Vivado Tutorial

Digital Design Using Verilog &

FPGA Flow Using Vivado Diploma - V11

Eng. Kareem Waseem

Made by: Eng. Magdy Ahmed

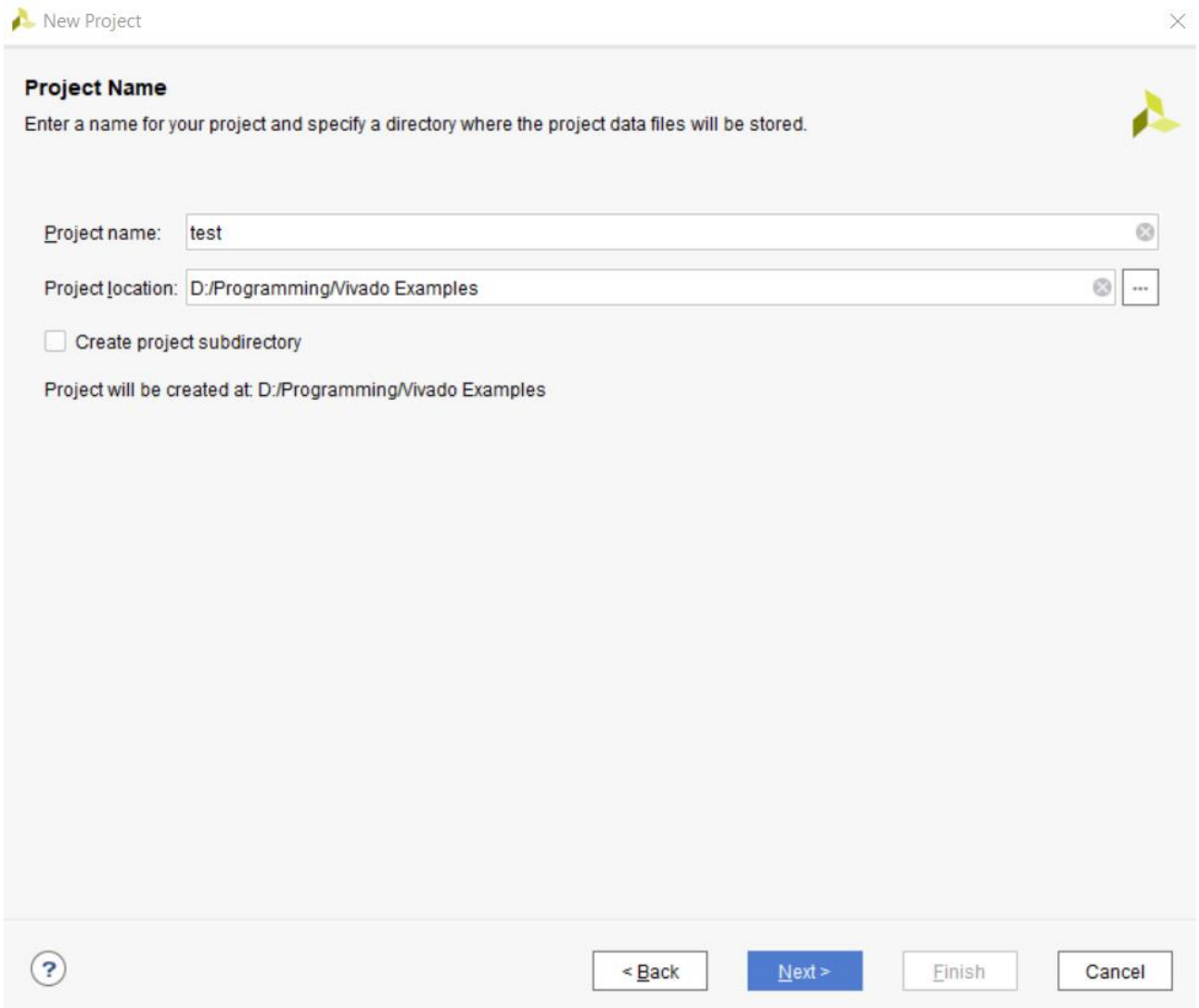
VIVADO™

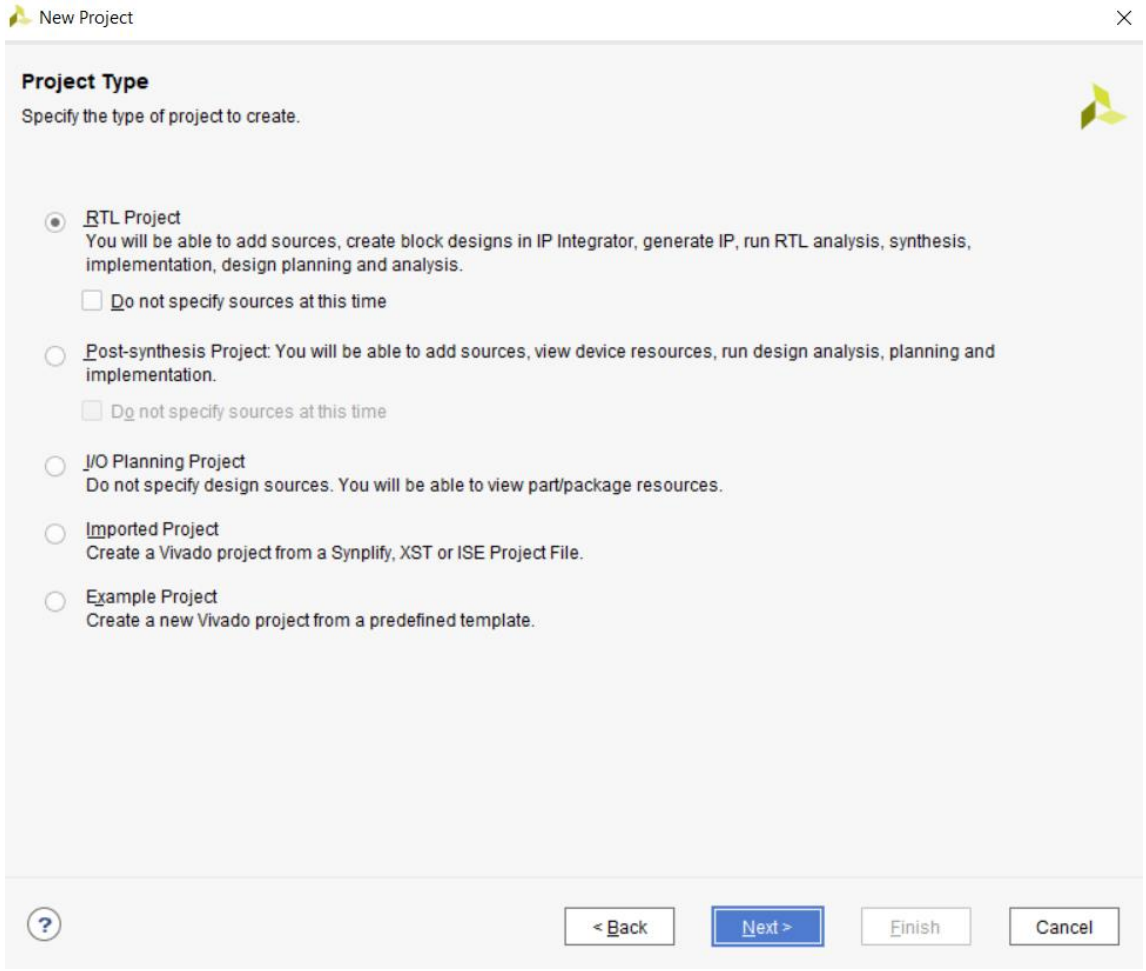
XILINX

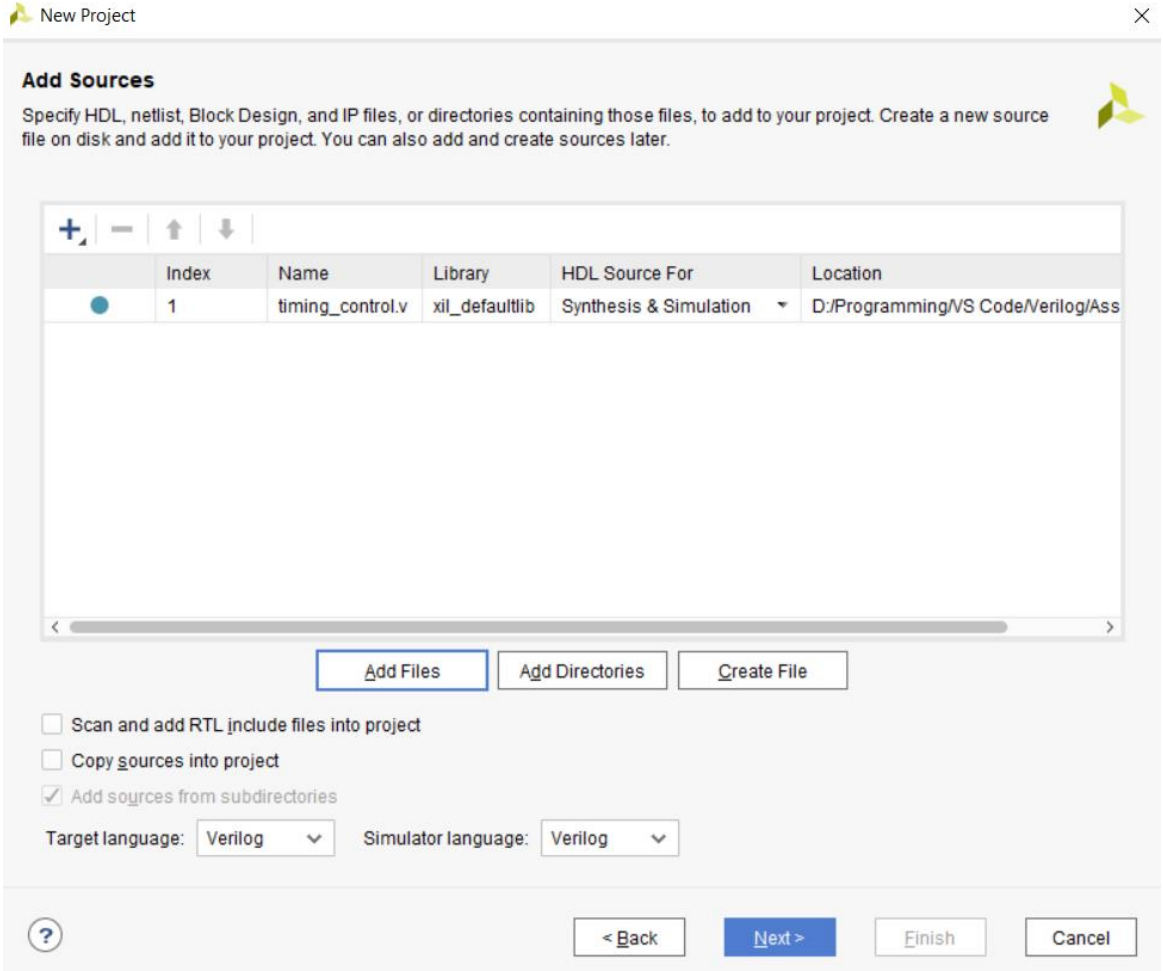
VIVADO Tool Flow

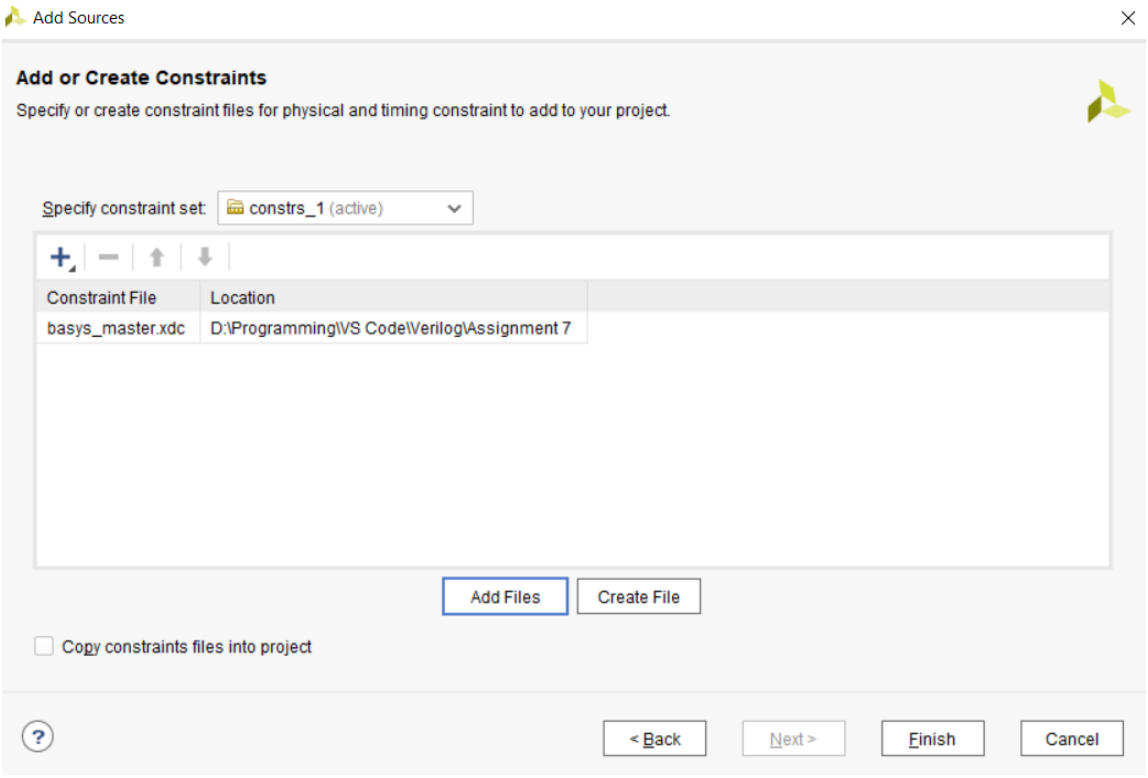
PROJECT CREATION

- First Open Vivado: From File Create New Projects, then Add Source. V Files and Constraints (basys_master. xdc) TCL file if possible where (xdc = xilinix design constraint), Put testbench. V in Simulation.
- Emulated Board Name: Ex: XC7A35TICPG236-1L (Artix - 7).




New Project
Project Type
Specify the type of project to create.
RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
Do not specify sources at this time
Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.
Do not specify sources at this time
I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.
Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.
Example Project
Create a new Vivado project from a predefined template.
< Back
Next >
Finish
Cancel


New Project
Add Sources
Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.
Index
Name
Library
HDL Source For
Location
1
timing_control.v
xil_defaultlib
Synthesis & Simulation
D:/Programming/VS Code/Verilog/Ass
Add Files
Add Directories
Create File
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
Target language:
Verilog
Simulator language:
