# Xilinx Vivado Basics (Part I)

October 11, 2022

Ahmet Can Mert
ahmet.mert@iaik.tugraz.at

TU Graz

# Overview

- Xilinx Vivado tool is a software for simulation, synthesis, implementation and analysis of HDL designs for Xilinx FPGAs.
  - HW development
- Xilinx SDK (Software Development Kit) tool is a design environment for creating embedded applications
  - SW development

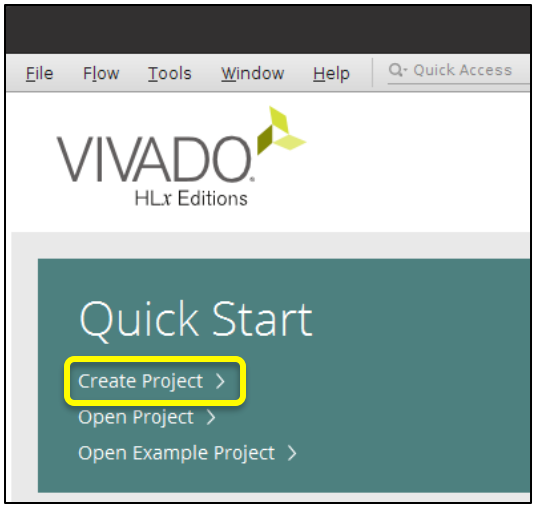
- In this tutorial, you will only focus on HW development:
  - Xilinx Vivado interface
  - How to create a project
  - How to add/create design and simulation files
  - How to run simulation/debug
  - How to perform synthesis/implementation (with constraint files)
  - How to add Xilinx IPs to your project

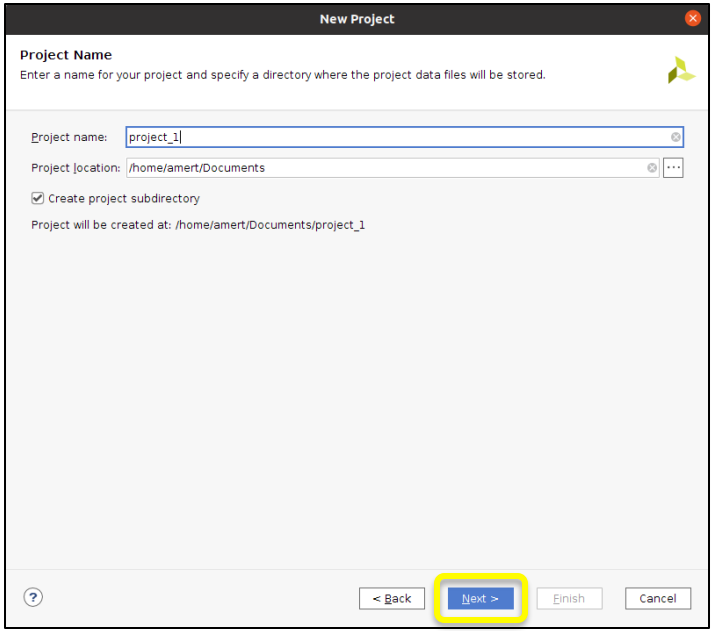
- Vivado Design Suite User Guide: https://docs.xilinx.com/r/2021.1-English/ug893-vivado-ide/Introduction

# Creating a New Project

# Creating a New Project

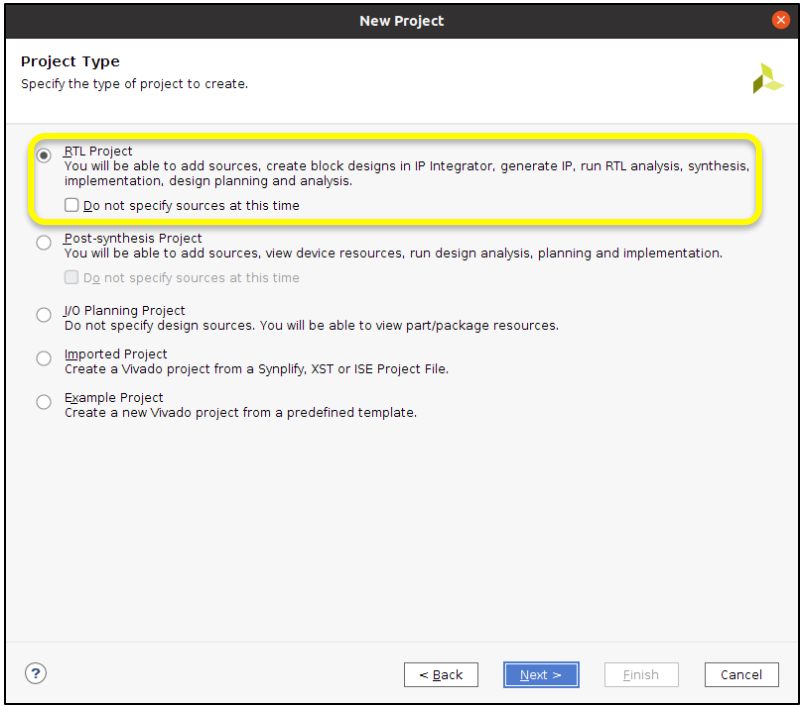
- Launch Xilinx Vivado 2019.1 and click on *Create Project* in Quick Start tab (or click on *File -> Project -> New*). Set your project name and location, then click on *Next*.
  - You can also open an existing project by clicking on *Open Project* in Quick Start tab (or click on *File -> Project -> Open*), then browsing and selecting *project_name.xpr* file.

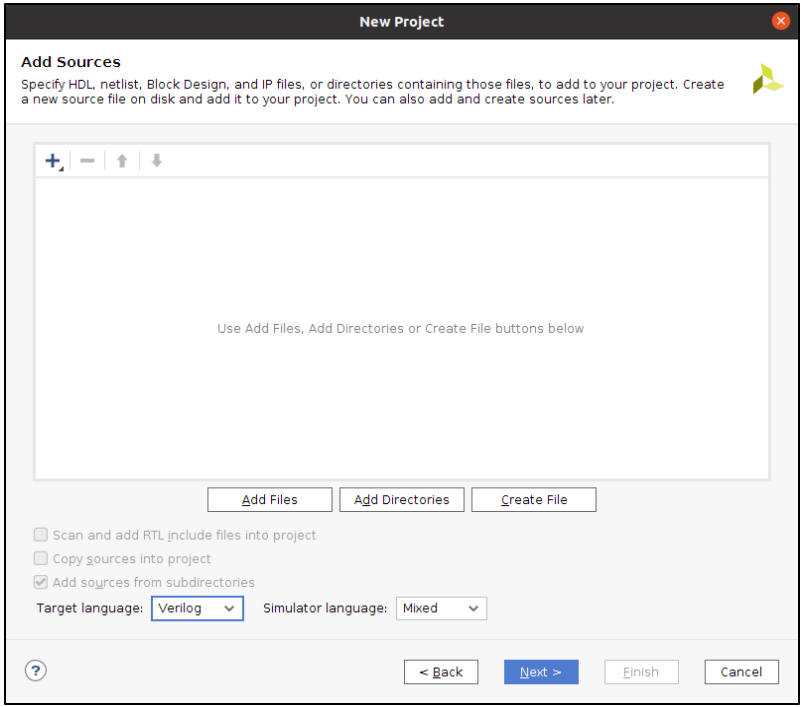


# Creating a New Project

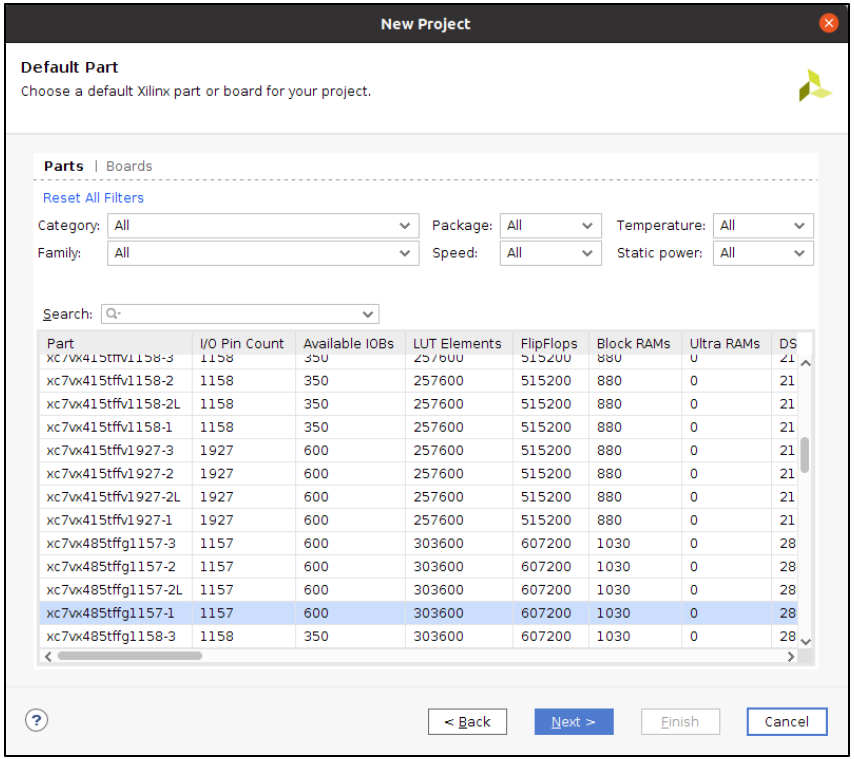
- Select project type as *RTL Project* and click on Next. In the next step, you can either add design sources/constraint files by clicking on *Create File* button or you can skip this step by clicking on *Next* (you can create files after you created the project).

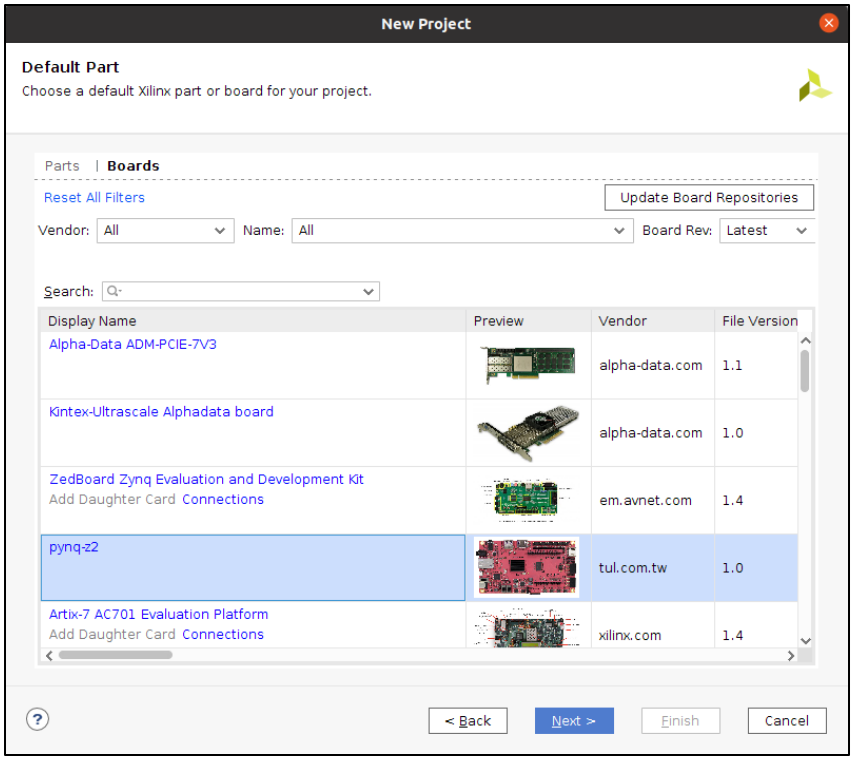


# Creating a New Project

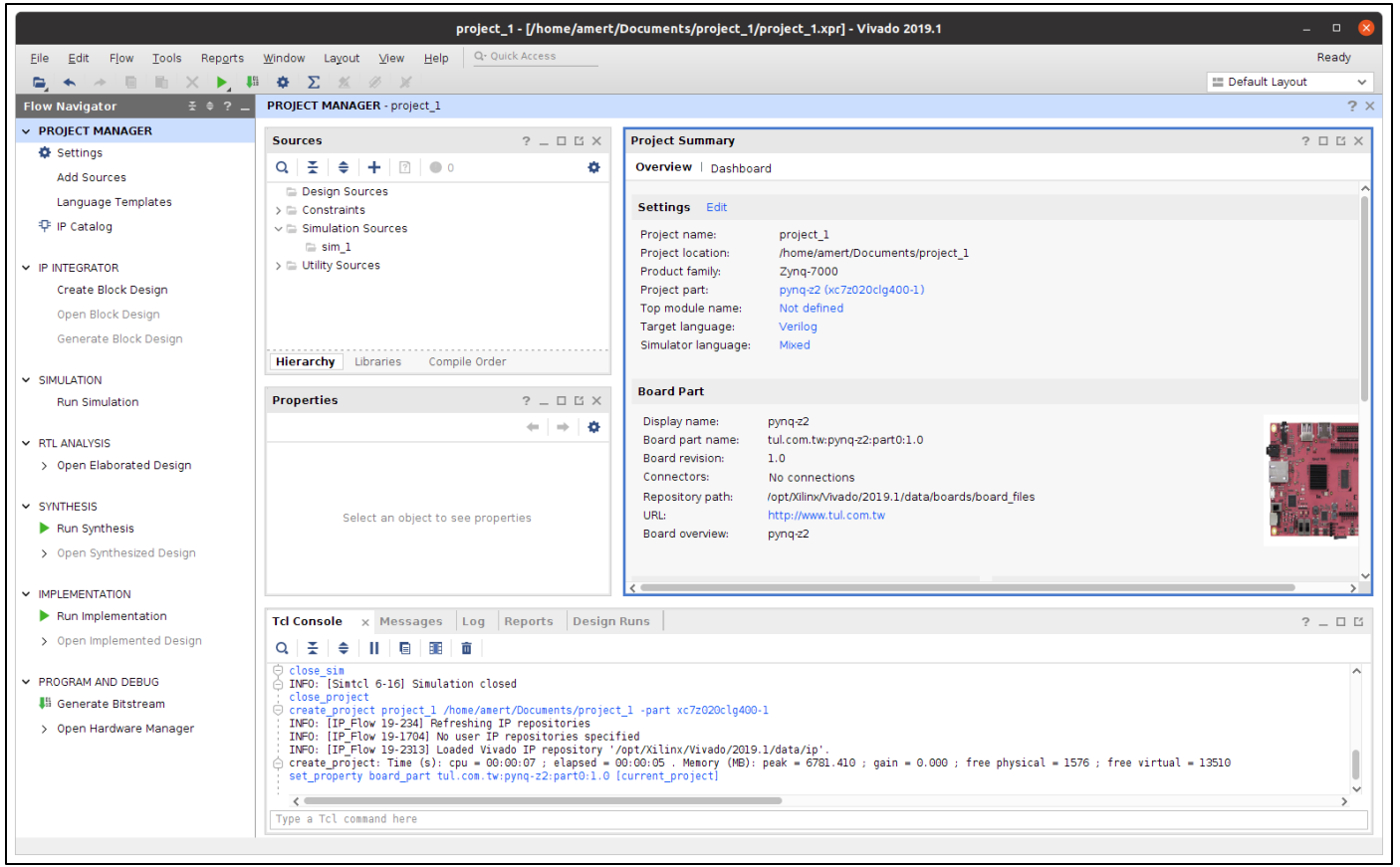
- You should select an *FPGA device* or *Board* for your project. In order to select PYNQ-Z2 board, switch to *Boards* tab and select *pynq-z2*. Then click on *Next* and *Finish*.



