



# Using Vitis HLS

# Objectives

- ▶ After completing this module, you will be able to:
  - List various OS under which Vitis HLS is supported
  - Describe how projects are created and maintained in Vitis HLS
  - State various steps involved in using Vitis HLS project creation wizard
  - Distinguish between the role of top-level module in test bench and design to be synthesized
  - List various verifications which can be done in Vitis HLS
  - List Vitis HLS project directory structure

# Outline

- ▶ Invoking Vitis HLS
- ▶ Project Creation using Vitis HLS
- ▶ Synthesis to with Vivado IP Flow
- ▶ Design Analysis
- ▶ Other Ways to use Vitis HLS
- ▶ Summary

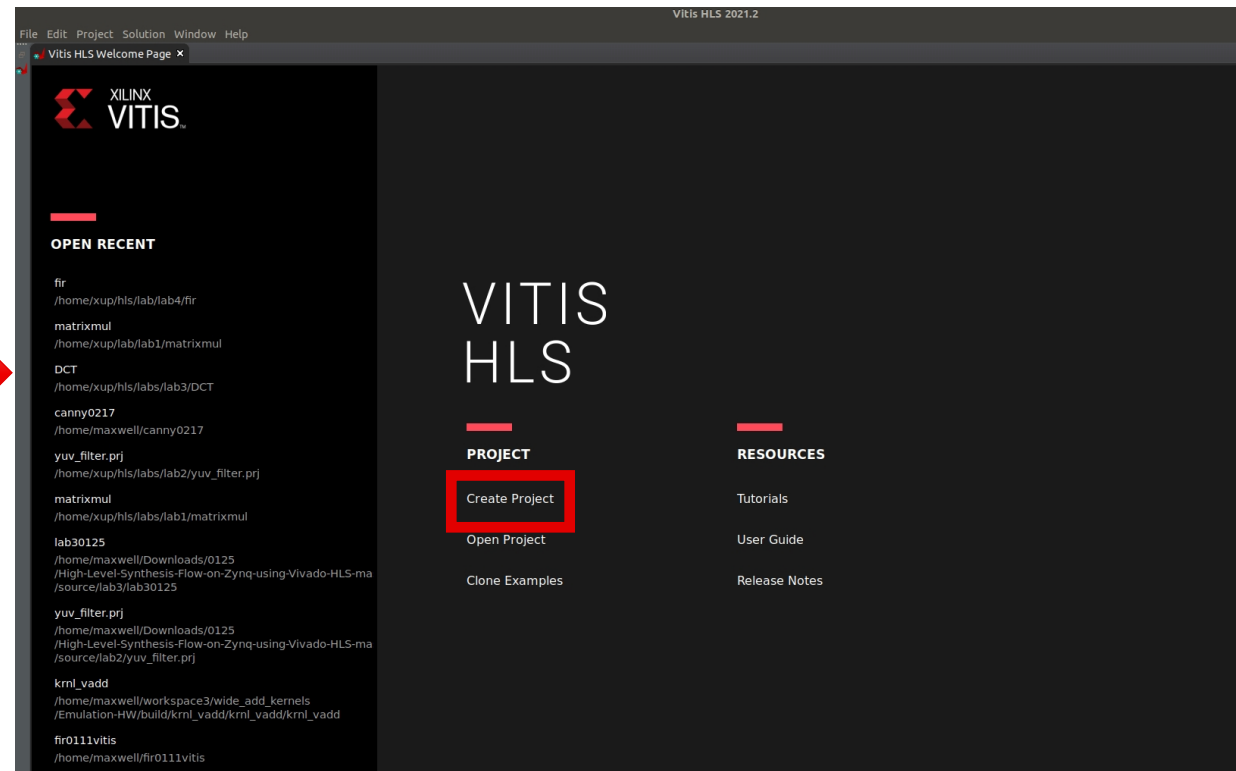
# Invoke Vitis HLS from the terminal

```
maxwell@maxwell-OptiPlex-7080:/home$ source /tools/Xilinx/Vitis/2021.2/settings64.sh
maxwell@maxwell-OptiPlex-7080:/home$ vitis_hls

***** Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2021.2 (64-bit)
**** SW Build 3367213 on Tue Oct 19 02:47:39 MDT 2021
**** IP Build 3369179 on Thu Oct 21 08:25:16 MDT 2021
** Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.

source /tools/Xilinx/Vitis_HLS/2021.2/scripts/vitis_hls/hls.tcl -notrace
INFO: Applying HLS Y2K22 patch v1.2 for IP revision
INFO: [HLS 200-10] Running '/tools/Xilinx/Vitis_HLS/2021.2/bin/unwrapped/lx64.o/vitis_hls'
INFO: [HLS 200-10] For user 'maxwell' on host 'maxwell-OptiPlex-7080' (Linux_x86_64 version 5.4.0-96-generic) on Sat Mar 05 16:51:11 CST 2022
INFO: [HLS 200-10] On os Ubuntu 18.04.2 LTS (beaver-osp1-ygritte X40)
INFO: [HLS 200-10] In directory '/home'
INFO: [HLS 200-10] Bringing up Vitis HLS GUI ...
```

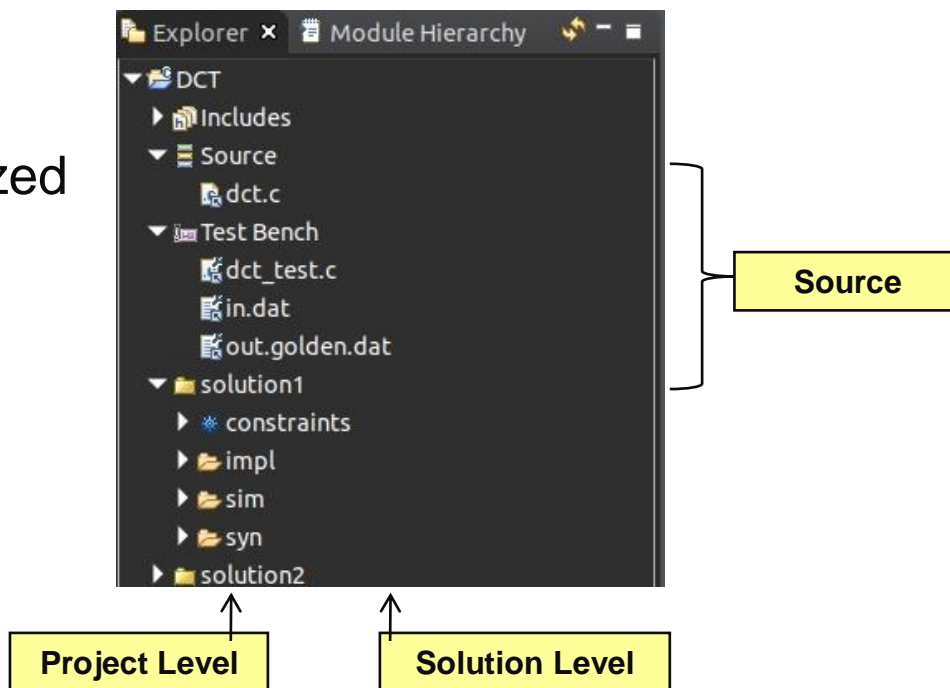
The first step is to open or create a project



# Project Creation Using Vitis HLS

# Vitis HLS Projects and Solutions

- ▶ Vitis HLS is project based
  - A project specifies the source code which will be synthesized
  - Each project is based on one set of source code
  - Each project has a user specified name
- ▶ A project can contain multiple solutions
  - Solutions are different implementations of the same code
  - Auto-named solution1, solution2, etc.
  - Supports user specified names
  - Solutions can have different clock frequencies, target technologies, synthesis directives
- ▶ Projects and solutions are stored in a hierarchical directory structure
  - Top-level is the project directory
  - The disk directory structure is identical to the structure shown in the GUI project explorer (except for source code location)



# Compare the Vivado IP flow with the Vitis kernel Flow

Project based and can contain multiple variations called solutions to drive synthesis and simulation

## Vivado IP Flow

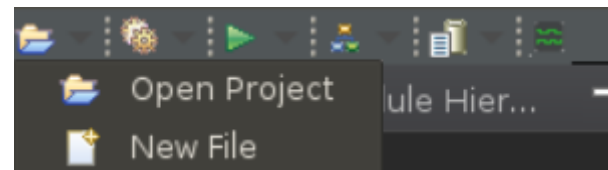
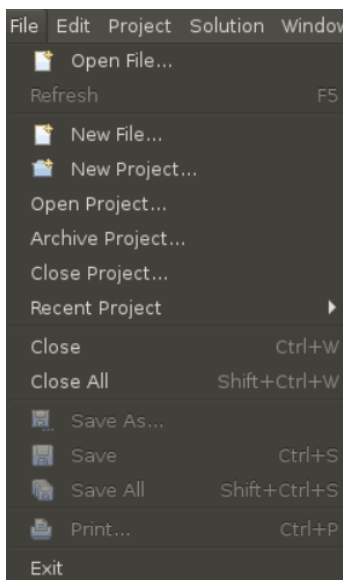
- Configuring Vitis HLS tool to generate RTL IP files for:
  - Use in the Vivado Design Suite
  - Inclusion in the IP catalog
  - Use in block designs of the IP integrator tool
- HLS synthesis transforms your C or C++ code into register transfer level (RTL) code
- More flexible and less structured than Vitis kernel flow
- Supports a wide variety of interface specifications and data transfer protocols
- Does not naturally support the Xilinx runtime (XRT) requirements of the Vitis system
- Provides much greater discretion in your design choices
- Leaves the integration and management of IP to user

## Vitis Kernel Flow

- Configuring Vitis HLS tool to generate the compiled kernel object (.xo) for the Vitis application acceleration flow
- More restrictive than the Vivado IP flow
- Kernels produced by the HLS tool must meet the specific requirements of the platforms and Xilinx runtime (XRT)
- After enabling the Vitis kernel flow in the IDE, the Vitis HLS tool implements interface ports using the AXI standard

# Vitis HLS Step 1: Create or Open a project

- ▶ Start a new project
  - The GUI will start the project wizard to guide you through all the steps



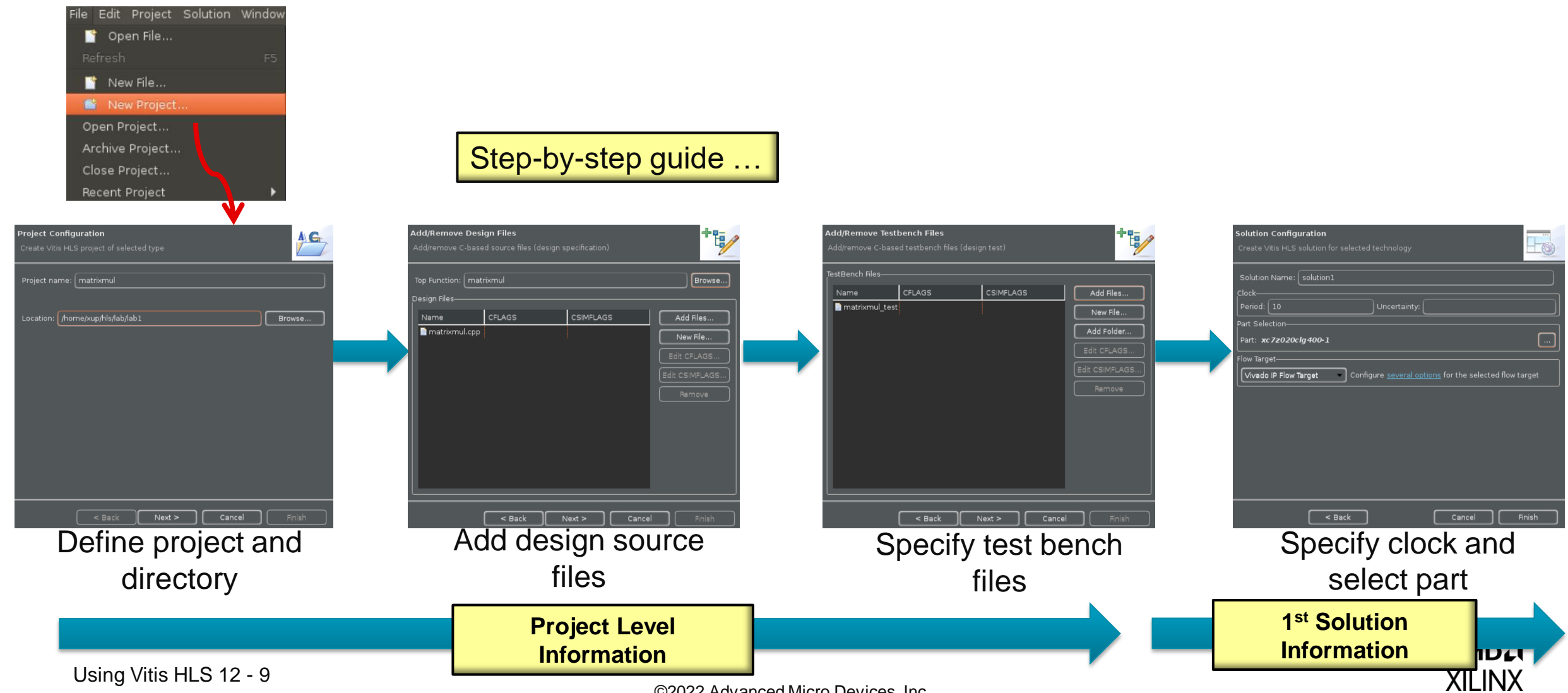
Optionally use the Toolbar Button to  
Open Project

