

# Xilinx Vivado Basics (Part I)

October 11, 2022

Ahmet Can Mert

[ahmet.mert@iaik.tugraz.at](mailto:ahmet.mert@iaik.tugraz.at)



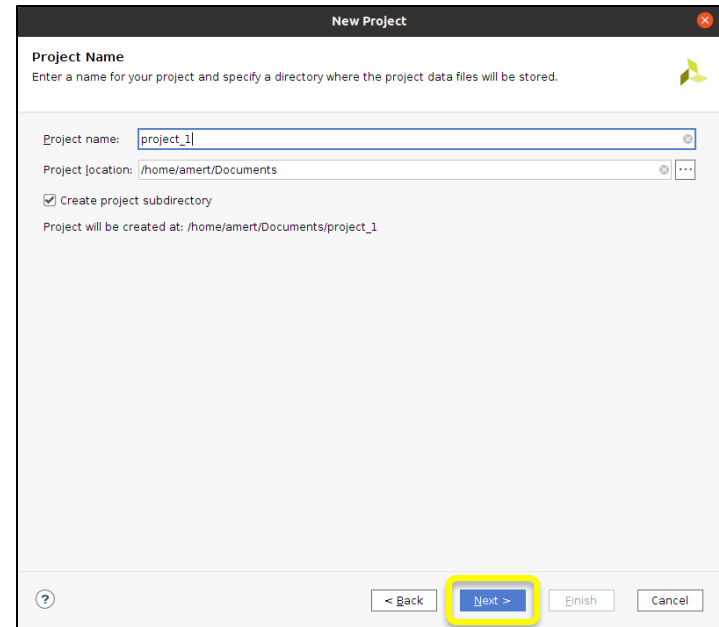
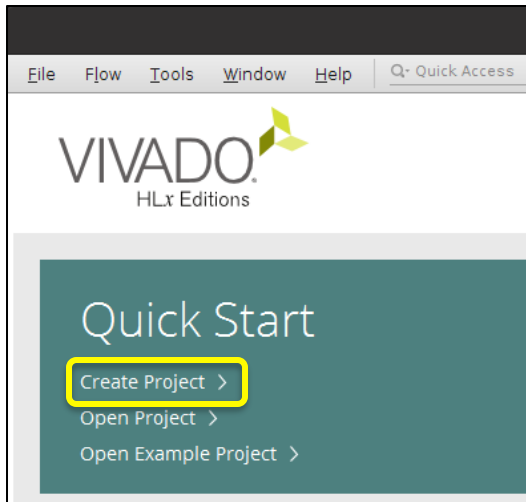
# Overview

- 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

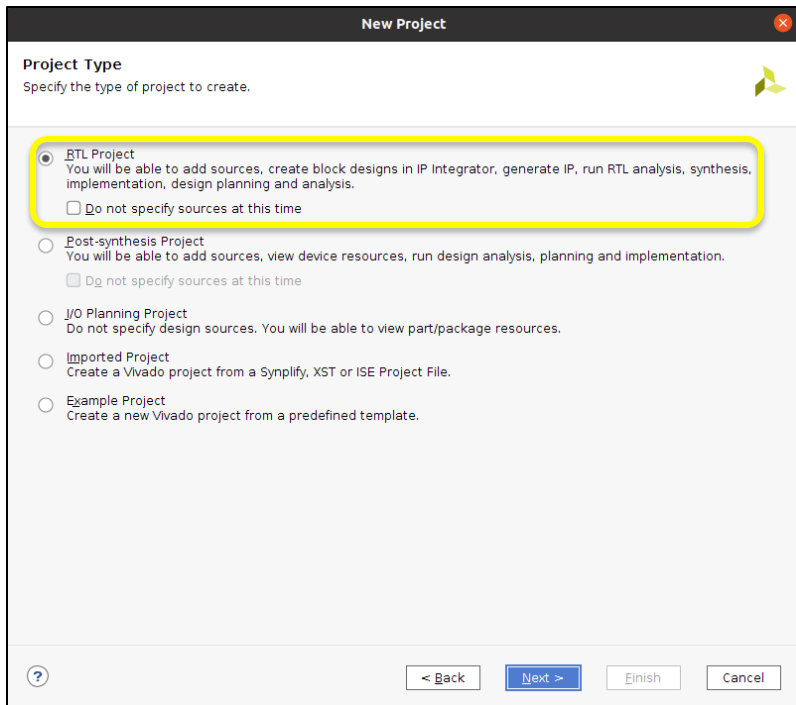
# Creating a New Project

- 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).



The screenshot shows the 'New Project' dialog box with the 'Project Type' section. The 'RTL Project' option is selected and highlighted with a yellow box. Below it, there are five other project types: 'Post-synthesis Project', 'I/O Planning Project', 'Imported Project', and 'Example Project'. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

**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

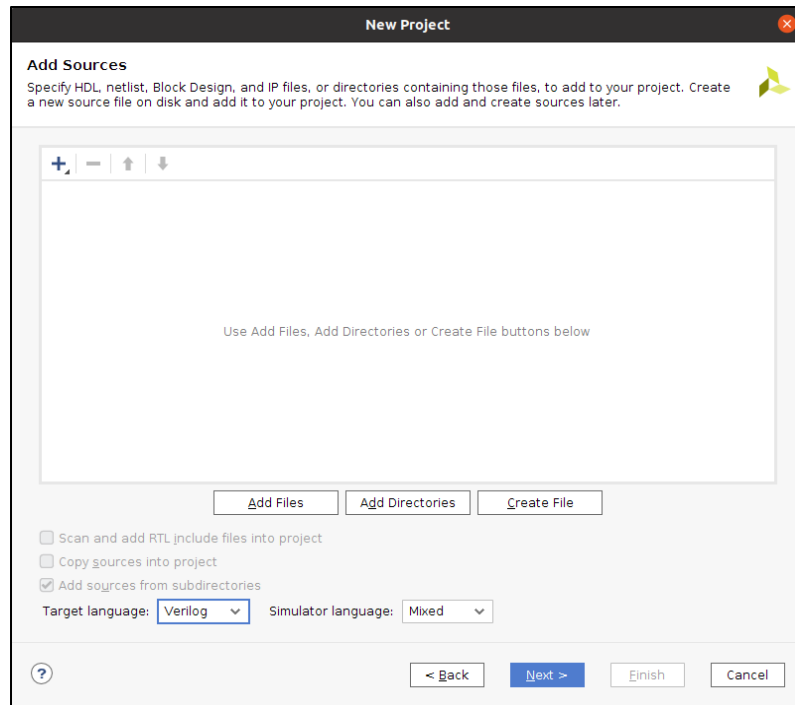
**Post-synthesis Project**  
You will be able to add sources, view device resources, run design analysis, planning and implementation.  
 Do not specify sources at this time

**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



The screenshot shows the 'New Project' dialog box with the 'Add Sources' section. It contains a large empty area with a toolbar at the top (plus, minus, up, down arrows) and a message: 'Use Add Files, Add Directories or Create File buttons below'. Below this are three buttons: 'Add Files', 'Add Directories', and 'Create File'. At the bottom, there are checkboxes for 'Scan and add RTL include files into project', 'Copy sources into project', and 'Add sources from subdirectories' (checked). There are also dropdown menus for 'Target language: Verilog' and 'Simulator language: Mixed'. At the bottom, there are buttons for '?', '< Back', 'Next >', 'Finish', and '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.

+ - ↑ ↓

Use Add Files, Add Directories or Create File buttons below

Add Files Add Directories Create File

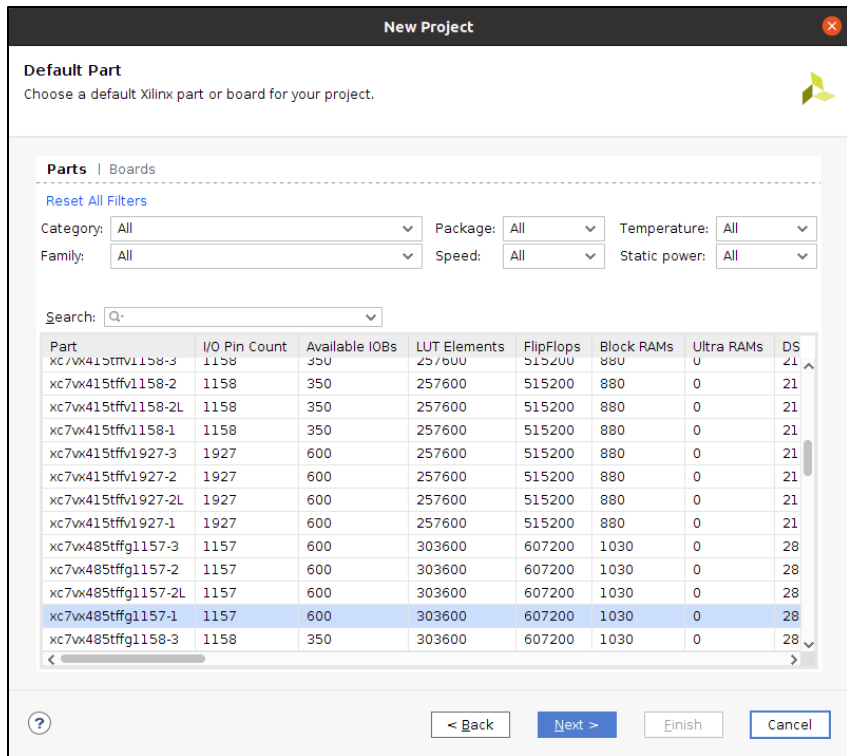
Scan and add RTL include files into project  
 Copy sources into project  
 Add sources from subdirectories

Target language: Verilog Simulator language: Mixed

? < Back Next > Finish Cancel

# 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*.



**New Project**

Default Part  
Choose a default Xilinx part or board for your project.

Parts | **Boards**

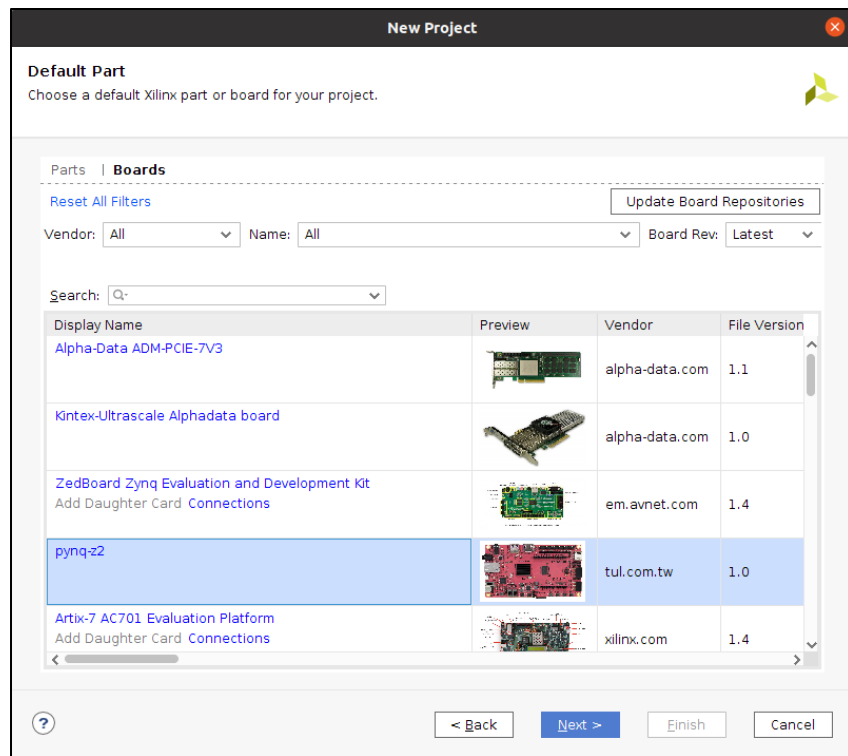
[Reset All Filters](#)

Category: All Package: All Temperature: All  
Family: All Speed: All Static power: All

Search: Q-

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DS
xc7vx415tffv1158-3	1158	350	257600	515200	880	0	21
xc7vx415tffv1158-2	1158	350	257600	515200	880	0	21
xc7vx415tffv1158-2L	1158	350	257600	515200	880	0	21
xc7vx415tffv1158-1	1158	350	257600	515200	880	0	21
xc7vx415tffv1927-3	1927	600	257600	515200	880	0	21
xc7vx415tffv1927-2	1927	600	257600	515200	880	0	21
xc7vx415tffv1927-2L	1927	600	257600	515200	880	0	21
xc7vx415tffv1927-1	1927	600	257600	515200	880	0	21
xc7vx485tffg1157-3	1157	600	303600	607200	1030	0	28
xc7vx485tffg1157-2	1157	600	303600	607200	1030	0	28
xc7vx485tffg1157-2L	1157	600	303600	607200	1030	0	28
xc7vx485tffg1157-1	1157	600	303600	607200	1030	0	28
xc7vx485tffg1158-3	1158	350	303600	607200	1030	0	28

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)



**New Project**






Default Part  
Choose a default Xilinx part or board for your project.

