Modul 2
Bahasa Verilog

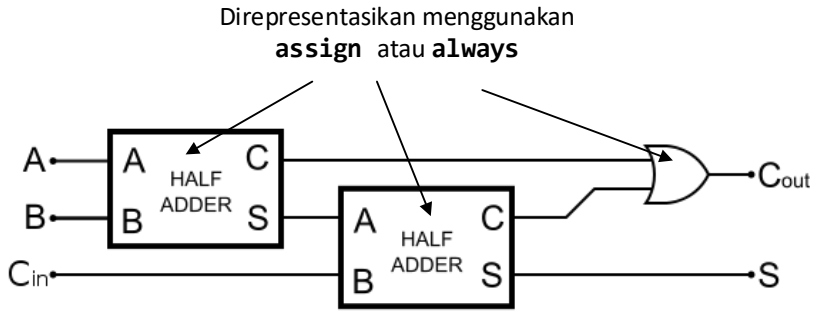
1. Tujuan

a. Memahami bahasa pemrograman Verilog.
b. Memahami timing constraint.
c. Melakukan debugging dengan internal logic analyzer.

2. Materi

a. Verilog

Verilog merupakan bahasa HDL yang digunakan untuk memodelkan rangkaian digital yang akan diimplementasikan pada FPGA atau ASIC. Verilog telah distrandarisasi pada IEEE 1364. Syntax yang dimiliki pada Verilog mirip seperti pada bahasa permrograman C. Setiap blok digital dapat direpresentasikan pada Verilog dengan blok `assign` atau `always`, seperti yang diilustrasikan pada gambar 1.



Gambar 1. Representasi Verilog

Pada modul 1, blok half adder telah dibuat menggunakan blok assign. Kode Verilog untuk half adder tersebut yaitu sebagain berikut:

```
`timescale 1ns / 1ps

module half_adder
    (
        input wire  a,
        input wire  b,
        output wire sum,
        output wire carry
    );

    assign sum = a ^ b;
    assign carry = a & b;

endmodule
```

Pada kode tersebut terlihat bahwa untuk mengimplementasikan blok half adder digunakan blok assign. Perbandingan jika menggunakan blok always yaitu ditampilkan pada source code berikut ini:

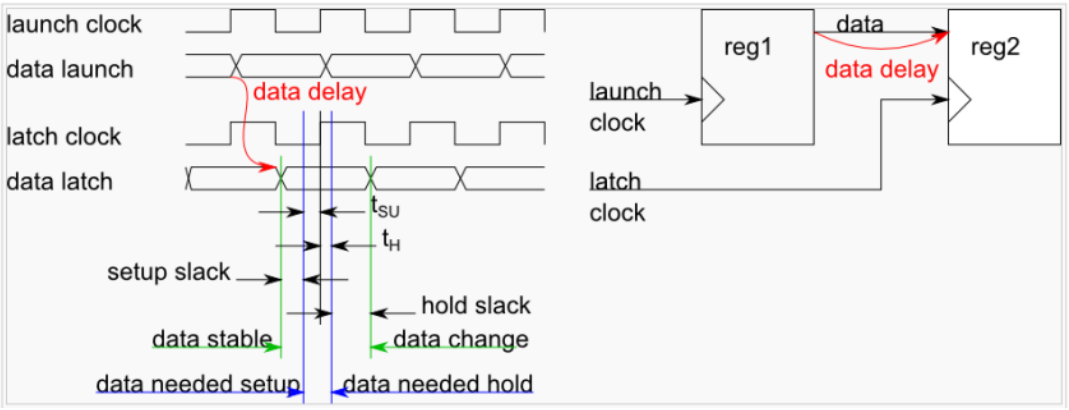
```
module half_adder
    (
        input wire  a,
        input wire  b,
        output reg sum,
        output reg carry
    );

    always @(a, b)
    begin
        sum_tmp = a ^ b;
        carry_tmp = a & b;
    end

endmodule
```

Beberapa hal yang perlu diperhatikan saat membuat module menggunakan assign dibandingkan dengan always adalah:

- Jika menggunakan blok assign, tipe output yang digunakan wire, sedangkan pada always menggunakan reg.
- Pada blok always terdapat sensitivity list yang berisi semua sinyal input dari blok kombinasional.
- Pada rangkaian kombinasional menggunakan always, untuk assignment gunakan tanda =, bukan <=. Tanda <= digunakan pada rangkaian sequensial menggunakan always.

b. Timing Constraint

Design rangkaian digital menggunakan HDL sering disebut juga RTL (Register Transfer Level), karena data biasanya diproses, dan dipindahkan dari satu register ke register lain. Pada pemrosesan data dengan rangkaian kombinasional terdapat delay propagasi dari logic gates. Semakin kompleks rangkaian kombinasional yang dibuat, maka delay propagasinya semakin besar. Delay propagasi terpanjang dari suatu sistem RTL disebut juga sebagai *cirital path*.