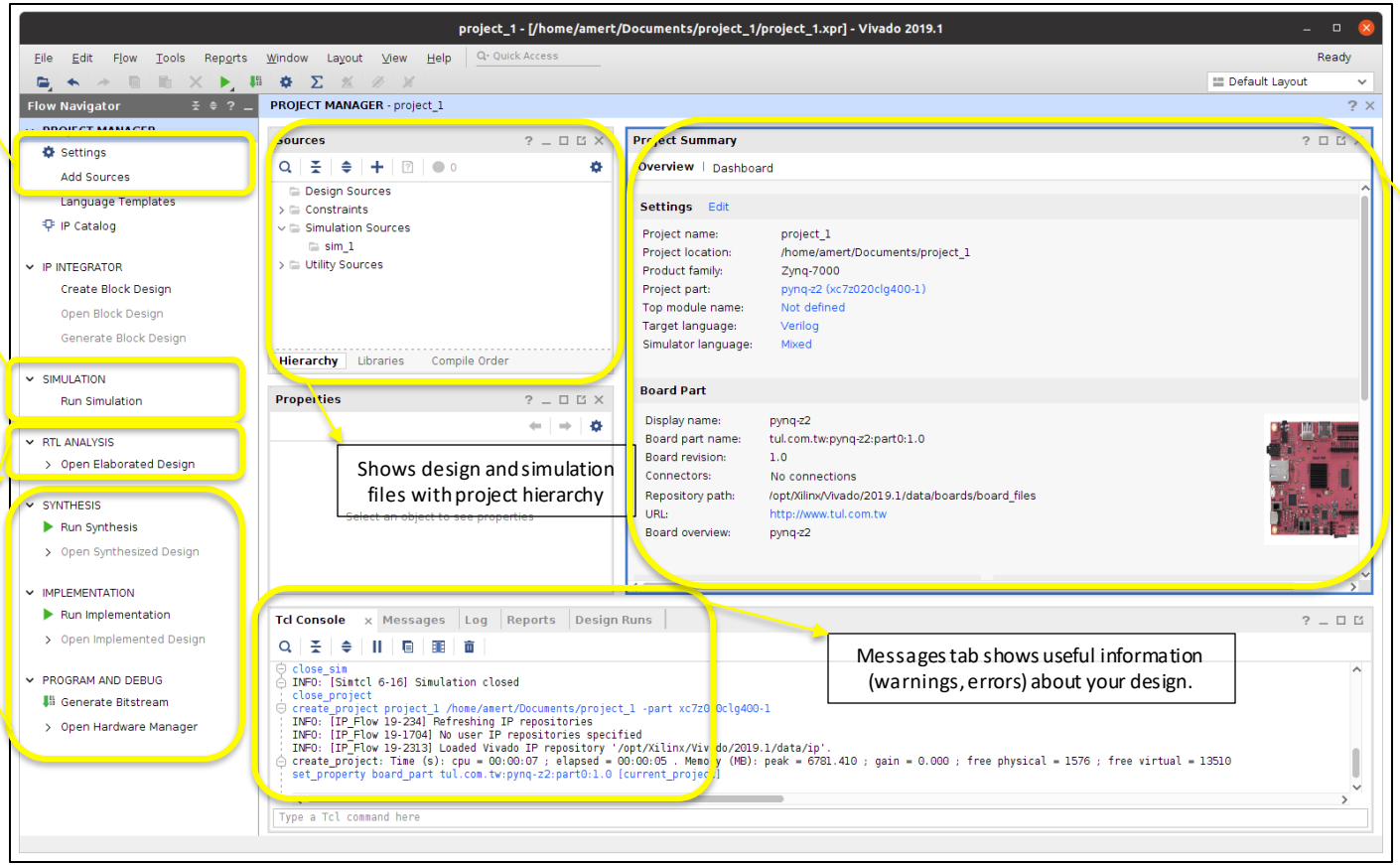



# Vivado Interface

# Vivado Interface

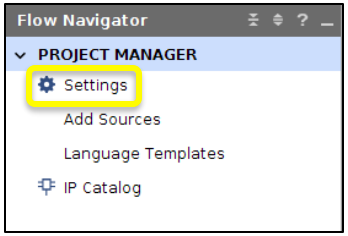


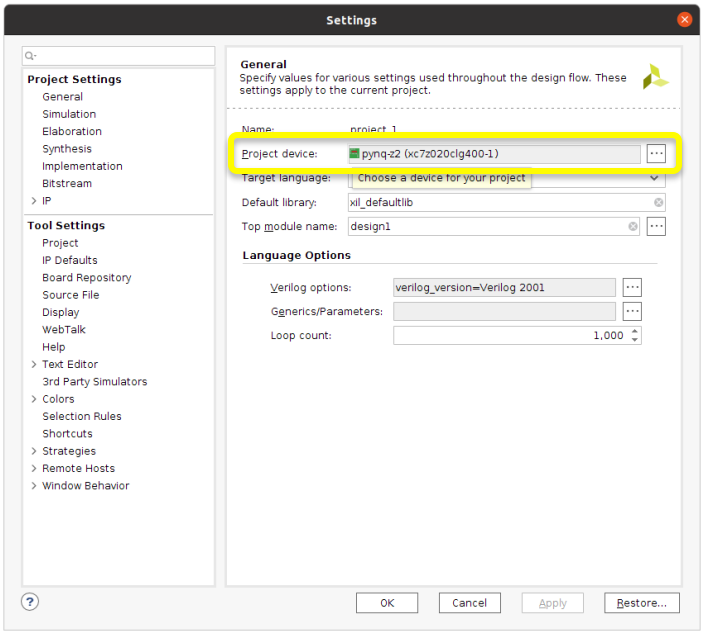
# Vivado Interface



# Vivado Interface

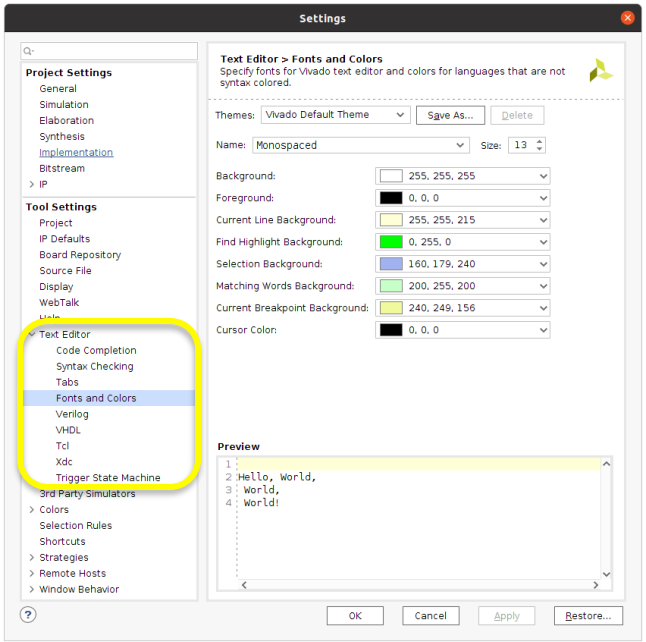
- From *Settings* under *PROJECT MANAGER* tab, you can change the settings of your project (e.g., Board or FPGA).



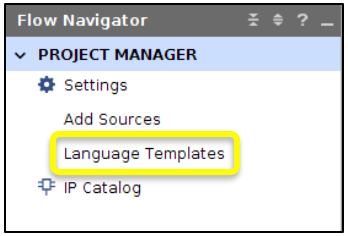
# Vivado Interface

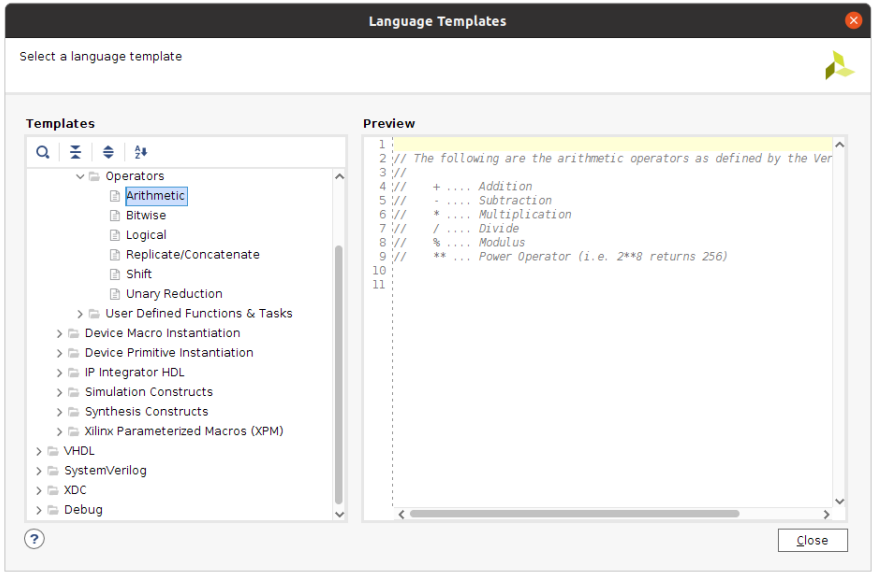
- From *Settings* under *PROJECT MANAGER* tab, you can change the settings of your project (e.g., Board or FPGA).
  - For example, you can change the font type and size of the text editor.



# Vivado Interface

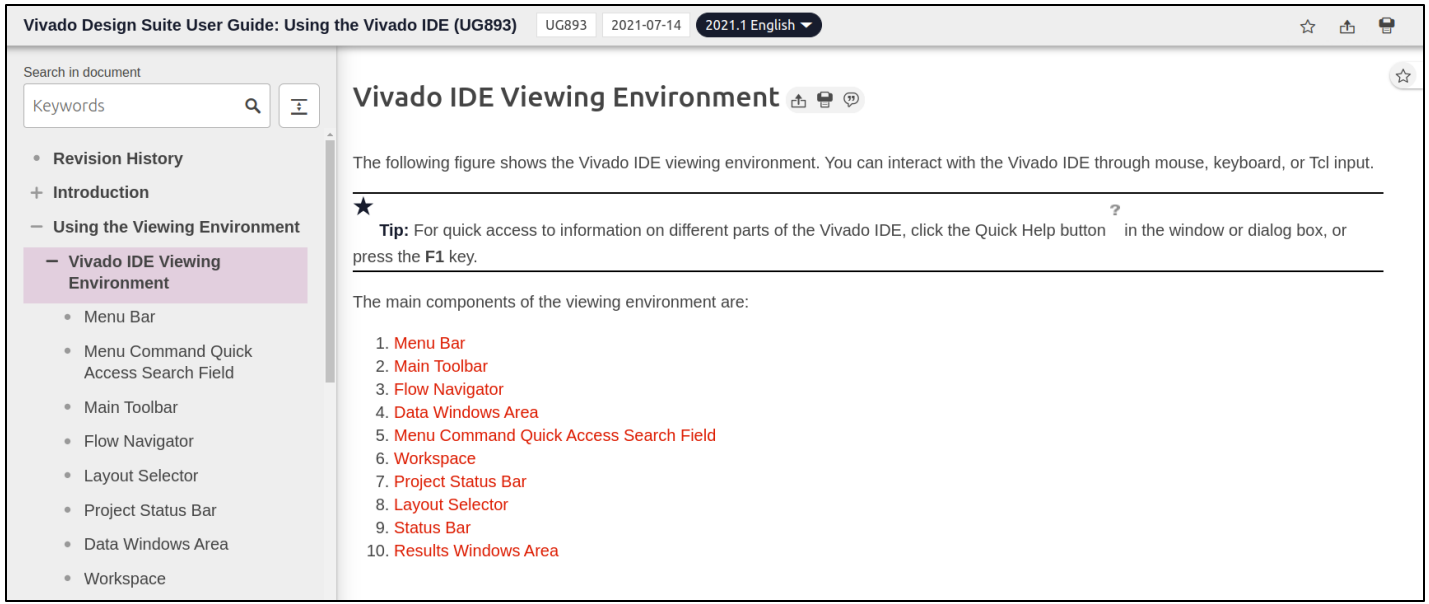
- *Language Templates* under *PROJECT MANAGER* tab shows useful Verilog and Xilinx IP constructions.



# Vivado Interface

- See *Vivado IDE Viewing Enviroment* chapter of *Vivado Design Suite User Guide*[1].

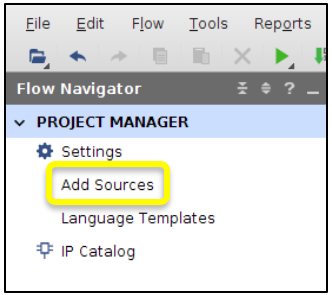


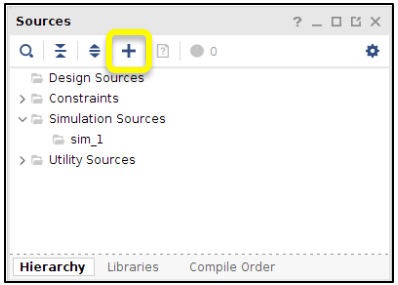

[1] https://docs.xilinx.com/r/2021.1-English/ug893-vivado-ide/Vivado-IDE-Viewing-Environment

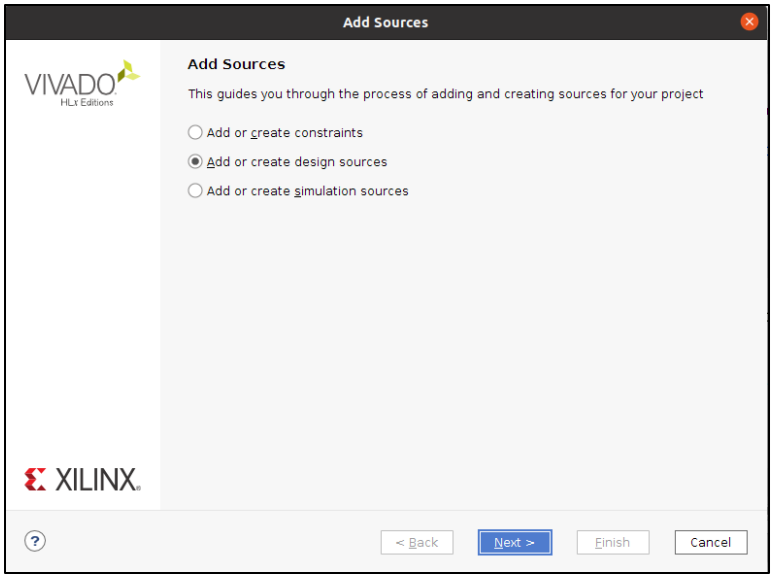
# Adding/Creating Design and Simulation Files

# Create a Design Source

- For your design, you should add a design source (*.v file). Click on *Add Sources* in Flow Navigator Window or "+" button on Sources window or *File -> Add Sources* on menu bar, select *Add or create design resources* and then click on Next.