Verilog
< Back
Next >
Finish
Cancel



Constraint File

- Constraints:

– physical: – To Choose Specific places for input and output we want.

– Determine location of IO ports on FPGA, a must for uploading.

– timing: – To Choose the Used CLK Frequency we want

– FPGA has no oscillators to generate clock so it gets clock from an outer circuit on the Same Board can divide multiply recieved clk depending on the constraints.

```
## Clock signal
set_property – dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports clk]
```

// pin W5 is the clock pin in the fpga

```
create_clock – period 10.000 – name sys_clk_pin – waveform {0.000 5.000} – add [get_ports clk]
```

– Name "sys_clk_pin" is the name of the clk the vivado uses period T,

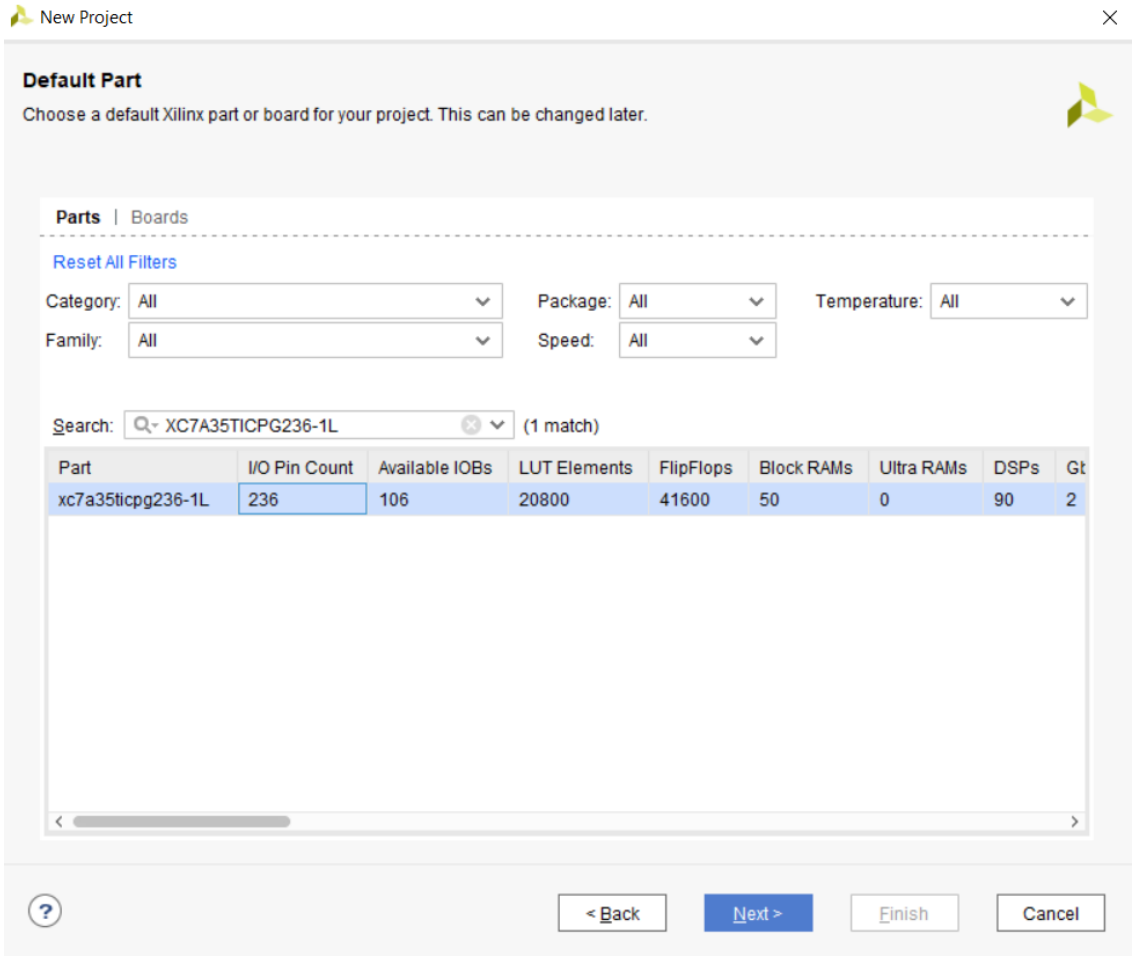
waveform{rising_time failing_time}, get_ports clk signal name in HDL CODE

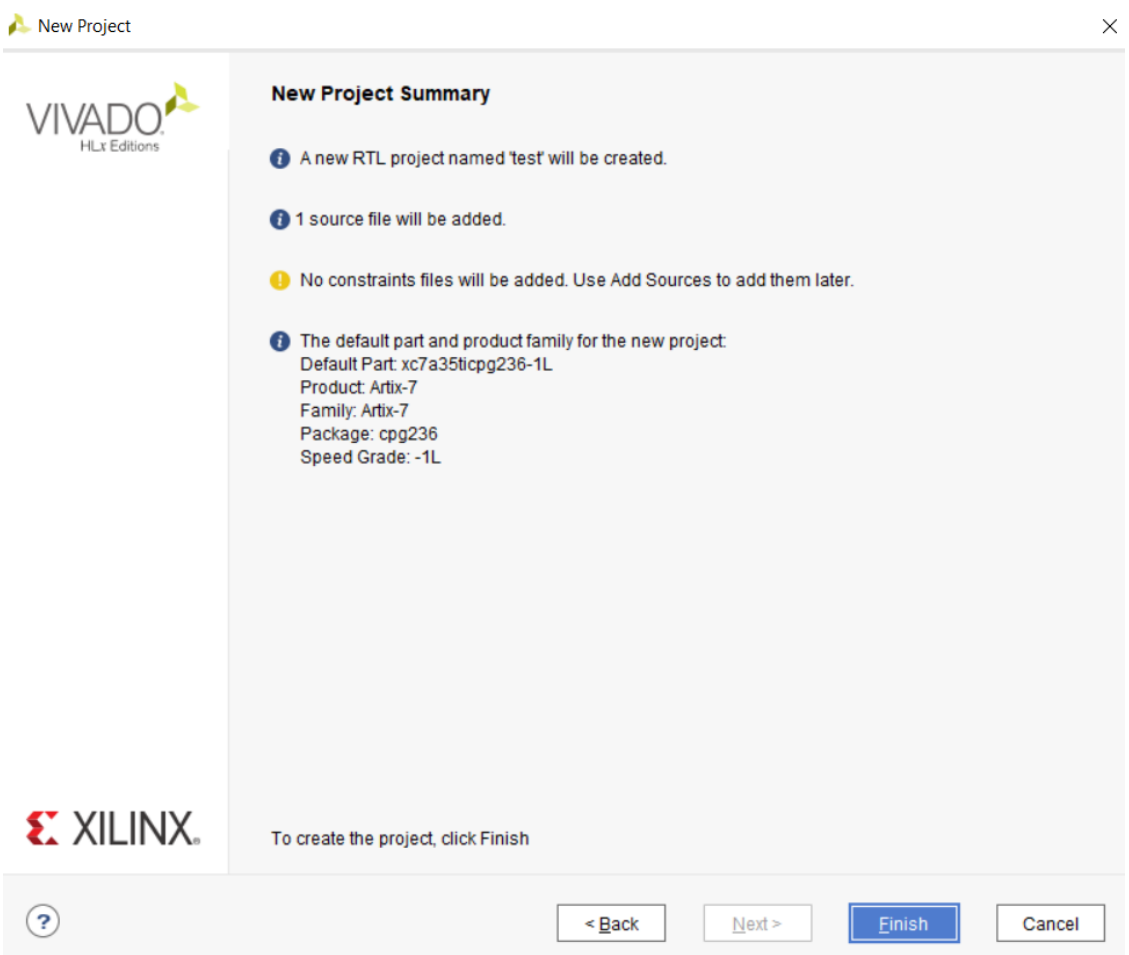
```
## LEDs
set_property – dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {result[0]}]
```

// pin U16 found in the datasheet of fpga, LVCMOS(33) is the voltage to port (3.3 Volt).

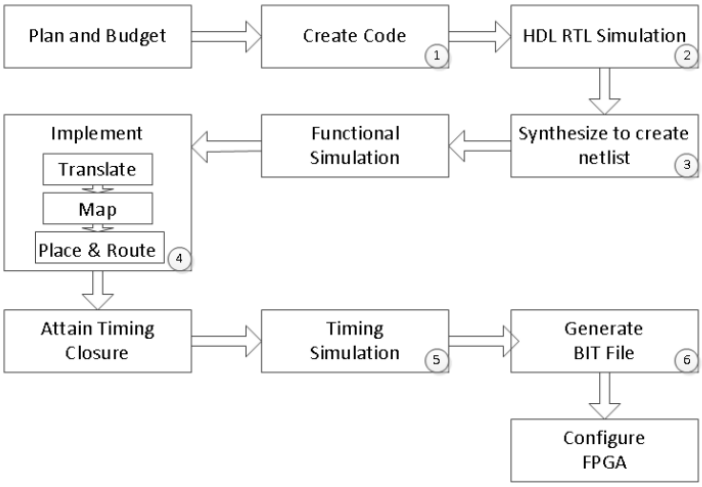
get_ports {your signal name in the HDL CODE}

- ToDo: Check Constraints (basys_master. xdc) TCL file check clock genertd and the inputs & outputs used use ctrl + '/' to Uncomment the used lines in the sw (for inputs) & leds (for outputs) & button ports (for reset signal) and the clk signal.