Parts | **Boards**

[Reset All Filters](#) [Update Board Repositories](#)

Vendor: All Name: All Board Rev: Latest

Search: Q-

Display Name	Preview	Vendor	File Version
Alpha-Data ADM-PCIE-7V3		alpha-data.com	1.1
Kintex-Ultrascale Alphadata board		alpha-data.com	1.0
ZedBoard Zynq Evaluation and Development Kit Add Daughter Card <a href="#">Connections</a>		em.avnet.com	1.4
pynq-z2		tul.com.tw	1.0
Artix-7 AC701 Evaluation Platform Add Daughter Card <a href="#">Connections</a>		xilinx.com	1.4

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

# Vivado Interface

# Vivado Interface

The screenshot displays the Vivado 2019.1 software interface. The main window title is "project\_1 - [/home/amert/Documents/project\_1/project\_1.xpr] - Vivado 2019.1". The interface is divided into several panes:

- Flow Navigator:** A vertical sidebar on the left containing project management tasks such as "Settings", "Add Sources", "Language Templates", "IP Catalog", "IP INTEGRATOR" (with options like "Create Block Design"), "SIMULATION", "RTL ANALYSIS", "SYNTHESIS", "IMPLEMENTATION", and "PROGRAM AND DEBUG".
- PROJECT MANAGER - project\_1:** The central workspace, divided into three sections:
  - Sources:** Lists "Design Sources", "Constraints", "Simulation Sources" (containing "sim\_1"), and "Utility Sources".
  - Hierarchy:** Includes tabs for "Libraries" and "Compile Order".
  - Properties:** A section for viewing object properties, currently showing "Select an object to see properties".
- Project Summary:** A panel on the right providing an overview of the project:
  - Overview | Dashboard:** Contains "Settings" and "Edit" options.
  - Settings:** Lists project details: Project name (project\_1), Project location (/home/amert/Documents/project\_1), Product family (Zynq-7000), Project part (pynq-z2 (xc7z020clg400-1)), Top module name (Not defined), Target language (Verilog), and Simulator language (Mixed).
  - Board Part:** Lists board information: Display name (pynq-z2), Board part name (tul.com.tw:pynq-z2:part0:1.0), Board revision (1.0), Connectors (No connections), Repository path (/opt/Xilinx/Vivado/2019.1/data/boards/board\_files), URL (http://www.tul.com.tw), and Board overview (pynq-z2). A small image of the PYNQ-Z2 board is shown.
- Tcl Console:** A bottom panel showing the execution of Tcl commands and their output:

```
close sim
INFO: [Sintcl 6-16] Simulation closed
close_project
create_project project_1 /home/amert/Documents/project_1 -part xc7z020clg400-1
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-233] Loaded Vivado IP repository: /opt/Xilinx/Vivado/2019.1/data/ip/
create_project: Time (s): cpu = 00:00:07 ; elapsed = 00:00:05 . Memory (MB): peak = 6781.410 ; gain = 0.000 ; free physical = 1576 ; free virtual = 13510
set_property board_part tul.com.tw:pynq-z2:part0:1.0 [current_project]
```



# Vivado Interface

The screenshot displays the Vivado 2019.1 interface for a project named 'project\_1'. The interface is divided into several panes:

- Flow Navigator:** A vertical sidebar on the left containing project settings and simulation options.
- Project Manager:** A central pane showing the project's sources (Design, Constraints, Simulation, Utility) and a hierarchy view.
- Project Summary:** A pane on the right providing an overview of the project, including settings and board part information.
- Tcl Console:** A pane at the bottom showing the execution of Tcl commands and their output.

Annotations with yellow boxes and arrows point to specific features:

- Project Settings:** Points to the 'Settings' option in the Flow Navigator.
- Runs Behavioral Simulation:** Points to the 'Run Simulation' option in the SIMULATION section of the Flow Navigator.
- Compiles RTL code and loads RTL netlist:** Points to the 'Run Synthesis' option in the SYNTHESIS section of the Flow Navigator.
- Implementation run and settings:** Points to the 'Run Implementation' option in the IMPLEMENTATION section of the Flow Navigator.
- Shows design and simulation files with project hierarchy:** Points to the 'Hierarchy' tab in the Project Manager pane.
- Summary of project and results:** Points to the 'Project Summary' pane.
- Messages tab shows useful information (warnings, errors) about your design:** Points to the 'Messages' tab in the Tcl Console pane.

Project Settings

Runs Behavioral Simulation

Compiles RTL code and loads RTL netlist

Implementation run and settings

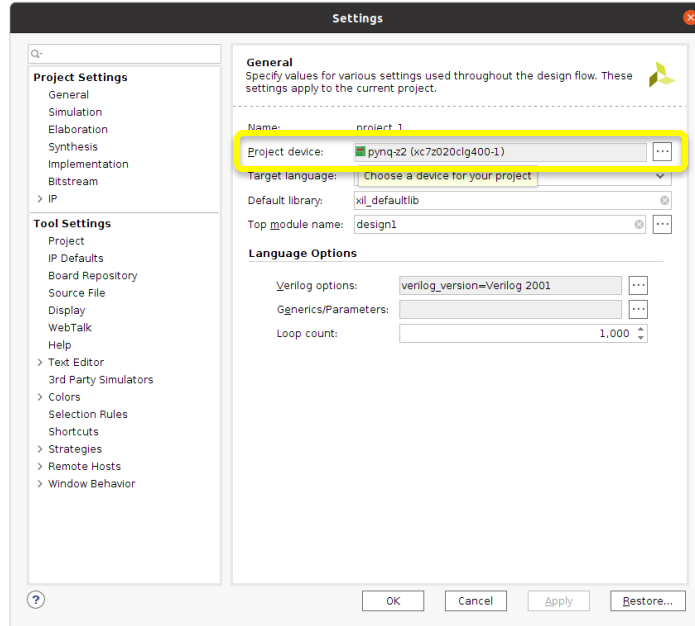
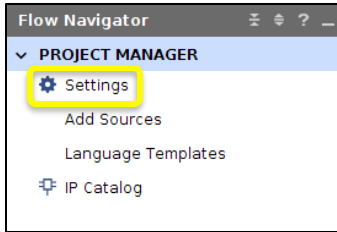
Shows design and simulation files with project hierarchy

Summary of project and results

Messages tab shows useful information (warnings, errors) about your design.

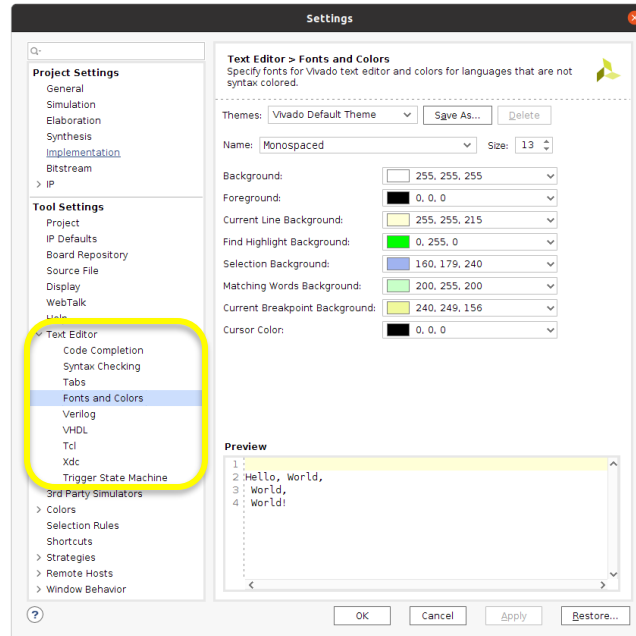
# Vivado Interface

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



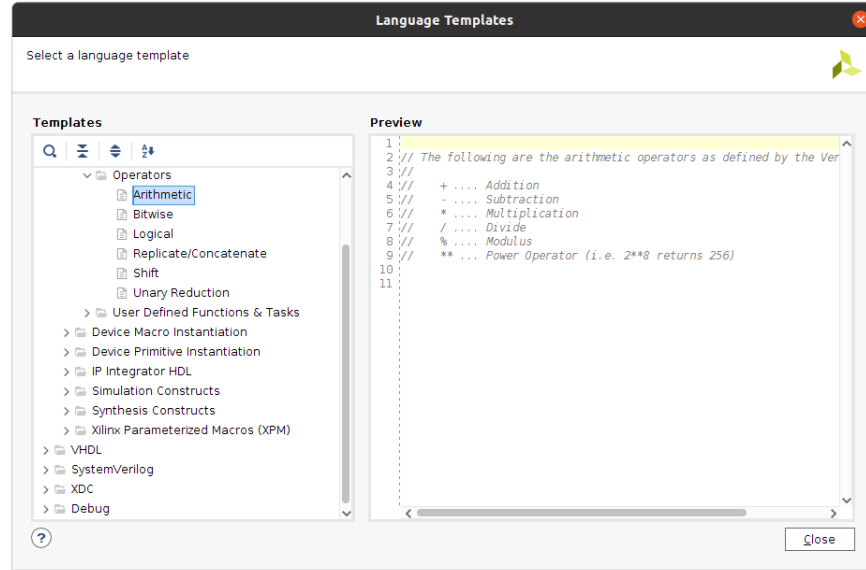
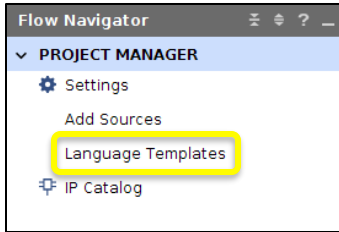
# Vivado Interface

- 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.



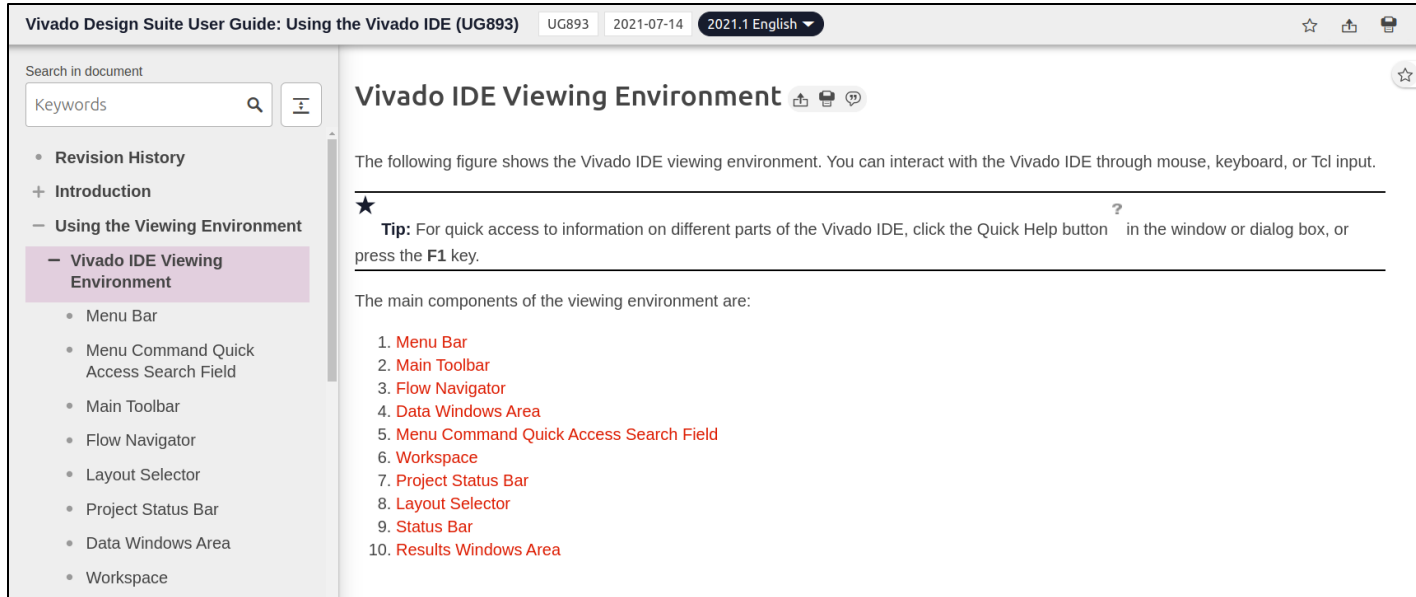
# Vivado Interface

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



# Vivado Interface

- See *Vivado IDE Viewing Environment* chapter of *Vivado Design Suite User Guide*<sup>[1]</sup>.