- ▶ Open an existing project
  - All results, reports and directives are automatically saved/remembered
  - Use "Recent Project" menu for quick access



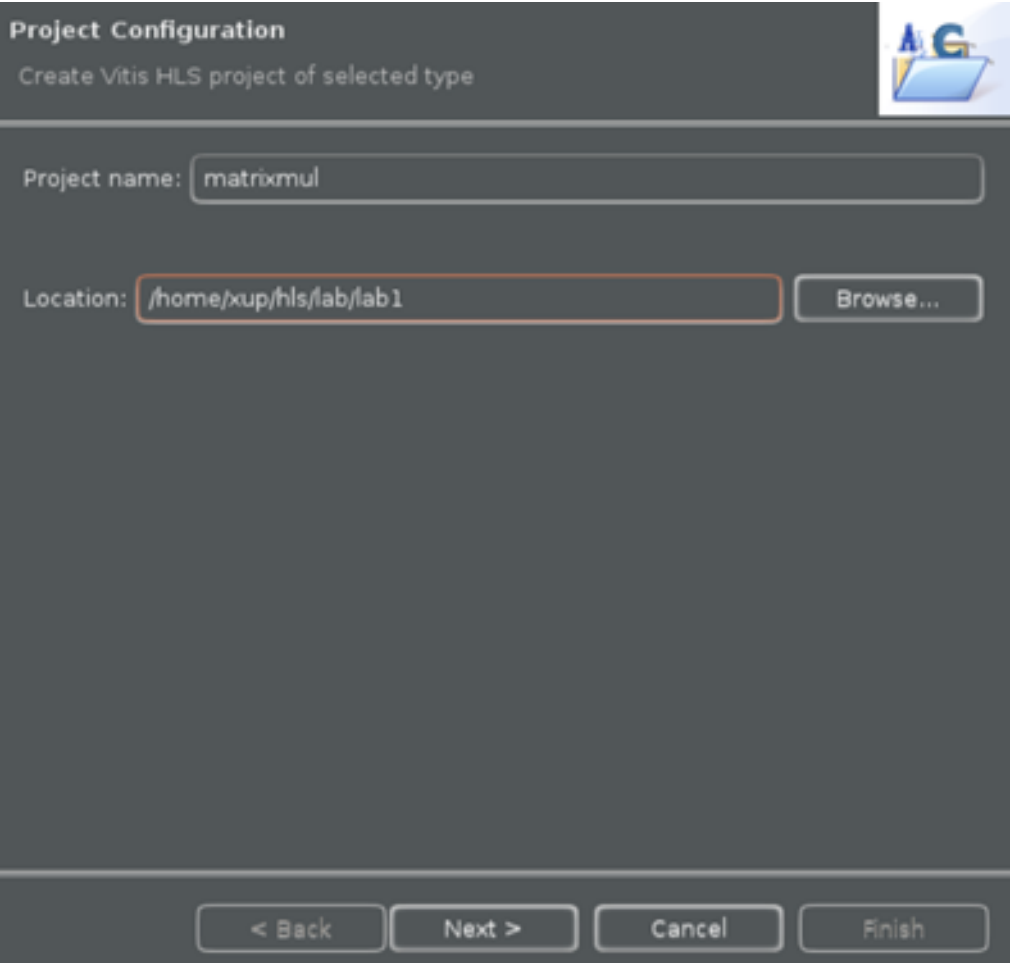
# Project Wizard

- ▶ The Project Wizard guides users through the steps of opening a new project



# Define Project & Directory

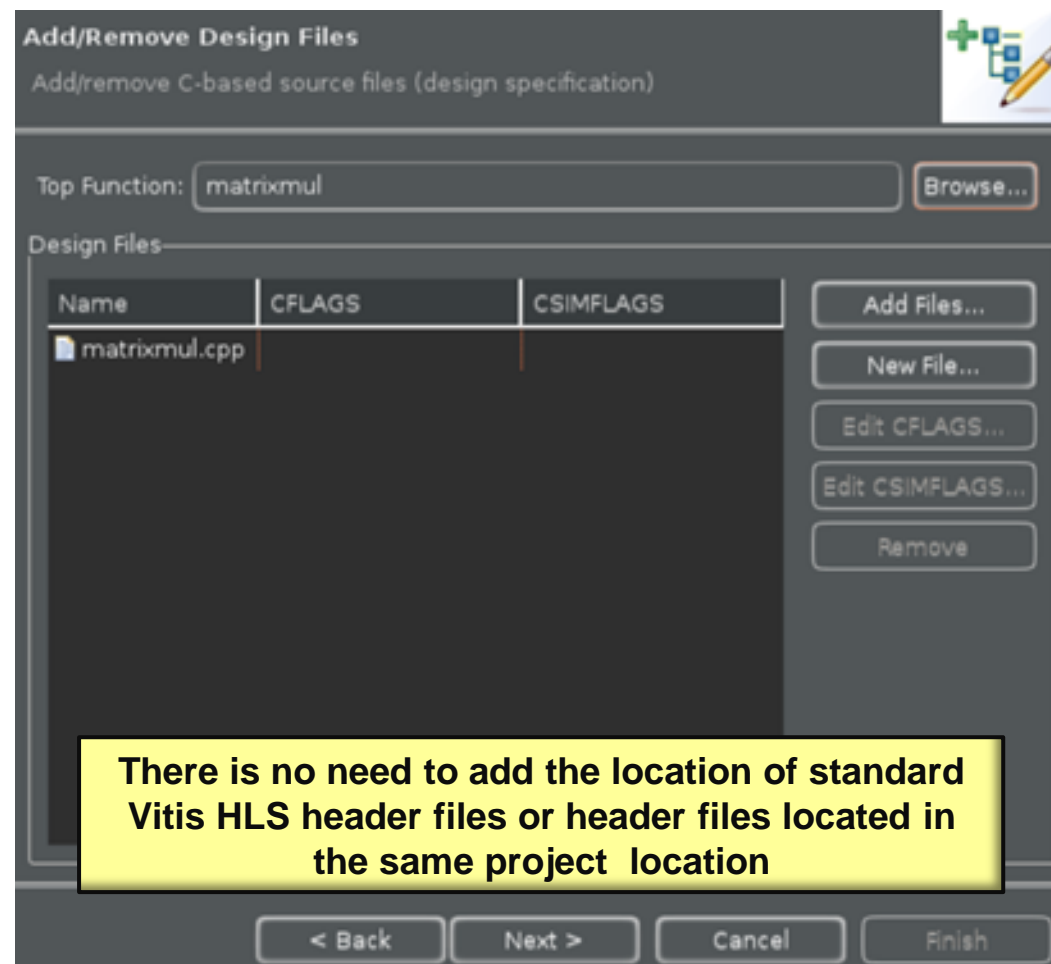
- ▶ Define the project name
- ▶ Browse to the location of the project
  - In this example, project directory “matrixmul” will be created inside directory “lab1”



The screenshot shows the 'Project Configuration' dialog box in Vitis. The title bar says 'Project Configuration' and the subtitle is 'Create Vitis HLS project of selected type'. There is a small icon of a folder with a blue 'G' in the top right corner. The dialog has two main input fields: 'Project name:' with the text 'matrixmul' entered, and 'Location:' with the text '/home/xup/hls/lab/lab1' entered. To the right of the 'Location:' field is a 'Browse...' button. At the bottom of the dialog are four buttons: '< Back', 'Next >', 'Cancel', and 'Finish'.

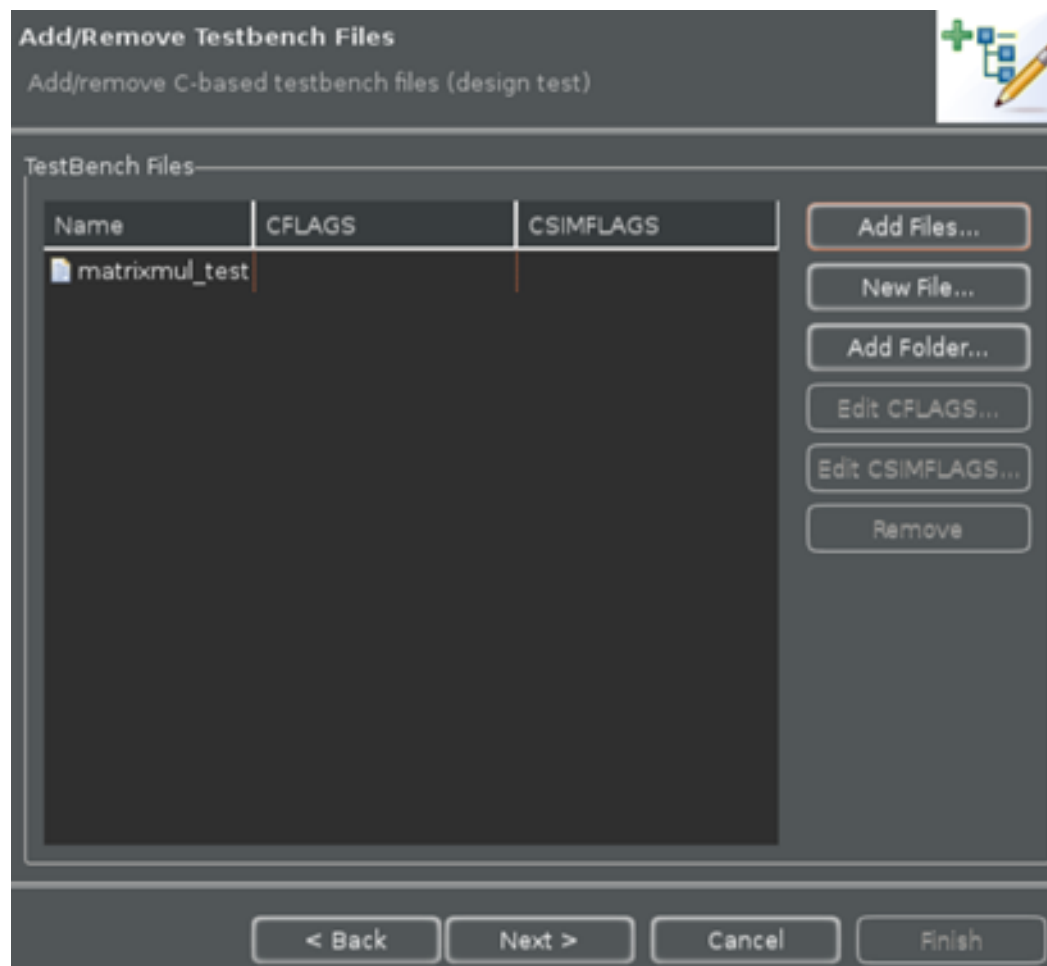
# Add Design Source Files

- ▶ Add Design Source Files
  - This allows Vitis HLS to determine the top-level design for synthesis, from the test bench and associated files
- ▶ Add Files...
  - Select the source code file(s)
  - The CTRL and SHIFT keys can be used to add multiple files
  - No need to include headers (.h) if they reside in the same directory
- ▶ Select File and Edit CFLAGS...
  - If required, specify C compile arguments using the “Edit CFLAGS...”
    - Define macros: `-DVERSION1`
    - Location of any (header) files not in the same directory as the source: `-I../include`