New Project
Default Part
Choose a default Xilinx part or board for your project. This can be changed later.
Parts | Boards
Reset All Filters
Category: All
Package: All
Temperature: All
Family: All
Speed: All
Search: XC7A35TICPG236-1L (1 match)
Part
I/O Pin Count
Available IOBs
LUT Elements
FlipFlops
Block RAMs
Ultra RAMs
DSPs
xc7a35ticpg236-1L
236
106
20800
41600
50
0
90
< Back
Next >
Finish
Cancel


New Project
VIVADO
HLx Editions
New Project Summary
A new RTL project named 'test' will be created.
1 source file will be added.
No constraints files will be added. Use Add Sources to add them later.
The default part and product family for the new project:
Default Part: xc7a35ticpg236-1L
Product: Artix-7
Family: Artix-7
Package: cpg236
Speed Grade: -1L
XILINX
To create the project, click Finish
< Back
Next >
Finish
Cancel

- Explain program layout window



Example 1 In Class: Switch_Leds

```
module Switch_leds (sw, clk, rst, led);

  input [7: 0] sw;
  input clk, rst;
  output reg [7: 0] led;
  always @(posedge clk or posedge rst) begin
    if (rst)
      led  <=  0;
    else
      led  <=  sw;
  end

endmodule
```

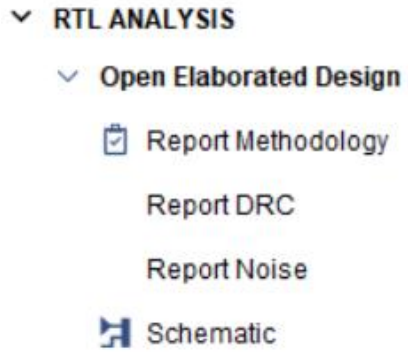
Editing the Basys3 Master XDC

```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }];
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { sw[5] }];
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { sw[6] }];
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { sw[7] }];
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }];
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }];
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }];
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }];
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }];
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }];
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }];
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }];


## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }];
```

RTL Analysis

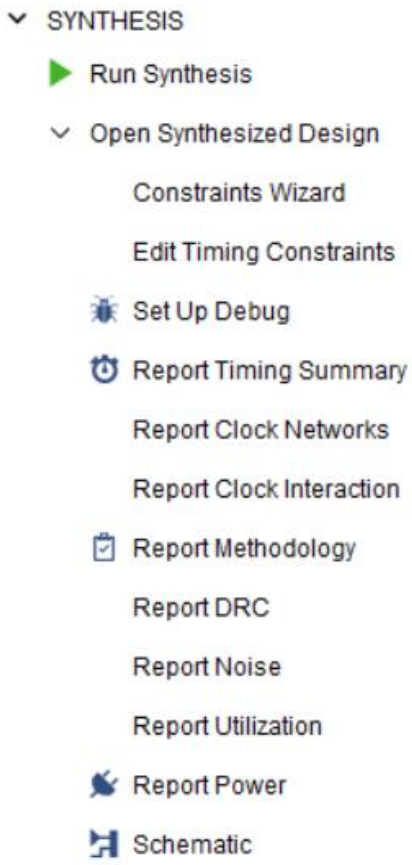
• First Step: Run RTL Analysis:

- Open Elaboration → Make Linting, Generic Synthesis Schematic using Gates



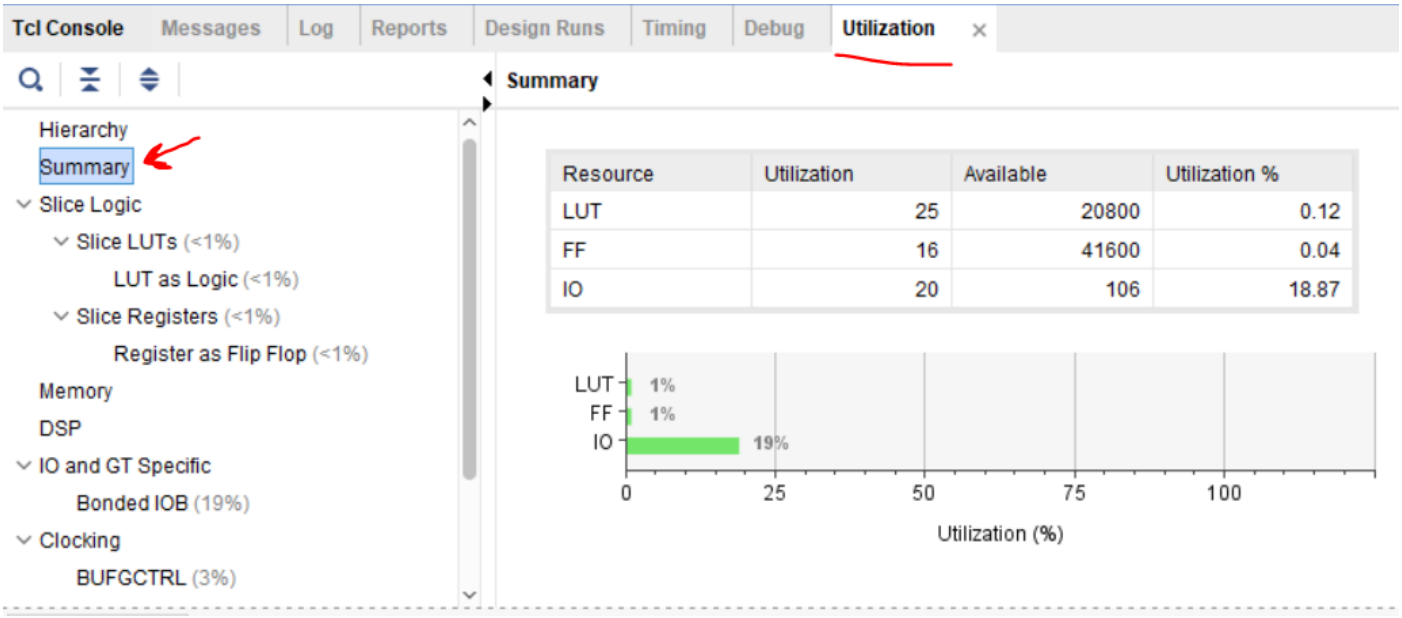
Synthesis

• Second Step: Run Synthesis:



- Open Design → Mapped Schematic on FPGA using the Emulated Board

- Report Utilization: Give me Number of Components Used from FPGA in % or numbers.

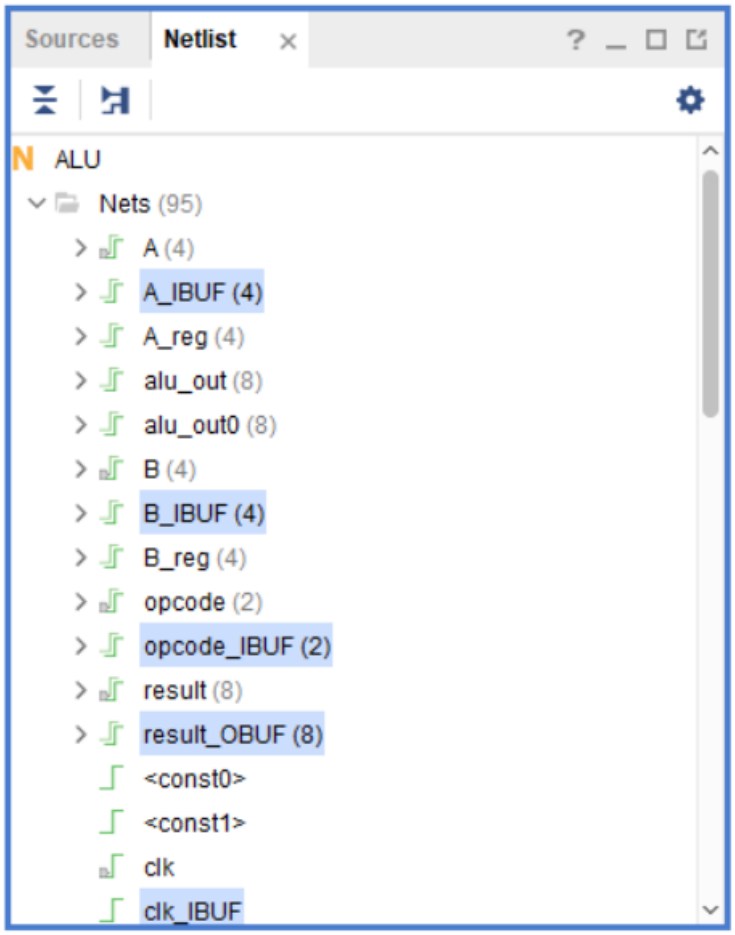


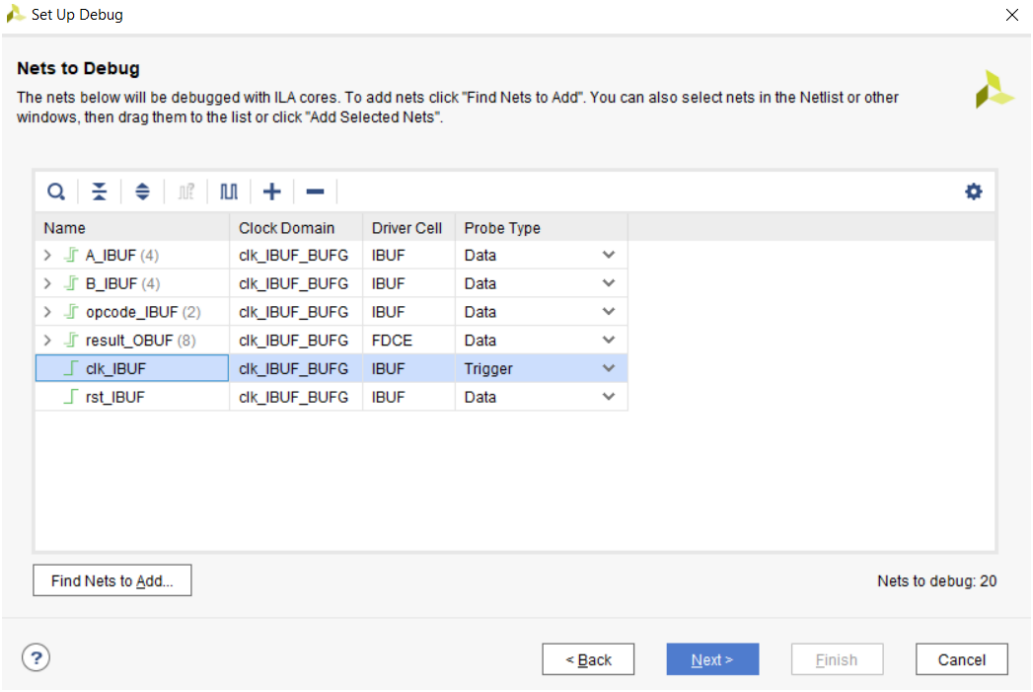
- To make Debug Core (its Role is to Show the Signals in the Waveform & Schematic) After Synthesis we

go to Netlist → Nets → choose inputs & outputs (IBUF) then click on Setup Debug.