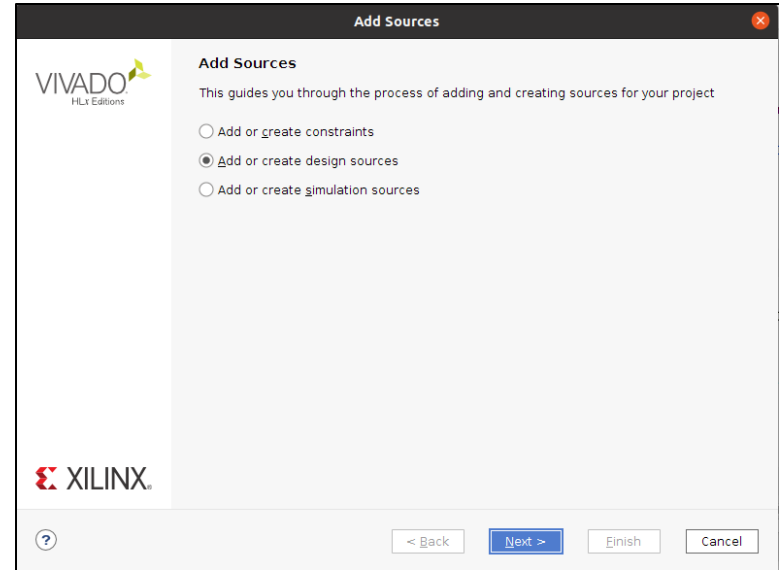
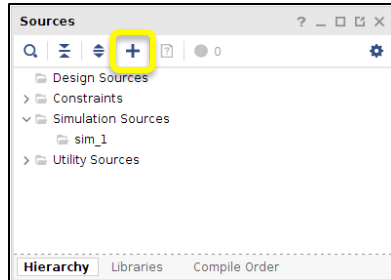
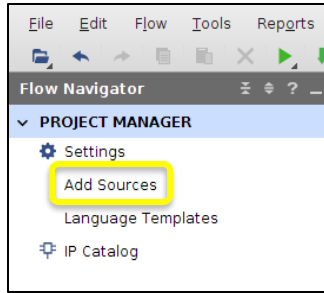


[1] <https://docs.xilinx.com/r/2021.1-English/ug893-vivado-ide/Vivado-IDE-Viewing-Environment>

## **Adding/Creating Design and Simulation Files**

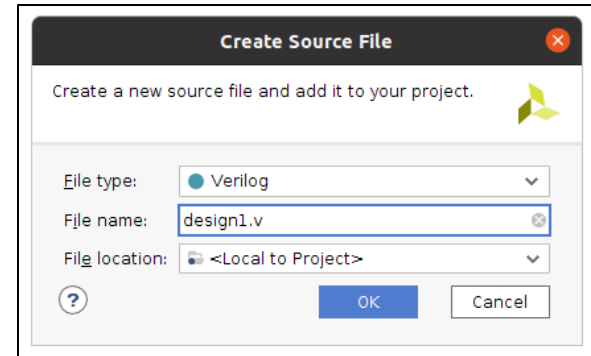
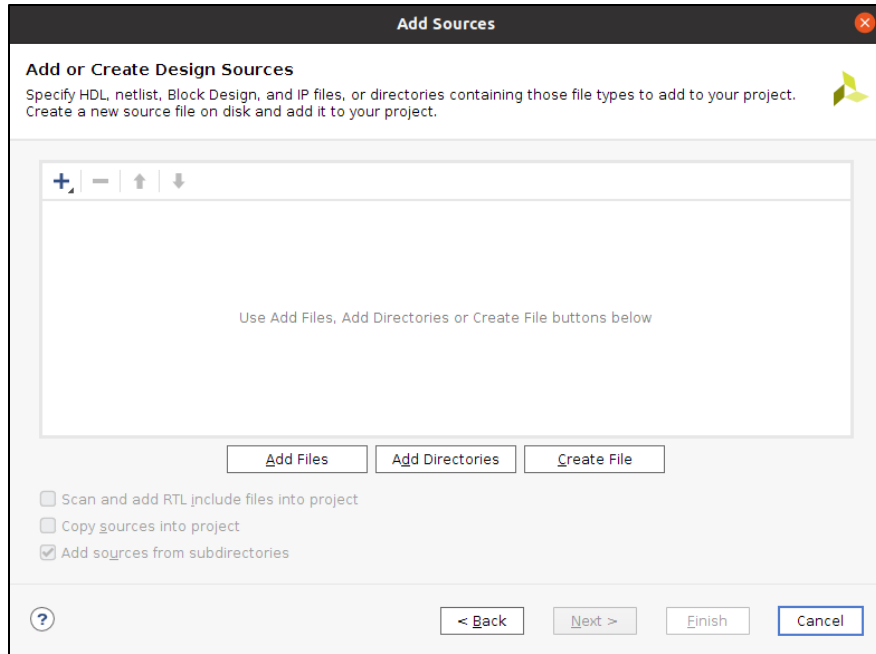
# Create a Design Source

- 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

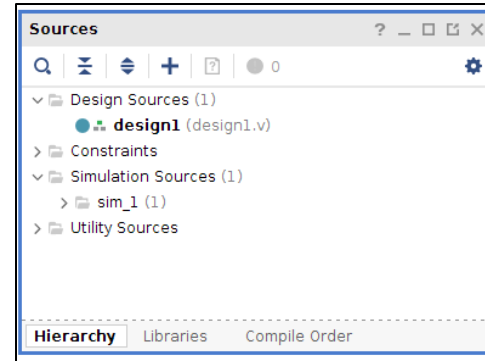
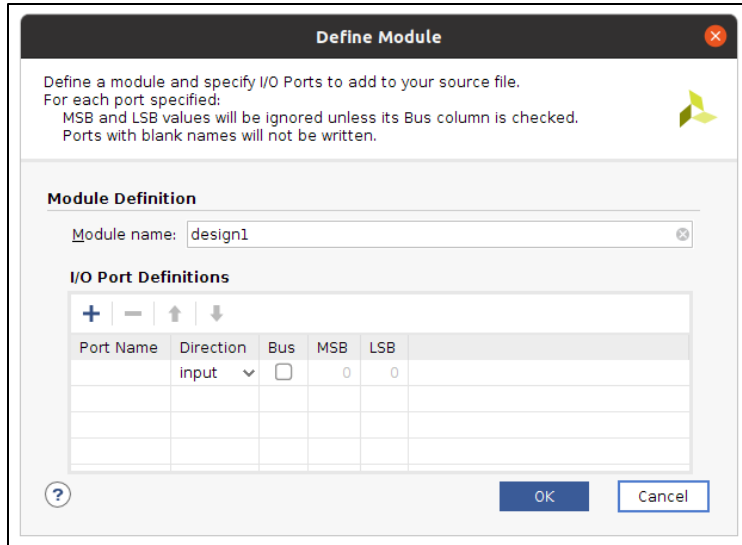
- Click on *Create File* button, name your source file (e.g., design1.v) and click on *Next*.
  - If you want to add an existing file to your project, click on *Add Files* and select the files that you want to add to the project.





# 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

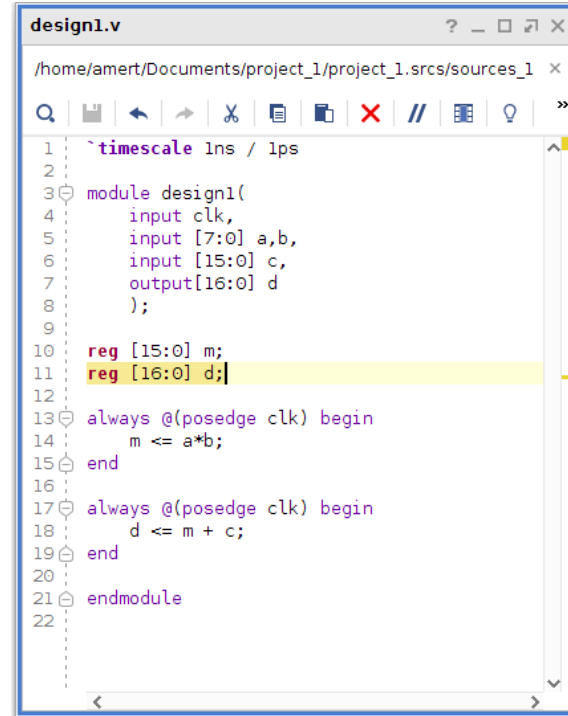
module design1(
    input clk,
    input [7:0] a,b,
    input [15:0] c,
    output [16:0] d
);

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
```

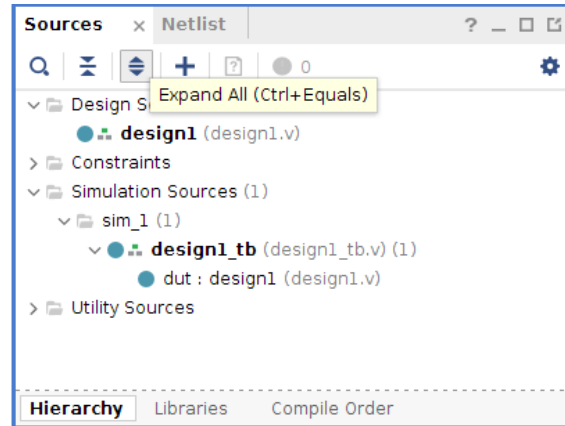


The screenshot shows a code editor window titled "design1.v" with the following content:

```
design1.v
/home/amert/Documents/project_1/project_1.srcs/sources_1
1 `timescale 1ns / 1ps
2
3 module design1(
4     input clk,
5     input [7:0] a,b,
6     input [15:0] c,
7     output [16:0] d
8 );
9
10 reg [15:0] m;
11 reg [16:0] d;
12
13 always @(posedge clk) begin
14     m <= a*b;
15 end
16
17 always @(posedge clk) begin
18     d <= m + c;
19 end
20
21 endmodule
22
```

# 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;

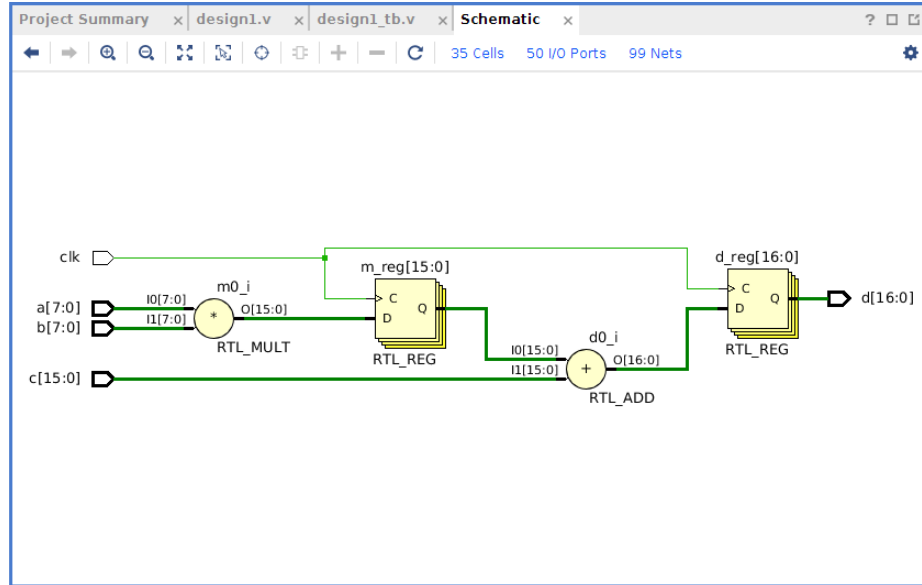
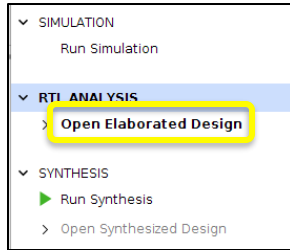
design1 dut (a,b,c,d);

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

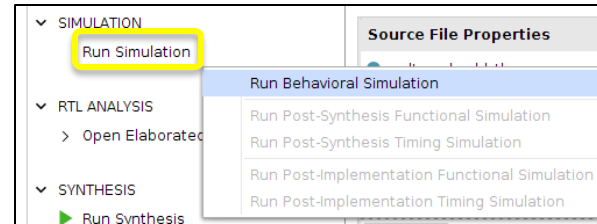
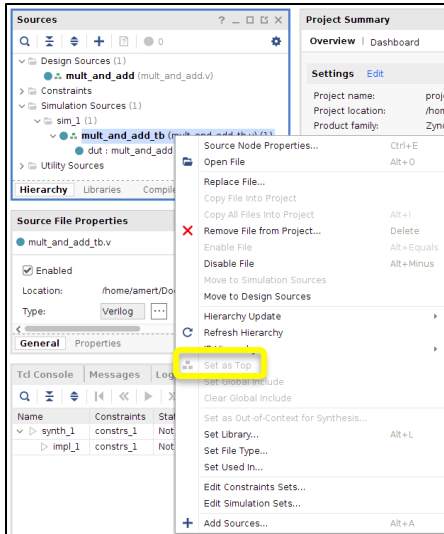
- Design Elaboration and RTL Analysis
  - It compiles RTL code and loads RTL netlist
  - You can check RTL structure, syntax, and logic definitions
  - You can view the schematic of your design



# Running Behavioral Simulation

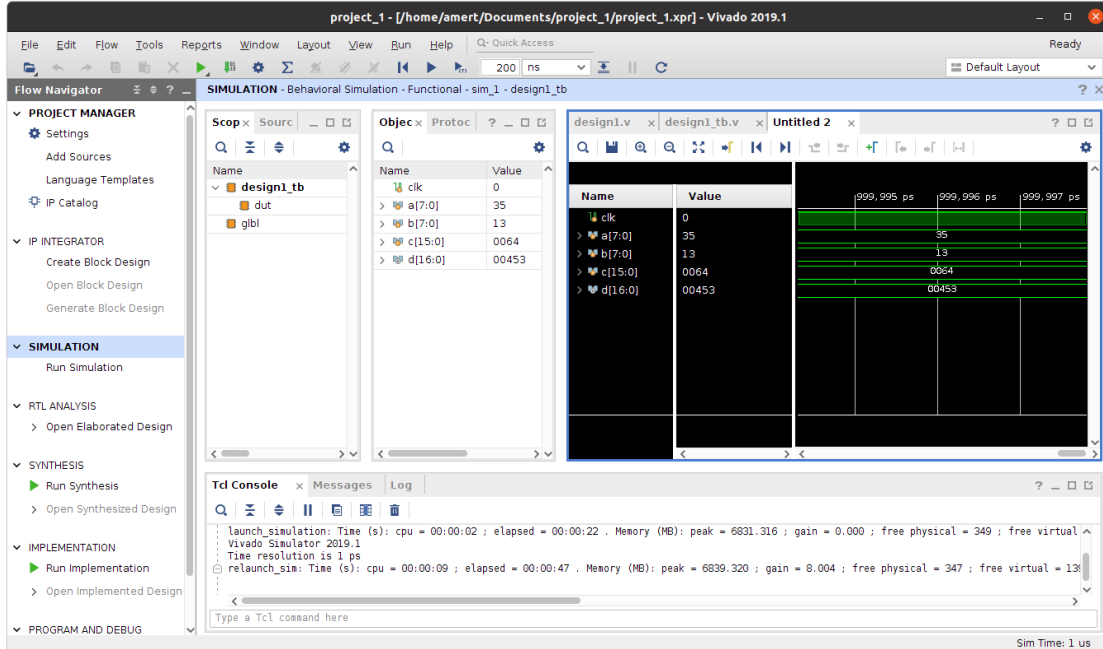
# Behavioral Simulation

- 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

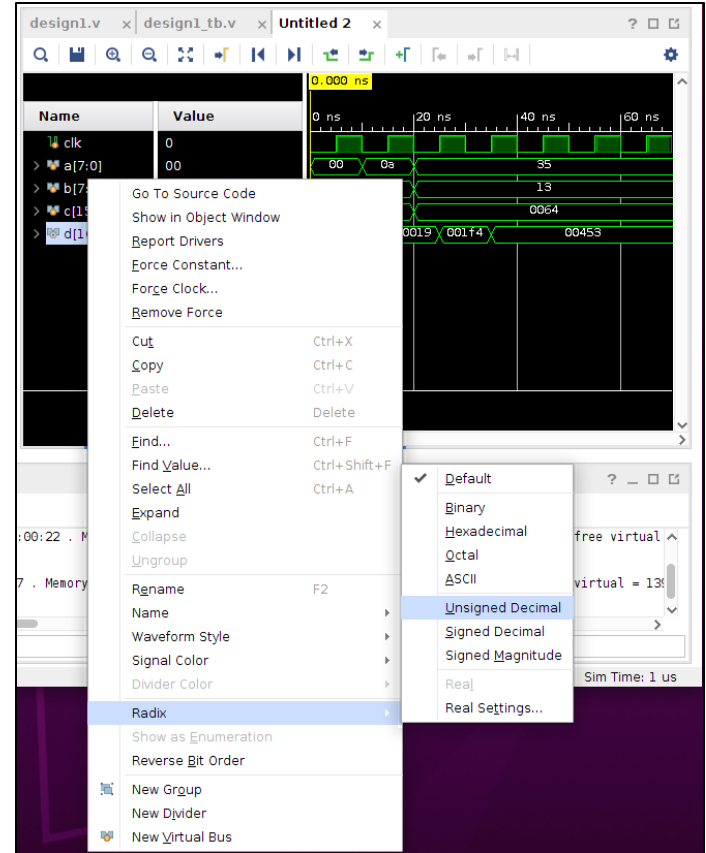


Go to the last time



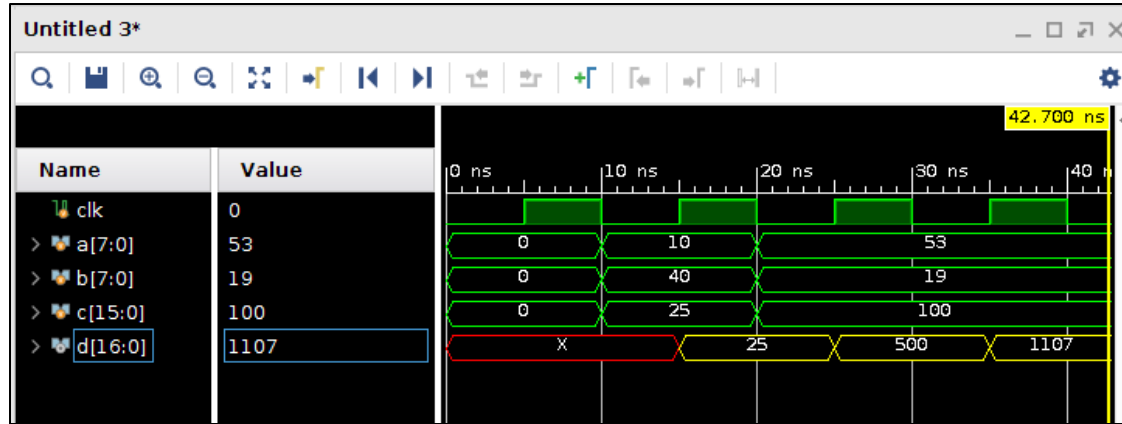
# Behavioral Simulation

- 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.

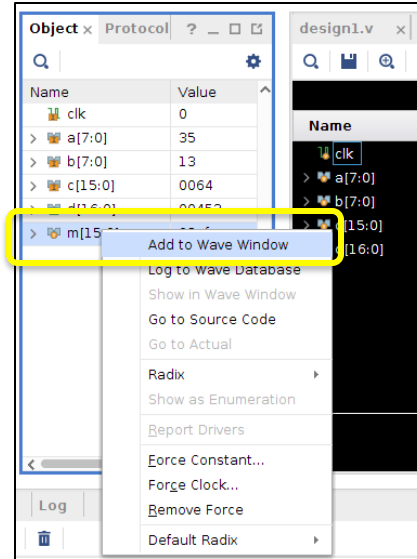
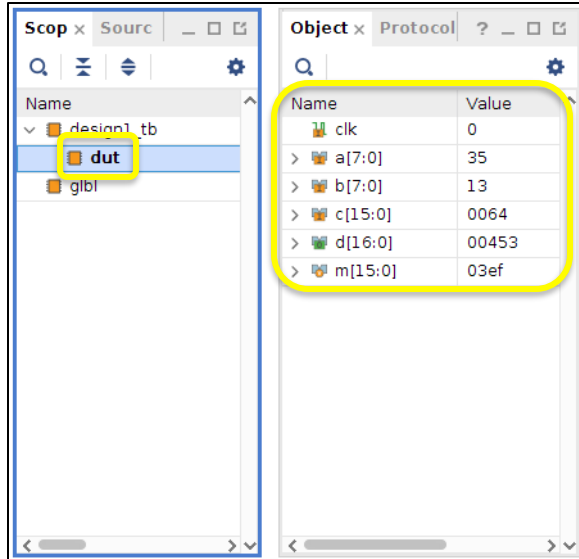
The screenshot displays a behavioral simulation tool interface with three main windows:

- Scope Window:** Shows the design hierarchy. The selected scope is `design1_tb`, which contains `dut` and `glbl`.
- Object Window:** Shows the HDL objects visible from the selected scope. The objects and their values are:

Name	Value
clk	0
a[7:0]	35
b[7:0]	13
c[15:0]	0064
d[16:0]	00453
- Waveform Window:** Shows a timing diagram for the selected scope. The time axis ranges from 0 ns to 60 ns. The signals shown are:
  - `clk`: A square wave signal.
  - `a[7:0]`: A signal that transitions from 00 to 0a at approximately 10 ns and then to 35 at approximately 20 ns.
  - `b[7:0]`: A signal that transitions from 00 to 28 at approximately 10 ns and then to 13 at approximately 20 ns.
  - `c[15:0]`: A signal that transitions from 0000 to 0019 at approximately 10 ns and then to 0064 at approximately 20 ns.
  - `d[16:0]`: A signal that transitions from xxxxx to 00019 at approximately 10 ns and then to 001f4 at approximately 20 ns, and finally to 00453 at approximately 30 ns.

# 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

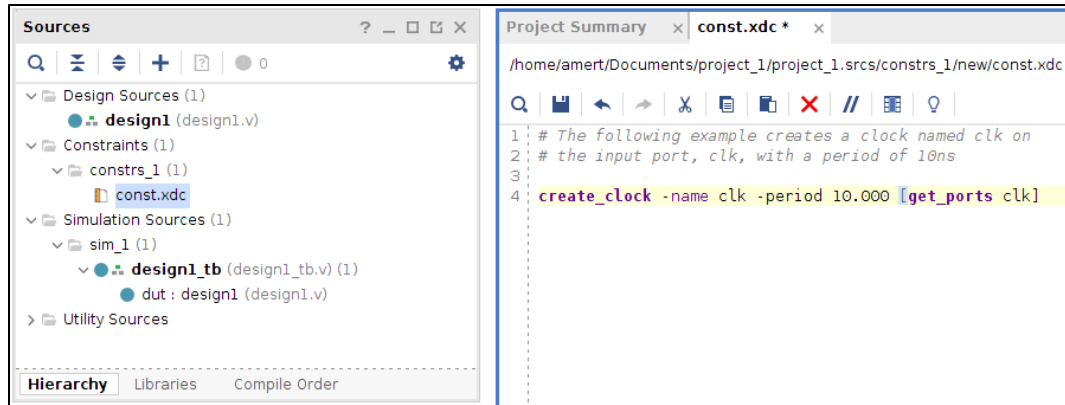
# Constraint File

- 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 <sup>[1]</sup>

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

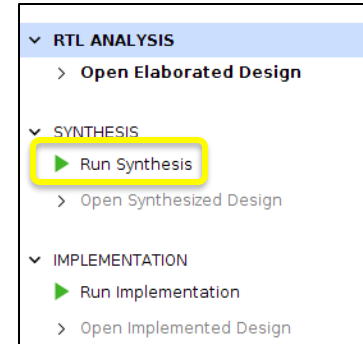
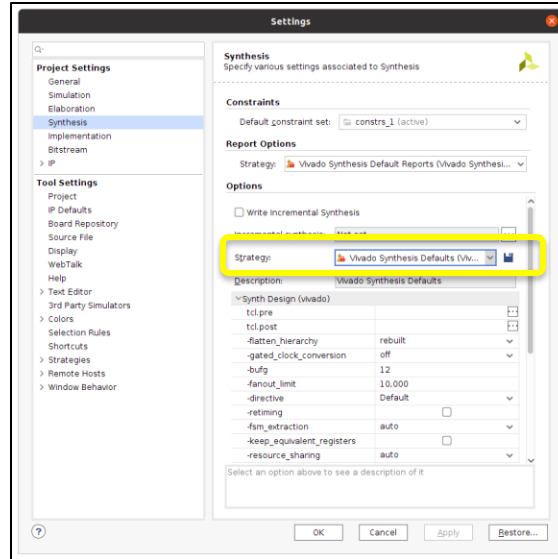
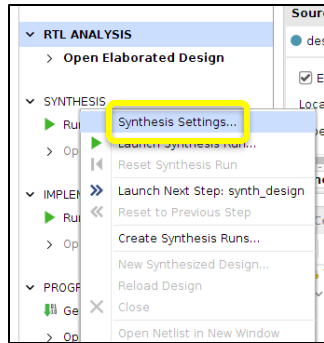


[1] [https://docs.xilinx.com/r/2021.2-English/ug835-vivado-tcl-commands/create\\_clock](https://docs.xilinx.com/r/2021.2-English/ug835-vivado-tcl-commands/create_clock)