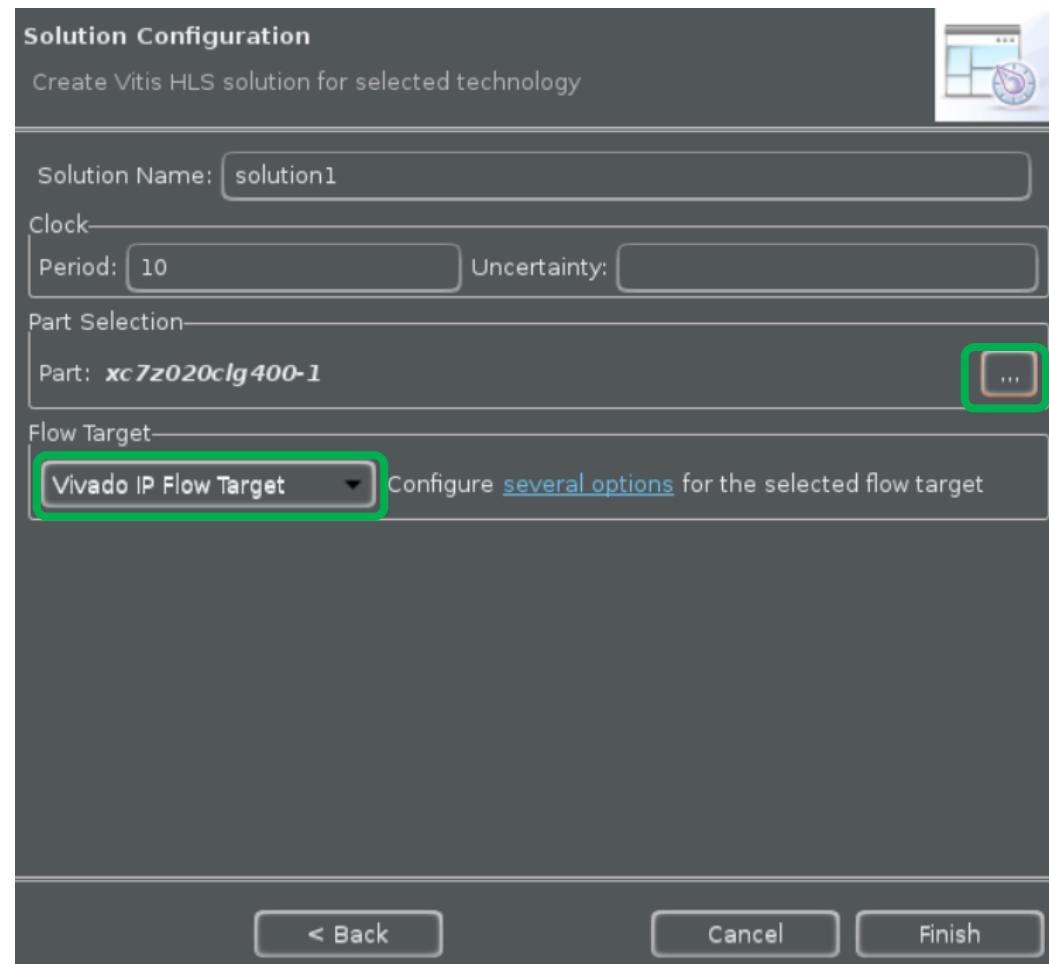
# Specify Test Bench Files

- ▶ Use “Add Files” to include the test bench
  - Vitis HLS will re-use these to verify the RTL using co-simulation
- ▶ And all files referenced by the test bench
  - The RTL simulation will be executed in a different directory (Ensures the original results are not overwritten)
  - Vitis HLS needs to also copy any files accessed by the test bench
    - E.g. Input data and output results
- ▶ Add Folders
  - If the test bench uses relative paths like “sub\_directory/my\_file.dat” you can add “sub\_directory” as a folder/directory
- ▶ Use “Edit CFLAGS...”
  - To add any C compile flags required for compilation



# Solution Configuration

- ▶ Provide a solution name
  - Default is solution1, then solution2 etc.
- ▶ Specify the clock
  - The clock uncertainty is subtracted from the clock to provide an “effective clock period”
  - Vitis HLS uses the “effective clock period” for Synthesis
  - Provides users defined margin for downstream RTL synthesis, P&R
- ▶ Select the part
  - Select a device family after applying filters such as family, package and speed grade (see next slide) or a board after applying boards specify
- ▶ Select the Target flow
  - Each solution can target either the Vivado IP flow, or the Vitis Kernel flow.



The screenshot shows the 'Solution Configuration' dialog box in Vitis. The title bar says 'Solution Configuration' and the subtitle is 'Create Vitis HLS solution for selected technology'. The dialog has several sections: 'Solution Name' with a text box containing 'solution1'; 'Clock' with 'Period' set to '10' and an empty 'Uncertainty' box; 'Part Selection' with 'Part' set to 'xc7z020clg400-1' and a green box around the selection icon; and 'Flow Target' with a dropdown menu set to 'Vivado IP Flow Target' (also highlighted with a green box) and a link to 'Configure several options for the selected flow target'. At the bottom are three buttons: '< Back', 'Cancel', and 'Finish'.

# Selecting Part and Implementation Engine

- ▶ Select the target part either through Parts or Boards specify

Select: 

Parts

Boards

Filter

Product Category: 

General Purpose

Family: 

zynq

Sub-Family: 

Zynq-7000

Package: 

clg400

Speed grade: 

-1

Temp grade: 

All

Reset All Filters

Search:

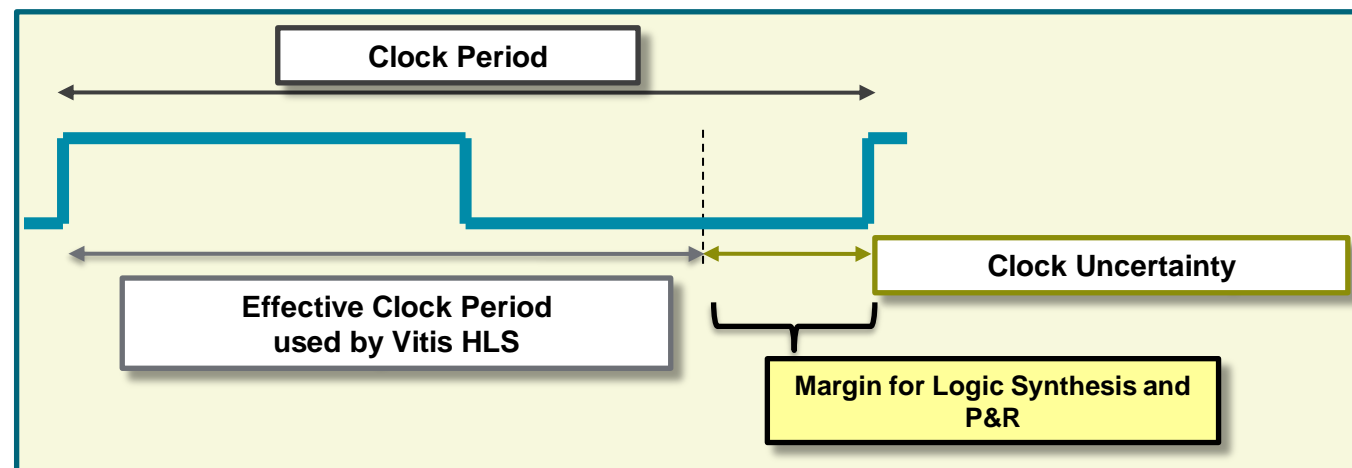
Part	Family	Package	Speed	SLICE	LUT	FF
<div>xc7z020clg400-1</div>	Zynq-7000	clg400	-1	13300	53200	106
<div>xc7z014sclg400-1</div>	Zynq-7000	clg400	-1	13300	40600	812
<div>xc7z010clg400-1</div>	Zynq-7000	clg400	-1	4400	17600	352
<div>xc7z007sclg400-1</div>	Zynq-7000	clg400	-1	4400	14400	288

Cancel

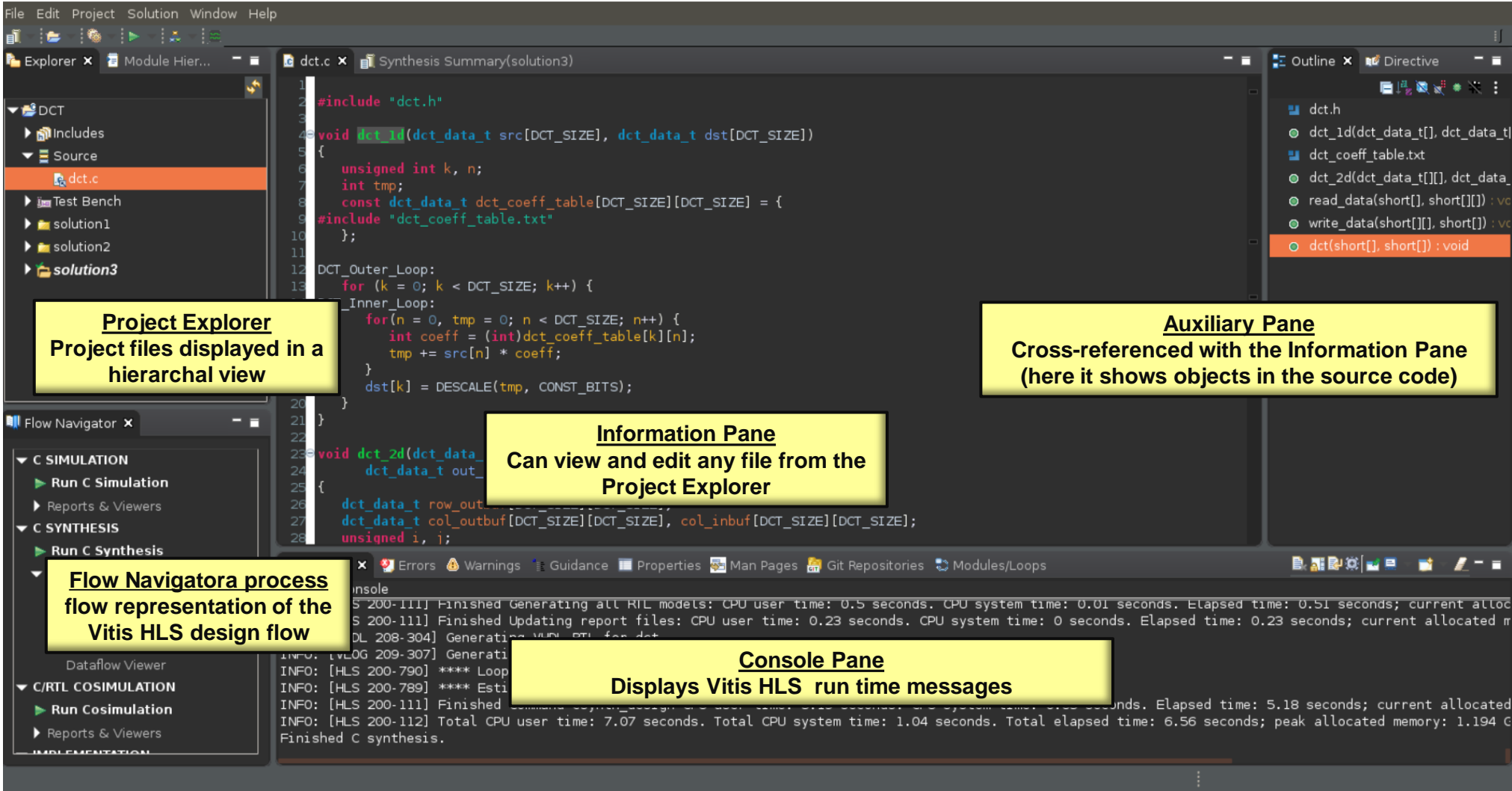
OK

# Clock Specification

- ▶ Clock frequency must be specified
  - Only 1 clock can be specified for C/C++ functions
- ▶ Clock uncertainty can be specified
  - Subtracted from the clock period to give an effective clock period
  - The effective clock period is used for synthesis
    - Should not be used as a design parameter
    - Do not vary for different results: this is your safety margin
  - A user controllable margin to account for downstream RTL synthesis and P&R