- Set Up Debug: See Input. IBUF, output. IBUF, clk. IBUF:

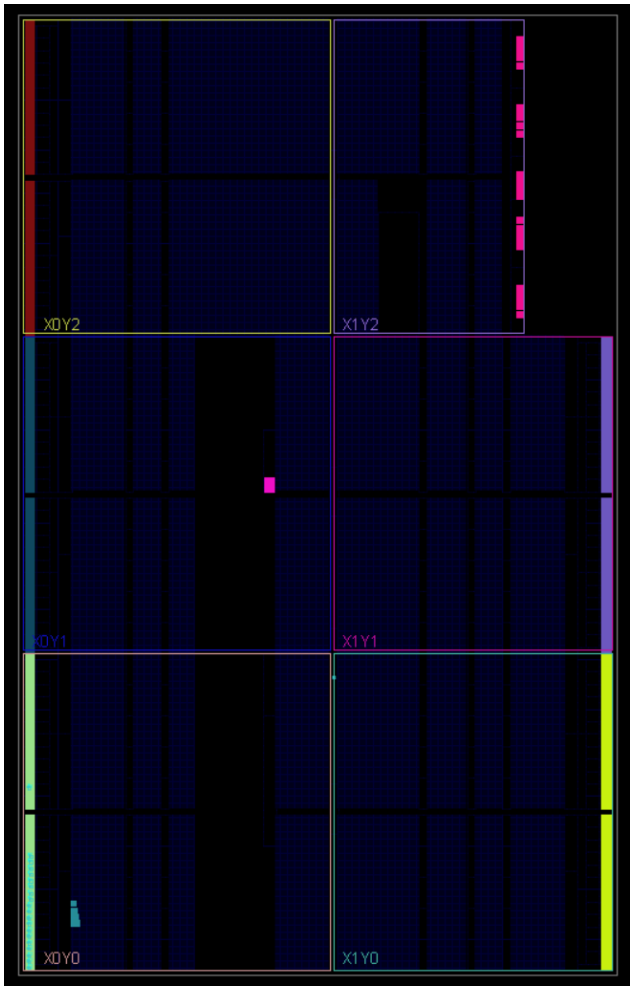
probe type for (input & output) → (Data), (Clock) → (Triggered).



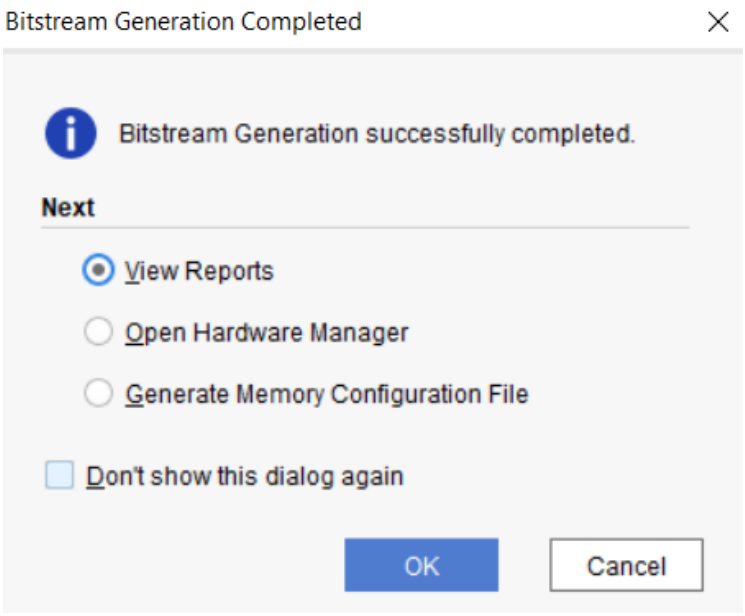


IMPLEMENTATION

• Third Step: Run Implementation Open Device → Place & Route & Packaging on FPGA Board + You Can Zoom on specific place to see the internal Logic.



BITSTREAM



• Fourth Step: Generate Bitstream

• Burn on FPGA

• open hardware manager

- auto connect target
- navigate yourprojectname.run
- choose the .bit file
- program the FPGA board
- test your design!

// VTR is an emulator like fpga tool but its open source so a little bit difficult to deal with.

• How to see changes in vivado after uploading to board:

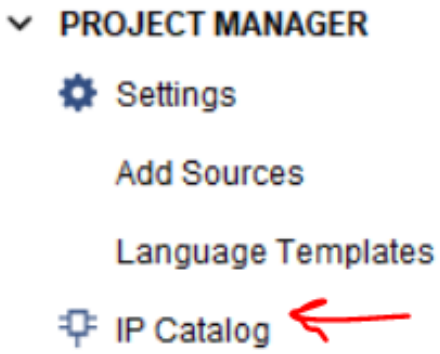
- select signals I/O and clock
- Set up Debug from Open synthesis
- choose data for I/O and trigger for clock
- regenerate bitstream
- resave the new constrain file
- in constrain file you will see new part (create_debug_core u_ila_0 ila)
- in the new schematic you will find dbghub block and (u_ila) (logic analyzer) block
- u_ila ready to use block on fpga that is used in decode setup
- after connecting choose bit stream file and debug file
  (Found in same directory of bit file)
- Questasim wave like window appear and it displays the data selected before
  toggle auto trigger auto mode to keep sampling values as changes happen.
  (This is a live display not like Questa's simulation. actual hardware is being tested)

IP CATALOG

• IP Catalog

• advantage: Already built in block that you can use and then can instant it in my design file for this built in block.

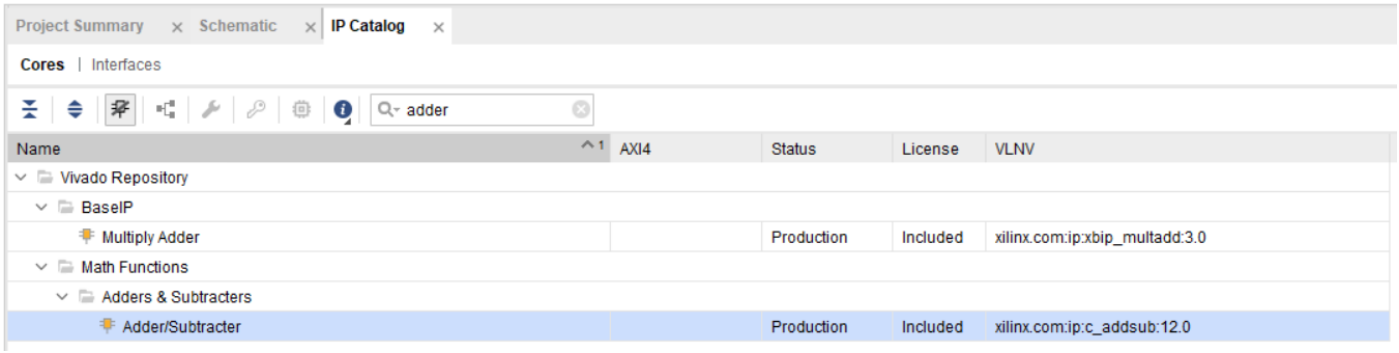
Like: Use DSP, memory, adders and multipliers, etc. (Adders & Multipliers Used in the example below).



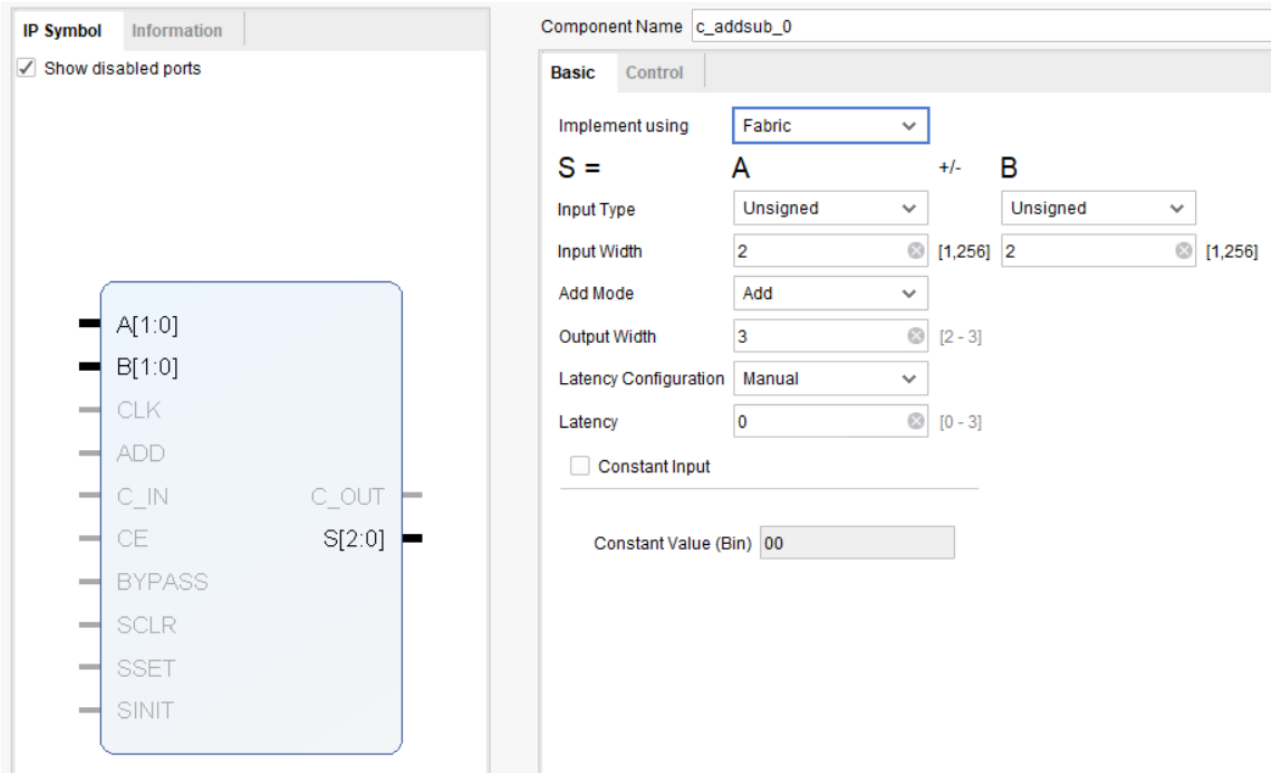
• First Click on IP Catalog on Left: Search For

- adder: double click on it

• From Basic choose the specs on input and output



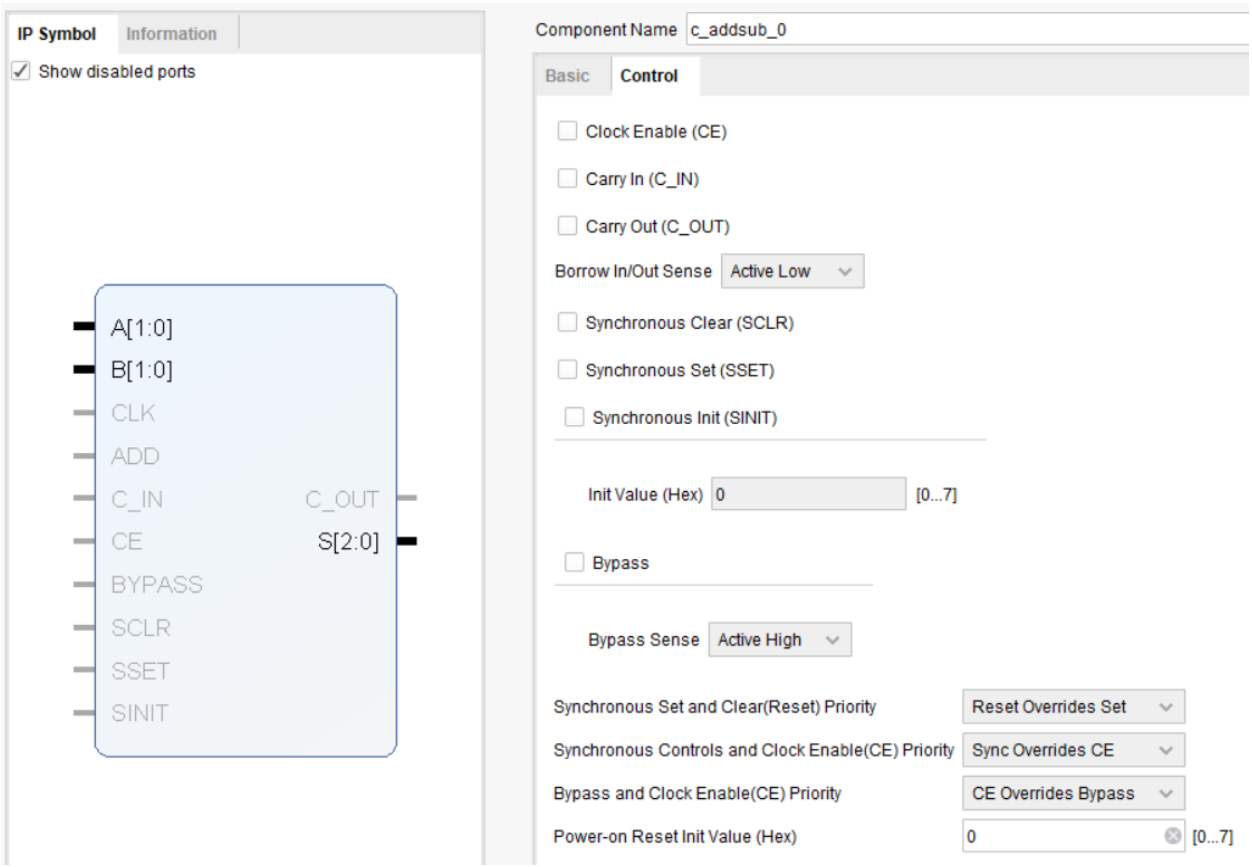
• From Control: Uncheck the Clock Enable (CE)