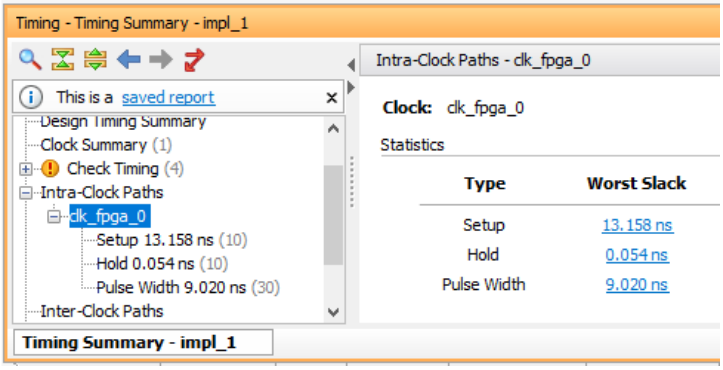


Gambar 2. Timing analysis

Delay tersebut berpengaruh pada performa sistem, dan dianalisis menggunakan timing analysis tool. Hasil dari timing analysis tool pada umumnya berisi informasi seperti pada gambar 2. Beberapa yang perlu diperhatikan adalah:

- **Setup time**: t_{su} merupakan waktu minimum data harus stabil sebelum rising/falling edge dari clock terjadi agar data tersimpan dengan benar pada flip-flop.
- **Hold time**: t_H merupakan waktu minimum data harus stabil setelah rising/falling edge dari clock terjadi agar data tersimpan dengan benar pada flip-flop.

Pada synthesis report Vivado, akan menghasilkan hasil seperti contoh berikut pada gambar 3. Pada gambar tersebut, terlihat bahwa worst setup slack (WNS) dari sebuah design adalah 13.158 ns. Jika WNS positif, maka timing design tidak bermasalah, tetapi jika negative, maka design tidak dapat berjalan dengan baik. Worst setup slack ini dapat digunakan untuk menghitung maximum frequency clock untuk sistem ini dengan persamaan (1).



Gambar 2. Timing analysis Vivado

$$T_{clock\_min} = T_{clock\_constraint} - WNS \quad (1)$$

Sebagain contoh jika $T_{clock\_constraint} = 10ns$, maka $T_{clock\_min} = 20ns - 13.158ns =$ 6.842ns. Jadi maximum frequency-nya adalah $\frac{1}{6.842\text{ns}} = 146.15$ MHz

c. ILA IP core

ILA (Internal Logic Analyzer) merupakan IP core Xilinx yang digunakan untuk mengcapture signal untuk debug. Modul ini ditnbahkan pada design setelah dilakukan synthesis.

3. Latihan

1. Implementasikan rangkaian full adder seperti pada modul 1 menggunakan blok always.

4. Tutorial

Pada tutorial ini, kita akan melakukan beberapa hal sebagai berikut:

- Menganalisis timing constraint dan menghitung maximum clock frequency yang dapat digunakan.
- Melakukan debugging dengan core ILA.

Menganalisis timing constraint

1. Tambahkan modul Verilog untuk D flip-flop dan top sebagai berikut:

File dff.v untuk D flip-flop.

```
`timescale 1ns / 1ps

module dff
    (
        input wire clk,
        input wire rst,
        input wire d,
        output reg q
    );

    always @(posedge clk)
    begin
        if (rst)
            q <= 0;
        else
            q <= d;
    end

endmodule
```

