

Kanad Basu

Outline

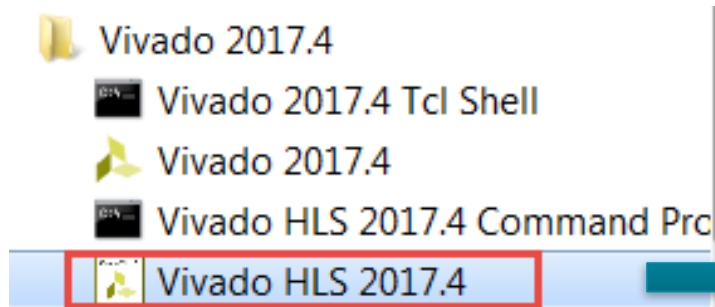
- *Invoking Vivado HLS*
- Project Creation using Vivado HLS
- Synthesis to IPXACT Flow
- Design Analysis
- Other Ways to use Vivado HLS
- Summary

Vivado HLS OS Support

- Vivado HLS is supported on both Linux and Windows
- Vivado HLS tool available under two licenses
 - HLS license
 - HLS license come with Vivado System Edition
 - Supports all 7 series devices including Zynq® All Programmable SoC
 - Does not support Virtex®-6 and earlier devices
 - Use older version of Vivado HLS for Virtex-6 and earlier

Operating System	Version
Windows	Windows 10 Professional (64-bit) Windows 7 SP1 Professional (64-bit)
Red Hat Linux	RHEL Enterprise Linux 6.6-6.9 (64-bit) RHEL Enterprise Linux 7.2 and 7.3 (64-bit)
SUSE	SUSE Linux Enterprise 11.4 and 12.2 (64-bit)
Cent OS	Cent OS 7.2 and 7.3 (64-bit) Cent OS 6.7, 6.8, and 6.9 (64-bit)
Ubuntu	Ubuntu Linux 16.04.2 LTS (64-bit)