From Basic implement using Fabric (LUT) (Preferred) or DSP, input Type (Unsigned), input Width (=2)

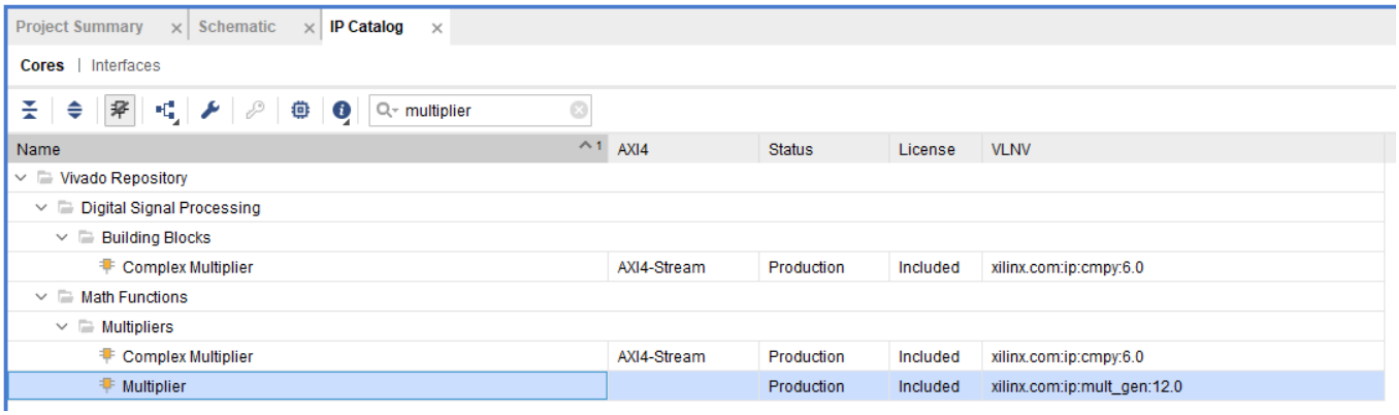
Add mode (choose Add), output width (=3), Latency Configuration (Manual),

Latency (Represent How Much We Delay the Output?) here we use Combinational so Latency = 0



- multiplier: double click on it

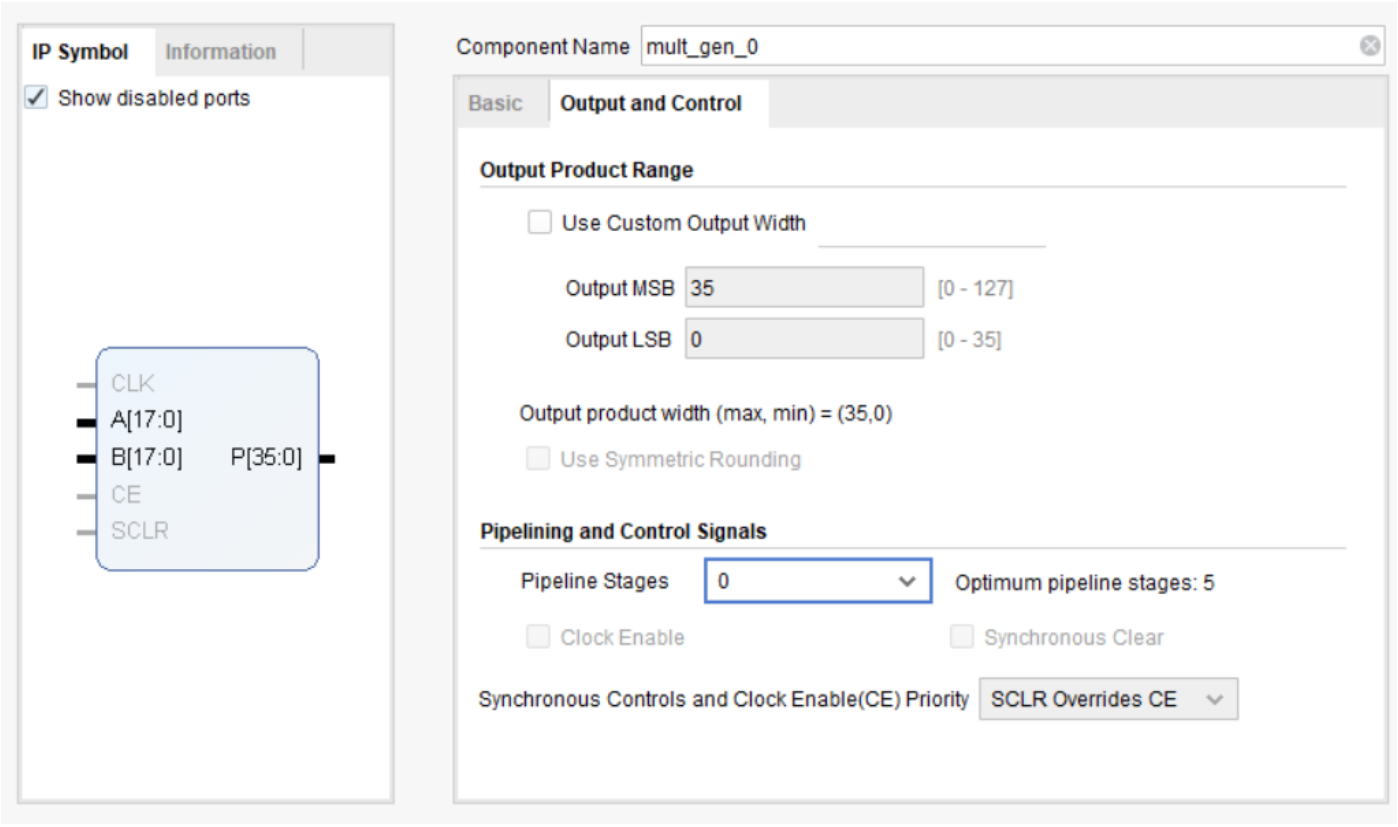
• From Basic choose the specs on input and output From Data Type (Unsigned) and put their Width.

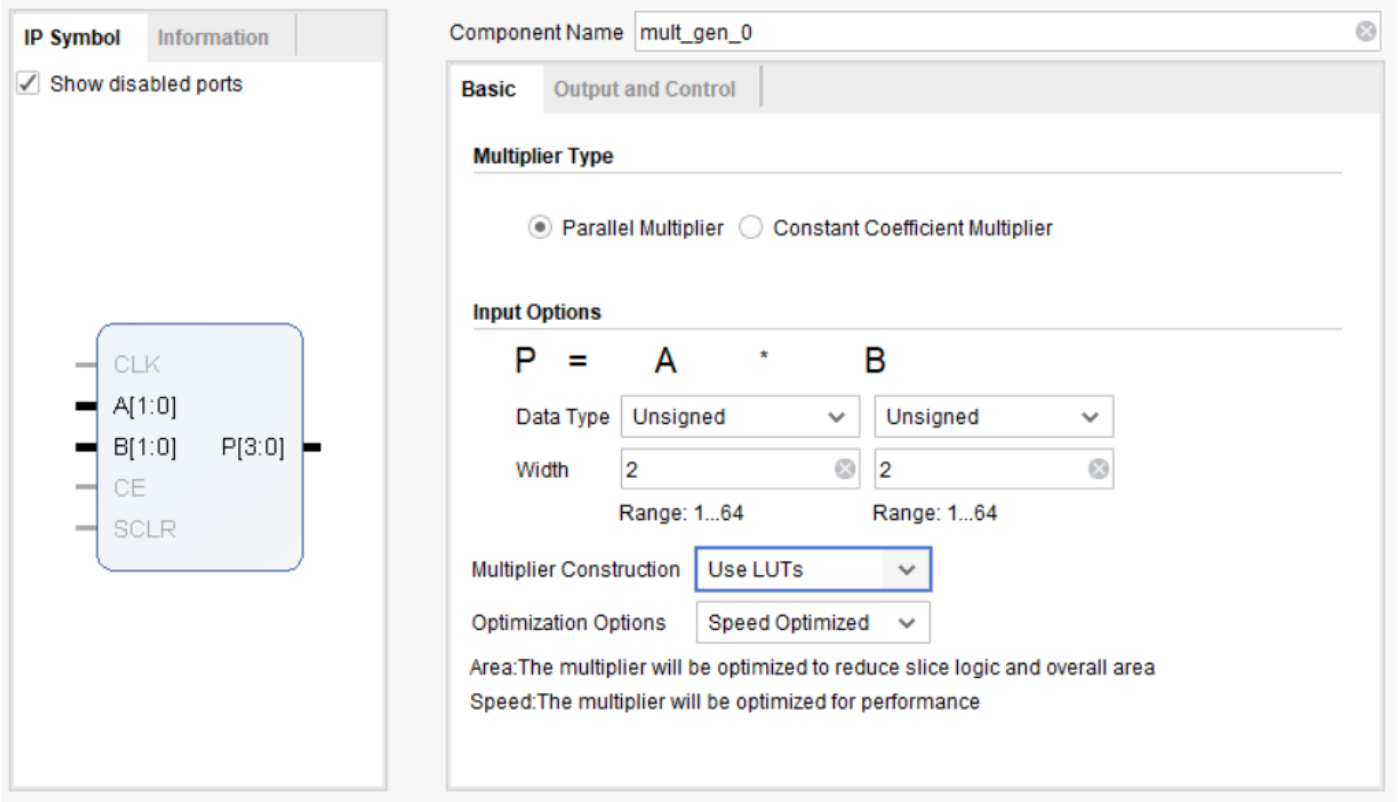


• From Output and Control from Pipelining and Control Signals

(Pipelining Represent the Stage) in Pipeline Stages = Choose

(0 = Combinational if we choose (1 = Sequential with CLK)) Like Latency in Adders.