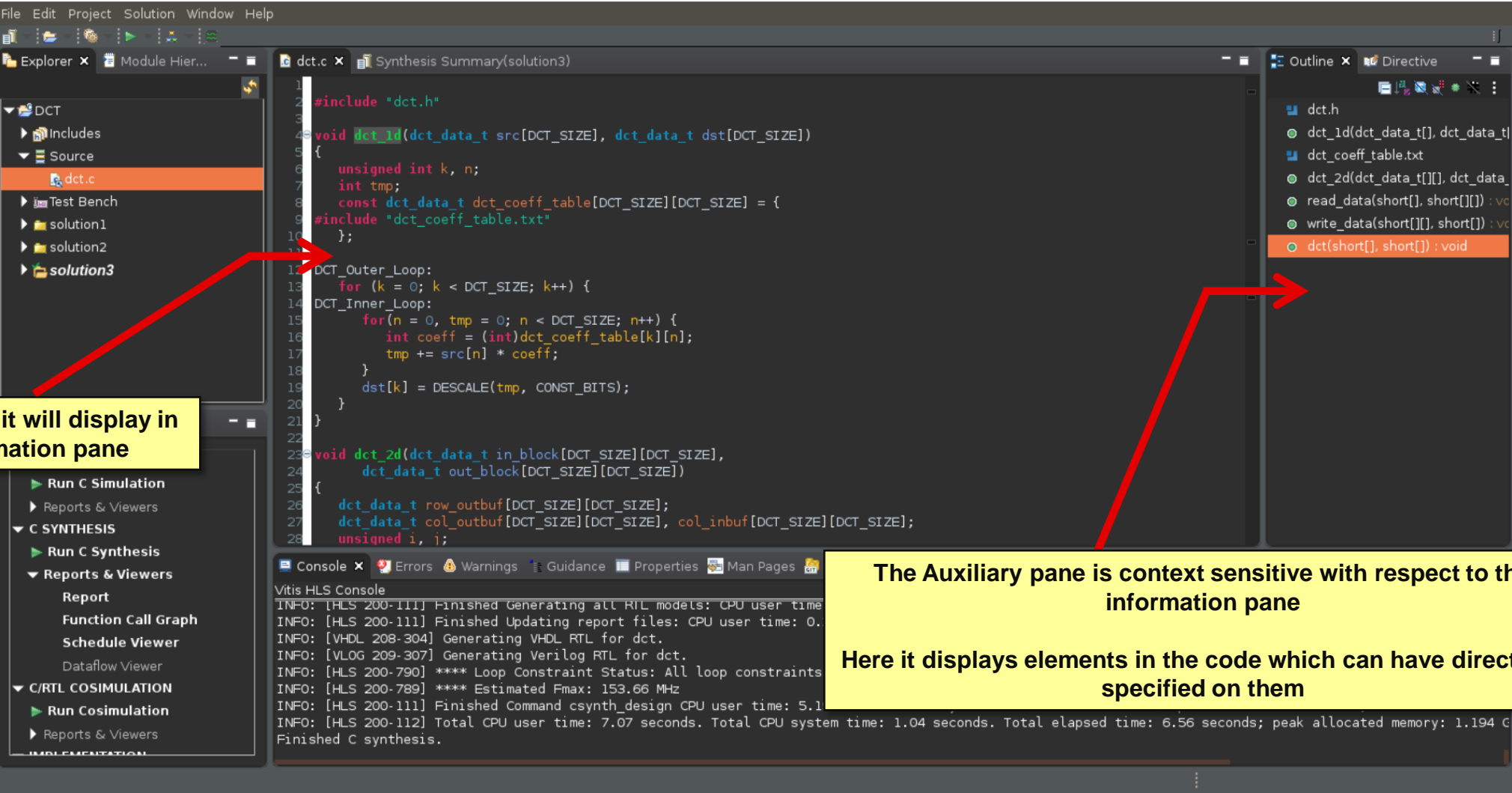


# A Vitis HLS Project



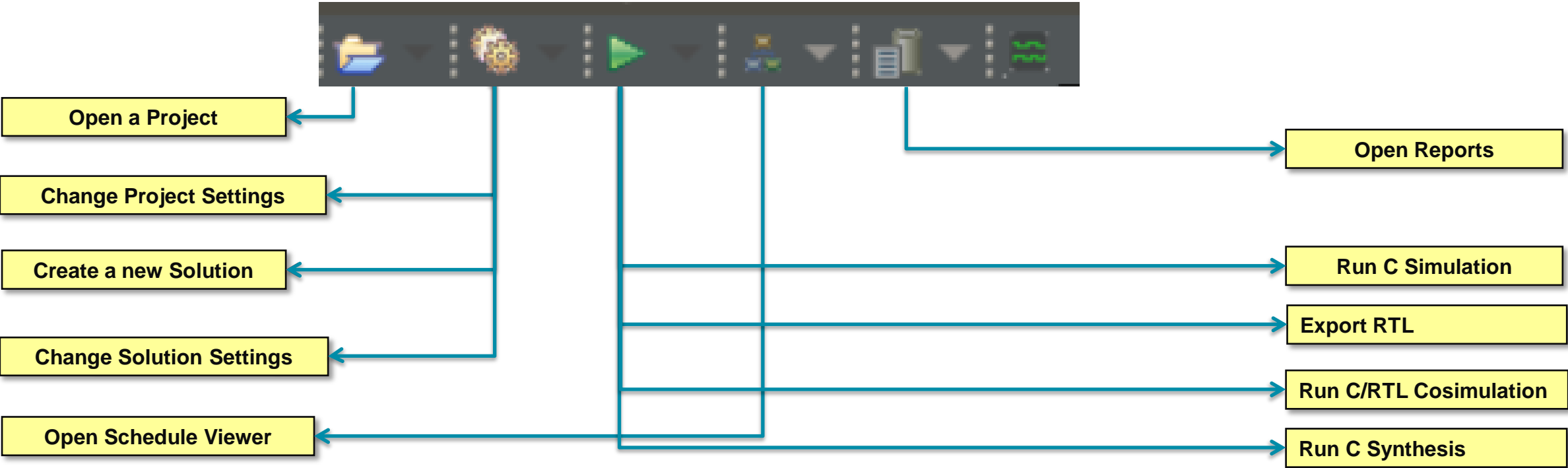


# Files: Views, Edits & Information



# Vitis HLS GUI Toolbar

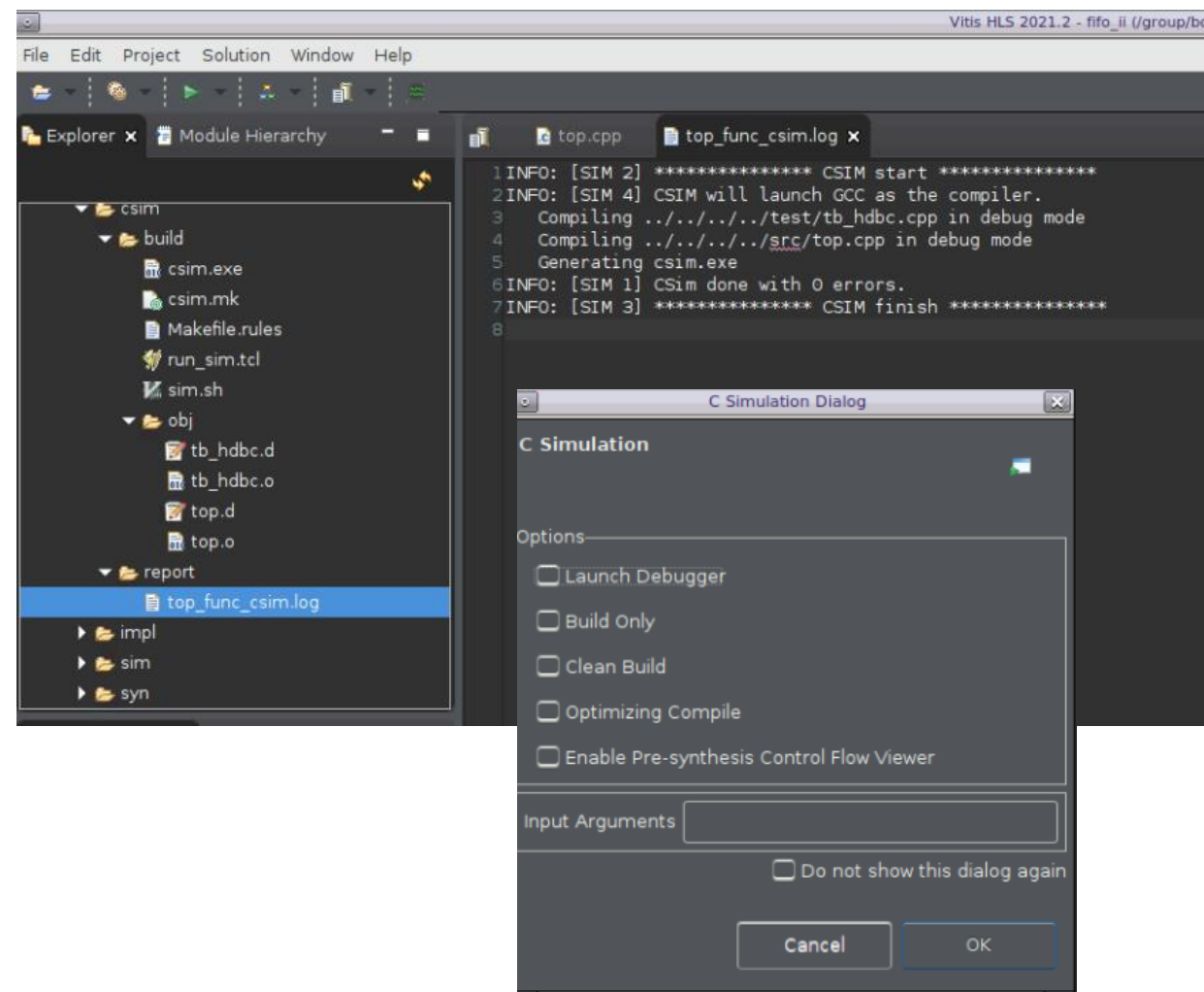
- ▶ The primary commands have toolbar buttons
  - Easy access for standard tasks
  - Button highlights when the option is available
    - E.g. cannot perform C/RTL simulation before synthesis



# Vitis HLS Design Analysis with Vivado IP Flow

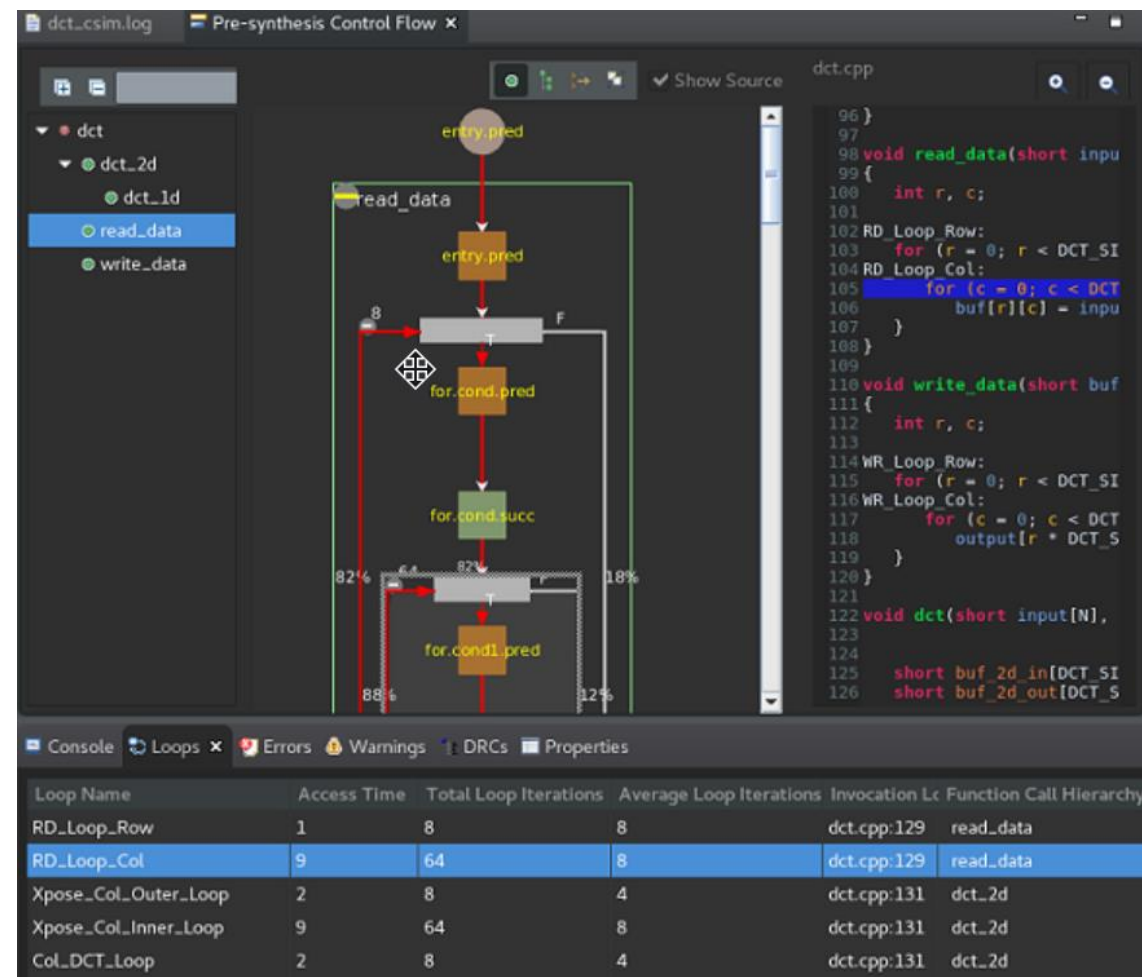
# C Simulation

- ▶ Run C Simulation
- ▶ Console
  - Will show run time information
  - Examine for failed constraints
- ▶ A “CSIM” directory is created
  - Build folder with csim.exe and object files
  - Reports for CSIM results
- ▶ Report is outlined in the Auxiliary pane
- ▶ Pre-synthesis viewer will show outlines of the function control flow