File top.v untuk menggabungkan full adder dengan D flip-flop.

```
`timescale 1ns / 1ps

module top
    (
        input wire  clk,
        input wire  rst,
        input wire  a,
        input wire  b,
        input wire  c,
        output wire sum,
        output wire carry
    );

    wire a_w, b_w, c_w, sum_w, carry_w;

    dff
    (
        .clk(clk),
        .rst(rst),
        .d(a),
        .q(a_w)
```

```
    );

    dff
    (
        .clk(clk),
        .rst(rst),
        .d(b),
        .q(b_w)
    );

    dff
    (
        .clk(clk),
        .rst(rst),
        .d(c),
        .q(c_w)
    );

    full_adder full_adder_0
    (
        .a(a_w),
        .b(b_w),
        .c(c_w),
        .sum(sum_w),
        .carry(carry_w)
    );

    dff
    (
        .clk(clk),
        .rst(rst),
        .d(sum_w),
        .q(sum)
    );

    dff
    (
        .clk(clk),
        .rst(rst),
        .d(carry_w),
        .q(carry)
    );

endmodule
```

2. Ubah file constraint menjadi seperti berikut ini:

Fungsi dari constraint ini yaitu membuat clock dengan nama sys_clk_pin yang terhubung ke sinyal Verilog `clk`. Sinyal `clk` ini tehubung ke external osilator 125 MHz melalui PIN L16.

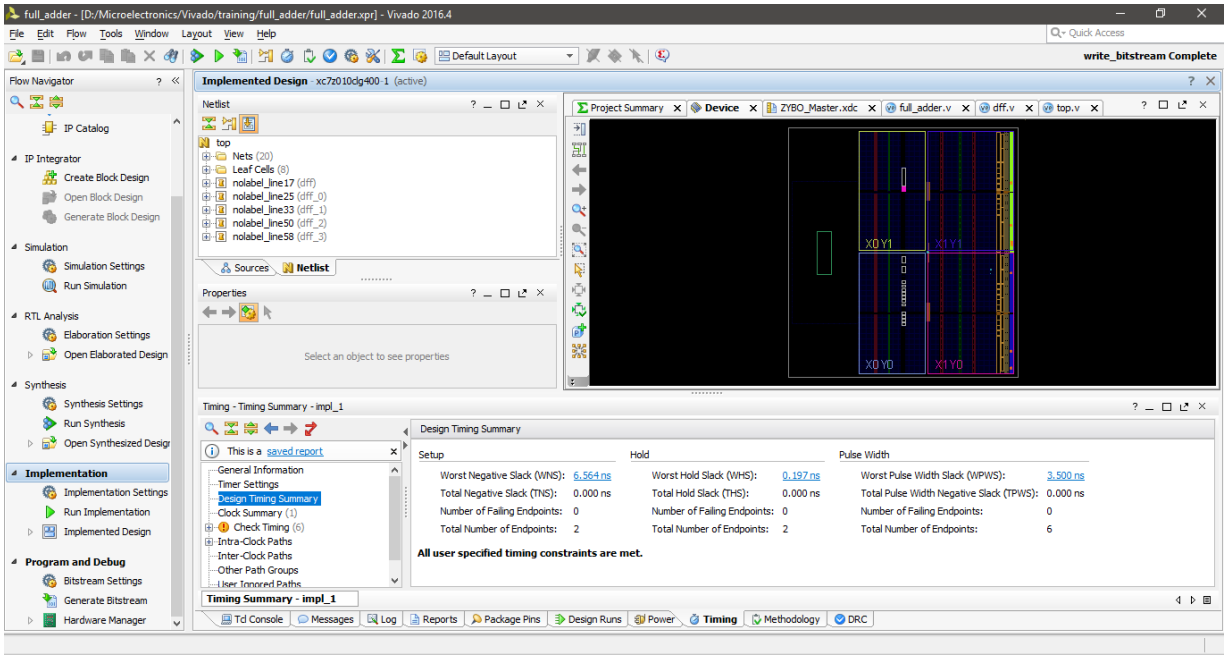
```
set_property PACKAGE_PIN L16 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports
clk]
```

Fungsi dari constraint ini yaitu menghubungkan sinyal rst ke PIN R18 yang terhubung ke button untuk reset.