# Synthesis

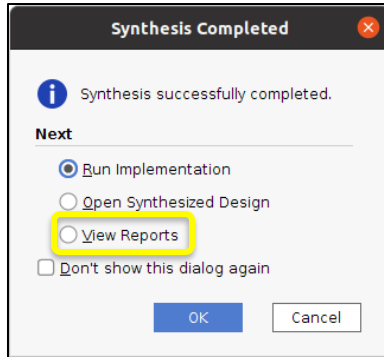
- Synthesis translates RTL code to a netlist which defines the circuit<sup>[1]</sup>
  - You can change synthesis strategy (i.e., area optimized or performance optimized) from synthesis settings (right click on *SYNTHESIS* -> *Synthesis Settings*)
  - For starting synthesis, click on *Run Synthesis*.



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

# Synthesis

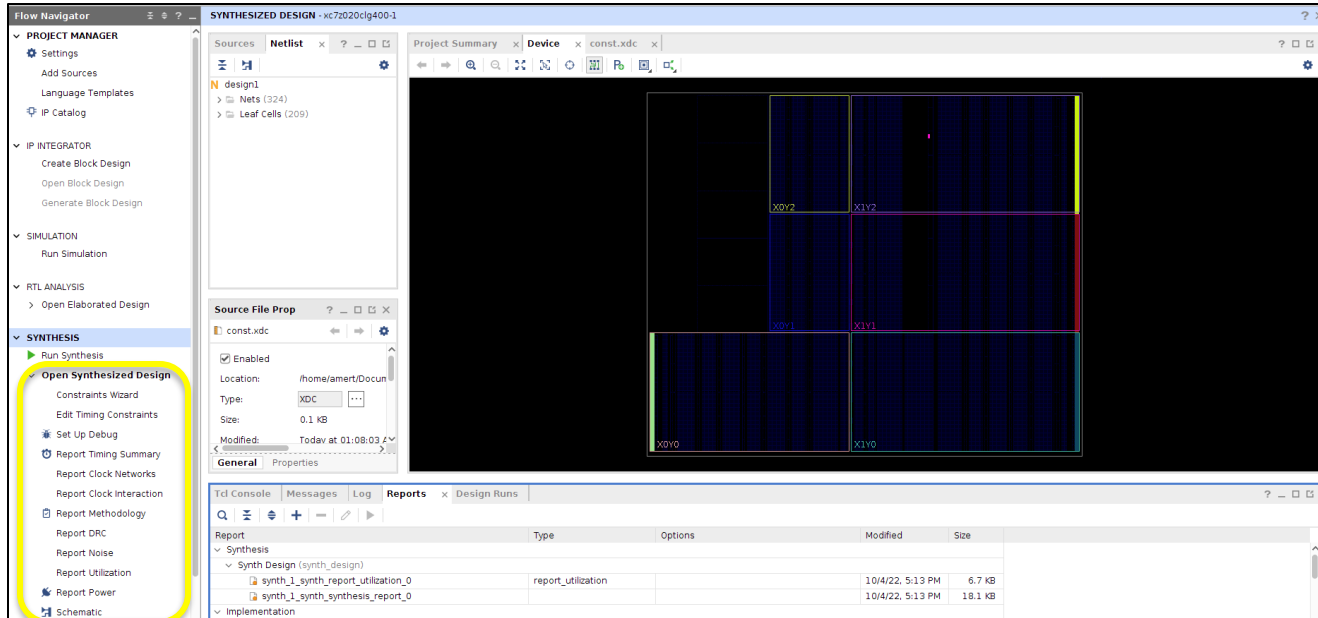
- After synthesis is finished, you can directly start implementation, open the synthesized design or view synthesis reports.
  - You can see synthesis report summary in *Project Summary*.



Utilization		Post-Synthesis   Post-Implementation		
		Graph   Table		
Resource	Estimation	Available	Utilization %	
LUT	87	53200	0.16	
FF	33	106400	0.03	
IO	50	125	40.00	
BUFG	1	32	3.13	

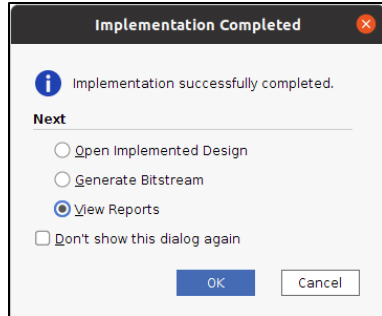
# Synthesis

- 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<sup>[1]</sup>.
  - 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*.



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

Timing	Setup	Hold	Pulse Width
Worst Negative Slack (WNS):	7.509 ns		
Total Negative Slack (TNS):	0 ns		
Number of Failing Endpoints:	0		
Total Number of Endpoints:	17		
<a href="#">Implemented Timing Report</a>			

[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.

The screenshot displays the implementation tool interface. On the left, the 'Flow Navigator' shows the 'IMPLEMENTATION' section with 'Open Implemented Design' highlighted. A callout box points to this menu item with the text 'Open Implemented Design'. Below it, another callout box points to 'Report Timing Summary' and 'Report Utilization' with the text 'For detailed timing report' and 'For detailed utilization report' respectively. The main window shows the 'Placed design' window with a circuit diagram. A callout box points to this window with the text 'Placed design'. At the bottom, the 'Design Timing Summary' report is displayed, which is highlighted with a yellow box and labeled 'Timing summary'.

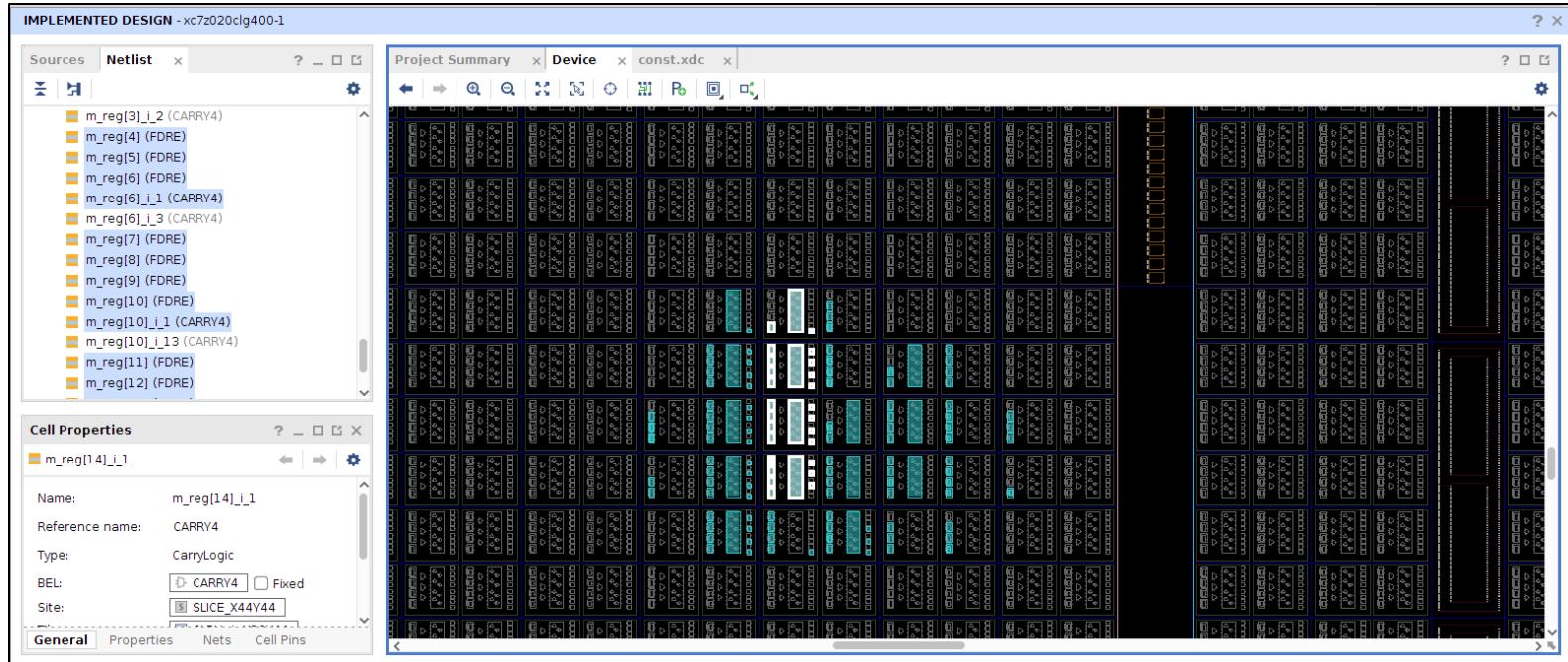
**Design Timing Summary**

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.
  - 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	
<b>All user specified timing constraints are met.</b>					

# Implementation

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

The screenshot displays the Xilinx ISE interface with the 'Timing Summary' report open. The report shows a critical path with a delay of 7.509 ns. The path is highlighted in yellow in the original image. The path components are:

- net (fo=33, routed) 1.667 ns
- FDRE (clk\_IBUF\_BUF6) 5.048 ns
- net (fo=2, routed) 0.816 ns
- LUT2 (Prop Lut2\_I0\_0) 0.124 ns
- net (fo=1, routed) 0.000 ns