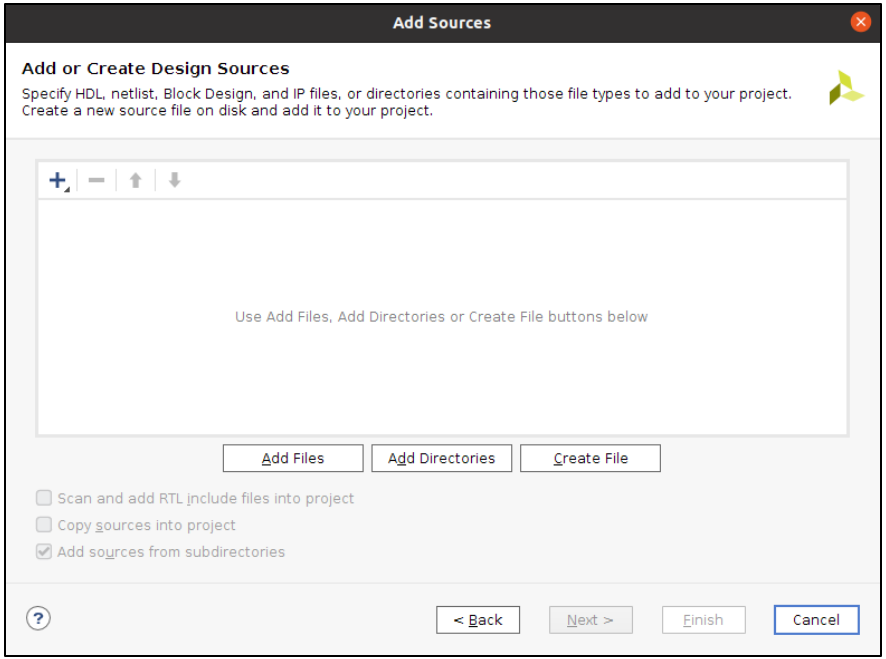


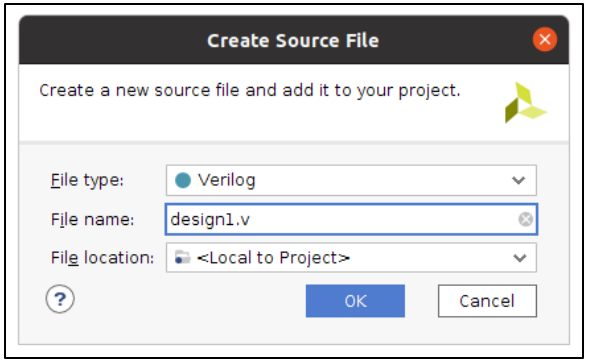



# Create a Design Source

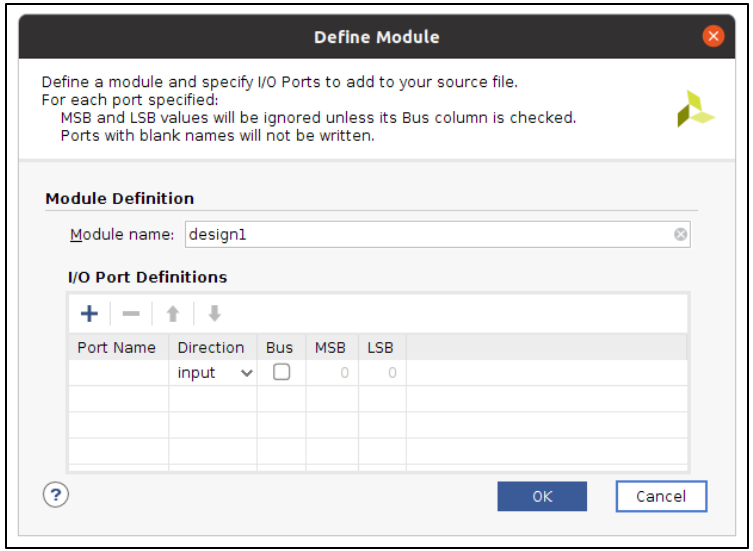
- Click on *Create File* button, name your source file (e.g., design1.v) and click on *Next*.
  - If you want to add an existing file to your project, click on *Add Files* and select the files that you want to add to the project.

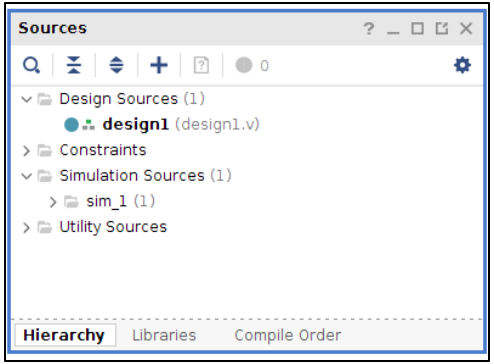


# Create a Design Source

- Set your module name (it is set as source file name per default) and click on OK. Then, you will see your source file under Design Sources tab on Sources window.

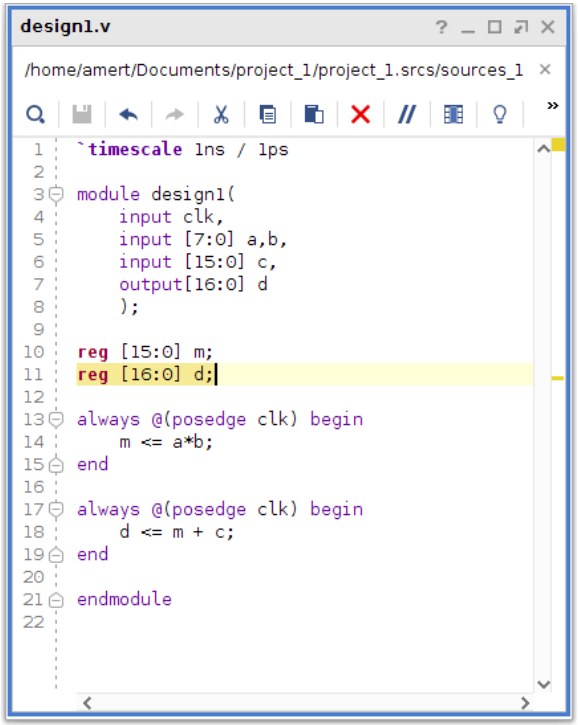


# Create a Design Source

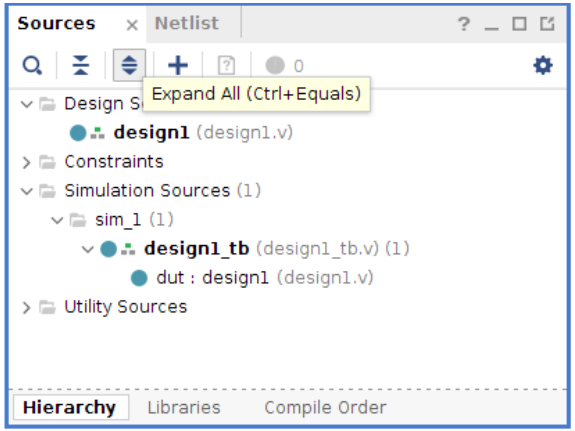
- Open design1.v and write the following (or any other) Verilog code as an example and save it.

```
`timescale 1ns / 1ps

module design1(
    input clk,
    input [7:0] a,b,
    input [15:0] c,
    output[16:0] d
    );

reg [15:0] m;
reg [16:0] d;

always @(posedge clk) begin
    m <= a*b;
end

always @(posedge clk) begin
    d <= m + c;
end

endmodule
```



# Create a Simulation Source

- In order to test your design, you should add a simulation source. Click on *Add Sources* in Flow Navigator Window, select *Add or create simulation resources* and then click on Next.
- Click on *Create File* button, name your simulation file (e.g., design1_tb.v) and click on *Next*.
- Set your module name (it is set as simulation file name per default) and click on OK. Then, you will see your simulation file under Simulation Sources tab on Sources window.



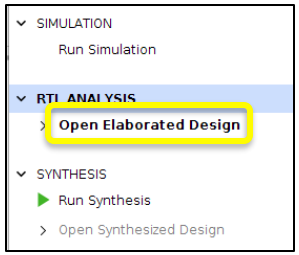
# Create a Simulation Source

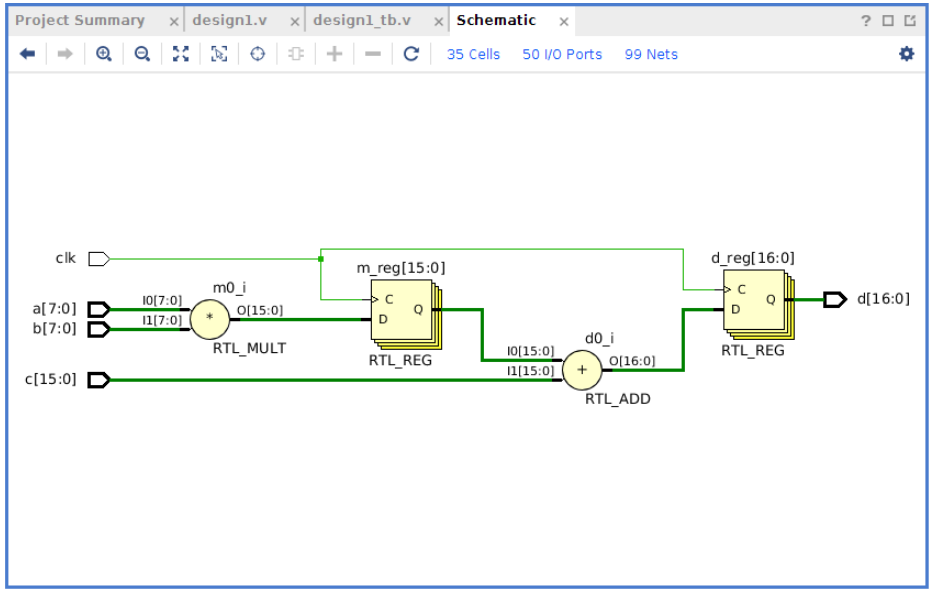
- Open design1_tb.v and write the following example Verilog testbench code and save it.

```
` timescale 1ns / 1ps

module design1_tb();

reg [7:0] a,b;
reg [15:0] c;
wire [16:0] d;

design1 dut (a,b,c,d);

initial begin
      a=0; b=0; c=0;
      #10;
      a=10; b=40; c=25;
      #10;
      a=53; b=19; c=100;
end

endmodule
```

# Design Elaboration

- Design Elaboration and RTL Analysis
  - It compiles RTL code and loads RTL netlist
  - You can check RTL structure, syntax, and logic definitions
  - You can view the schematic of your design

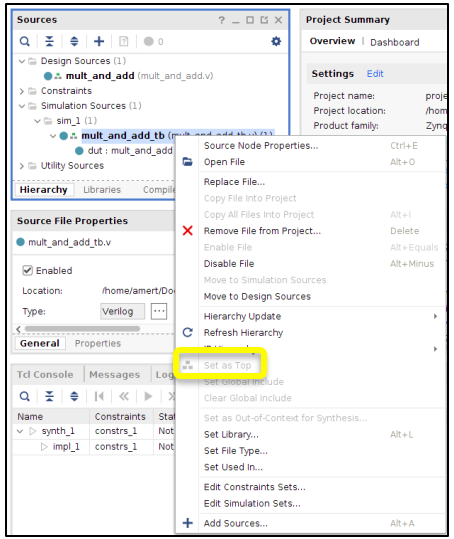


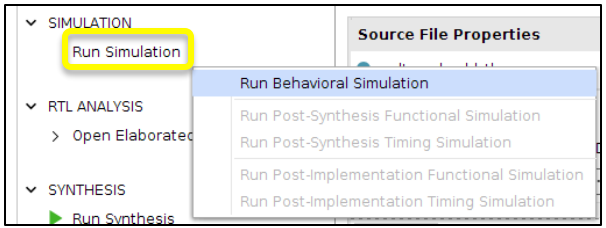



# Running Behavioral Simulation

# Behavioral Simulation

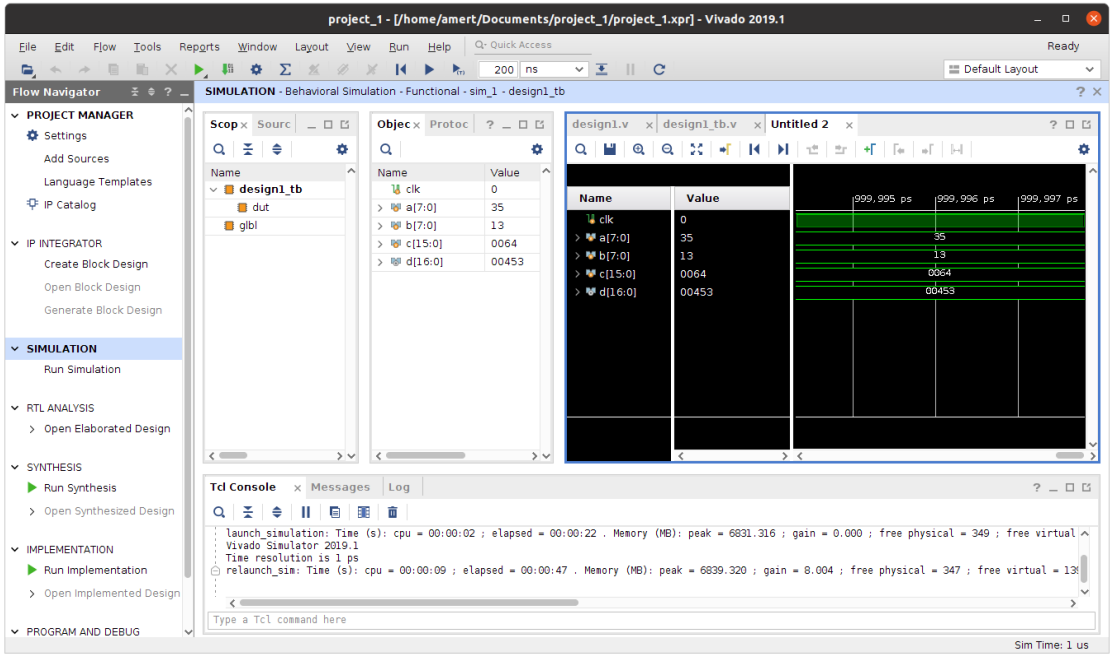
- Before running simulation, first make sure that your simulation source (testbench) is selected as top simulation module (its module name should be in boldcase letters) under Simulation Sources tab on Sources window.
- If it is not, right click on its name and click on *Set as Top*.
- Then, click on *Run Simulation* in Flow Navigator Window and click *Run Behavioral Simulation*.



# Behavioral Simulation

- If there is no error in your design and testbench files, you will see the following simulation screen.




Save waveform configuration

Zoom in to area where cursor is

Zoom out from area where cursor is

Fit whole run into waveform

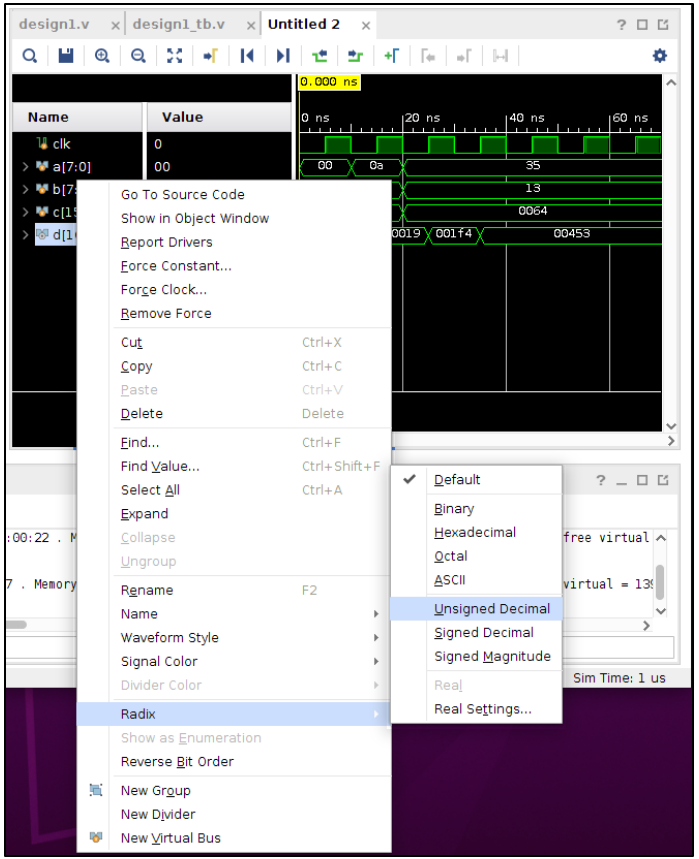
Go to time=0

Go to the last time

# Behavioral Simulation

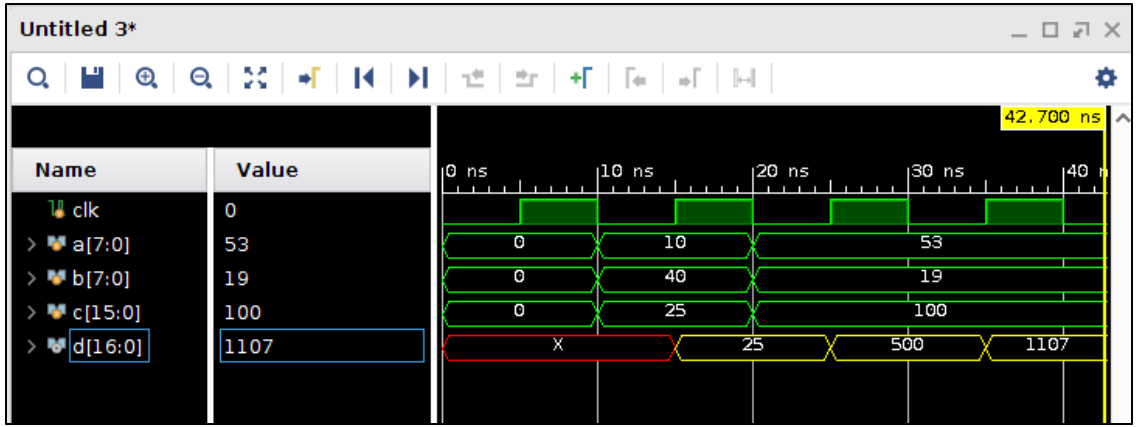
- You can change radix of the signals in the waveform window. For example, right click on signal name and then select *Radix -> Unsigned decimal* to change representation of this signal from hexadecimal to unsigned decimal.