# Pre-Synthesis Control Flow Viewer

- ▶ As shown in the figure above, the Pre-Synthesis Control Flow viewer has multiple elements
  - Function Call Tree on the upper left.
  - Control Flow Graph (CFG) in the middle.
  - Source Code viewer on the upper right.
  - Loops view in the lower Console area that is associated with, provides cross-probing with the CFG viewer.
  - identify the hot spots in your function, the compute-intensive control structures
- ▶ The Pre-Synthesis Control Flow viewer helps you
  - apply pragmas or directives to improve or optimize the results.
  - shows the control flow through your C code
  - visualize the top-level function
  - provides static profiling, such as the trip-count of loops, and dynamic profiling information to analyze the design.



# Synthesis

- ▶ Run C Synthesis
- ▶ Console
  - Will show run time information
  - Examine for failed constraints
- ▶ A “syn” directory is created
  - Verilog & VHDL RTL
  - Synthesis reports for all non-inlined functions
- ▶ Report opens automatically
  - When synthesis completes
- ▶ Report is outlined in the Auxiliary pane

The screenshot displays the Vivado IDE interface. The left pane shows the Project Explorer with the 'matrixmul' project selected. The right pane shows the 'Synthesis Details Report for 'matrixmul''. The report includes the following sections:

**General Information**

- Date: Sun Feb 13 22:39:57 2022
- Version: 2021.2 (Build 3367213 on Tue Oct 19 02:47:39 MDT 2021)
- Project: matrixmul
- Solution: solution1 (Vivado IP Flow Target)
- Product family: zynq
- Target device: xc7z020-clg400-1

**Performance Estimates**

**Timing**

**Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.270 ns	2.70 ns

**Latency**

**Summary**

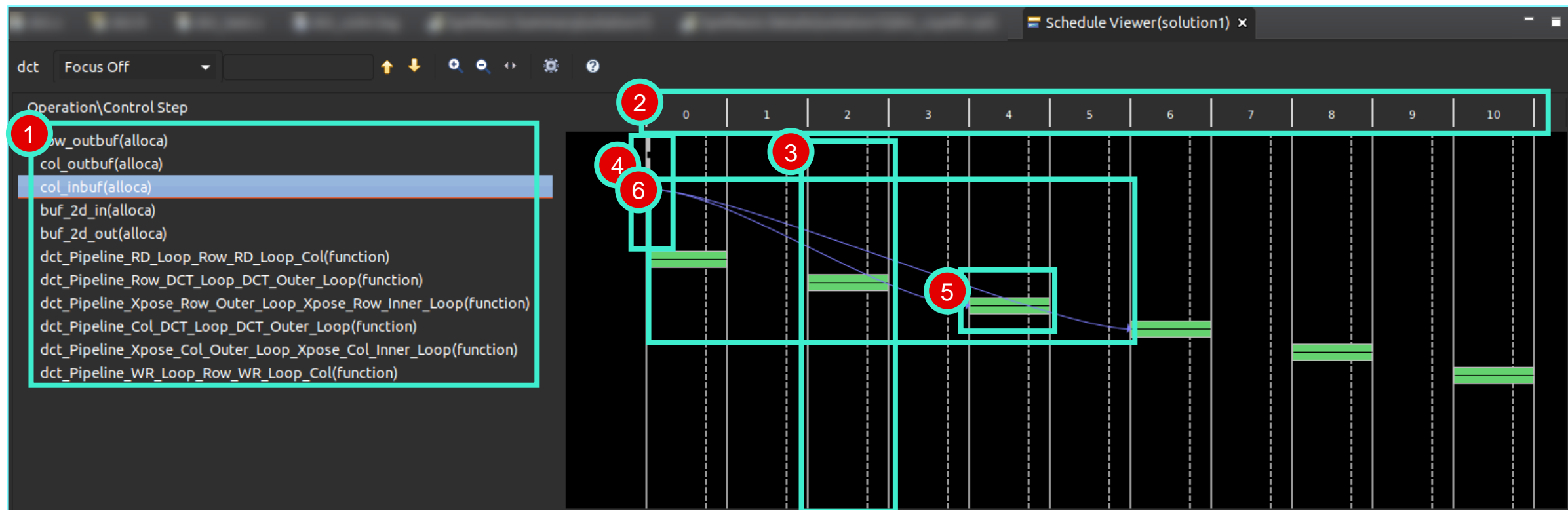
Latency (cycles)	Latency (absolute)	Interval (cycles)

The bottom pane shows the Console with 19 Guidance-Infos, 1 Guidance-Warnings, and 0 Guidance-Errors. A warning is displayed:

[HLS 200-885] The II Violation in module 'matrixmul' (loop 'Row\_Col'): Unable to schedule 'load' operation ('b\_load', ms (II = 1). Please consider using a memory core with more ports or partitioning the array 'h'.

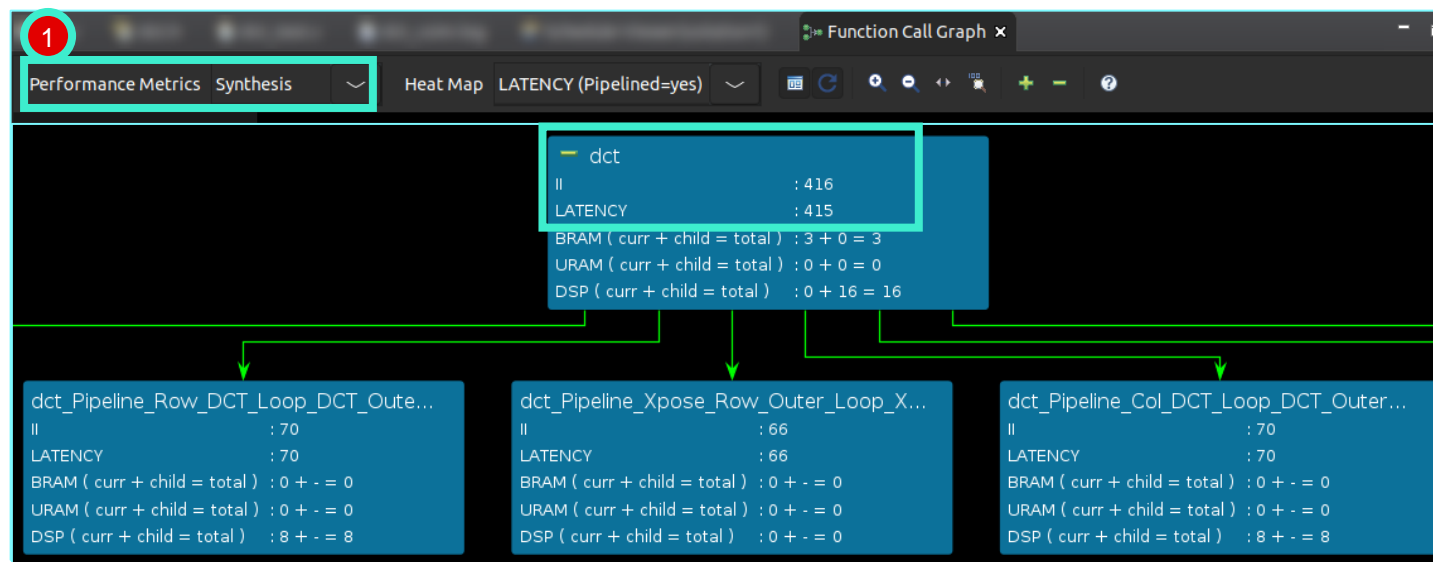
# Schedule Viewer

- ▶ Provides a detailed view of the synthesized RTL and the clock cycle that it executes in
- ▶ Helps identify any loop dependencies preventing parallelism, timing violations, and data dependencies



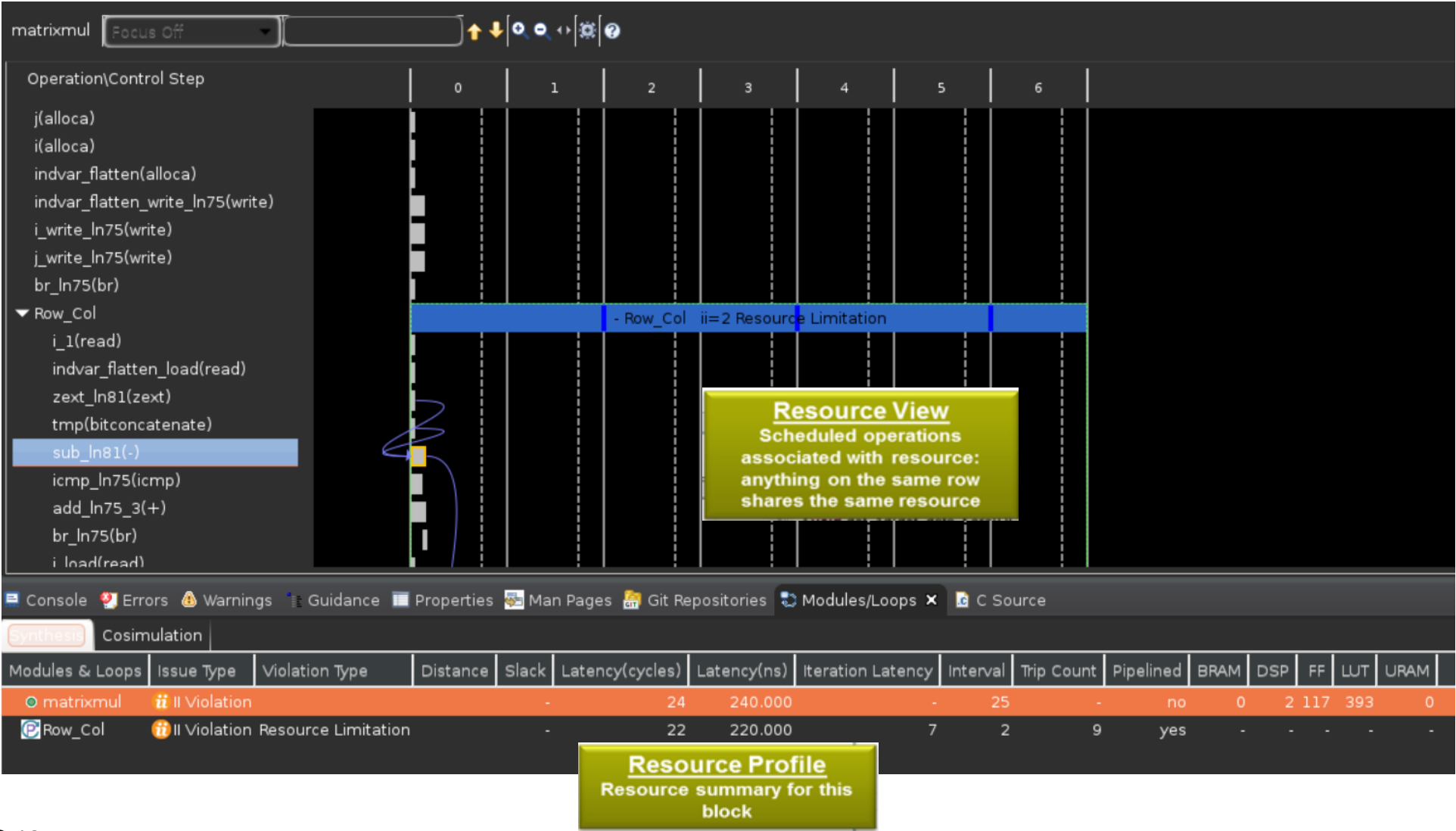
# Function Call Graph Viewer

- Illustrates full design after C synthesis or C/RTL co-simulation
- Goal: Show the throughput of the design in terms of latency and II
- Helps identify the critical path and bottlenecks in the design
- Shows the paths through the design where throughput may be imbalanced—leading to FIFO stalls and/or deadlock.
- Functions in-lined will no longer be visible