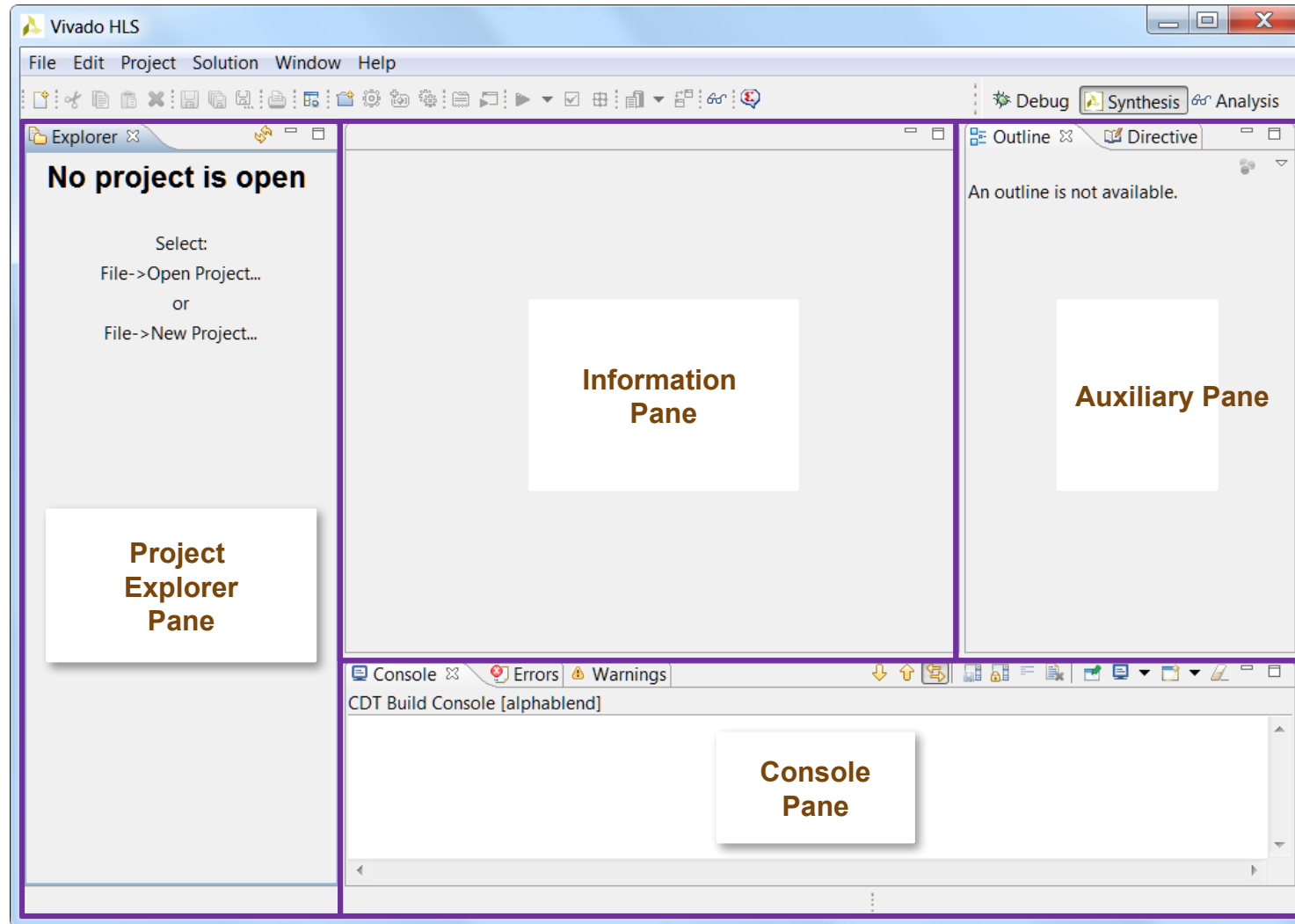
Invoke Vivado HLS from Windows Menu



The first step is to open or create a project



Vivado HLS GUI



Outline

- Invoking Vivado HLS
- *Project Creation using Vivado HLS*
- Synthesis to IPXACT Flow
- Design Analysis
- Other Ways to use Vivado HLS
- Summary

Vivado HLS Projects and Solutions

➤ Vivado HLS is project based

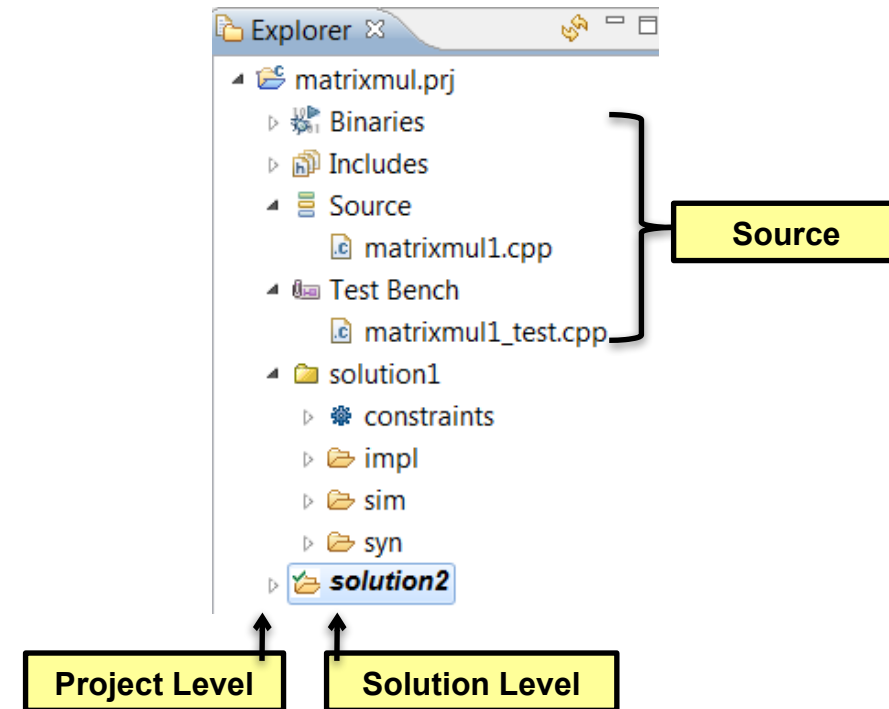
- A project specifies the source code which will be synthesized
- Each project is based on one set of source code
- Each project has a user specified name