- CARRY4 (Prop\_c\_4\_S11\_CO[3]) 0.550 ns
- net (fo=1, routed) 0.000 ns
- CARRY4 (Prop\_carry4\_CI\_CO[3]) 0.114 ns
- net (fo=1, routed) 0.000 ns
- CARRY4 (Prop\_carry4\_CI\_CO[3]) 0.114 ns
- net (fo=1, routed) 0.000 ns
- CARRY4 (Prop\_carry4\_CI\_CO[0]) 0.271 ns
- net (fo=1, routed) 0.000 ns
- FDRE (d0[16]) 7.492 ns

The 'Arrival Time' is 7.492 ns and the 'Destination Clock Path' is d0[16].

The 'Timing Summary' table shows the following critical paths:

Path	Delay (ns)	Setup (ns)	Hold (ns)	Source	Destination	Setup (ns)	Hold (ns)	Slack (ns)	Source	Destination	
Path 1	7.509	5	2	m_reg[5]/C	d_reg[16]/D	2.445	1.629	0.816	10.0	clk	clk
Path 2	7.509	4	2	m_reg[3]/C	d_reg[13]/D	2.394	1.578	0.816	10.0	clk	clk
Path 3	7.597	4	2	m_reg[5]/C	d_reg[15]/D	2.373	1.557	0.816	10.0	clk	clk
Path 4	7.671	4	2	m_reg[5]/C	d_reg[14]/D	2.299	1.483	0.816	10.0	clk	clk
Path 5	7.687	4	2	m_reg[5]/C	d_reg[12]/D	2.283	1.467	0.816	10.0	clk	clk
Path 6	7.690	3	2	m_reg[5]/C	d_reg[9]/D	2.280	1.464	0.816	10.0	clk	clk
Path 7	7.711	3	2	m_reg[5]/C	d_reg[11]/D	2.259	1.443	0.816	10.0	clk	clk
Path 8	7.785	3	2	m_reg[5]/C	d_reg[10]/D	2.185	1.369	0.816	10.0	clk	clk
Path 9	7.801	3	2	m_reg[5]/C	d_reg[8]/D	2.169	1.353	0.816	10.0	clk	clk
Path 10	7.846	3	2	m_reg[5]/C	d_reg[15]/D	2.156	1.341	0.816	10.0	clk	clk



# Implementation

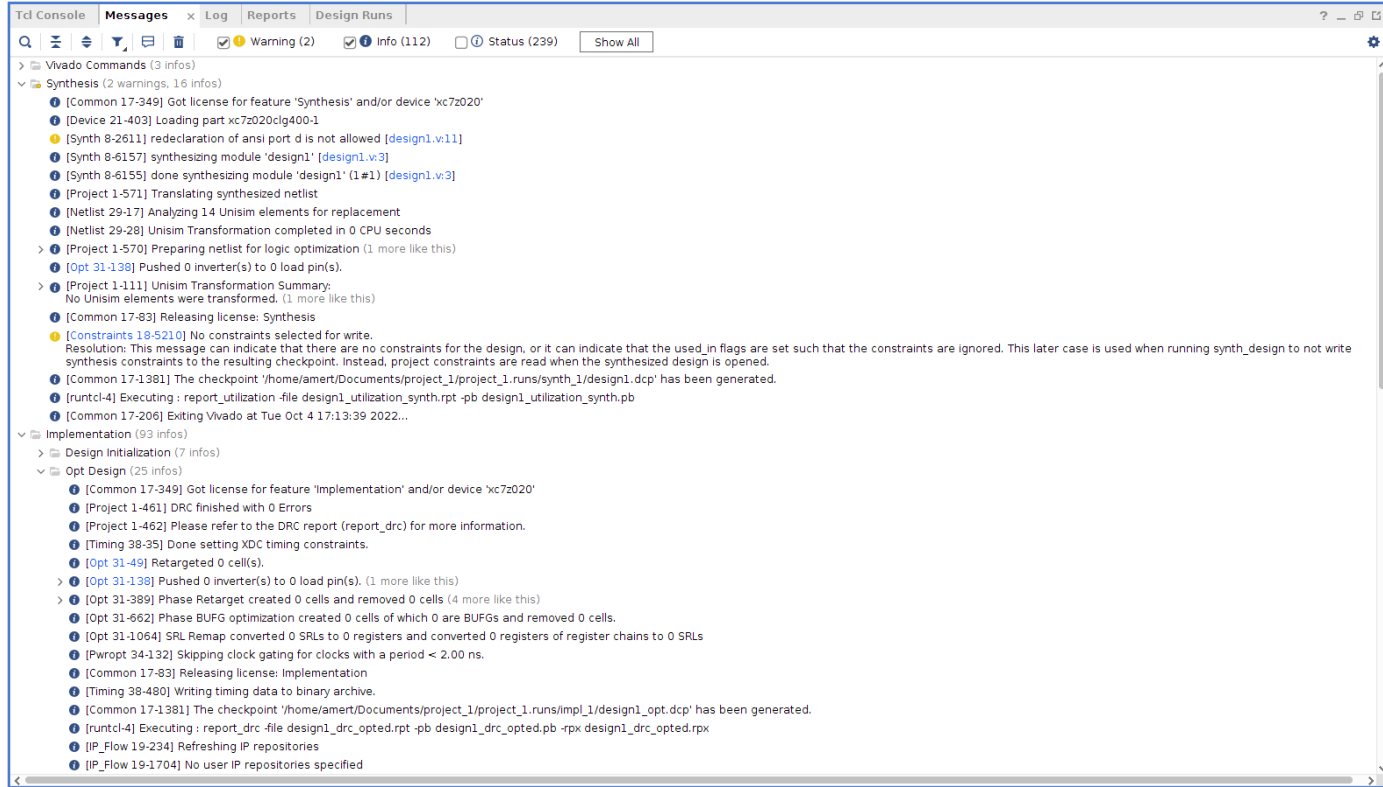
- For detailed report/results and the placed & routed design, open the implemented design.
  - Detailed area (utilization) result.

The screenshot shows the Utilization report in a design tool. The report is titled "Hierarchy" and displays utilization data for a design named "design1". The data is presented in a table with columns for Name, Slice LUTs, Slice Registers, Slice, LUT as Logic, Block RAM Tile, Bonded IPADs, and BUFI0. The utilization values are 87, 33, 27, 87, 33, 50, and 1 respectively. The total available resources are 53200 for LUTs, 106400 for Registers, 13300 for Slices, 53200 for LUT as Logic, 140 for Block RAM Tiles, 2 for Bonded IPADs, and 16 for BUFI0.

Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	Block RAM Tile (140)	Bonded IPADs (2)	BUFI0 (16)
design1	87	33	27	87	33	50	1

# Synthesis & Implementation Messages

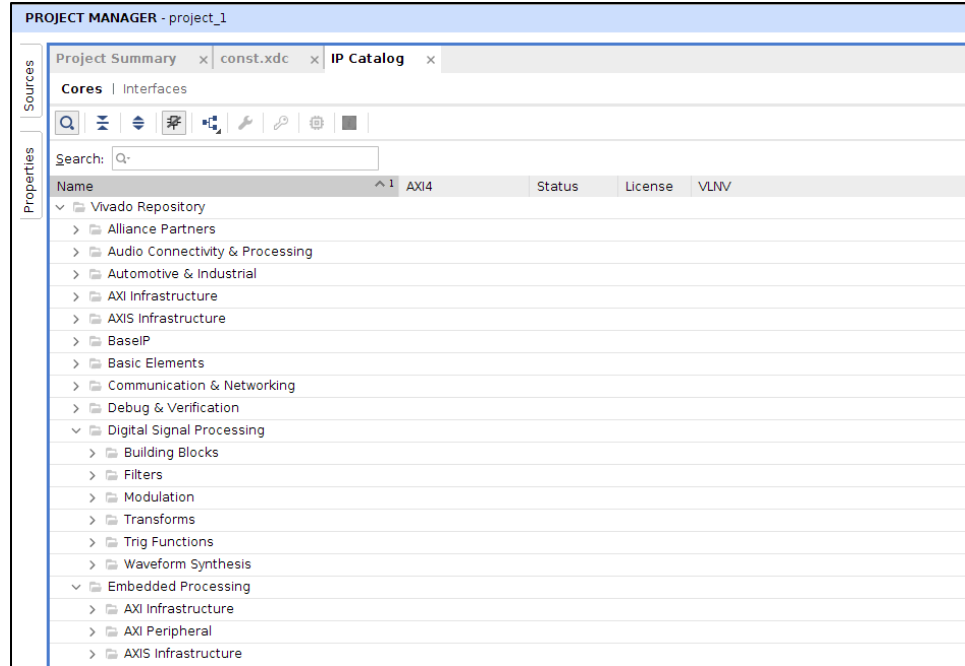
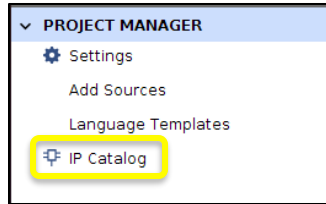
- Messages window displays a filtered list of the Vivado log. This list includes useful information such as the main messages, warnings, errors.



# **Adding Xilinx IPs to your Design**

# IP Catalog

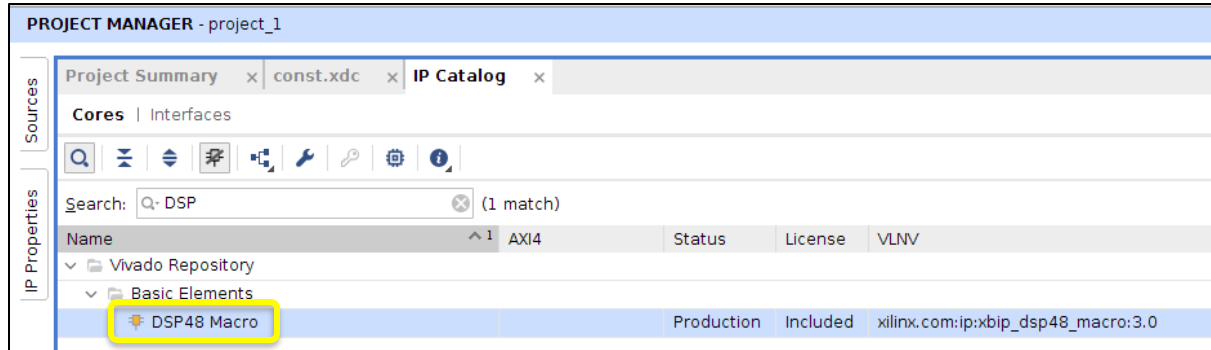
- Click on IP Catalog under Flow Navigator to see the list of available Xilinx IPs.
  - You can use Search bar to find IPs.



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

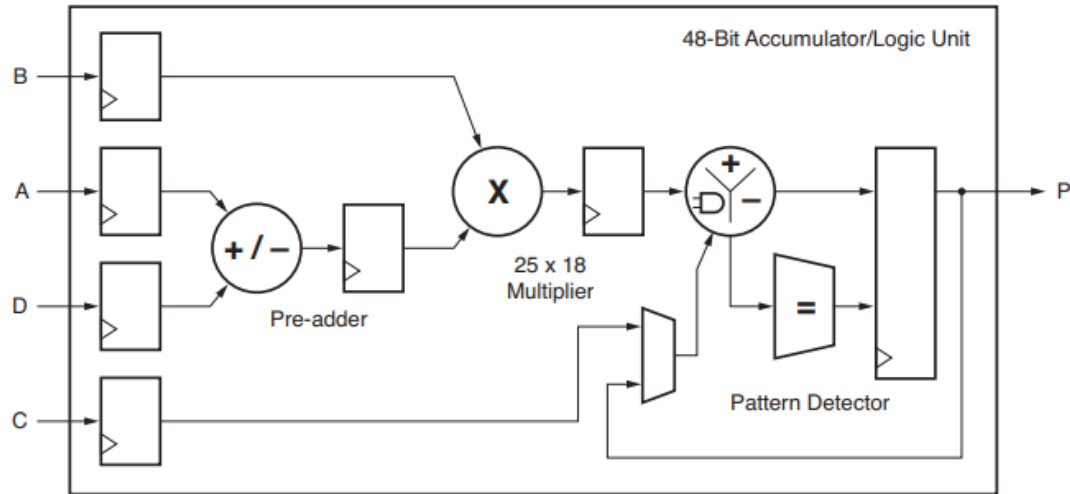
# IP Catalog Example: DSP48 Macro

- 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](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