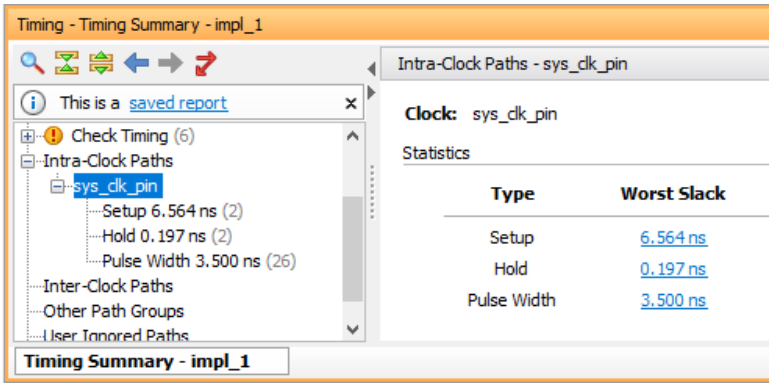
```
##Buttons
##IO_L20N_T3_34
set_property PACKAGE_PIN R18 [get_ports {rst}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst}]
```

3. Lakukan **synthesis**, **implementation**, dan **generate bitstream.**

4. Pilih menu **Open Implementated Design** pada Project Explorer.



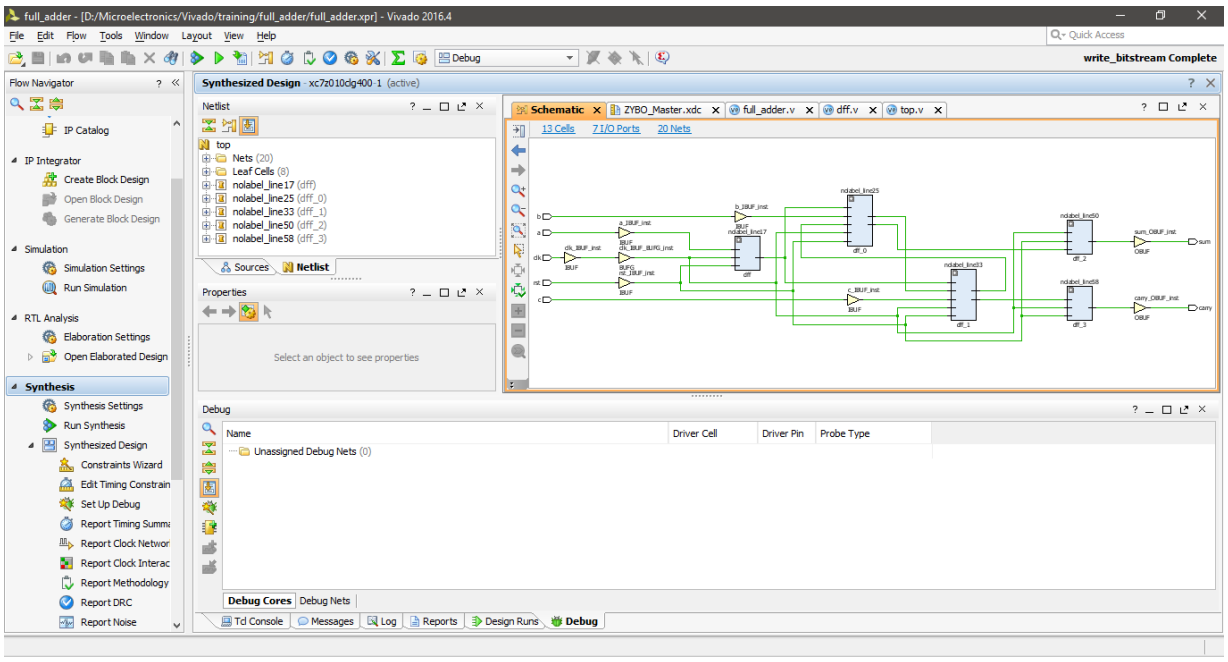
5. Pada timing summary akan tampil WNS sebagai berikut.