# Resources Analysis



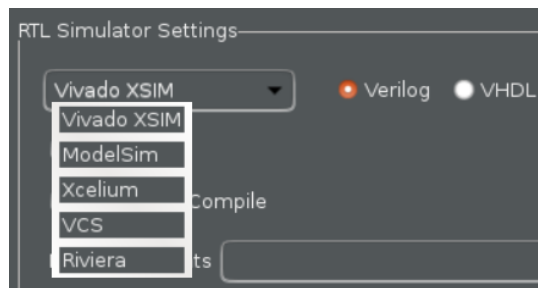
# C/RTL Co-simulation

## ▶ Start Simulation

- Opens the dialog box

## ▶ Select the RTL

- Verilog and VHDL require the appropriate simulator
  - Select the desired simulator

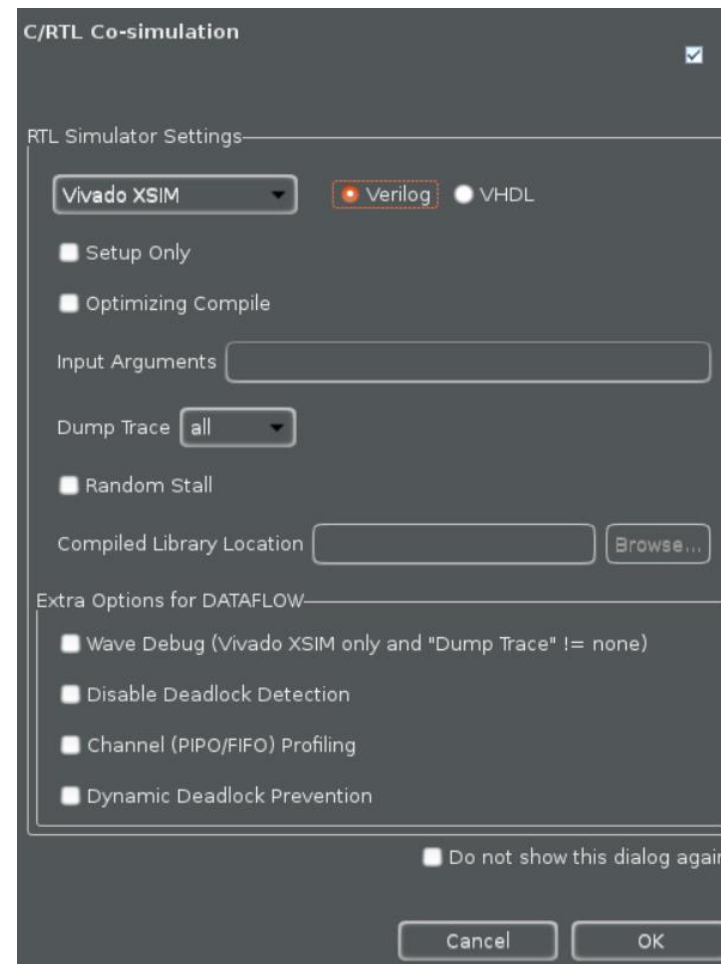


## ▶ Options

- Can output trace file (VCD format)
- Optimize the C compilation & specify test bench linker flags
- The “setup only” option will not execute the simulation
- Enables waveform visualization of all processes in the RTL simulation.

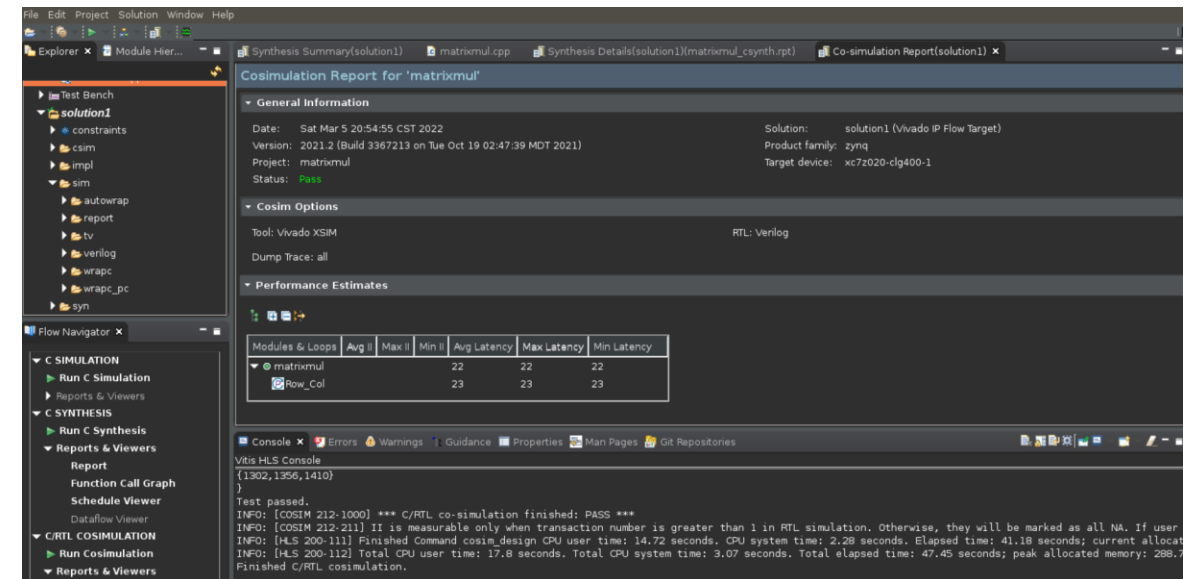
## ▶ OK will run the simulator

- Output files will be created in a “sim” directory



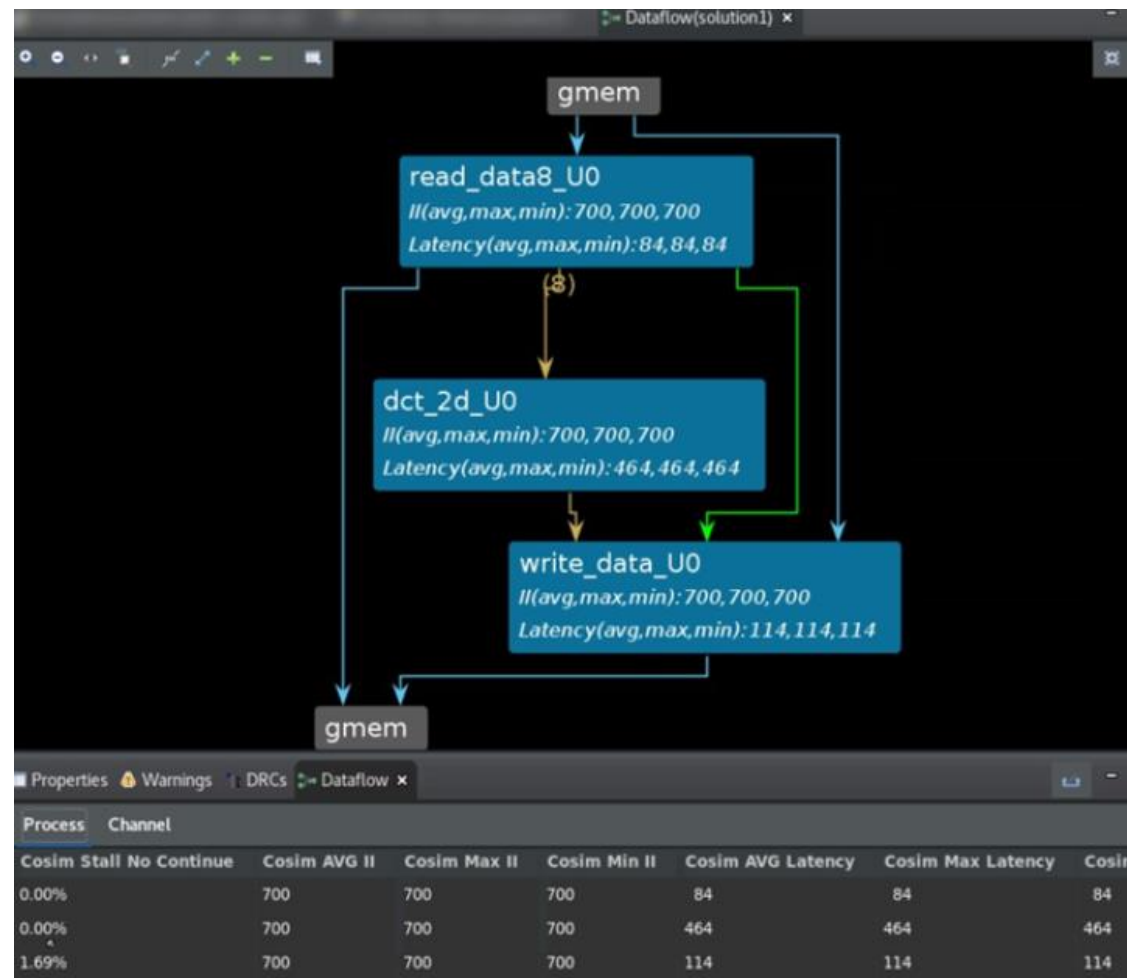
# Simulation Results

- ▶ Simulation output is shown in the console
- ▶ Expect the same test bench response
  - If the C test bench plots, it will with the RTL design (but slower)
- ▶ Sim Directory
  - Will contain a sub-directory for each RTL which is verified
- ▶ Report
  - A report is created and opened automatically



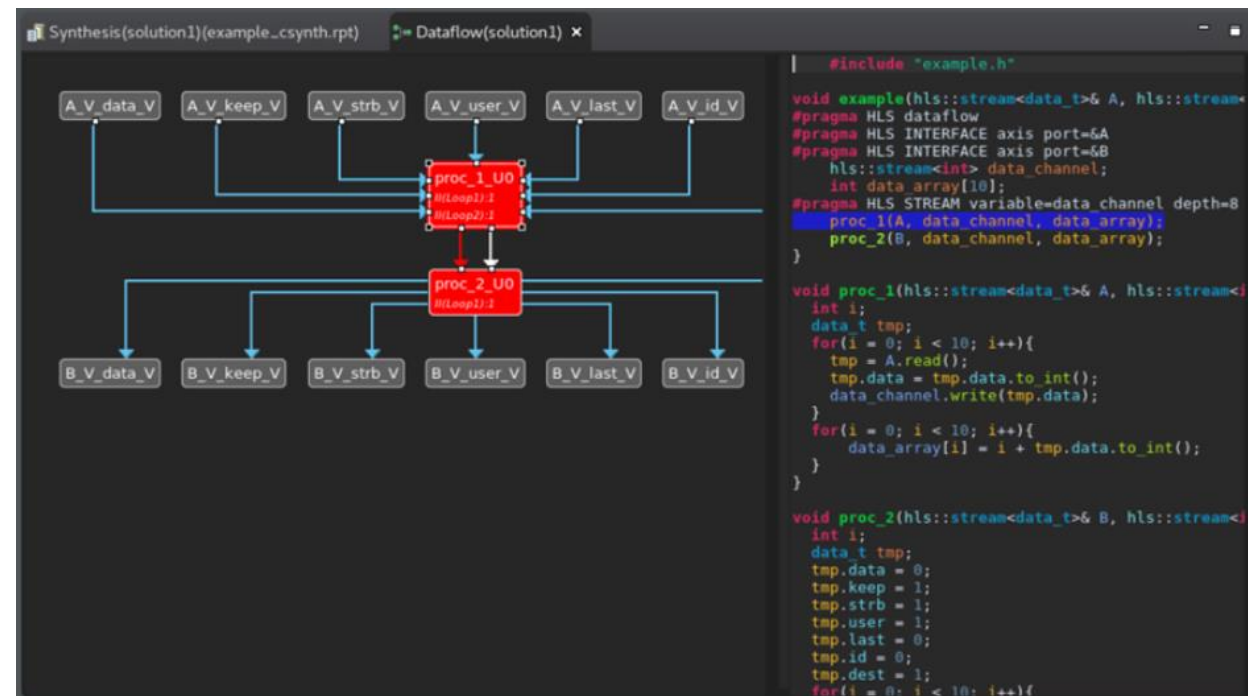
# Dataflow Viewer

- ▶ Allows you to see the dataflow structure inferred by the tool
- ▶ Inspect the channels (FIFO/PIPO)
- ▶ Examine the effect of channel depth on performance
- ▶ Performance data is back-annotated to the dataflow viewer from the co-simulation results
- ▶ Displays a representation of the dataflow graph structure, showing different processes and underlying connections
- ▶ Helps with performance debugging of the design
- ▶ Highlights the channels and processes involved in the design deadlock during RTL co-simulation
- ▶ DATAFLOW pragma/directive must be applied to your design for the Dataflow viewer to be populated