➤ 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

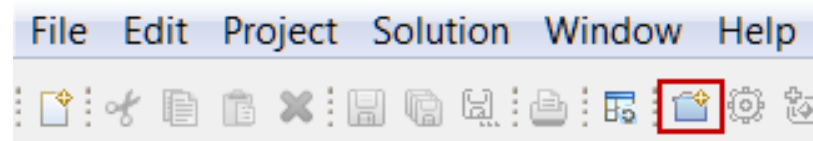
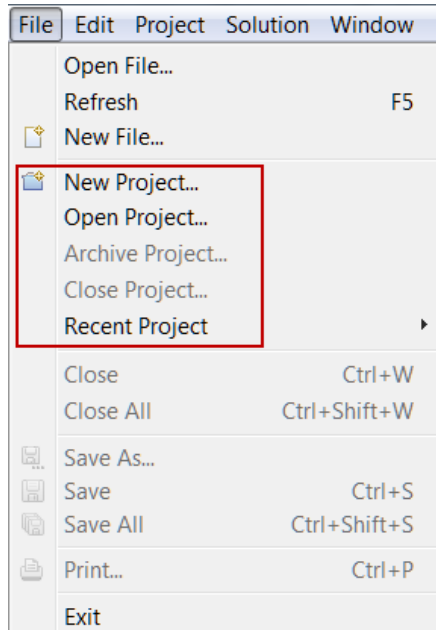
- Top-level is the project directory
- The disk directory structure is identical to the structure shown in the GUI project explorer (except for source code location)



Vivado 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 New Project**

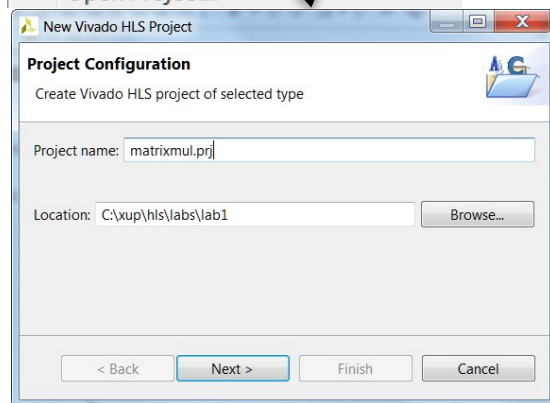
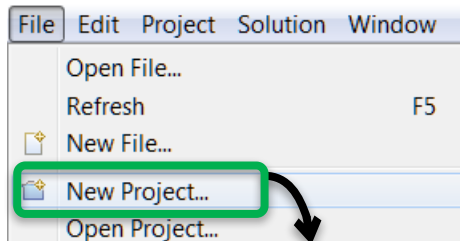
➤ Open an existing project

- All results, reports and directives are automatically saved/remembered
- Use “Recent Project” menu for quick access

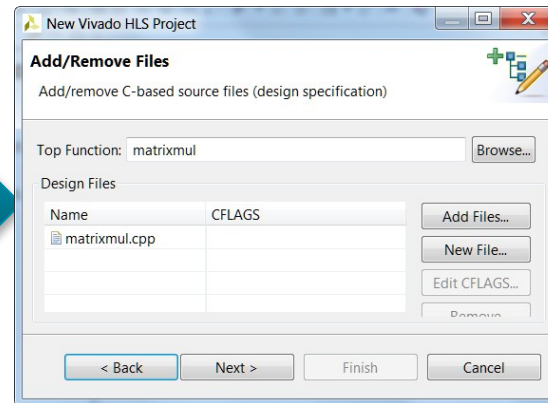
Project Wizard

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

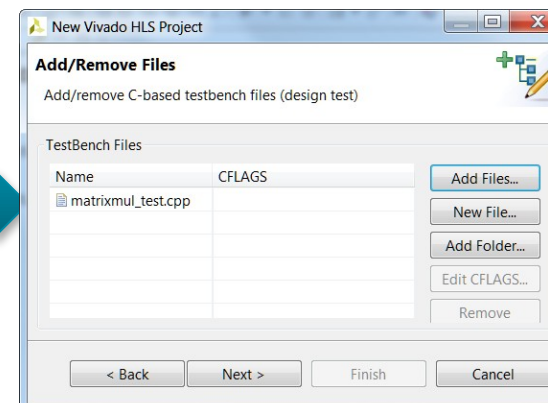
Step-by-step guide ...