- Similarly, you can change the color of the signal. For example, right click on signal name and then select *Signal Color -> Yellow* to change the color of this signal from lime to yellow.



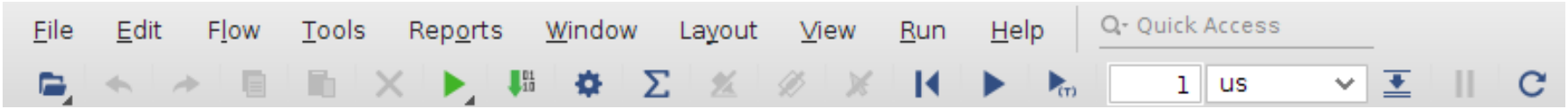
# Behavioral Simulation

- Change radix of the signals to the unsigned decimal. Then, zoom in to the beginning of the simulation. Now you can observe output *d*.



# Behavioral Simulation

- In the toolbar on the top, you will see some shortcut buttons for simulation.




Restart the simulation. Use this to restart the simulation with current state of the design.

Run All. Use this to run simulation until it reaches a stop/finish command in testbench .

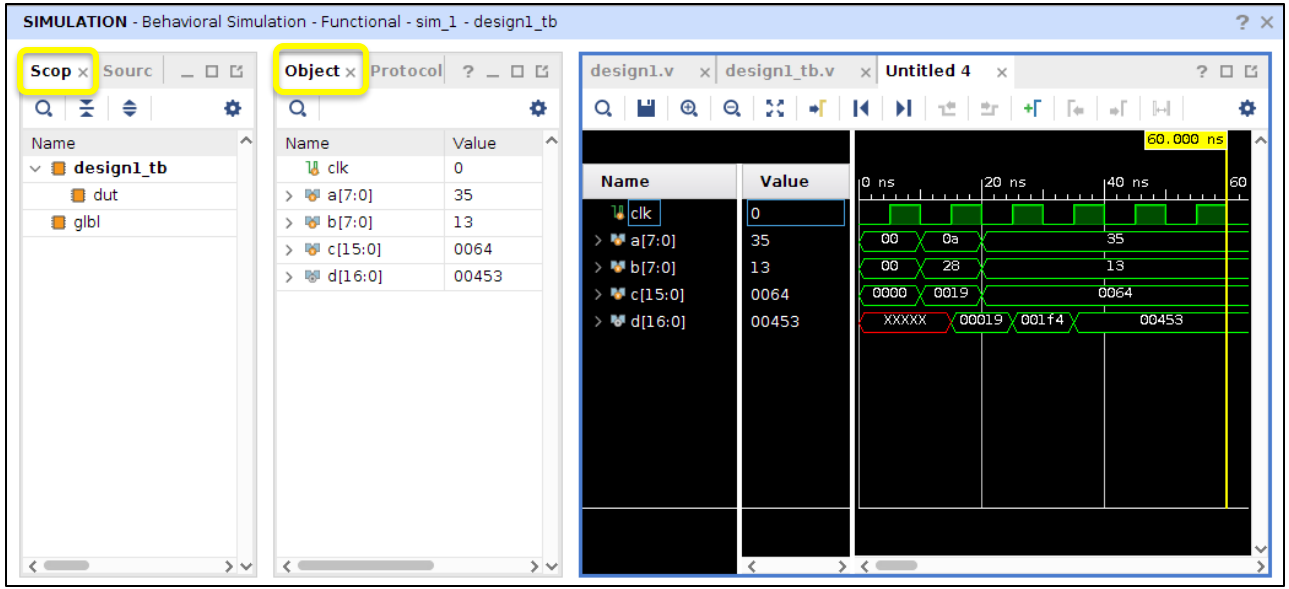
Run for specified time. Use this to run simulation for specified time.


Time and unit. Use this to specify run time and unit for the command above.

Relaunch the simulation. Use this to relaunch the simulation for elaborating the changes you made in your design.

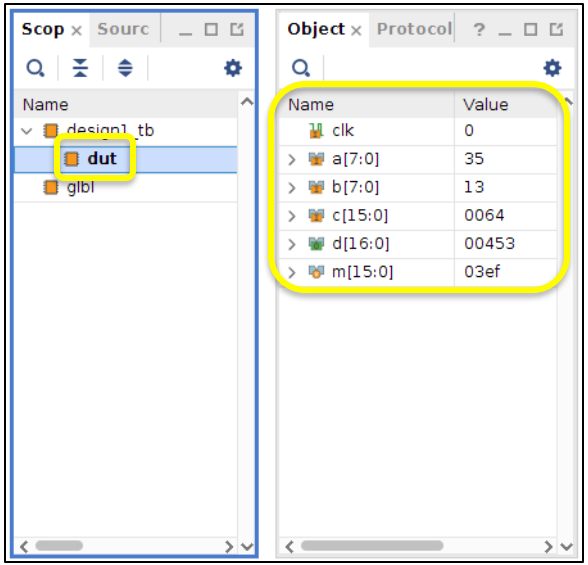
# Behavioral Simulation

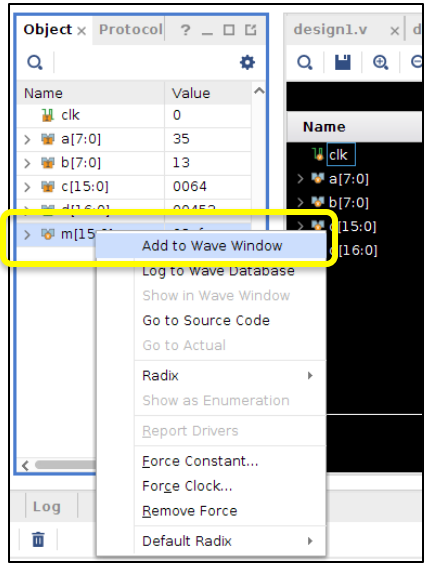
- In the Scope window, you can see the design hierarchy. When you select a scope in the Scope window, all HDL objects visible from that scope appear in the Objects window.



# Behavioral Simulation

- When you start simulation, you will see signals defined in testbench on the waveform. In order to observe internal signals (for debugging), click on the module you want to investigate on Scope window. Then right click on the signal you want to observe in the *Objects window* and click on *Add to wave window.*



# Synthesis/Implementation

# Constraint File

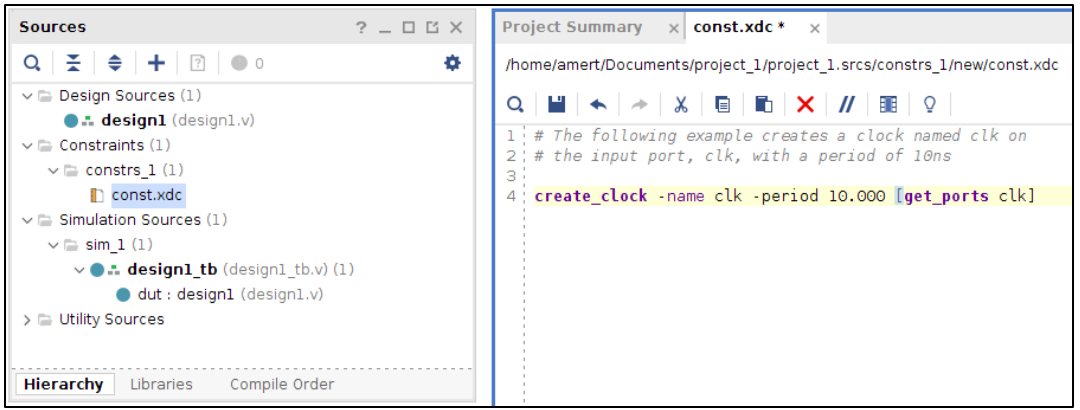
- Constraint file
  - Timing: *For setting clock frequency of your design*
  - Placement: *For floorplanning*
  - I/O: *Assigning your design inputs/outputs to FPGA pins*
  - Other user-defined constraints (i.e., *false paths*)

- For adding the constraint file, click on *Add Sources* in Flow Navigator Window, select *Add or create constraints* and then click on Next.
- Click on *Create File* button, name your simulation file (e.g., const.xdc) and click on *Next*.
- Then, you will see your source file under Design Sources tab on Sources window.

# Constraint File

- We will only use timing constraint for clock frequency/period [1]

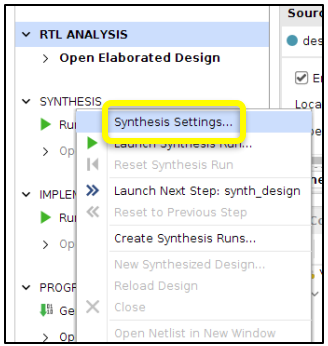
```
create_clock -name clk -period PERIOD [get_ports PORT_NAME]
```

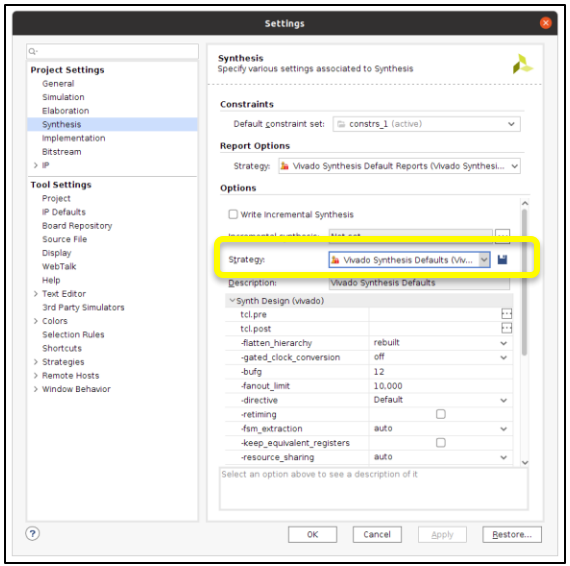


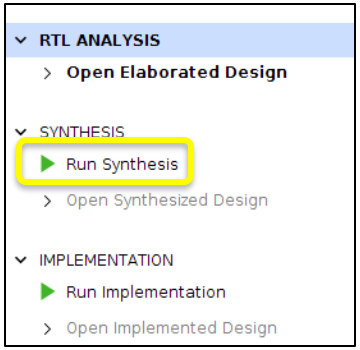

[1] https://docs.xilinx.com/r/2021.2-English/ug835-vivado-tcl-commands/create_clock

# Synthesis

- Synthesis translates RTL code to a netlist which defines the circuit[1]
  - You can change synthesis strategy (i.e., area optimized or performance optimized) from synthesis settings (right click on *SYNTHESIS -> Synthesis Settings*)
  - For starting synthesis, click on *Run Synthesis*.



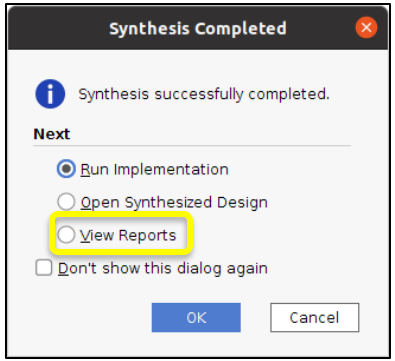



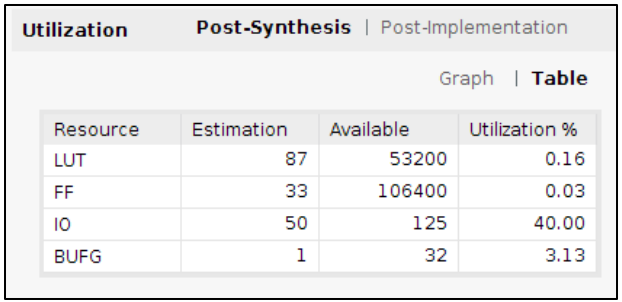




[1] https://docs.xilinx.com/v/u/2021.1-English/ug901-vivado-synthesis

# Synthesis

- After synthesis is finished, you can directly start implementation, open the synthesized design or view synthesis reports.
  - You can see synthesis report summary in *Project Summary*.

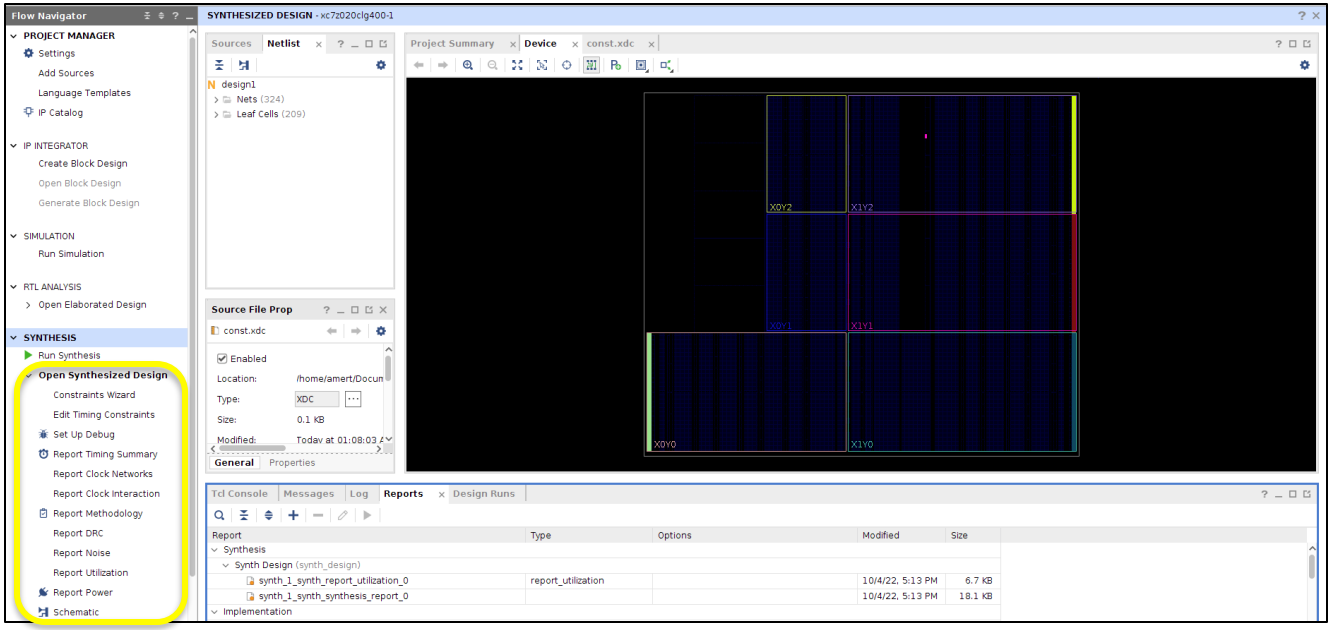



| Resource | Estimation | Available | Utilization % |
|---|---|---|---|
| LUT | 87 | 53200 | 0.16 |
| FF | 33 | 106400 | 0.03 |
| IO | 50 | 125 | 40.00 |
| BUFG | 1 | 32 | 3.13 |

# Synthesis

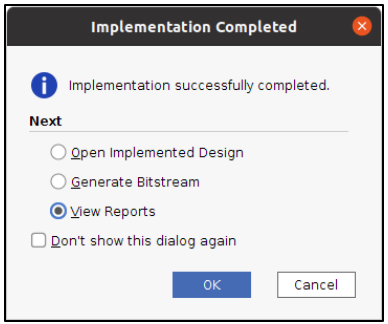
- After synthesis is finished, you can directly start implementation, open the synthesized design or view synthesis reports.
  - You can see synthesis report summary in *Project Summary*.
  - For detailed report/results, open the synthesized design.