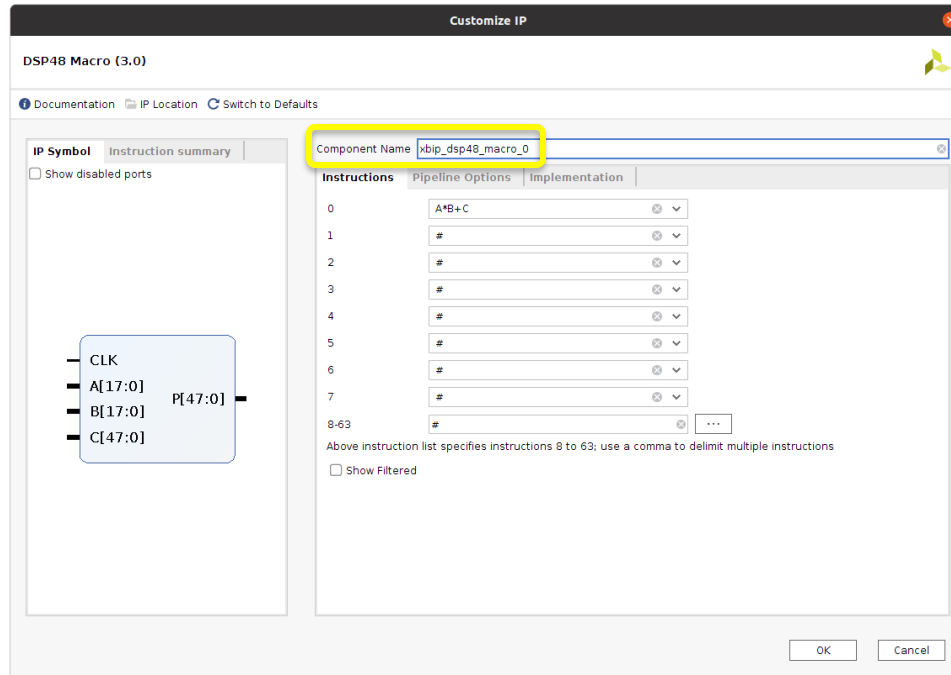
- Xilinx FPGAs have dedicated, full-custom, low-power DSP slices.
  - 7 Series DSP48E1 Slice User Guide  
[https://docs.xilinx.com/v/u/en-US/ug479\\_7Series\\_DSP48E1](https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1)
  - Search for DSP48 Macro and double click on it.



UG479\_c1\_21\_032111

# 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.

**Customize IP**

DSP48 Macro (3.0)

Documentation IP Location Switch to Defaults

IP Symbol Instruction summary

Show disabled ports

Component Name:

**Instructions** Pipeline Options Implementation

Instruction Index	Instruction
0	A*B+C
1	A*B+C
2	A*B+C+CARRYIN
3	A*B+C+CARRYIN
4	A*B+P
5	A*B+P+CARRYIN
6	A*B+P->>17
7	A*B+P->>17+CARRYIN
8	A*B+PCIN

8-63: # ...

Above instruction list specifies instructions 8 to 63; use a comma to delimit multiple instructions

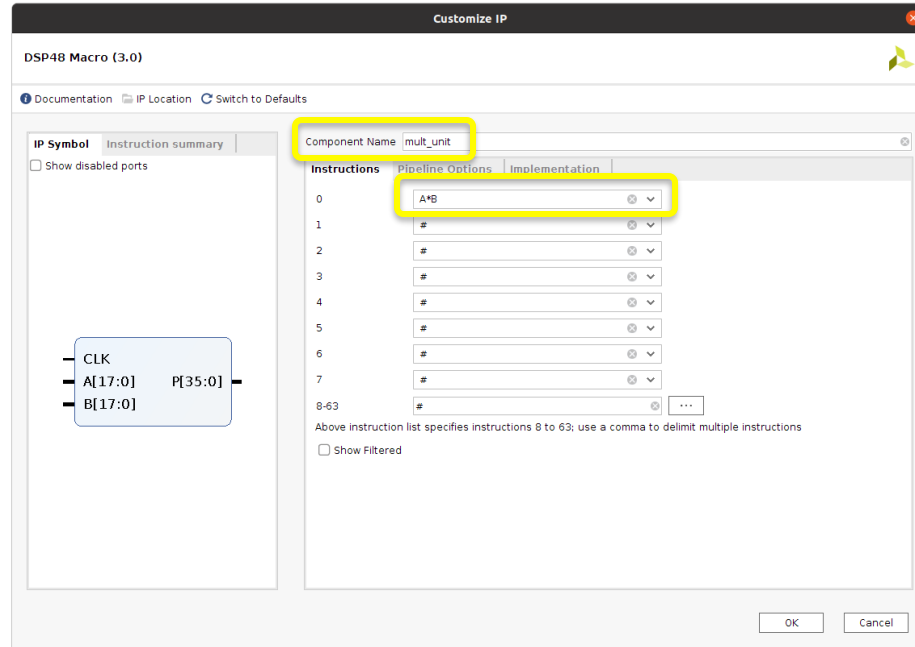
Show Filtered

OK Cancel

Block Diagram: CLK, A[17:0], B[17:0], C[47:0] → P[47:0]

# 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

Customize IP

DSP48 Macro (3.0)

Documentation IP Location Switch to Defaults

Component Name: mult\_unit

Instructions Pipeline Options Implementation

Pipeline Options: Expert

Custom Pipeline options

Tier	1	2	3	4	5	6
D	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
B	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
CONCAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CARRYIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SEL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
KEY:	Fabric register DSP register					

Control ports

	Global	D	A	B	CONCAT	C	M	P	SEL/CARRYIN
CE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SCLR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

IP Symbol | Instruction summary

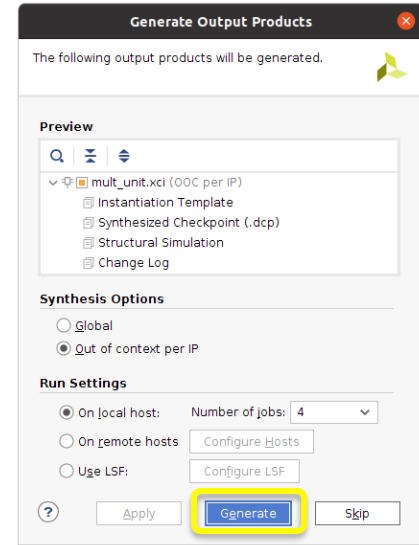
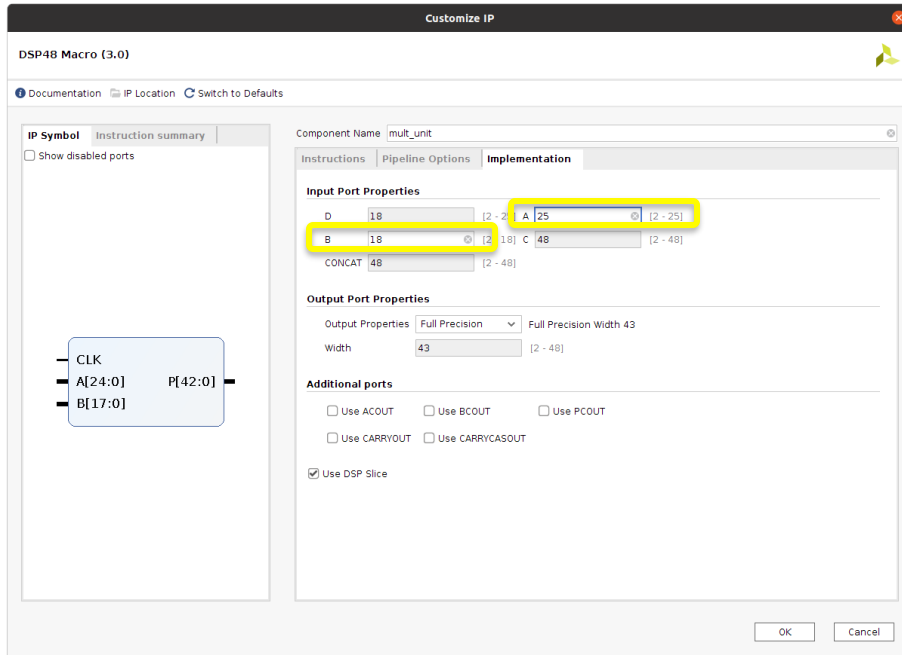
Show disabled ports

CLK A[17:0] B[17:0] P[35:0]

OK Cancel

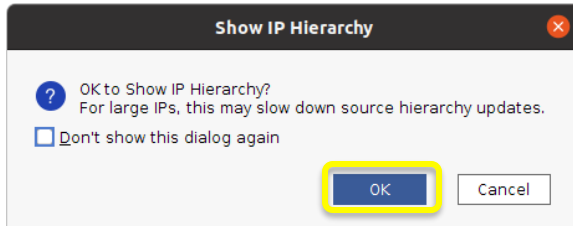
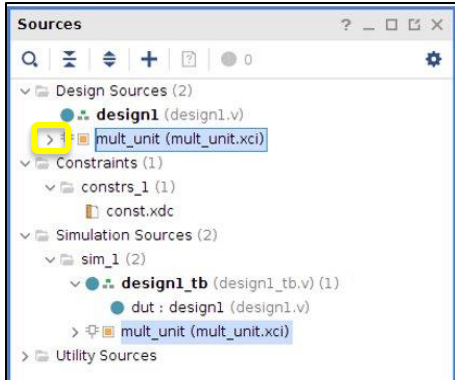
# IP Catalog Example: DSP48 Macro

- 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.



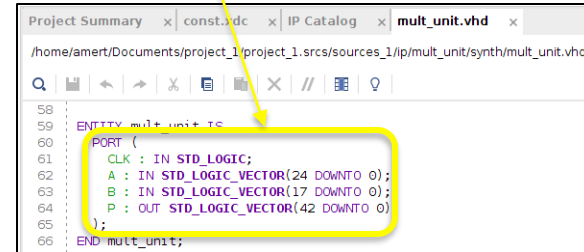
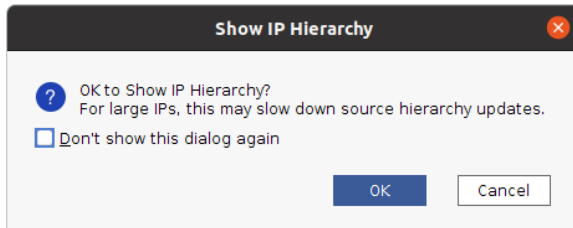
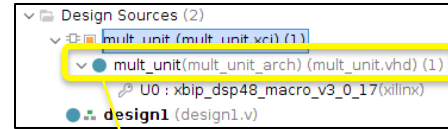
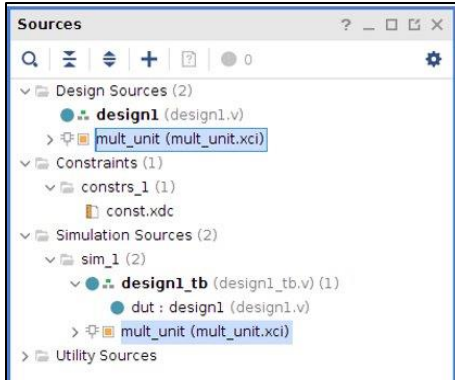
# IP Catalog Example: DSP48 Macro

- 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.

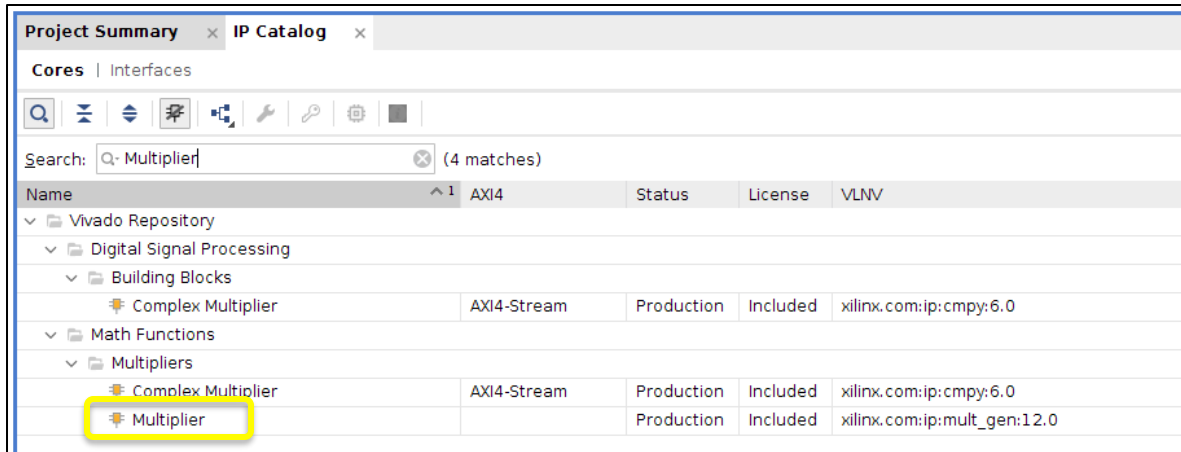
```
...  
wire [24:0] mult_unit_A;  
wire [17:0] mult_unit_B;  
wire [42:0] mult_unit_P;  
...  
mult_unit MU (clk,  
              mult_unit_A,  
              mult_unit_B,  
              mult_unit_P);  
...
```

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



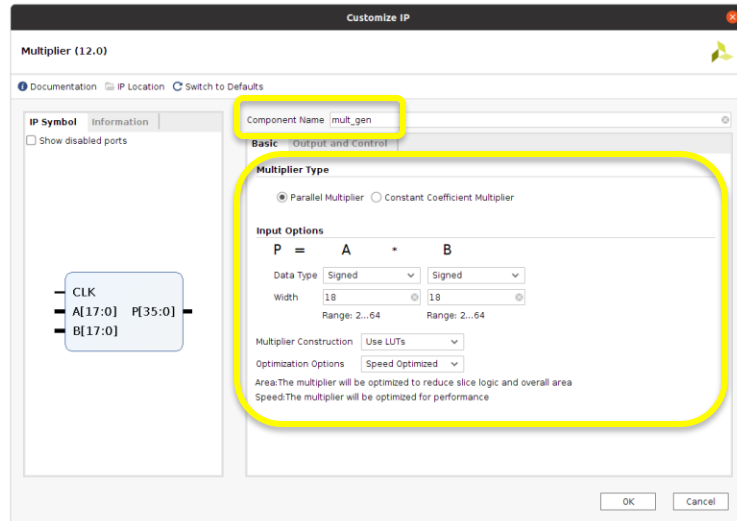
# IP Catalog Example: Multiplier

- 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.