• From Sources go to IP Sources Click in (c_addsub_0 or mult_gen_0):

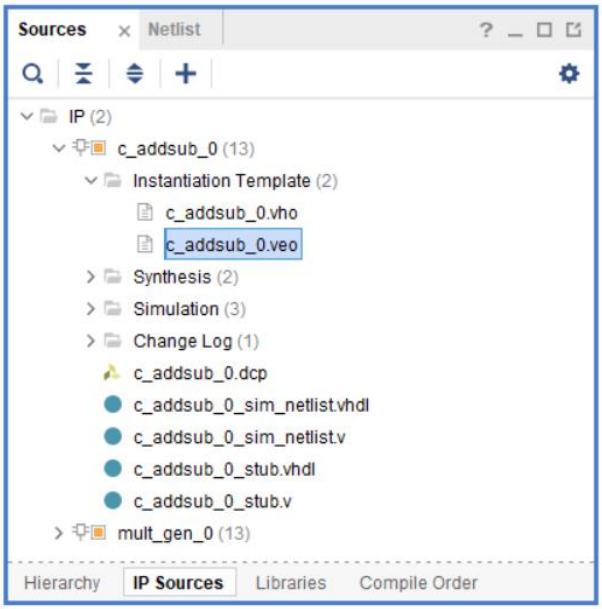
- Go to Instantiation Template and then Double Click on

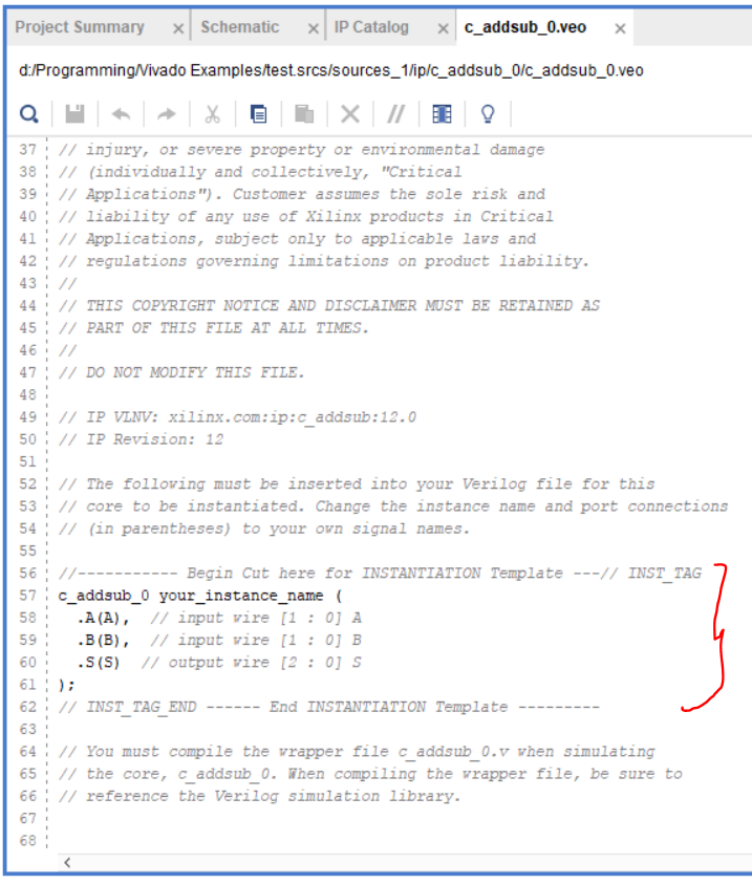
(c_addsub_0.veo or mult_gen_0.veo) Files.

And Copy the Instantiation in my Design.v in the Text Editor Give Inputs and Take Outputs.

• Don't Forget to Change the (your_instance_name) to A Readable Name and Make Wire to pass it to the output of the Instantiation.

(The Relation Between Structural and Behavioral is Wire).





Example 2 In Class: ALU

```
module ALU (A, B, opcode, clk, rst, result);

input [1: 0] A, B, opcode;
input clk, rst;
output reg [3: 0] result;

wire [3: 0] adder_out, mult_out;

c_addsub_0 Adder (
  .A(A),  // input wire [1 :  0] A
  .B(B),  // input wire [1 :  0] B
  .S(adder_out)  // output wire [2 :  0] S
  );

mult_gen_0 MULT (
  .A(A),  // input wire [1 :  0] A
  .B(B),  // input wire [1 :  0] B
  .P(mult_out)  // output wire [3 :  0] P
  );

always @(posedge clk or posedge rst) begin
  if (rst) begin
    result  <=  0;
  end
  else
    case (opcode)
      2'b00 :  result  <=  adder_out;
      2'b01 :  result  <=  mult_out;
      2'b10 :  result  <=  A | B;
      2'b11 :  result  <=  A ^ B;
    endcase
  end
endmodule
```

RUN. TCL

• RUN.TCL for Vivado // Like DO file for Questasim

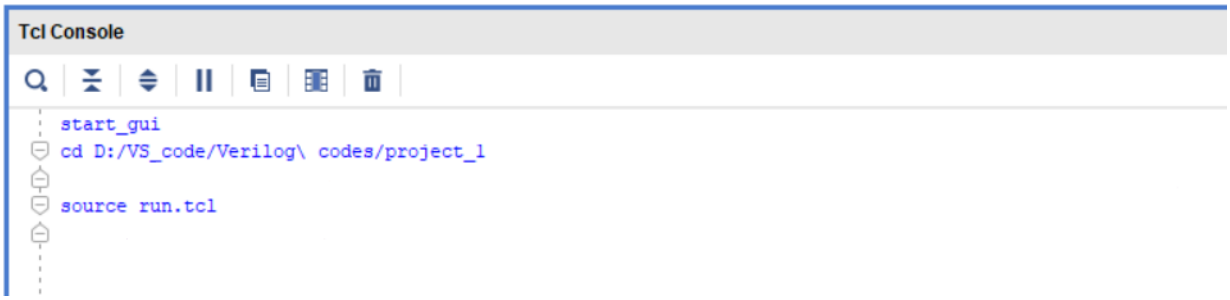
```
create_project project_1 D:/VS_code/Verilog\ codes/project_1 – part xc7a35ticpg236 – 1L – force

add_files switch_LEDs.v basys3_constraints.xdc

synth_design – rtl – top switch_LEDs > elab.log

write_schematic elaborated_schematic.pdf – format pdf – force

launch_runs synth_1 > synth.log

wait_on_run synth_1
open_run synth_1

write_schematic synthesized_schematic.pdf – format pdf – force

write_verilog – force switch_LEDs_netlist.v

launch_runs impl_1 – to_step write_bitstream

wait_on_run impl_1
open_run impl_1

open_hw

connect_hw_server
```

From the File “RUN.TCL” we change in the first 3 Lines Only.

- 1st Line: we put the Path of the folder Containing all my project files.
  Like: (file.v & file_tb.v & constraint_file.xdc & RUN.tcl)
  - Here my path is D:\VS_code\Verilog codes\project_1.

  We convert first all “ \ ” with “ / ” as Vivado understand Linux Languages
  - If I have A space in my path between words like here between Verilog and codes, there is a space we do like that “ Verilog\ codes ”
  - So, the Result of the edited path will be: D:/VS_code/Verilog\ codes/project_1
- 2nd Line: we add all (.v) files needed in this project Like: all Lower & Top Modules “Files Name Not Modules Name”
- 3rd Line: we add Only the Top Module Name (Wrapper Module Name).

- Last Step: we Open Vivado and in the TCL Console Below we Write First
  "Cd D:/VS_code/Verilog\ codes/project_1"
  And Then Write "source run.tcl"



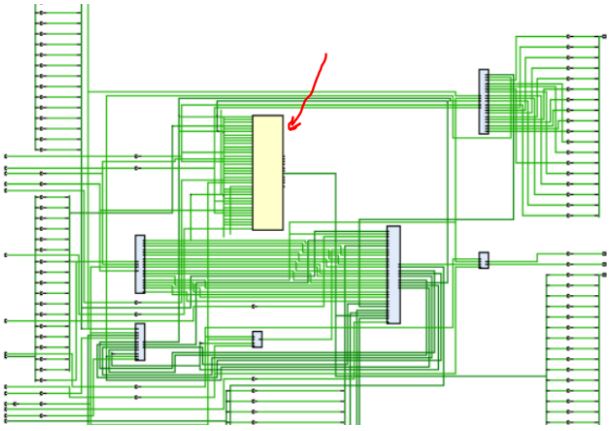
Then All the Vivado Flow Steps Will Run with addition to pdf files with clear schematics and Netlist.

DESIGN INFERRING

- Inferring The Designs

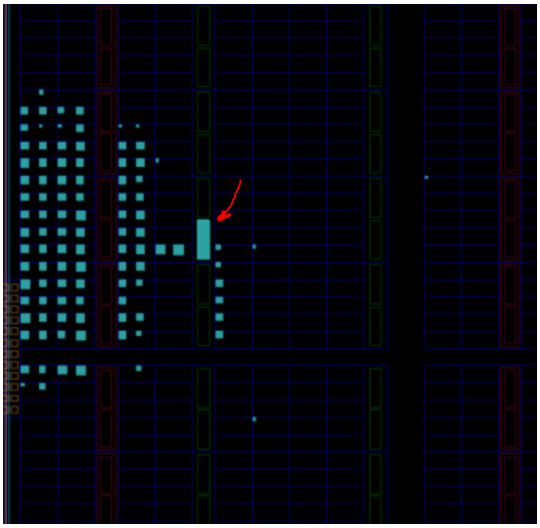
Example 1 DSP:

Any Design code have a DSP implementation in the code we must find A DSP Block in Vivado's Schematic and Implementation Device and use a number of blocks from the FPGA in Report Utilization Like this:



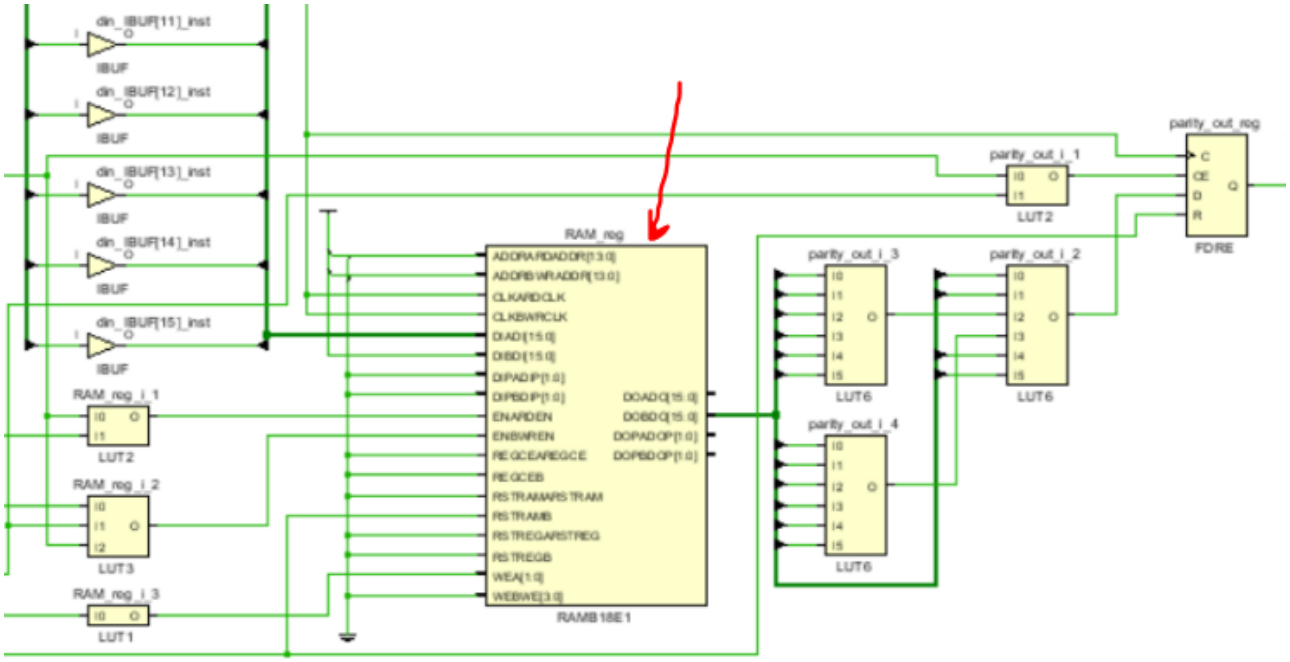
Summary

Resource	Utilization	Available	Utilization %
LUT	356	134600	0.26
FF	179	269200	0.07
DSP	1	740	0.14
IO	327	500	65.40