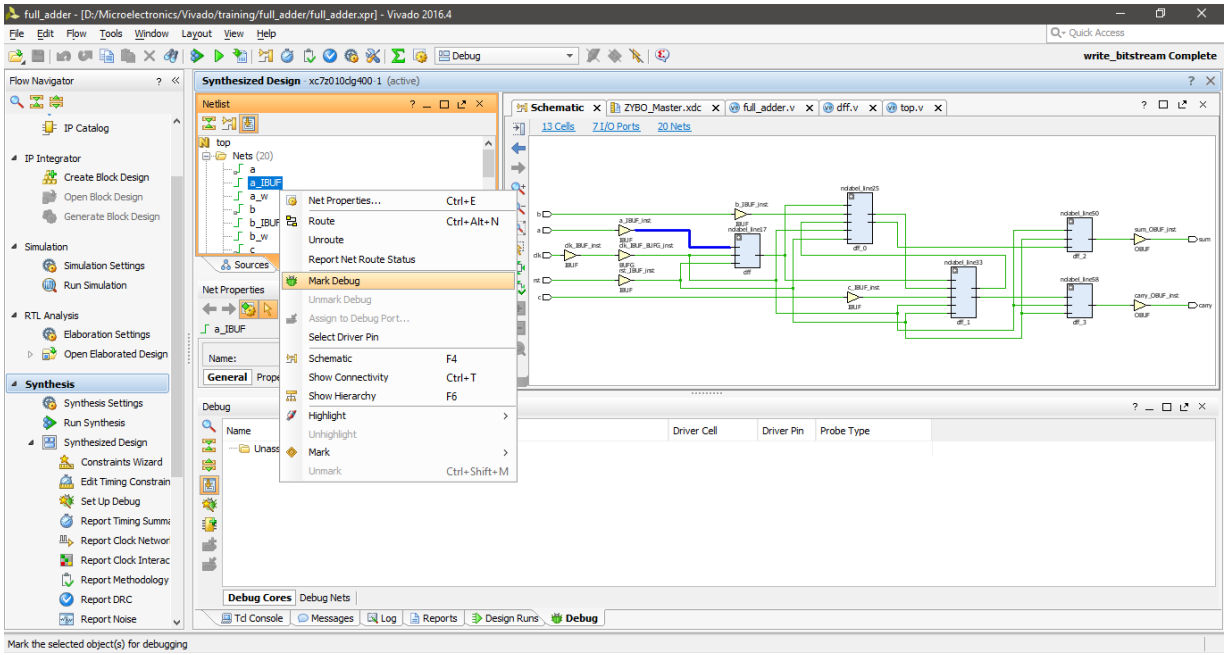
Maximum frequency dapat dihitung sebagai berikut $T_{clock\_min} = 8ns - 6.564ns =$ 1.436ns (696.37 MHz)

Melakukan debugging dengan core ILA

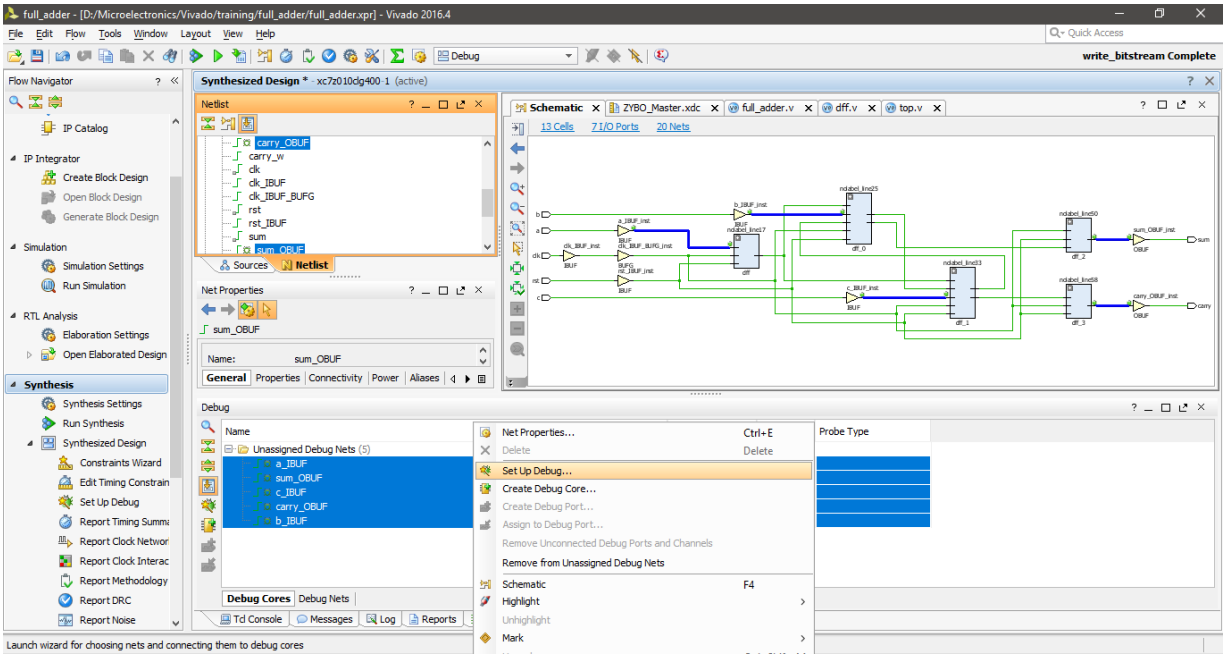
1. Pilih menu **Open Synthesized Design** pada Project Explorer.