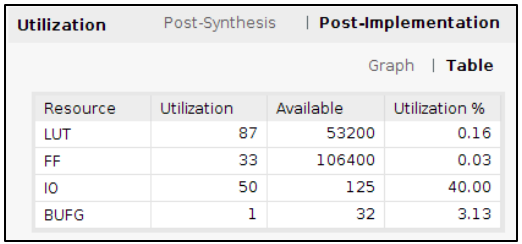


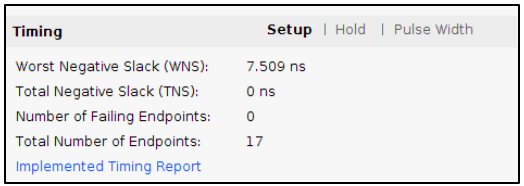
# Implementation

- Implementation takes the netlist with user constraint and maps it to actual FPGA components[1].
  - Similar to Synthesis, you can change implementation strategy
  - For starting implementation, click on *Run Implementation*.

- After implementation is finished, you can start bitstream generation, open the implemented design or view implementation report.
  - You can see implementation report (area and timing) in *Project Summary*.



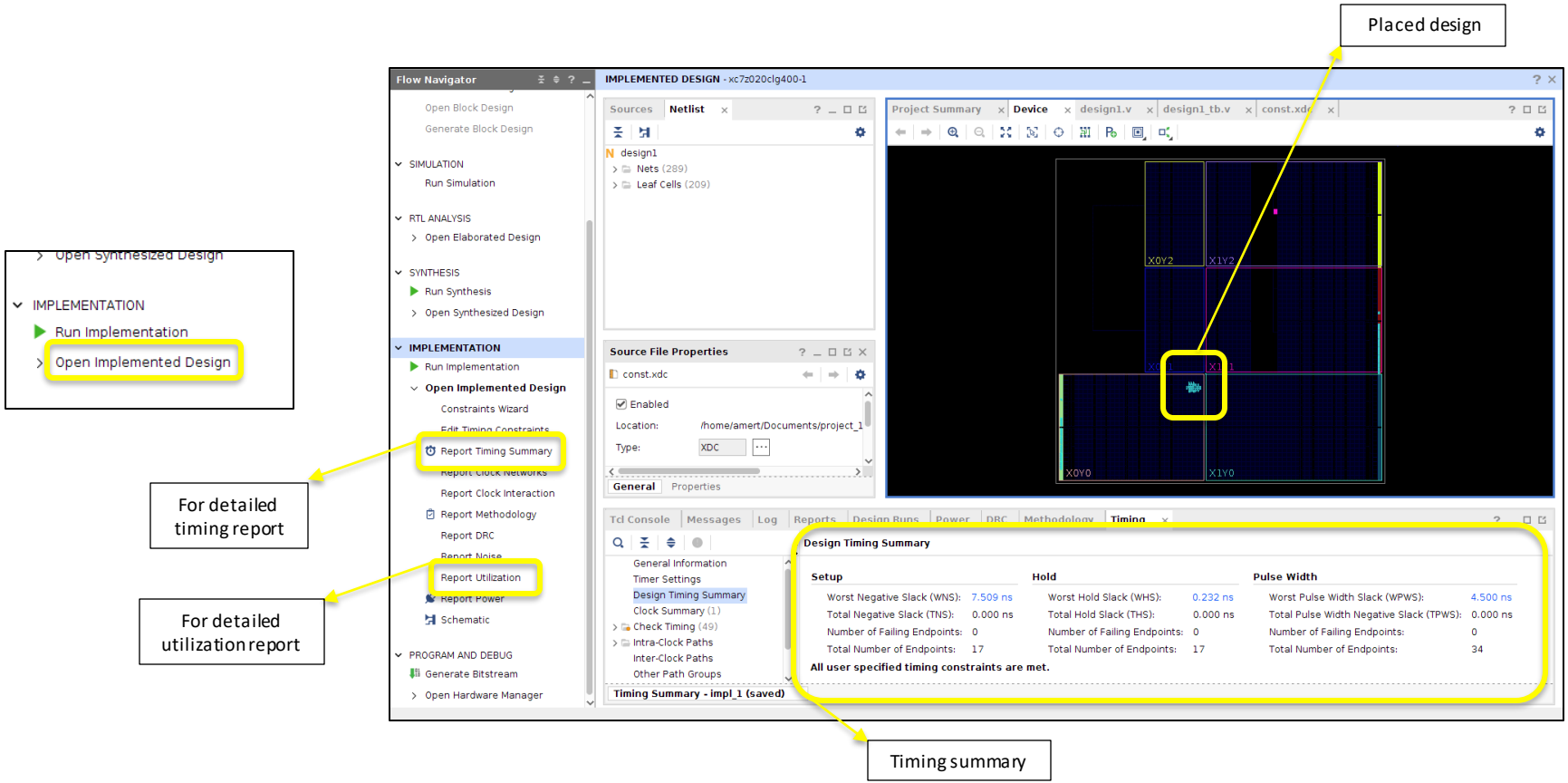




[1] https://docs.xilinx.com/r/2021.1-English/ug904-vivado-implementation/Revision-History

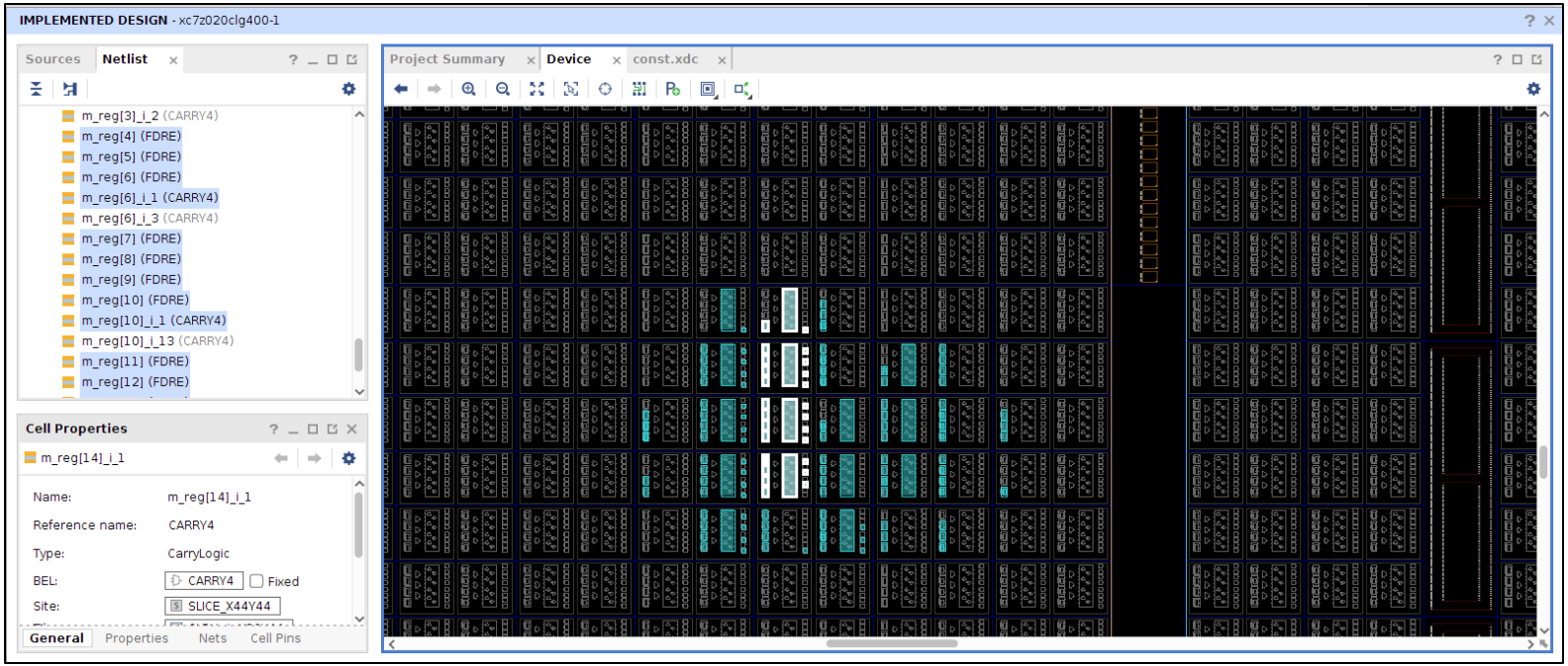
# Implementation

- For detailed report/results and the placed & routed design, open the implemented design.



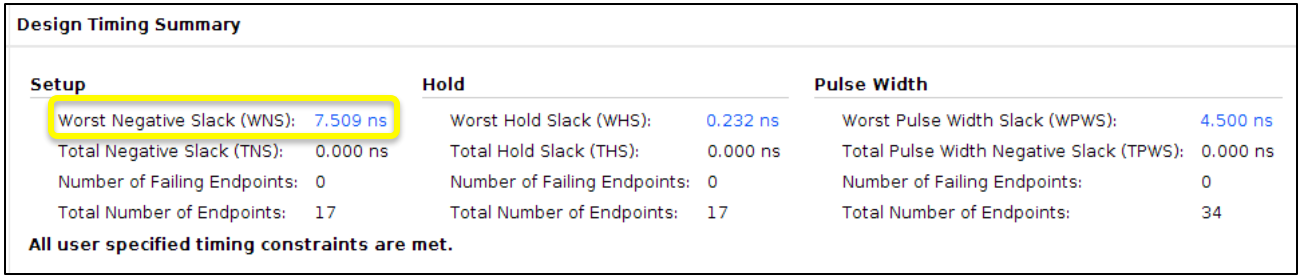
# Implementation

- For detailed report/results and the placed & routed design, open the implemented design.
  - Placed design



# Implementation

- For detailed report/results and the placed & routed design, open the implemented design.
  - Detailed timing result (showing critical paths in your design).
  - Worst Negative Slack (WNS): If it is negative, then it means your design could not meet the timing requirement (given in your constraint file) and your design has failing paths.
    - Click on WNS value to see critical paths.



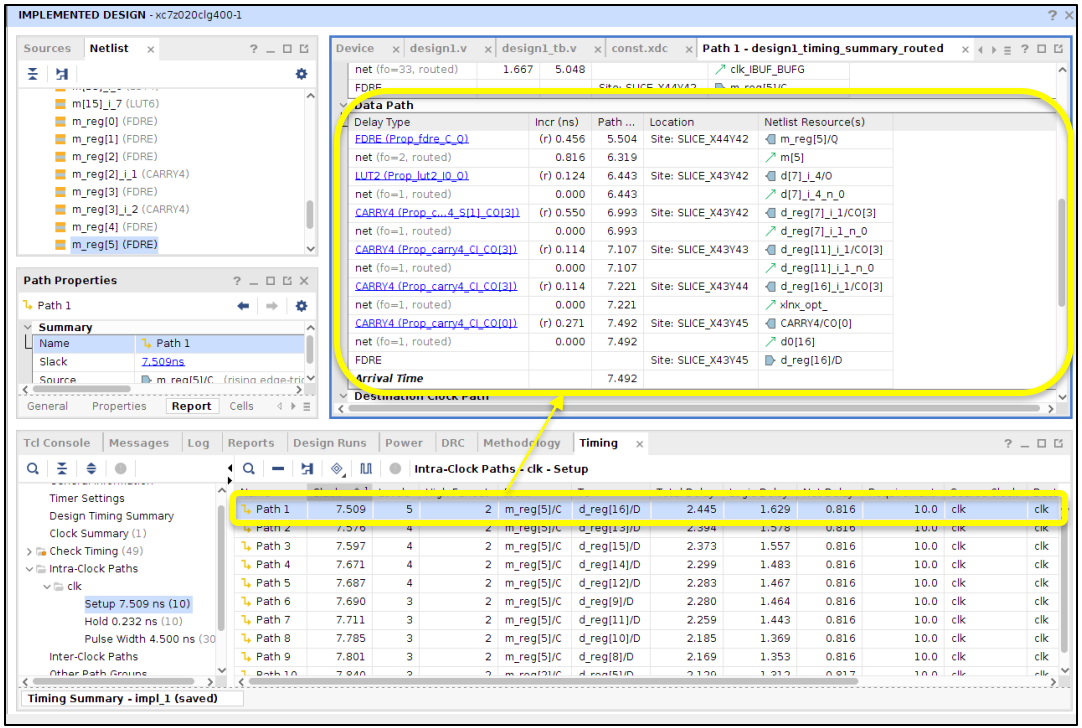
**Design Timing Summary**

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 7.509 ns | Worst Hold Slack (WHS): | 0.232 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: | 17 | Total Number of Endpoints: | 17 | Total Number of Endpoints: | 34 |

**All user specified timing constraints are met.**

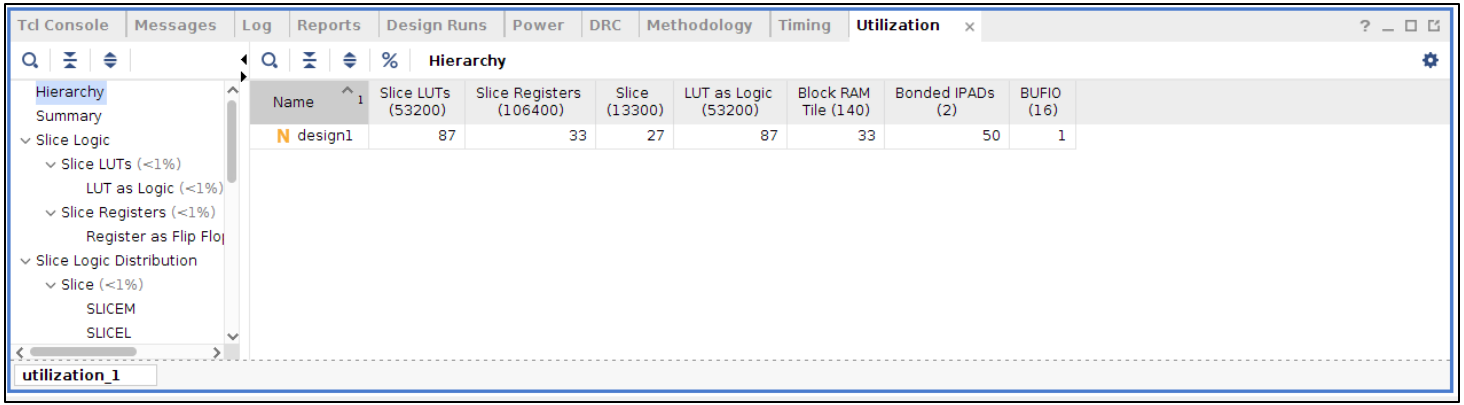
# Implementation

- For detailed report/results and the placed & routed design, open the implemented design.
  - Detailed timing result (showing critical paths in your design).