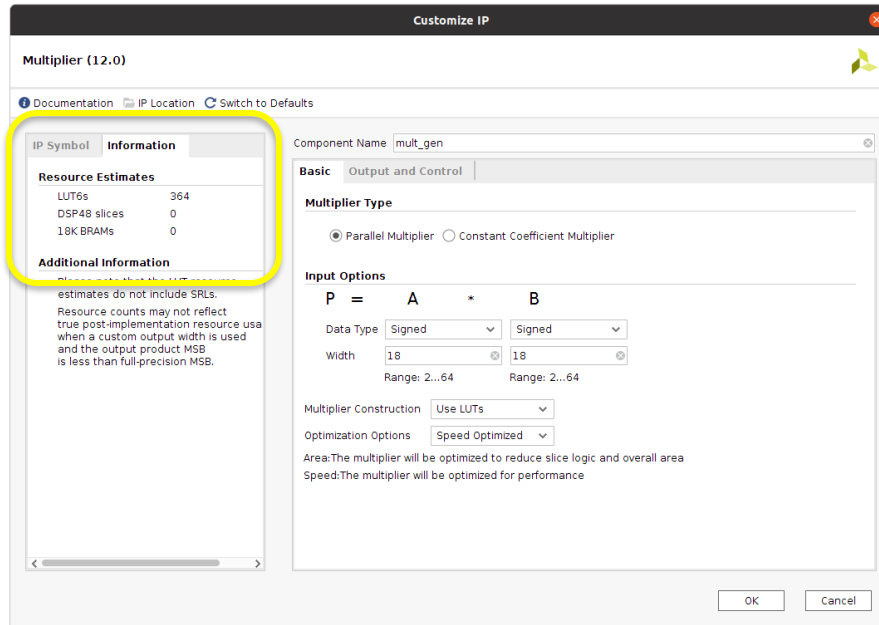
# IP Catalog Example: Multiplier

- 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



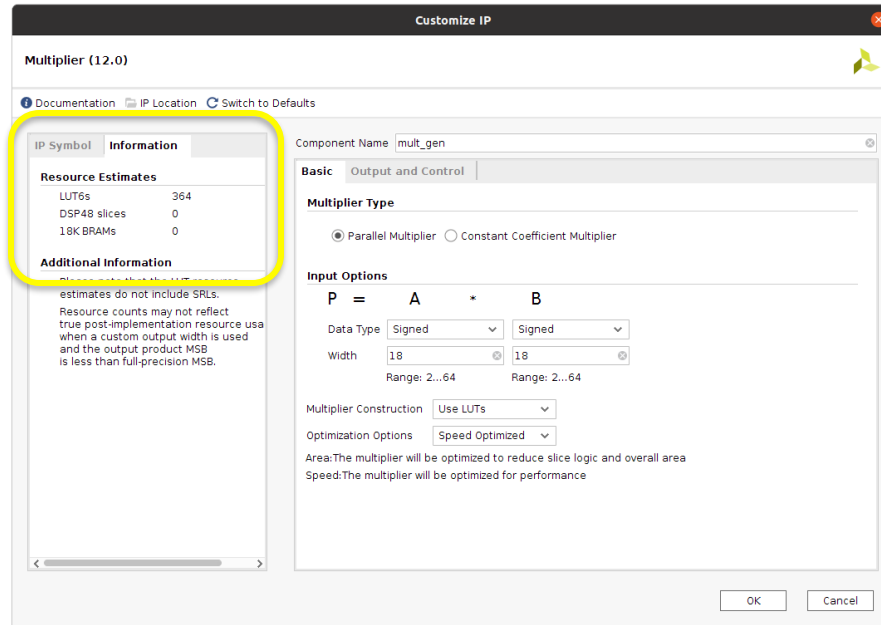
# IP Catalog Example: Multiplier

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



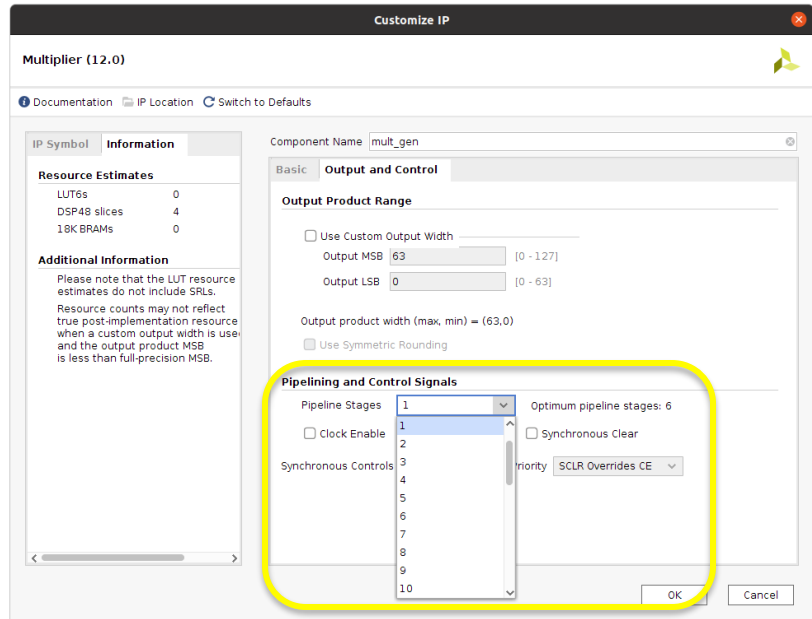
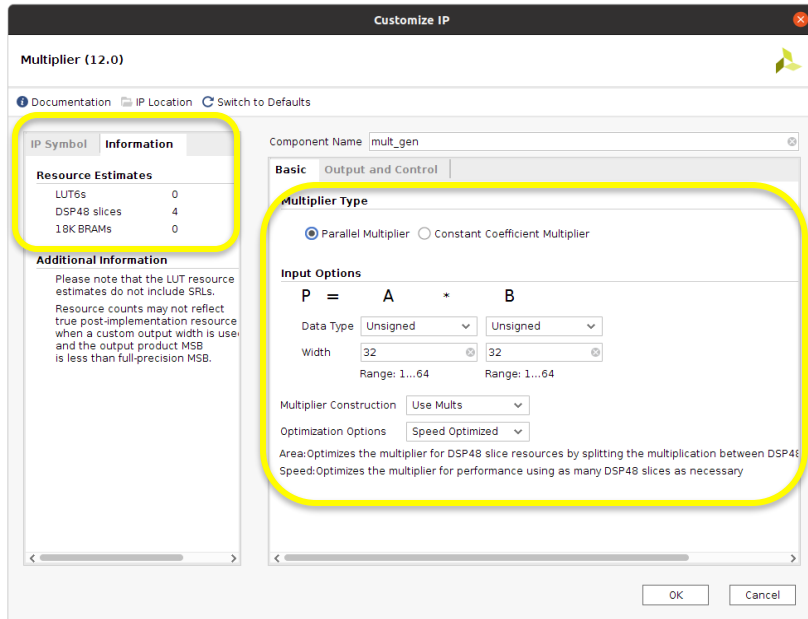
# IP Catalog Example: Multiplier

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



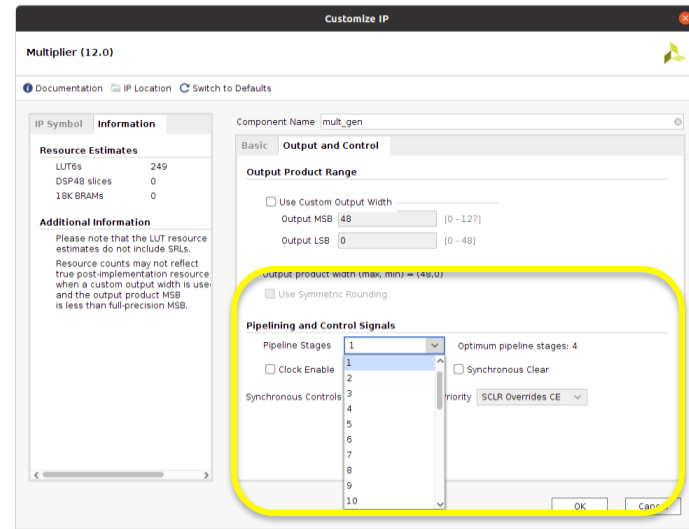
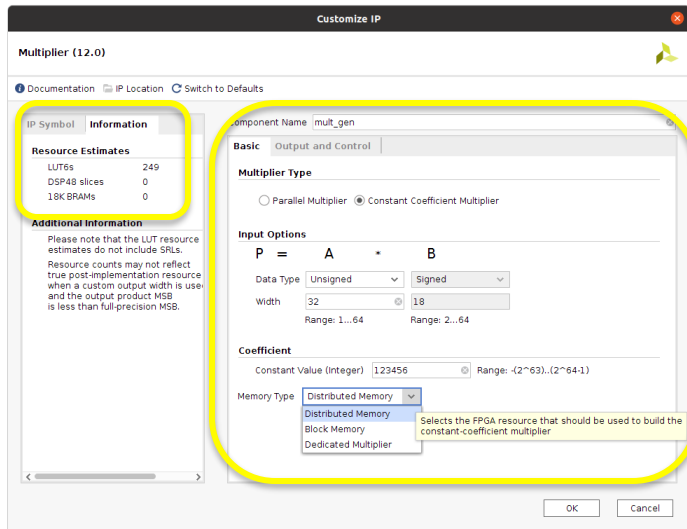
# IP Catalog Example: Multiplier

- 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

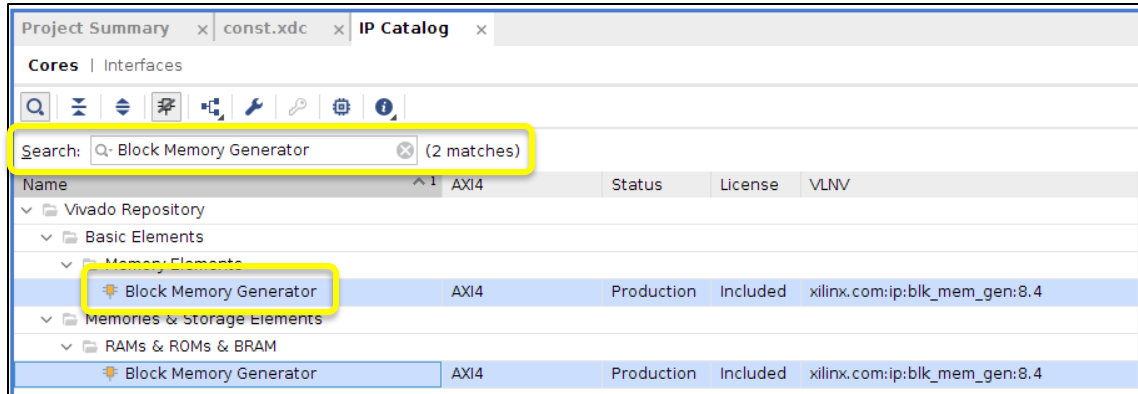
- Xilinx FPGAs provide a multiplier core.
  - *Constant Coefficient Multiplier* example
    - In *Basic* tab you can set the constant integer (second operand) and memory type to implement constant multiplier
    - In *Output and Control* tab, you can adjust the number of pipeline stages (optimum pipeline 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*

Port A is used for WRITE

Port B is used for READ

Block Memory Generator (8.4)

Component Name: bram\_unit

Interface Type: Native

Memory Type: Simple Dual Port RAM

ECC Options: No ECC

Write Enable: Byte Write Enable (unchecked), Byte Size (bits): 9

Algorithm Options: Algorithm: Minimum Area, Primitive: 8kx2

BRAM\_PORTA: addra[3:0], clka, dina[15:0], ena, wea[0:0]

BRAM\_PORTB: addrb[3:0], clkb, doutb[15:0], enb

OK Cancel

# 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)

Component Name: bram\_unit

Basic | **Port A Options** | Port B Options | Other Options | Summary

**Memory Size**

Port A Width: 32 (Range: 1 to 4608 (bits))

Port A Depth: 256 (Range: 2 to 1048576)

The Width and Depth values are used for Write Operations in Port A

Operating Mode: No Change | Enable Port Type: Use ENA Pin

**Port A Optional Output Registers**

Primitives Output Register  Core Output Register

SoftECC Input Register  REGCEA Pin

**Port A Output Reset Options**

RSTA Pin (set/reset pin) Output Reset Value (Hex): 0

Reset Memory Latch Reset Priority: CE (Latch or Register Enable)

**READ Address Change A**

Read Address Change A

Component Name: bram\_unit

Basic | Port A Options | **Port B Options** | Other Options | Summary

**Memory Size**

Port B Width: 32

Port B Depth: 256

The Width and Depth values are used for Read Operation in Port B

Operating Mode: Read First | Enable Port Type: Use ENB Pin

**Port B Optional Output Registers**

Primitives Output Register  Core Output Register

SoftECC Output Register  REGCEB Pin

**Port B Output Reset Options**

RSTB Pin (set/reset pin) Output Reset Value (Hex): 0

Reset Memory Latch Reset Priority: CE (Latch or Register Enable)

**READ Address Change B**

Read Address Change B

# 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.