2. Pada **Netlist**, pilih sinyal yang akan didebug, klik kanan pilih **Mark Debug**. Netlist yang dipilih yaitu **a_IBUF**, **b_IBUF**, **c_IBUF**, sum_**OBUF**, dan **carry_OBUF**.

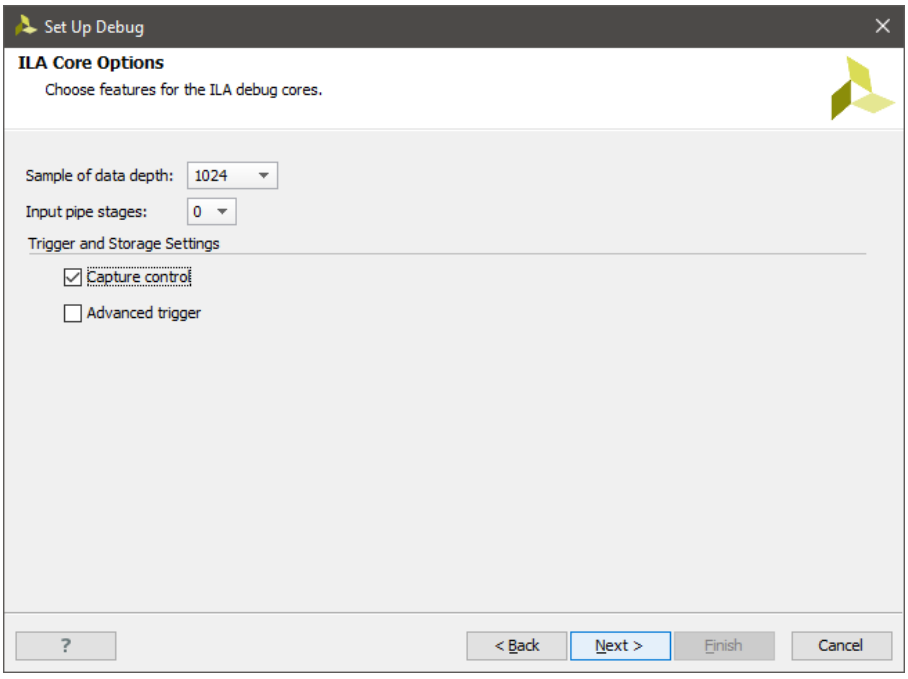


3. Pada **Debug Cores**, blok semua sinyal, klik kanan, pilih **Set Up Debug**.

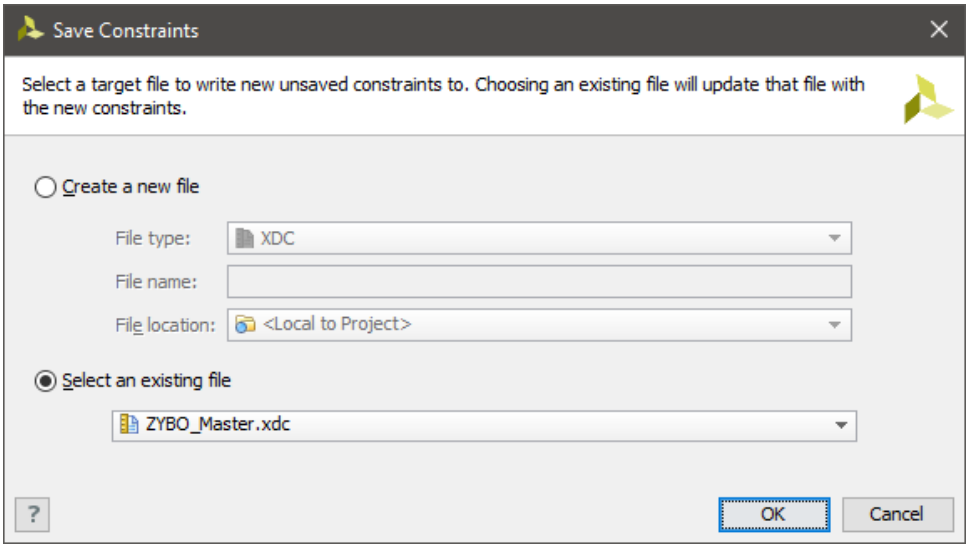


Pada wizard Set Up Debug, klik Next, Next.