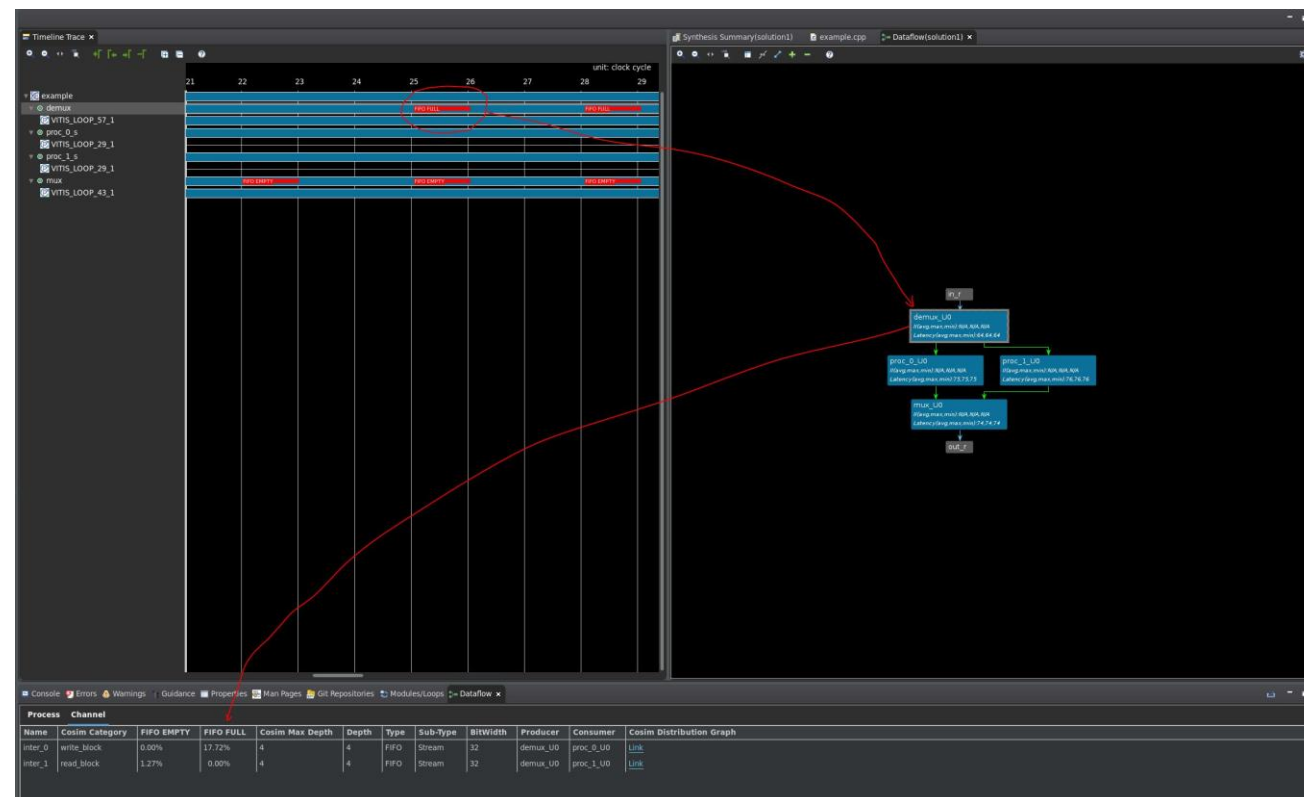
# Deadlock Viewer

- ▶ Automatically opens only after, the co-simulation detects the deadlock situation and the co-sim run has finished
- ▶ Visualizes this deadlock scenario on the static dataflow viewer
- ▶ Highlights the problematic processes in red
- ▶ Provides a cross-probing capability to link between the problematic dataflow channels and the associated source code
- ▶ Use the information in solving the issue with less time and effort



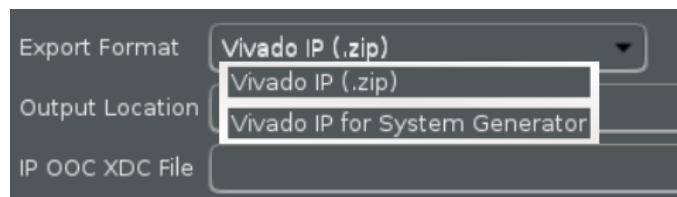
# Timeline Trace Viewer

- ▶ Timeline Trace Viewer
- ▶ Shows the run time profile of the functions of the design
- ▶ Useful to see the behavior of dataflow regions after co-simulation
- ▶ Displays multiple iterations through the various sub-functions of a dataflow region
- ▶ Shows where the functions are starting and ending
- ▶ Displays the co-simulation data in tables below the timeline

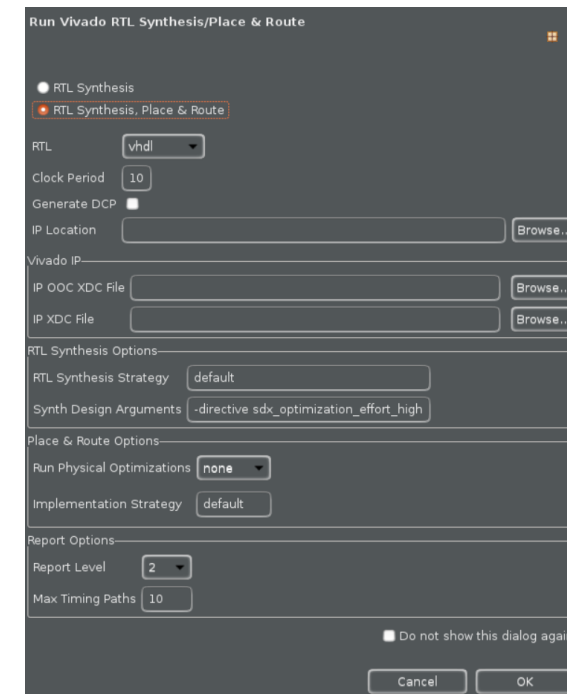
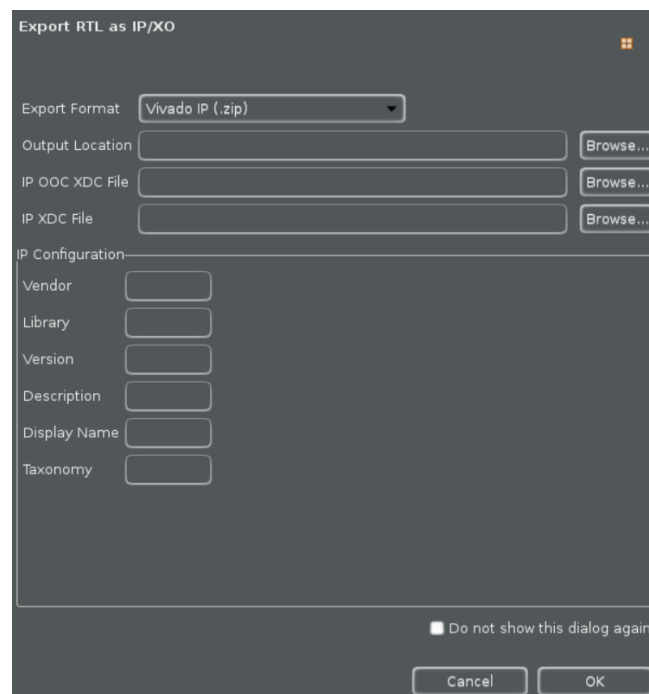


# RTL Export for Implementation

- ▶ Click on Export RTL
  - Export RTL Dialog opens
- ▶ Select the desired output format



- ▶ Optionally, configure the output
- ▶ Select the desired language
- ▶ Optionally, click on implementation and choose RTL Synthesis, Place & Route options for invoking implementation tools from within Vitis HLS
- ▶ Click OK to start the implementation





# RTL Export (Place and Route Option) Results

- ▶ Impl directory created
  - Will contain a sub-directory for each RTL which is synthesized
  - A report is created and opened automatically

Export Report for 'matrixmul'

General Information	
Report date:	Sat Mar 05 21:14:46 CST 2022
Project:	matrixmul
Solution:	solution1
Device target:	xc7z020-clg400-1
Implementation tool:	Xilinx Vivado v.2021.2

Run Constraints & Options	
Name	Value
▶ Design Constraints & Options	
▶ RTL Synthesis Options	
▶ Place & Route Options	
▶ Reporting Options	

Resource Usage	
	VHDL
SLICE	41
LUT	109
FF	53
DSP	2
BRAM	0
URAM	0
LATCH	0
SRL	0
CLB	0

Final Timing	
	VHDL
CP required	10.000
CP achieved post-synthesis	8.496
CP achieved post-implementation	9.360

Fail Fast Synth			
Created on Thu May 06 16:45:38 PDT 2021 with report_failfast (2020.12.07)			
Design Summary			
design_1			
xcvc1902-viva1596-1LP-e-S-es1			
Criteria	Guideline	Actual	Status
LUT	70%	7.45%	OK
FD	50%	3.99%	OK
LUTRAM+SRL	25%	1.49%	OK
LOOKAHEAD8	25%	0.07%	OK
DSP	80%	0.20%	OK
RAMB	80%	6.51%	OK
URAM	80%	0.00%	OK
DSP+RAMB+URAM (Avg)	70%	3.35%	OK
BUFGCE* + BUFGCTRL	24	0	OK
DONT_TOUCH (cells/nets)	0	0	OK
MARK_DEBUG (nets)	0	0	OK
Control Sets	16872	407	OK
Average Fanout for modules > 100k cells	4	3.56	OK
Non-FD high fanout nets > 10k loads	0	2	REVIEW
TIMING-6 (No common primary clock between related clocks)	0	0	OK
TIMING-7 (No common node between related clocks)	0	0	OK
TIMING-8 (No common period between related clocks)	0	0	OK
TIMING-14 (LUT on the clock tree)	0	0	OK
TIMING-35 (No common node in paths with the same clock)	0	0	OK
Number of paths above max LUT budgeting (0.449ns)	0	100	REVIEW
Number of paths above max Net budgeting (0.302ns)	0	100	REVIEW

Fail Fast Routed			
Created on Thu May 06 17:33:58 PDT 2021 with report_failfast (2020.12.07)			

Design  
Characteristics

Clocking  
Checks

LUT and Net  
Budgeting



# RTL Export Results (Place and Route Option Unchecked)

- ▶ Impl directory created
  - Will contain a sub-directory for both VHDL and Verilog along with the ip directory
- ▶ No report will be created
- ▶ Observe the console
  - No packing, routing phases