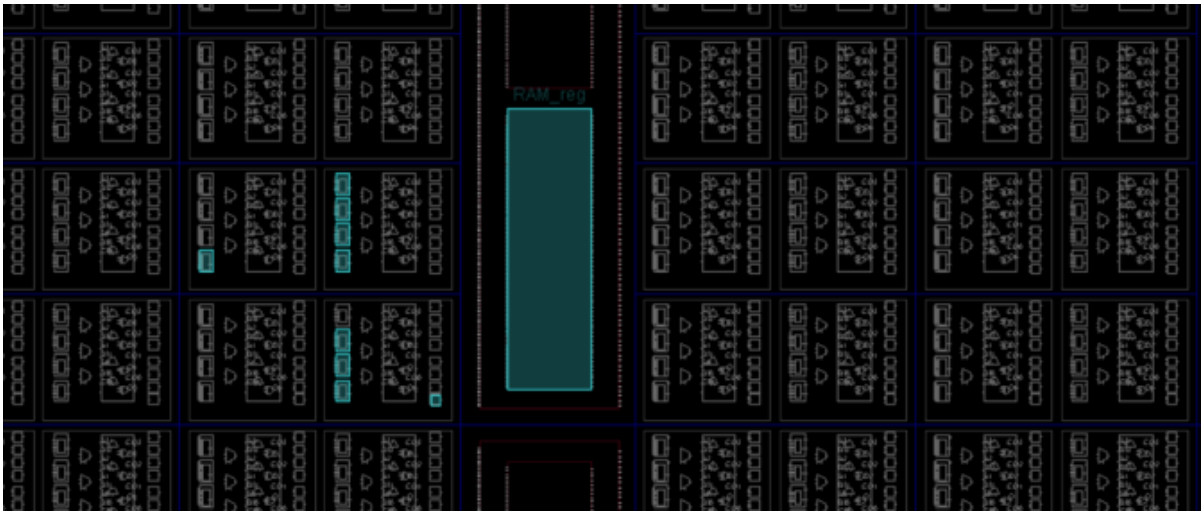
Example 2 RAM:

Any Design code have a RAM Memory implementation in the code we must find A RAM Block in Vivado's Schematic and Implementation Device and use a number of blocks from the FPGA in Report Utilization Like this:



Summary

Resource	Utilization	Available	Utilization %
LUT	7	20800	0.03
FF	1	41600	0.00
BRAM	0.50	50	1.00
IO	38	106	35.85



TIMING CONTROL

• Timing Control

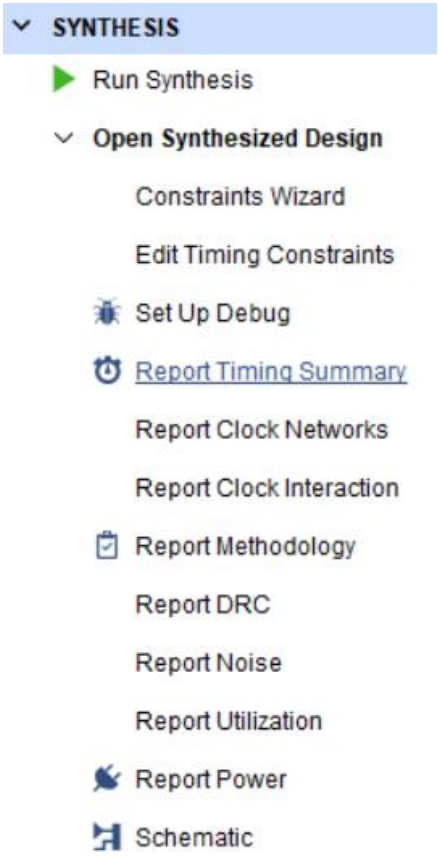
Like ALU Example Above but with FFs. For The inputs to see timing Violations.

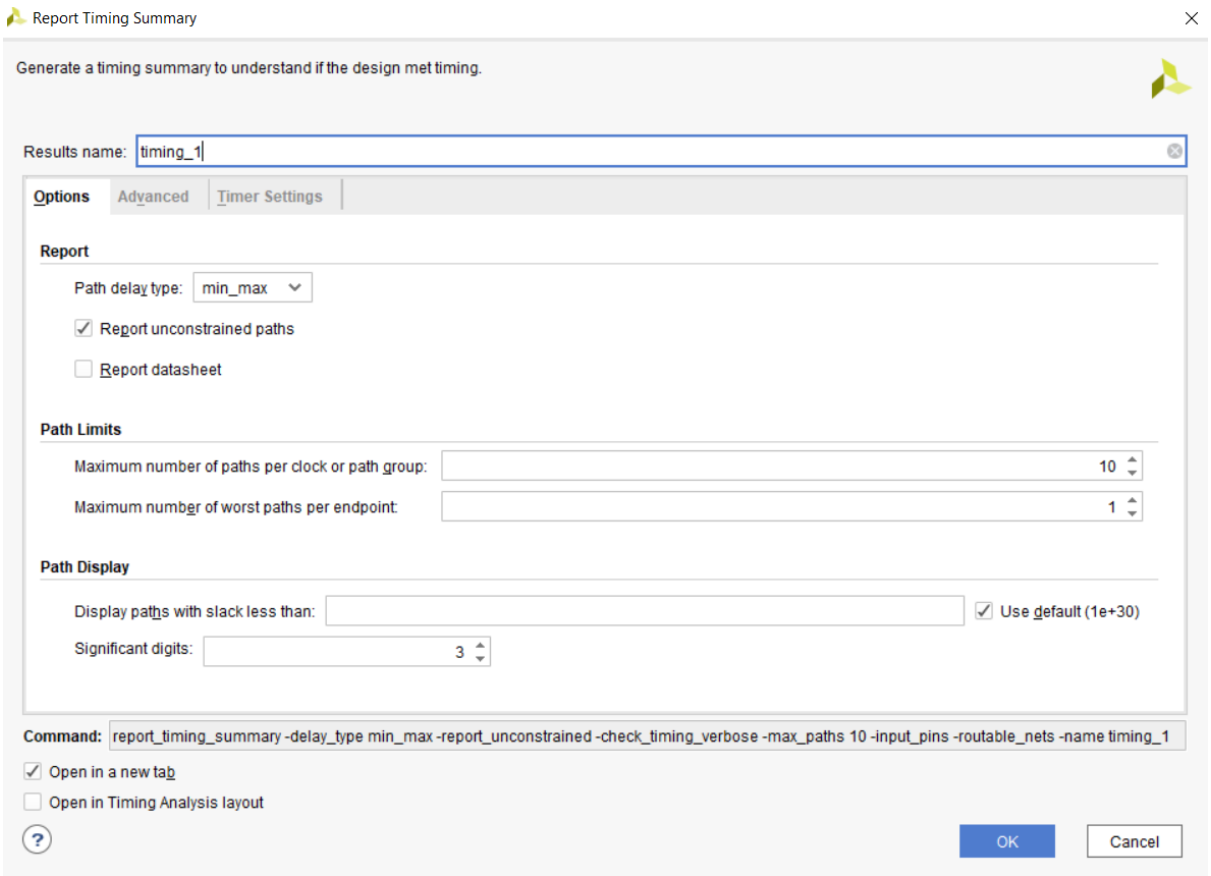
We Use This Example to Show The Timing Report Summary.

```
module ALU (A, B, opcode, clk, rst, result);
input [3: 0] A, B;
input [1: 0] opcode;
input clk, rst;
output reg [7: 0] result;
reg [7: 0] alu_out;
reg [3: 0] A_reg, B_reg;
reg [1: 0] opcode_reg;
always @(posedge clk or posedge rst) begin
  if(rst) begin
    A_reg <= 0;
    B_reg <= 0;
    opcode_reg <= 0;
  end
  else begin
    A_reg <= A;
    B_reg <= B;
    opcode_reg <= opcode;
  end
end
always @(*) begin
  case(opcode)
    2'b00 : alu_out = A_reg + B_reg;
    2'b01 : alu_out = A_reg * B_reg;
    2'b10 : alu_out = A_reg | B_reg;
    2'b11 : alu_out = A_reg ^ B_reg;
  endcase
end
always @(posedge clk or posedge rst) begin
  if(rst)
    result <= 0;
  else
    result <= alu_out;
end
endmodule
```

• 1st Step: We Must First Do the Constraint file and Definition for CLK and make the Report Time for it.

• 2nd Step: From Synthesis we Click on Report Timing Summary Then Click OK.





Timing × Debug

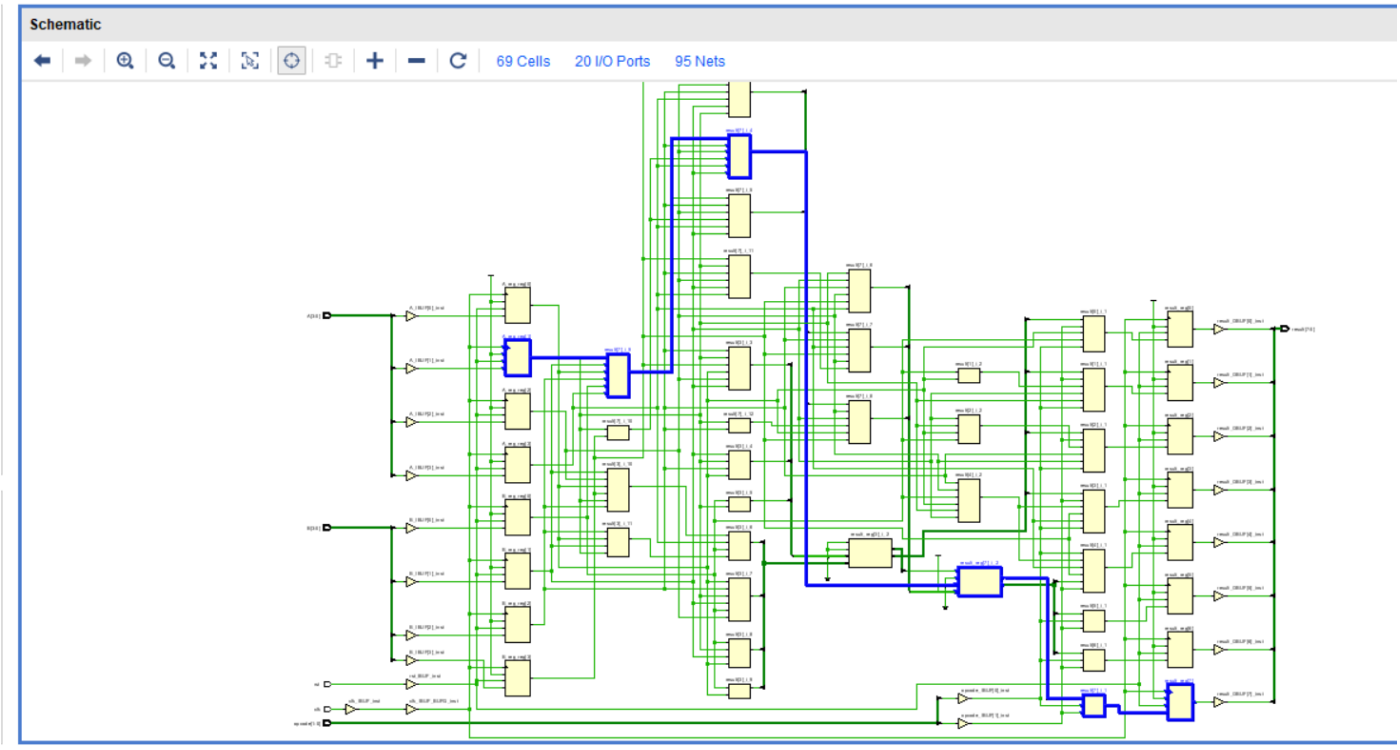
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.332 ns	Worst Hold Slack (WHS):	0.302 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	8	Total Number of Endpoints:	8	Total Number of Endpoints:	17

All user specified timing constraints are met.

• 3rd Step: We then Click on the Worst Negative Slack Blue Number (5.332 ns) which represent the Delay of the Critical (Longest) Path which is the Less time Slack Number.

• 4th Step: Then we Hold on to this Less Slack number then it points to the longest path in the Design Schematic.