4. Aktifkan **Capture Control**, kemudian **Finish**.

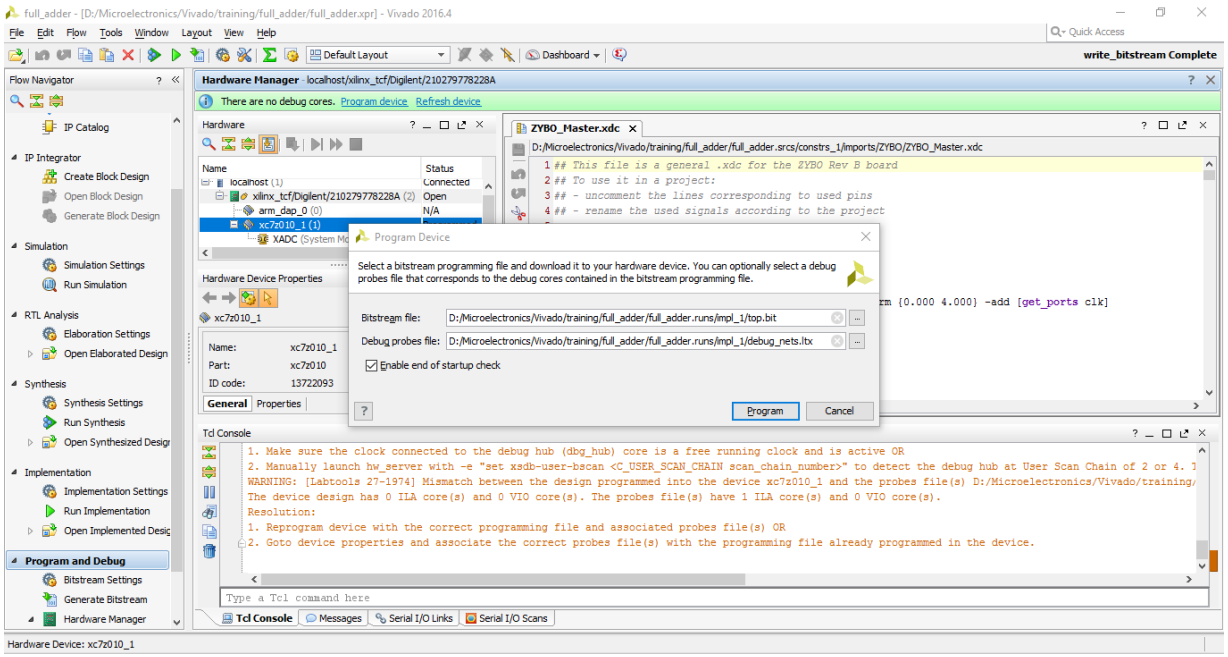


5. Klik **Ctrl+S** untuk save synthesized design.

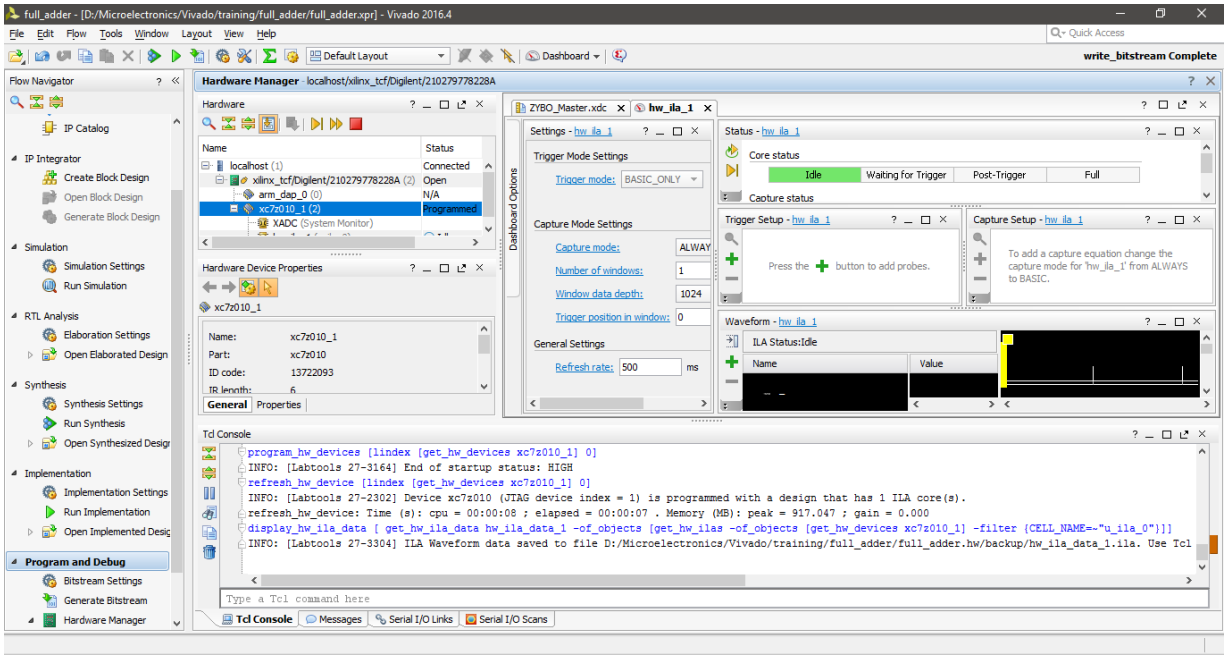


6. Lakukan **Run Implementation** dan **Generate Bitstream**.

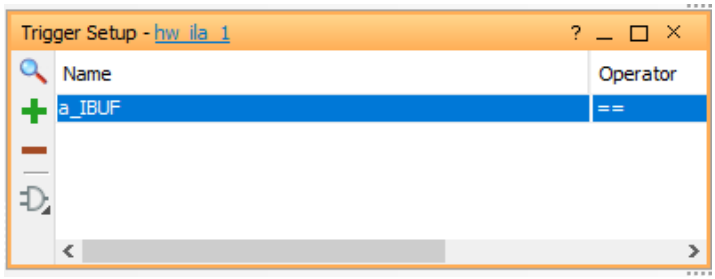
7. Program **bitstream**.

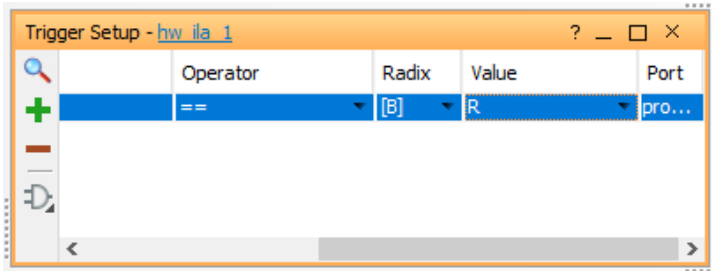


8. Window debug ILA akan terbuka seperti ini.

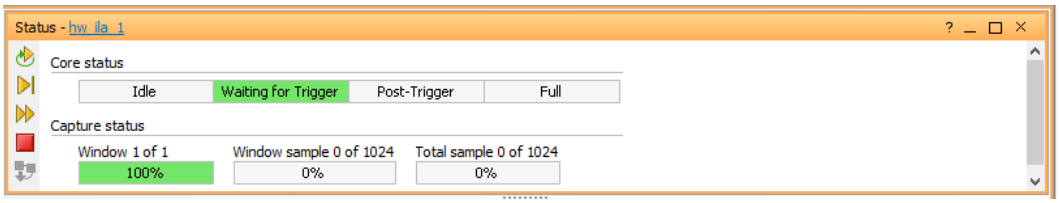


9. Tambahkan sinyal trigger sebagai contoh menggunakan sinyal **a_IBUF**. Kemudian ubah value menjadi **rising edge (R)**. Sehingga jika sinyal a_IBUF berubah dari 0 ke 1, maka logic analyzer akan mengcapture data.





10. Pada window **Status**, klik tombol **Run trigger for this ILA core**. Kemudian core akan berada pada state **Waiting for Trigger**.



11. Pada board Zybo, ubah SW0 dari 0 ke 1.

12. Hasil capture ILA akan tampil pada window **Waveform**.