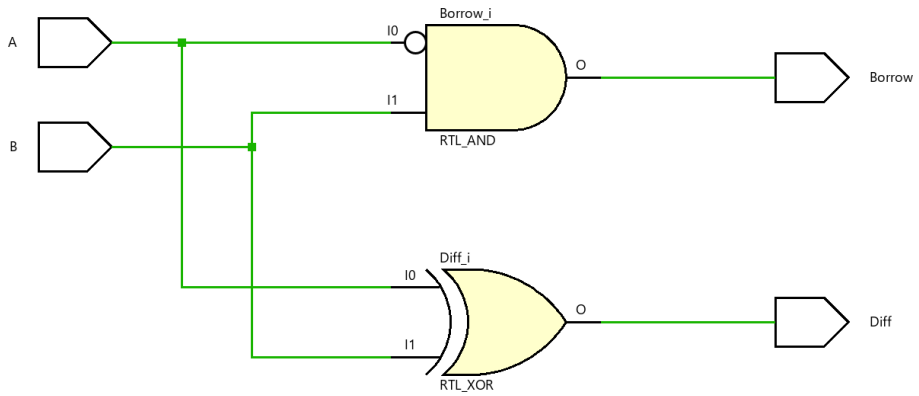
Half Subtractor

Half Subtractor adalah rangkaian logika kombinasional yang digunakan untuk melakukan operasi pengurangan pada bilangan biner. Rangkaian ini memiliki dua *input* yaitu A dan B, dan juga dua *output* yaitu Difference (D) dan Borrow (B). *Output* Difference merupakan hasil dari pengurangan A dan B (A – B), sedangkan *output* Borrow menunjukkan apakah terjadi peminjaman ("*borrow*") dari bit berikutnya atau tidak.

Tabel 5. 4 Truth Table Half Subtractor

INPUT		OUTPUT	
A	**B**	**Diff**	**Borrow**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Dari *Truth Table* diatas, *output* Difference (D) akan bernilai 1 jika nilai dari *input* A dan B berbeda, dan bernilai 0 jika nilai dari *input* A dan B sama. *Output* Borrow (B) akan bernilai 1 hanya ketika nilai dari *input* A bernilai 0 dan B bernilai 1, yang menunjukkan bahwa diperlukan peminjaman dari bit berikutnya. *Half Subtractor* dapat diimplementasikan menggunakan gerbang XOR untuk output Difference dan gerbang AND NOT pada *input* A untuk *output* Borrow.



Gambar 5. 5 Rangkaian Half Subtractor

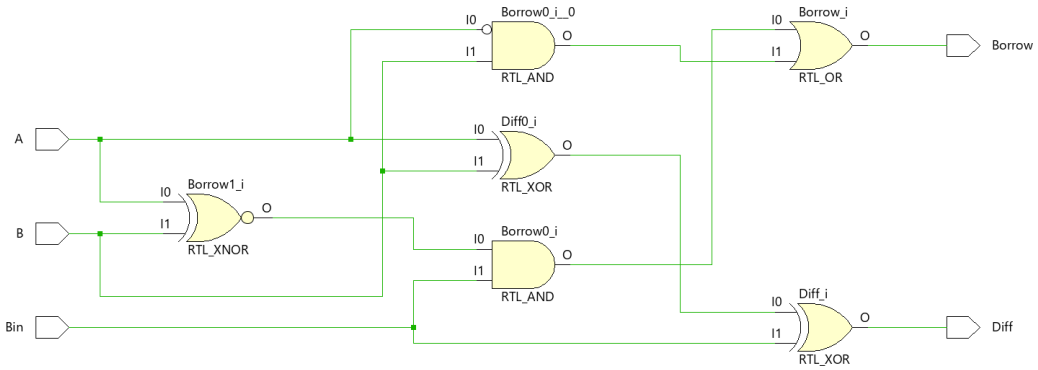
Full Subtractor

Full Subtractor digunakan untuk melakukan pengurangan pada tiga bit biner. *Input* tambahan pada *Full Subtractor* adalah Borrow-in (Bin), yang merupakan *borrow* dari pengurangan sebelumnya. *Output Full Subtractor* sama dengan *Half Subtractor*, yaitu Difference (D) dan Borrow-out (Bout).

Tabel 5. 5 Truth Table Full Subtractor

INPUT			**OUTPUT**	
A	**B**	**Bin**	**Diff**	**Bout**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Dari *Truth Table* diatas, *Full Subtractor* memiliki tiga *input* yaitu A, B, dan Bin, dan dua *output* yaitu D dan Bout. *Output* Diff menunjukkan hasil pengurangan dari *input* A dan B dengan memperhitungkan *input* Borrow Bin, sedangkan *output* Bout menunjukkan apakah ada kekurangan (*borrow*) dari pengurangan tersebut.



Gambar 5. 6 Rangkaian Full Subtractor