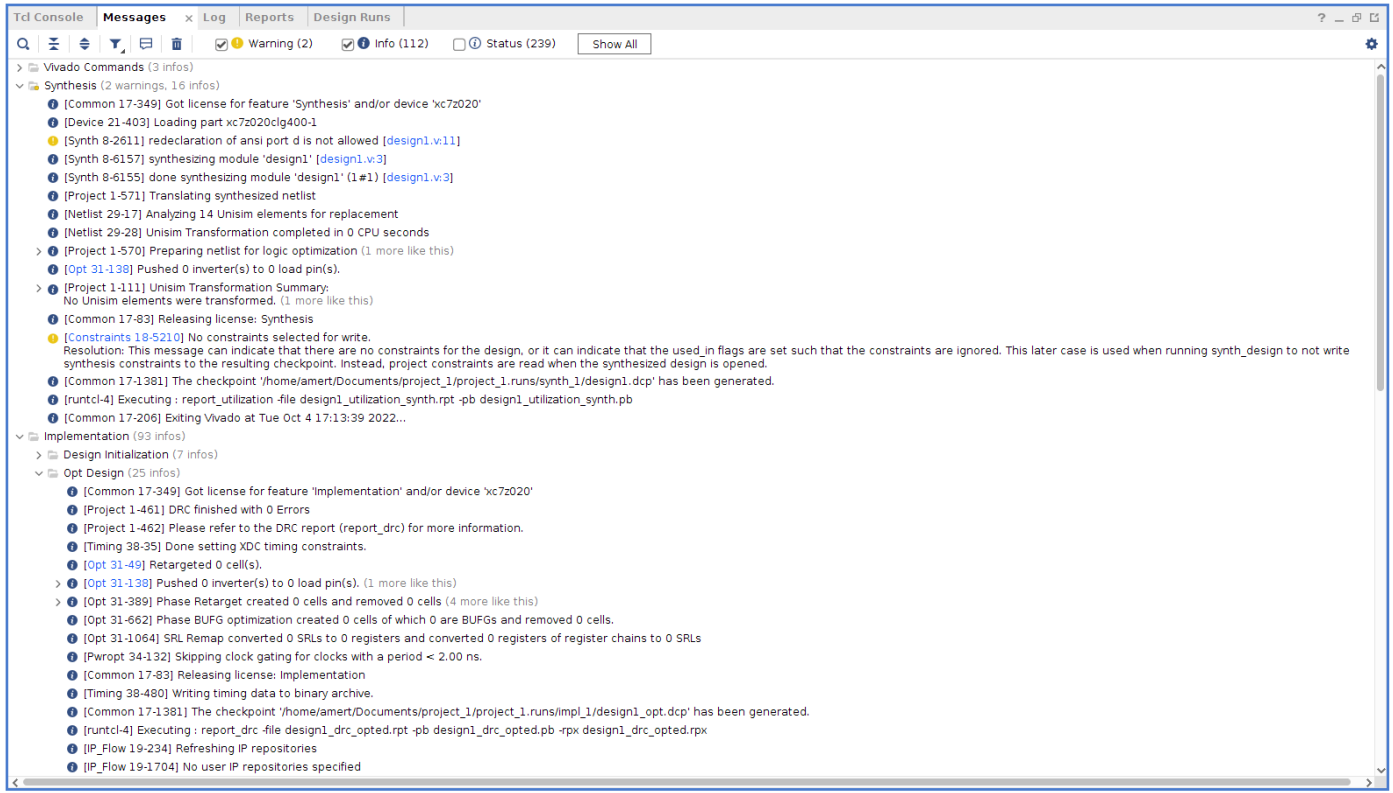
# Implementation

- For detailed report/results and the placed & routed design, open the implemented design.
  - Detailed area (utilization) result.



# Synthesis & Implementation Messages

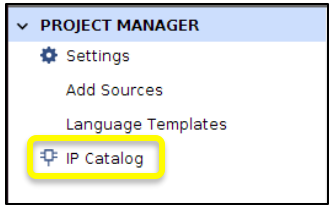
- Messages window displays a filtered list of the Vivado log. This list includes useful information such as the main messages, warnings, errors.

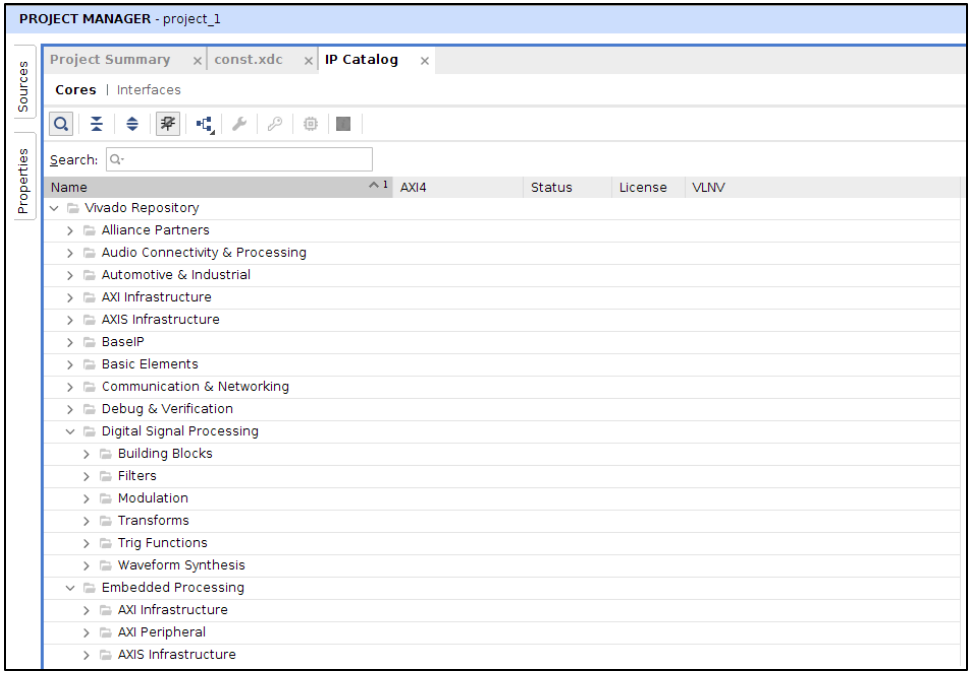


# Adding Xilinx IPs to your Design

# IP Catalog

- Click on IP Catalog under Flow Navigator to see the list of available Xilinx IPs.
  - You can use Search bar to find IPs.

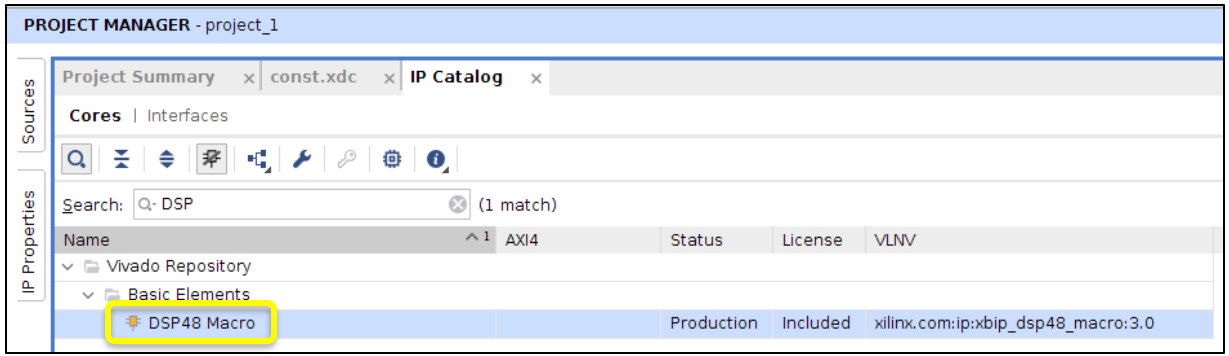


# Adding Xilinx IPs to your Design (Example: DSP48 Macro)

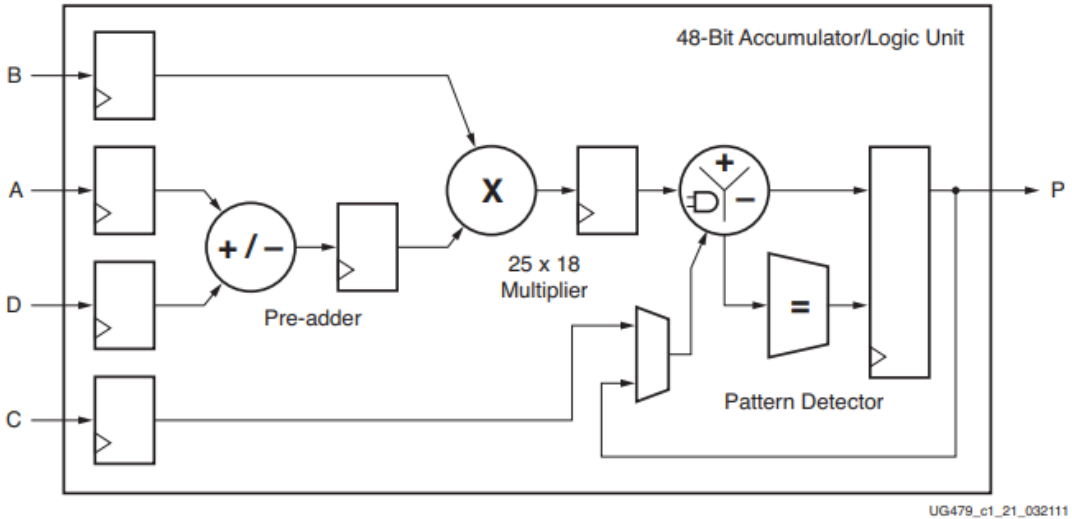
# IP Catalog Example: DSP48 Macro

- Xilinx FPGAs have dedicated, full-custom, low-power DSP slices.
  - 7 Series DSP48E1 Slice User Guide
    https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1
  - Search for *DSP48 Macro* and double click on it.



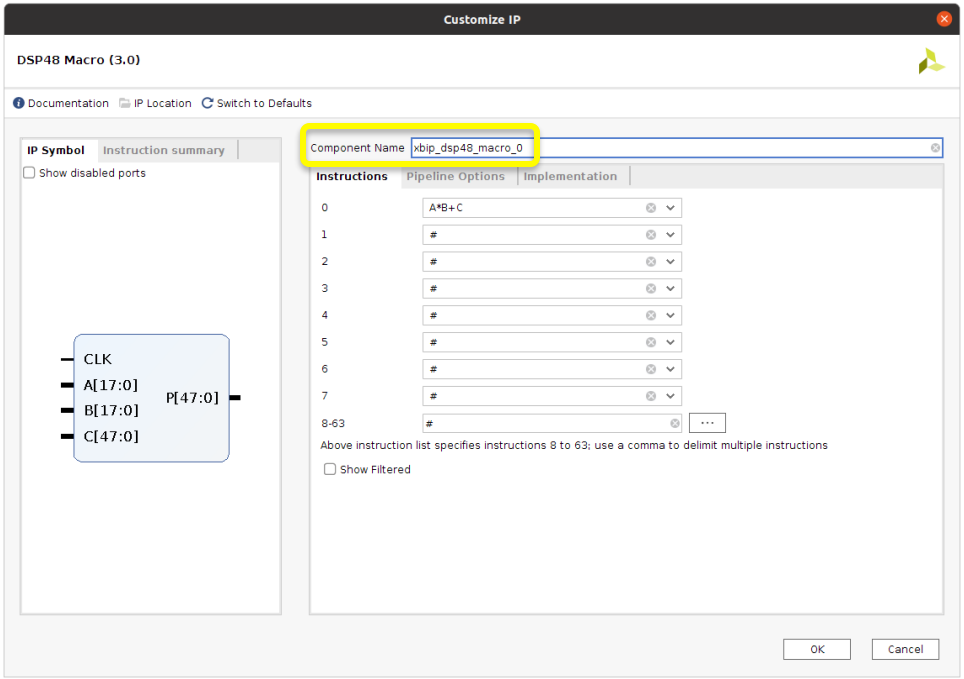
# IP Catalog Example: DSP48 Macro

- Xilinx FPGAs have dedicated, full-custom, low-power DSP slices.
  - 7 Series DSP48E1 Slice User Guide
    https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1
  - Search for DSP48 Macro and double click on it.



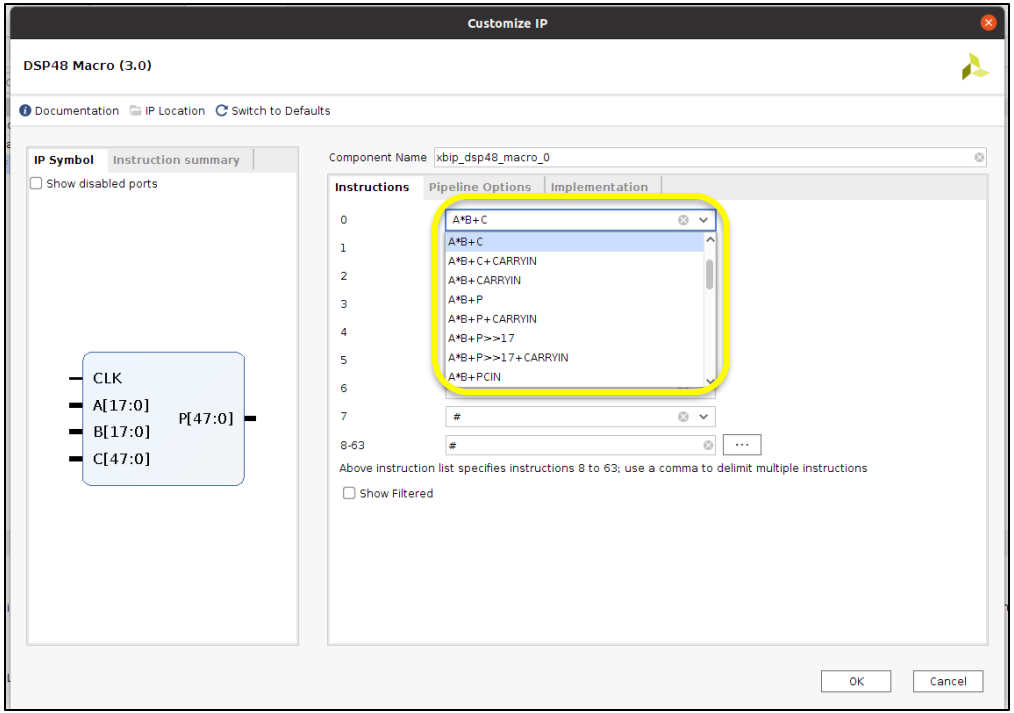
# IP Catalog Example: DSP48 Macro

- Xilinx FPGAs have dedicated, full-custom, low-power DSP slices.
  - First, set the component name.



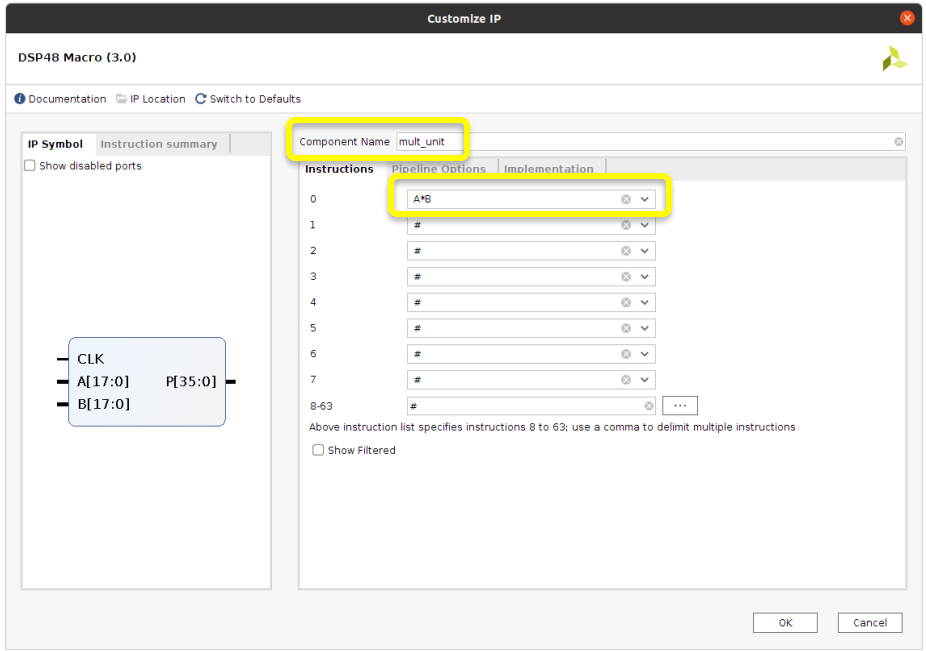
# IP Catalog Example: DSP48 Macro

- DSP unit provides various functionalities.