Timing × Debug

Intra-Clock Paths - sys_clk_pin - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	5.332	4	5	16	A_reg_reg[1]/C	result_reg[7]/D	4.517	1.500	3.017	10.0	sys_clk_pin	sys_clk_pin	
Path 2	5.568	4	5	16	A_reg_reg[1]/C	result_reg[6]/D	4.281	1.730	2.551	10.0	sys_clk_pin	sys_clk_pin	
Path 3	5.899	5	6	16	A_reg_reg[1]/C	result_reg[5]/D	3.950	1.729	2.221	10.0	sys_clk_pin	sys_clk_pin	
Path 4	6.088	5	6	16	A_reg_reg[1]/C	result_reg[4]/D	3.761	1.541	2.220	10.0	sys_clk_pin	sys_clk_pin	
Path 5	6.363	4	5	16	B_reg_reg[1]/C	result_reg[3]/D	3.486	1.430	2.056	10.0	sys_clk_pin	sys_clk_pin	
Path 6	6.970	3	4	16	A_reg_reg[1]/C	result_reg[2]/D	2.879	1.632	1.247	10.0	sys_clk_pin	sys_clk_pin	
Path 7	7.458	3	4	16	A_reg_reg[1]/C	result_reg[1]/D	2.391	1.284	1.107	10.0	sys_clk_pin	sys_clk_pin	
Path 8	7.732	3	4	10	B_reg_reg[0]/C	result_reg[0]/D	2.117	1.293	0.824	10.0	sys_clk_pin	sys_clk_pin	

Accuracy in Report Timing Summary in Implementation > In Synthesis Due to Violations and Place and Route Done and Report The Problems.

Where:

• Levels: Represents the Number of Blocks (Flip Flops or LUTs) I Pass in The Critical Path, Levels Refers to the number of logic stages or levels, that a signal path traverse.

• Total Delay: Represents the (t_{pd}) (Propagation Delay)→ Combinational Delays

• Logic Delay: Delay caused by the logic elements in the path (t_{qd}).

• Net Delay: Delay introduced by the physical interconnect (wires or traces) between logic elements (t_{pc}).

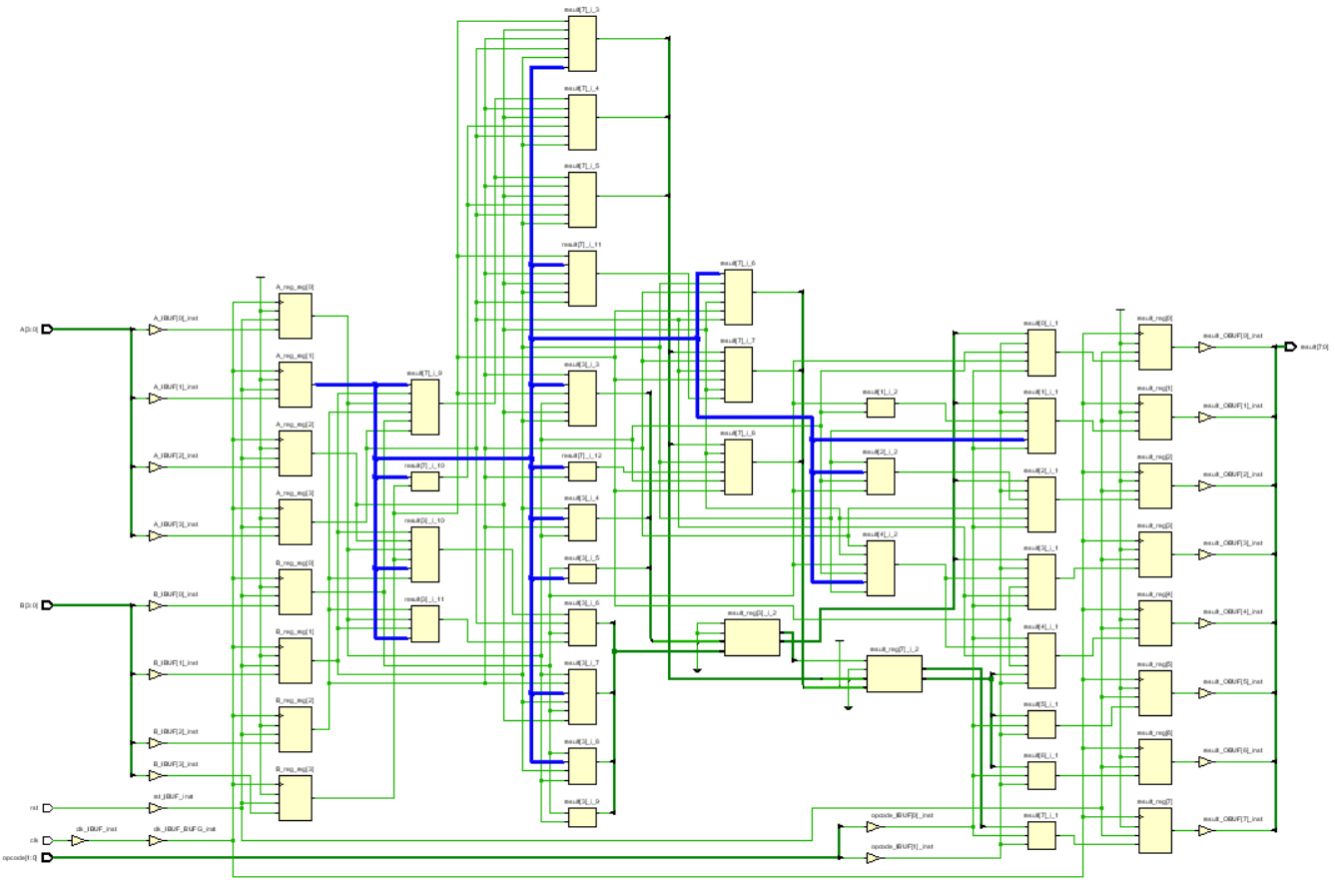
Note: Total Delay = Logic Delay + Net Delay.

• Requirement: Represents (t_{CLK}).

Note: t_{pcq} & t_{setup} are not Written in the Table.

• Routes: Refer to the paths that signals take as they traverse the interconnect resources on the FPGA chip.

• High Fanout: The Number of Connections out from the path Registers.



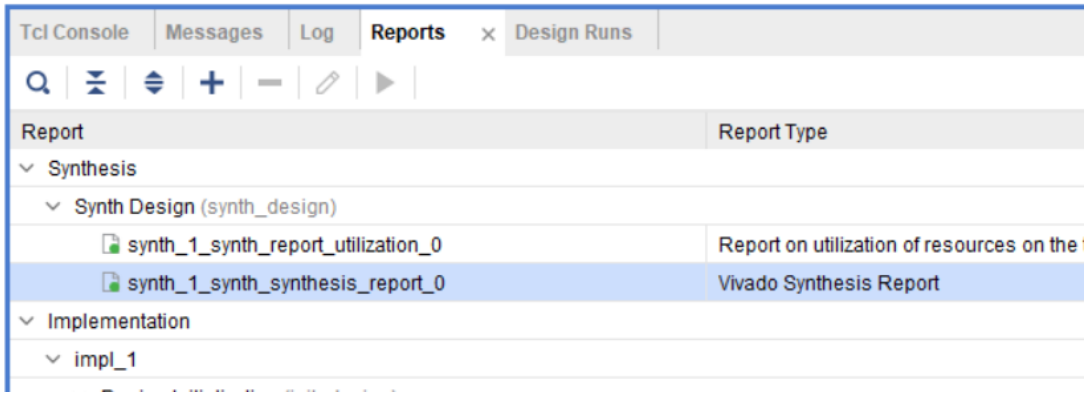
FSM Encoding

• FSM Encoding

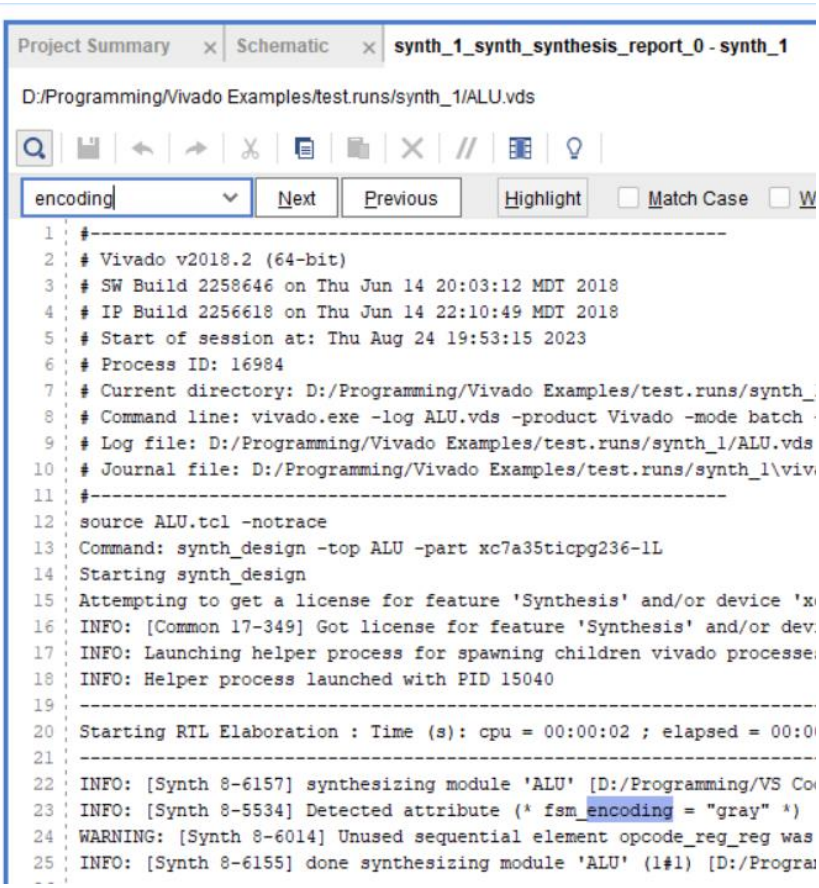
• 1st Step: We Write in the Design Code using only one of the following vivado attributes:

```
(* fsm_encoding = "gray" *)
(* fsm_encoding = "one_hot" *)
(* fsm_encoding = "sequential" *)
```

• 2nd Step: From Reports double Click on synth_1_synthesis_report_0



• 3rd Step: we can Search for encoding using ctrl + F and write encoding.



- 4th Step: we can see the difference between different types of encoding

Schematic | synth_1_synth_synthesis_report_0 - synth_1

D:/Vivado2018.2/fsm/project_1/project_1.runs/synth_1/FSM_div_three.vds

encodi | Next | Previous | Highlight | Match Case | Whole Words | 4 Match(es)

```
    Parameter S3 bound to: 2'b11
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "gray" *) [D:/Vivado2018.2/fsm/FSM_Div_three.v:12]
INFO: [Synth 8-6155] done synthesizing module 'FSM_div_three' (1#1) [D:/Vivado2018.2/fsm/FSM_Div_three.v:1]
---------------------------------------------------------------------------------
Finished Synthesize : Time (s): cpu = 00:00:02 ; elapsed = 00:00:02 . Memory (MB): peak = 436.543 ; gain = 129.902
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Finished Constraint Validation : Time (s): cpu = 00:00:02 ; elapsed = 00:00:03 . Memory (MB): peak = 436.543 ; gain = 129.902
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Start Loading Part and Timing Information
---------------------------------------------------------------------------------
Loading part: xc7vx485tffg1157-1
---------------------------------------------------------------------------------
Finished Loading Part and Timing Information : Time (s): cpu = 00:00:02 ; elapsed = 00:00:03 . Memory (MB): peak = 436.543 ; gain = 129.902
---------------------------------------------------------------------------------
INFO: [Device 21-403] Loading part xc7vx485tffg1157-1
INFO: [Synth 8-802] inferred FSM for state register 'cs_reg' in module 'FSM_div_three'
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5544] ROM "ns" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
---------------------------------------------------------------------------------------------------
                   State |                     New Encoding |               Previous Encoding
---------------------------------------------------------------------------------------------------
                      S0 |                               00 |                               00
                      S1 |                               01 |                               01
                      S2 |                               11 |                               10
                      S3 |                               10 |                               11
---------------------------------------------------------------------------------------------------
```

"FSM_Encoding": one_hot , sequential , johnson , gray , compact , user_encoding , none , auto