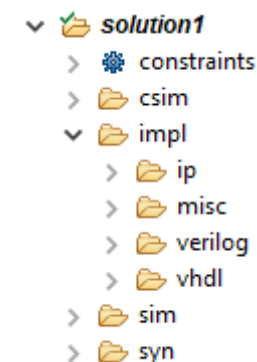
```

/tools/Xilinx/Vitis_HLS/2021.2/bin/vitis_hls /home/xup/hls/labs/lab1/matrixmul/solution1/export.tcl
INFO: Applying HLS Y2K22 patch v1.2 for IP revision
INFO: [HLS 200-10] Running '/tools/Xilinx/Vitis_HLS/2021.2/bin/unwrapped/lnx64.o/vitis_hls'
INFO: [HLS 200-10] For user 'maxwell' on host 'maxwell-OptiPlex-7080' (Linux_x86_64 version 5.4.0-96
INFO: [HLS 200-10] On os Ubuntu 18.04.2 LTS (beaver-ospl-ygritte X40)
INFO: [HLS 200-10] In directory '/home/xup/hls/labs/lab1'
Sourcing Tcl script '/home/xup/hls/labs/lab1/matrixmul/solution1/export.tcl'
INFO: [HLS 200-1510] Running: open_project matrixmul
INFO: [HLS 200-10] Opening project '/home/xup/hls/labs/lab1/matrixmul'.
INFO: [HLS 200-1510] Running: set_top matrixmul
INFO: [HLS 200-1510] Running: add_files matrixmul.cpp
INFO: [HLS 200-10] Adding design file 'matrixmul.cpp' to the project
INFO: [HLS 200-1510] Running: add_files -tb matrixmul_test.cpp -cflags -DHW_COSIM -Wno-unknown-pragm
INFO: [HLS 200-10] Adding test bench file 'matrixmul_test.cpp' to the project
INFO: [HLS 200-1510] Running: open_solution solution1 -flow_target vivado
INFO: [HLS 200-10] Opening solution '/home/xup/hls/labs/lab1/matrixmul/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-1611] Setting target device to 'xc7z020-clg400-1'
INFO: [HLS 200-1505] Using flow_target 'vivado'
Resolution: For help on HLS 200-1505 see www.xilinx.com/cgi-bin/docs/rdoc?v=2021.2;t=hls+guidance;d=
INFO: [HLS 200-1464] Running solution command: config_export -format=ip_catalog
INFO: [HLS 200-1464] Running solution command: config_export -rtl=vhdl
INFO: [HLS 200-1464] Running solution command: config_export -vivado_clock=10
INFO: [HLS 200-1510] Running: set_part xc7z020-clg400-1
INFO: [HLS 200-1510] Running: create_clock -period 10 -name default
INFO: [HLS 200-1510] Running: config_export -format ip_catalog -rtl vhdl -vivado_clock 10
INFO: [HLS 200-1510] Running: set_directive_top -name matrixmul matrixmul
INFO: [HLS 200-1510] Running: export_design -rtl vhdl -format ip_catalog
INFO: [IMPL 213-8] Exporting RTL as a Vivado IP.

***** Vivado v2021.2 (64-bit)
**** SW Build 3367213 on Tue Oct 19 02:47:39 MDT 2021
**** IP Build 3369179 on Thu Oct 21 08:25:16 MDT 2021
** Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.

source run_ippack.tcl -notrace
ipx::create_core: Time (s): cpu = 00:00:04 ; elapsed = 00:00:06 . Memory (MB): peak = 2577.934 ; gai
INFO: [IP_Flow 19-234] Refreshing IP repositories

```



# Other Ways to use Vitis HLS

# Using Vitis HLS CLI

▶ Invoke Vitis HLS in interactive mode

- Type Tcl commands one at a time

```
> vitis_hls -i
```

▶ Execute Vitis HLS using a Tcl batch file

- Allows multiple runs to be scripted and automated

```
> vitis_hls -f run_aesl.tcl
```

▶ Open an existing project in the GUI

- For analysis, further work or to modify it

```
> vitis_hls -p my.prj
```

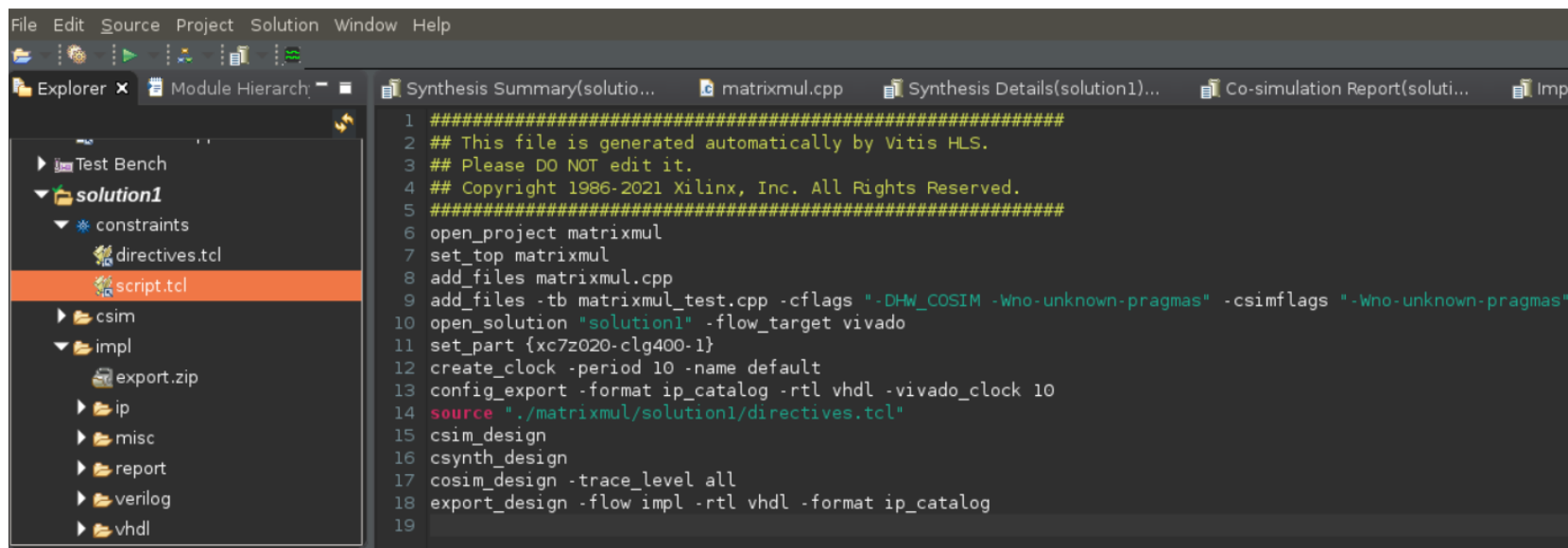
▶ Use the terminal to launch Vitis HLS GUI

```
> vitis_hls
```

```
open_project dct
set_top dct
add_files ../dct_src/dct.cpp
add_files -tb ../dct_src/out.golden.dat -cflags "-Wno-unknown-pragmas" -csimfl
add_files -tb ../dct_src/in.dat -cflags "-Wno-unknown-pragmas" -csimflags "-Wn
add_files -tb ../dct_src/dct_test.cpp -cflags "-Wno-unknown-pragmas" -csimflag
open_solution "solution1" -flow_target vitis
set_part {xcvu11p-flga2577-1-e}
create_clock -period 10 -name default
source "../dct/solution1/directives.tcl"
csim_design
csynth_design
cosim_design
export_design -format ip_catalog
```

# Using Tcl Commands

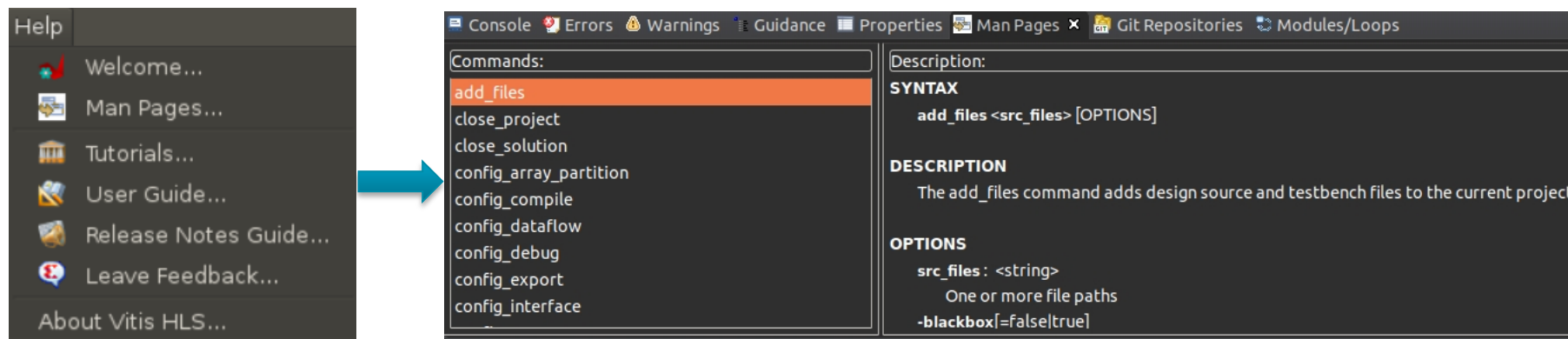
- ▶ When the project is created
  - All Tcl command to run the project are created in script.tcl
    - User specified directives are placed in directives.tcl
  - Use this as a template from creating Tcl scripts



The screenshot shows the Vitis IDE interface. On the left, the 'Module Hierarchy' pane displays the project structure for 'solution1', with 'script.tcl' selected under the 'constraints' folder. The main editor window shows the contents of 'script.tcl', which is a Tcl script generated by Vitis HLS. The script includes comments and commands for opening the project, setting the top module, adding files, opening the solution, setting the part, creating a clock, configuring export, sourcing directives, and performing synthesis and design export.

```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.
5 #####
6 open_project matrixmul
7 set_top matrixmul
8 add_files matrixmul.cpp
9 add_files -tb matrixmul_test.cpp -cflags "-DHW_COSIM -Wno-unknown-pragmas" -csimflags "-Wno-unknown-pragmas"
10 open_solution "solution1" -flow_target vivado
11 set_part {xc7z020-clg400-1}
12 create_clock -period 10 -name default
13 config_export -format ip_catalog -rtl vhdl -vivado_clock 10
14 source "./matrixmul/solution1/directives.tcl"
15 csim_design
16 csynth_design
17 cosim_design -trace_level all
18 export_design -flow impl -rtl vhdl -format ip_catalog
19
```

# Help



- ▶ Help is always available
  - The Help Menu
  - Opens User Guide, Reference Guide and Man Pages
- ▶ In interactive mode
  - The help command lists the man page for all commands

**Auto-Complete all commands using the tab key**

```
maxwell@maxwell-OptiPlex-7080:/home$ vitis_hls -help add_files

***** Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2021.2 (64-bit)
**** SW Build 3367213 on Tue Oct 19 02:47:39 MDT 2021
**** IP Build 3369179 on Thu Oct 21 08:25:16 MDT 2021
** Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.

source /tools/Xilinx/Vitis_HLS/2021.2/scripts/vitis_hls/hls.tcl -notrace
INFO: Applying HLS Y2K22 patch v1.2 for IP revision
SYNTAX
  vitis_hls <tclscripts> [OPTIONS]

DESCRIPTION
  Vitis HLS command line

OPTIONS
  tclscripts : <string>
    TCL file
  -eval <string>
    Run TCL commands
  -f[=false|true]
    start in shell mode
  -i[=false|true]
    start in interactive mode
  -l <string:vitis_hls.log>
    log file
  -n[=false|true]
    do not show splash screen
  -nosplash[=false|true]
    do not show splash screen
  -p <string>
    project name
  -terse[=false|true]
    Filter stdout to only show status INFO and WARNING messages, log file will contain all messages
  -version[=false|true]
    show product version

SEE ALSO
  www.xilinx.com/cgi-bin/docs/rdoc?v=2021.2;t=vitis+doc;d=wic1584802485878.html
```

# Summary

# Summary

- ▶ Vitis HLS can be run under Windows 10, Red Hat Linux, Cent OS, and Ubuntu
- ▶ Vitis HLS can be invoked through GUI and command line in Windows OS, and command line in Linux
- ▶ Vitis HLS project creation wizard involves
  - Defining project name and location
  - Adding design files
  - Specifying testbench files
  - Selecting clock and technology

# Summary

- ▶ Vitis HLS project directory consists of
  - \*.prj project file
  - Multiple solutions directories
  - Each solution directory may contain
    - CSim, Synth, Sim and Impl directories
- ▶ Vitis HLS analysis viewer includes:
  - Pre-Synthesis Control Flow Viewer
  - Schedule Viewer
  - Function Call Graph Viewer
  - Dataflow Viewer
  - Timeline Trace Viewer





Thank You

## Disclaimer and Attribution

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© Copyright 2022 Advanced Micro Devices, Inc. All rights reserved. Xilinx, the Xilinx logo, AMD, the AMD Arrow logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