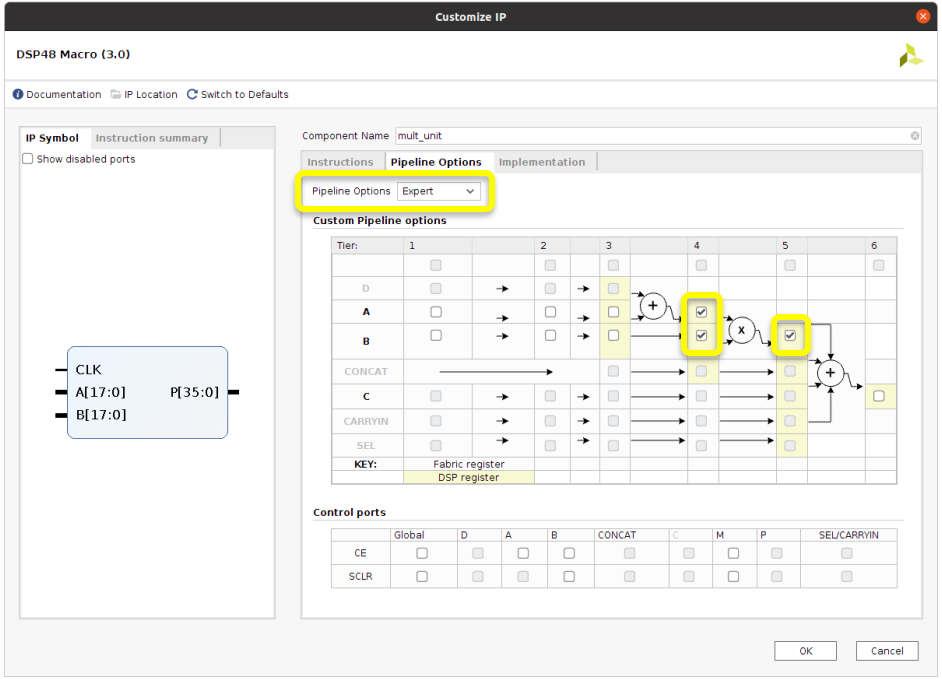
# IP Catalog Example: DSP48 Macro

- As an example, we select A*B as functionality and set the component name as *mult_unit*.
  - You will use the component name for instantiating the DSP unit in your design



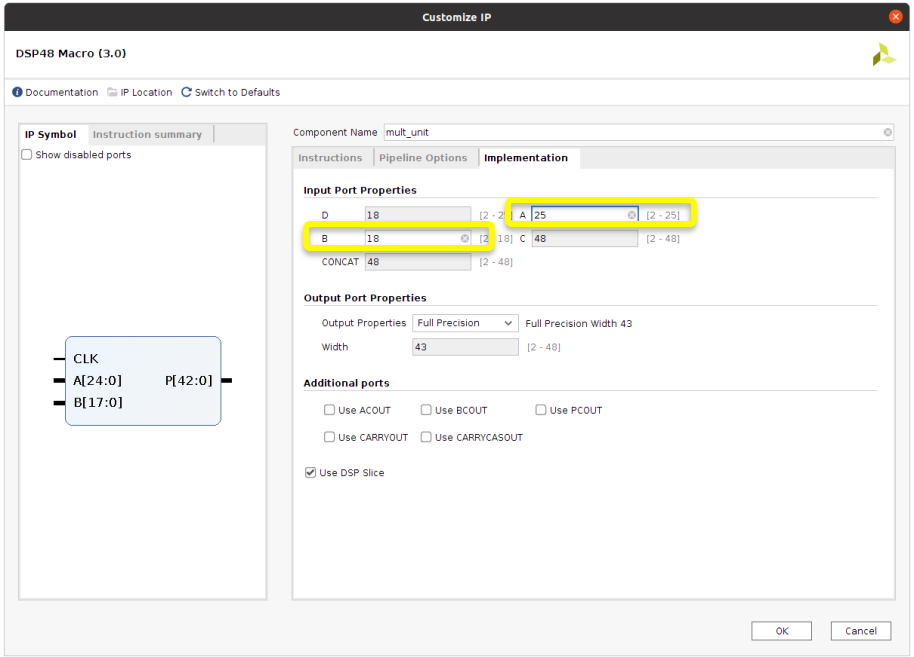
# IP Catalog Example: DSP48 Macro

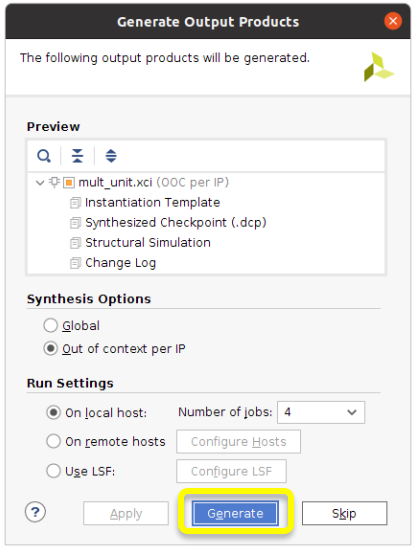
- You can set the pipeline options.
  - As an example, we use two level of pipeline registers



# IP Catalog Example: DSP48 Macro

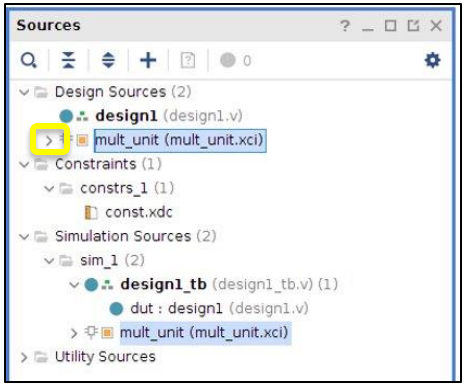
- You can set input/output port widths.
  - We set A and B width as 25 and 18 (an signed 25-bit x 18-bit multiplier)
  - Finally, click on *OK* and then *Generate* to create the DSP.

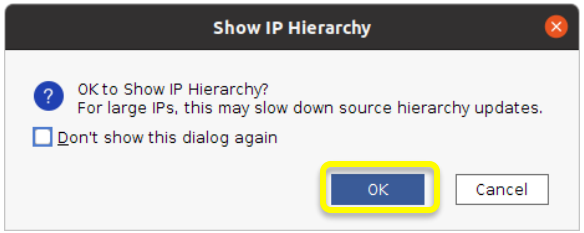


# IP Catalog Example: DSP48 Macro

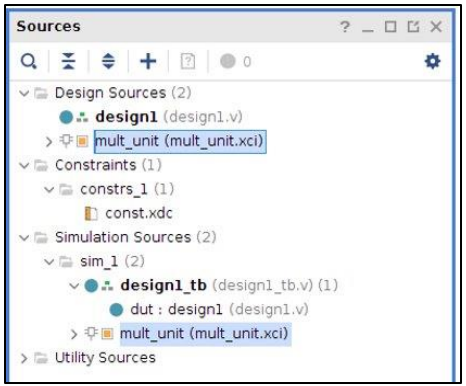
- You will see the DSP IP under Design Sources
  - In order to see the IP ports, expand the IP hierarchy and click on OK.

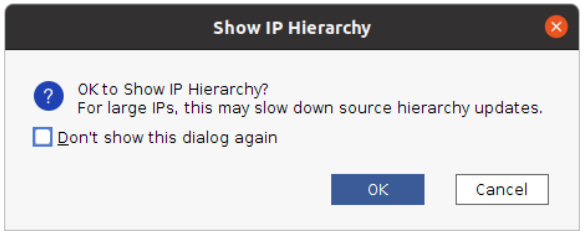


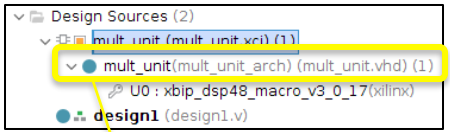



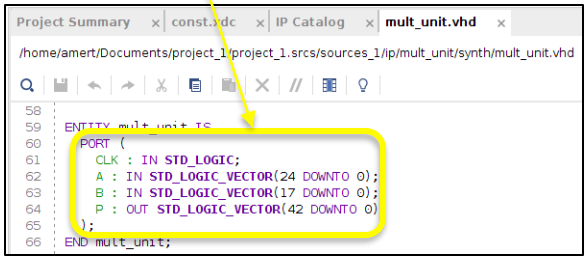
# IP Catalog Example: DSP48 Macro

- You will see the DSP IP under Design Sources
  - In order to see the IP ports, expand the IP hierarchy and click on OK.
  - Double click on *mult_unit.vhd* to see the module ports.



# IP Catalog Example: DSP48 Macro

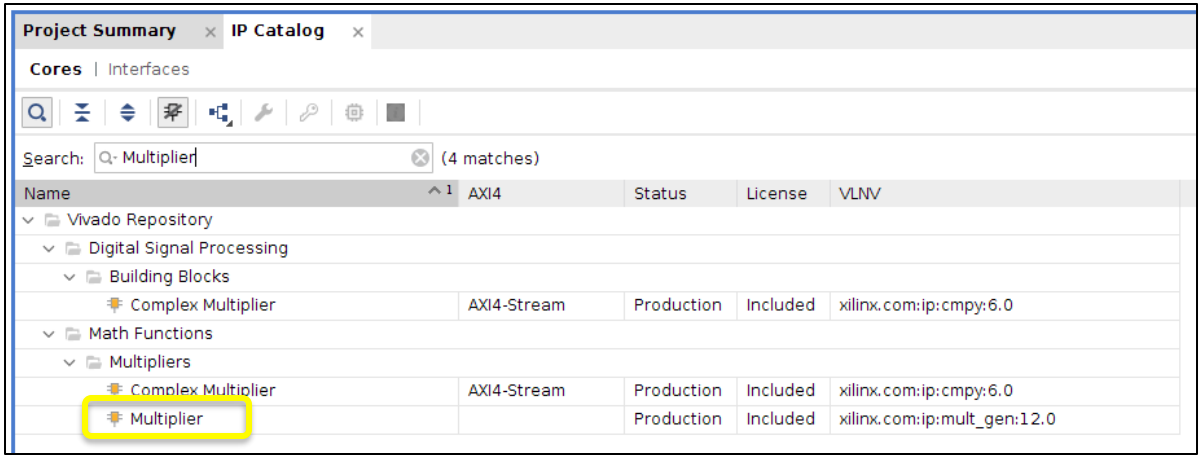
- You will see the DSP IP under Design Sources
  - In order to see the IP ports, expand the IP hierarchy and click on OK.
  - Double click on *mult_unit.vhd* to see the module ports.
    - You can instantiate the module (mult_unit) with module ports.

```
…
wire [24:0] mult_unit_A;
wire [17:0] mult_unit_B;
wire [42:0] mult_unit_P;
…
mult_unit MU(clk,
             mult_unit_A,
             mult_unit_B,
             mult_unit_P);
…
```

# Adding Xilinx IPs to your Design (Example: Multiplier)

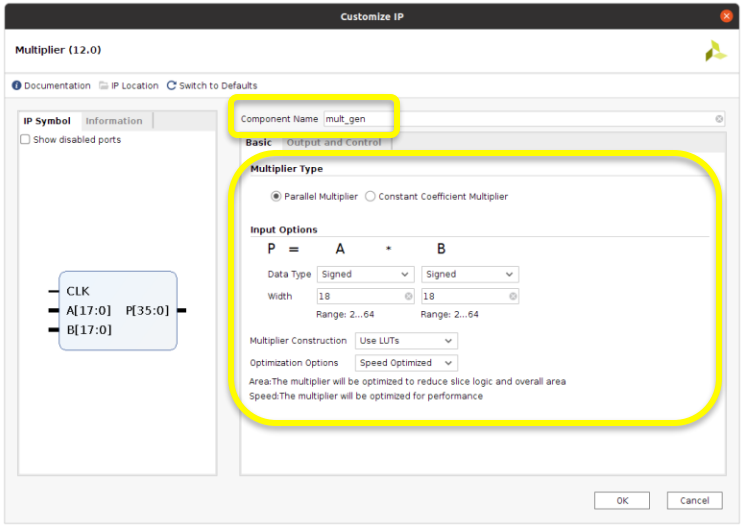
# IP Catalog Example: Multiplier

- Xilinx FPGAs provide a multiplier core. This core allows parallel and constant-coefficient multipliers to be generated. The user can specify if DSP48 Slices, LUTs or a combination of resources should be utilized.
  - 7 Series DSP48E1 Slice User Guide
    https://docs.xilinx.com/v/u/en-US/pg108-mult-gen
  - Search for *Multiplier* and double click on it.



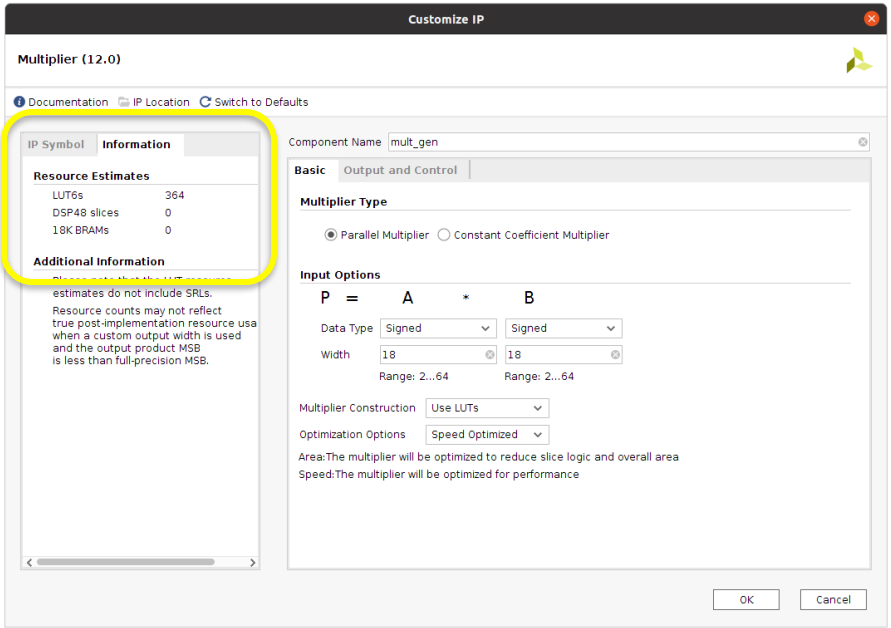
# IP Catalog Example: Multiplier

- Xilinx FPGAs provide a multiplier core.
  - Set the component name and select one of multiplier types
    - Parallel Multiplier or Constant Coefficient Multiplier
  - Set input options
    - Data type: Signed or Unsigned
    - Bit width: 2 to 64
    - Multiplier construction: Use LUT or Use Mults (DSPs)
    - Optimization options: Speed optimized or Area optimized



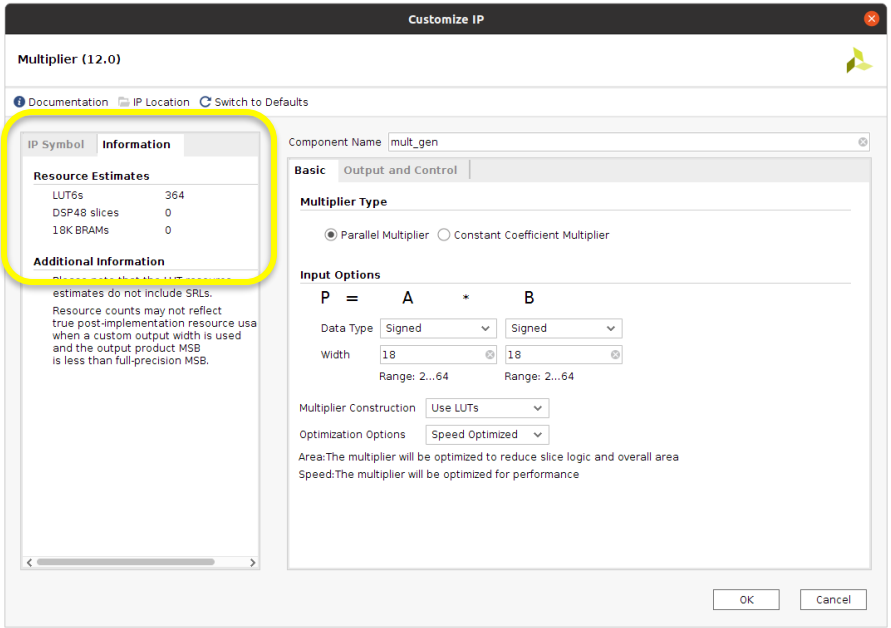
# IP Catalog Example: Multiplier

- Xilinx FPGAs provide a multiplier core.
  - Information tab shows resource estimate (LUT6s, DSP48 slices, 18K BRAMs) based on the Multiplier type and Input options.



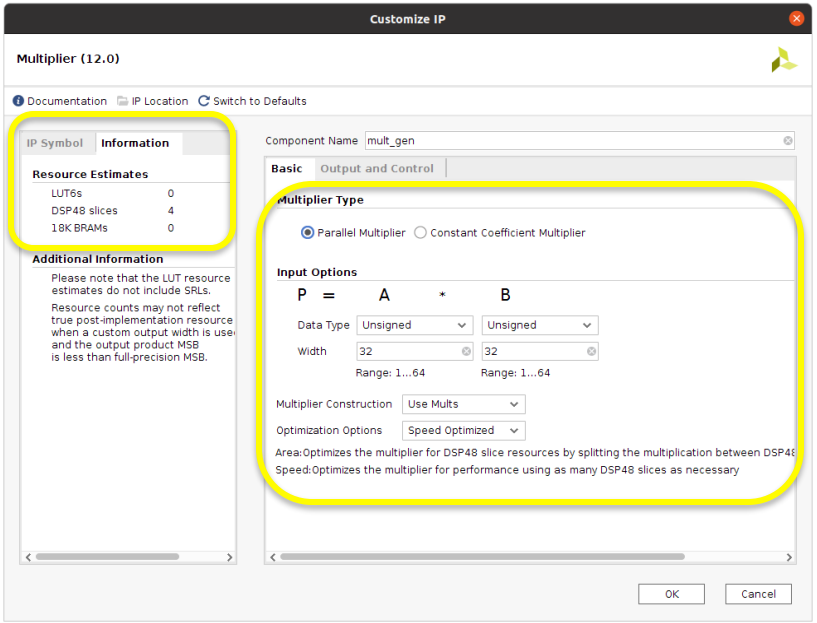
# IP Catalog Example: Multiplier

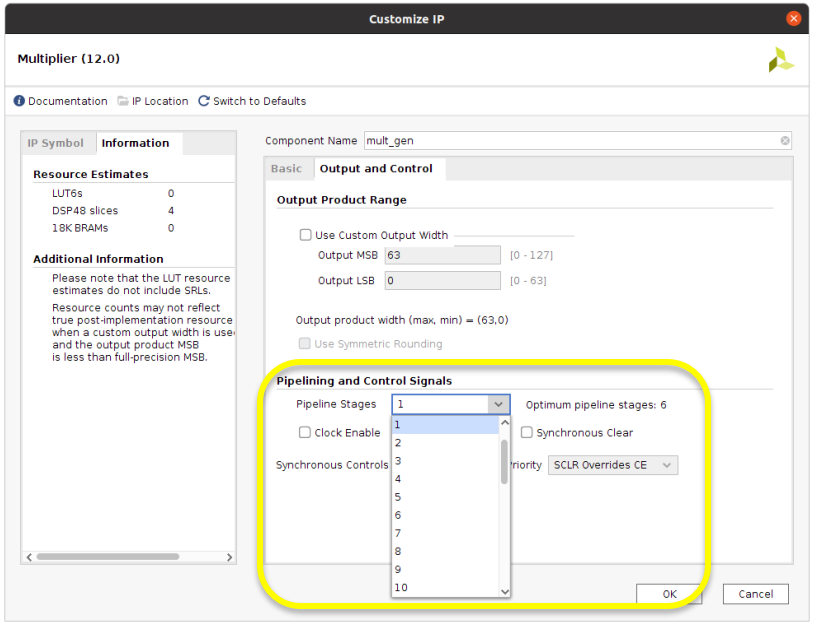
- Xilinx FPGAs provide a multiplier core.
  - Information tab shows resource estimate (LUT6s, DSP48 slices, 18K BRAMs) based on the Multiplier type and Input options.



# IP Catalog Example: Multiplier

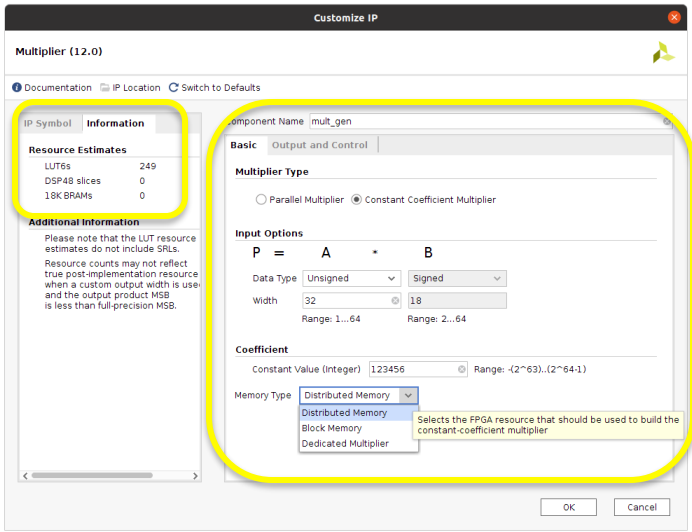
- Xilinx FPGAs provide a multiplier core.
  - *Parallel Multiplier* example
    - In *Output and Control* tab, you can adjust the number of pipeline stages (optimum pipeline stages are proposed by the core based on multiplier type and input options)
    - Finally click on OK and generate to create the IP block.

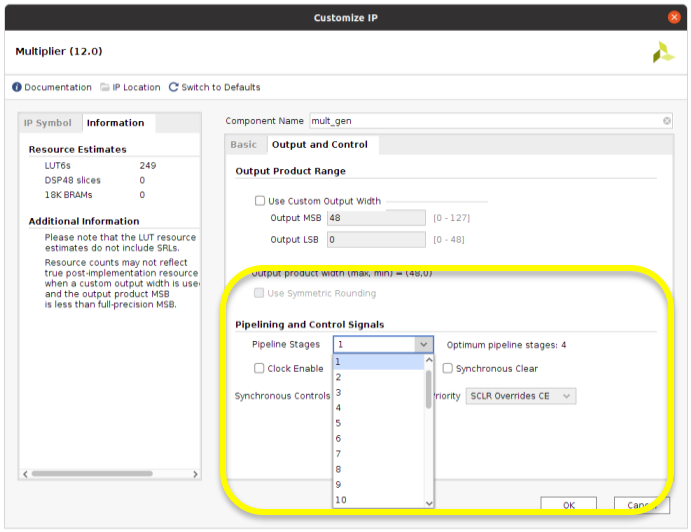


# IP Catalog Example: Multiplier

- Xilinx FPGAs provide a multiplier core.
  - *Constant Coefficient Multiplier* example
    - In *Basic* tab you can set the constant integer (second operand) and memory type to implement constant multiplier
    - In *Output and Control* tab, you can adjust the number of pipeline stages (optimum pipeline stages are proposed by the core based on multiplier type and input options)
    - Finally click on OK and generate to create the IP block.

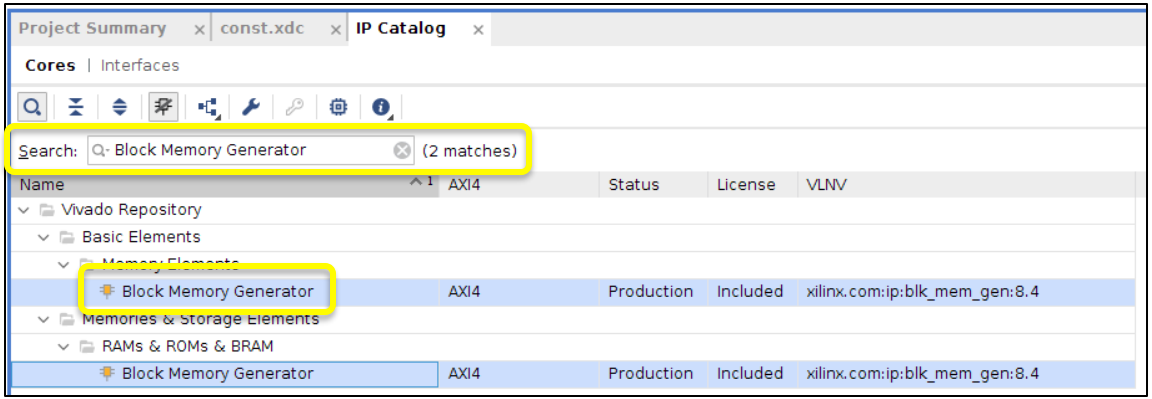


# Adding Xilinx IPs to your Design
# (Example: Block Memory Generator)

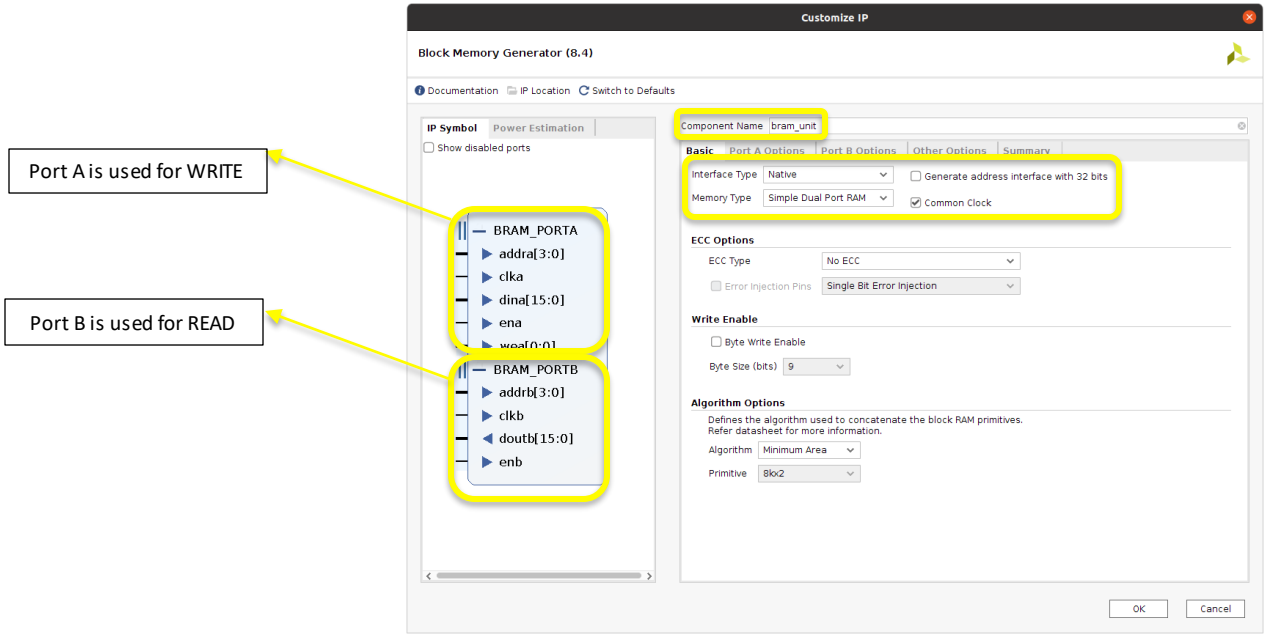
# IP Catalog Example: Block Memory Generator

- Xilinx FPGAs have dedicated block memory primitives
  - Block Memory Generator Product Guide
    - https://docs.xilinx.com/v/u/en-US/pg058-blk-mem-gen
  - Search for *Block Memory Generator* and double click on it.



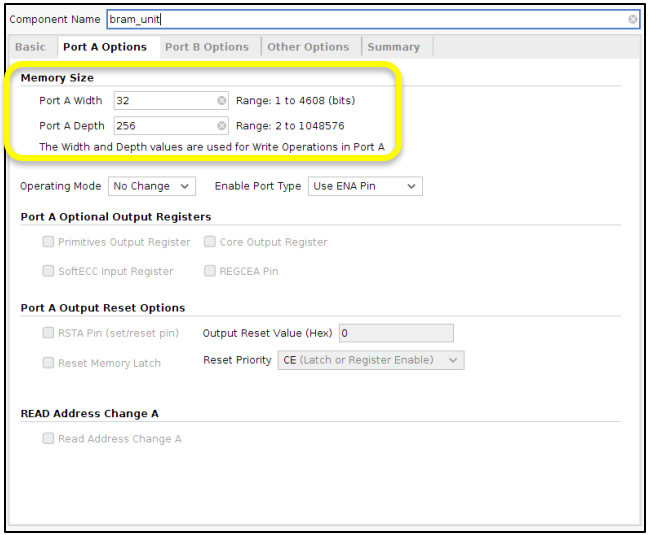
# IP Catalog Example: Block Memory Generator

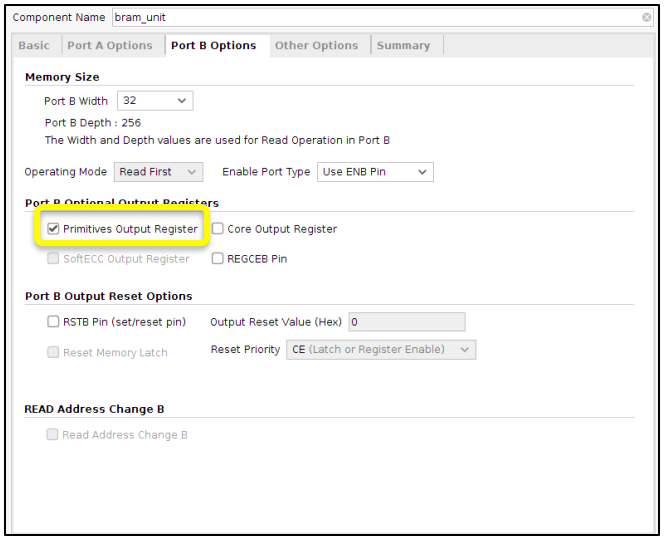
- Xilinx FPGAs have dedicated block memory primitives
  - Set the component name and select one of available memory types
  - As an example, we select *Simple Dual Port RAM* and name the component as *bram_unit*



# IP Catalog Example: Block Memory Generator

- You can set the data width and depth of Port A and Port B. We set width and depth as 32 and 256 respectively. This memory unit can store 256 of 32-bit words.
  - You can enable optional output register of Port B
    (If you enable it, read latency will be 2 cycles instead of 1)



# IP Catalog Example: Block Memory Generator

- You can see the summary of memory unit that you generated.
  - Bit-width of ports, read latency …
- Finally click on OK and generate to create the IP block.