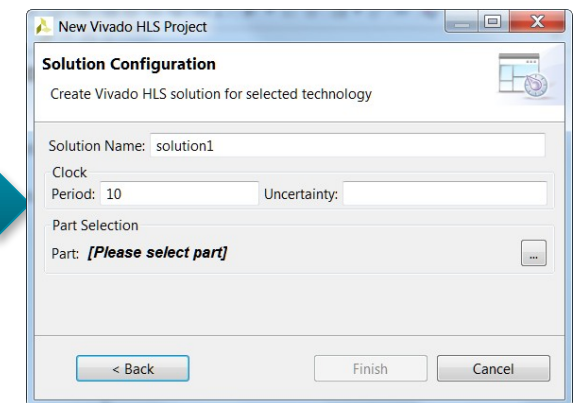
Define project and directory



Add design source files



Specify test bench files



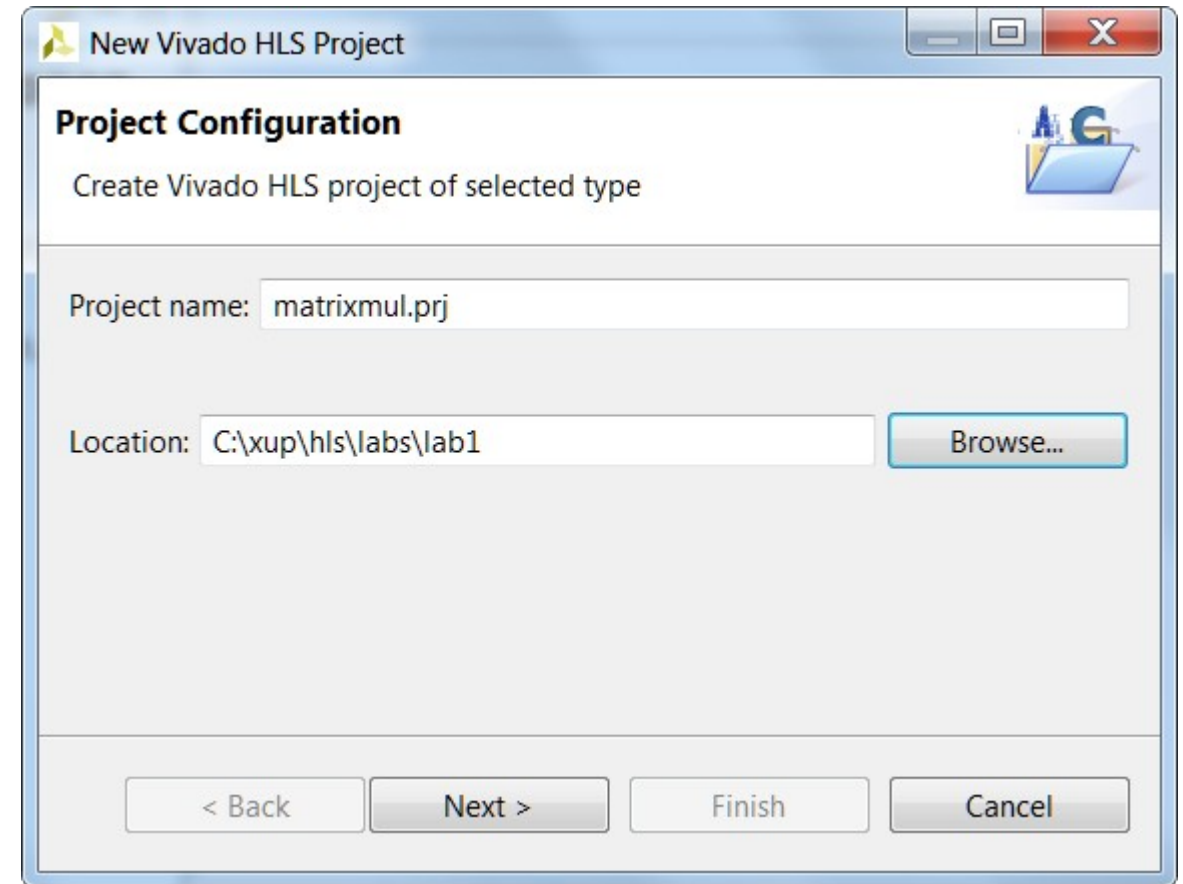
Specify clock and select part

Project Level Information

1st Solution Information

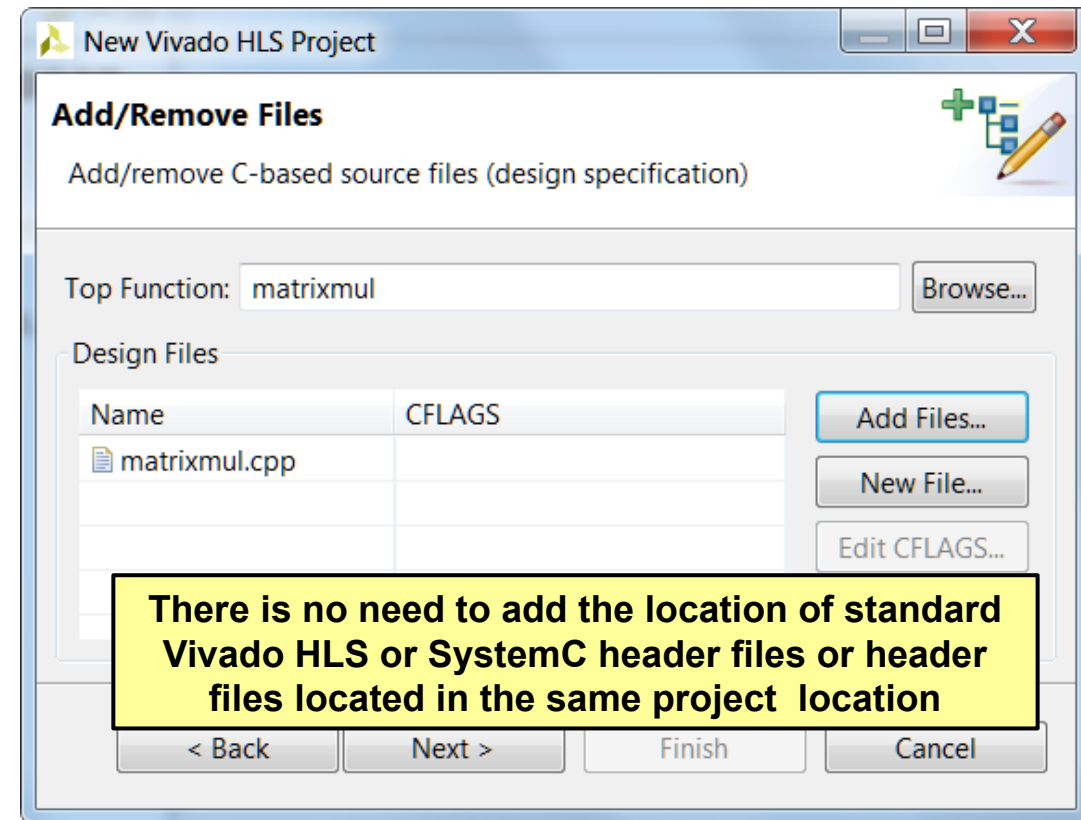
Define Project & Directory

- Define the project name
 - Note, here the project is given the extension .prj
 - A useful way of seeing it's a project (and not just another directory) when browsing
- Browse to the location of the project
 - In this example, project directory “matrixmul.prj” will be created inside directory “lab1”



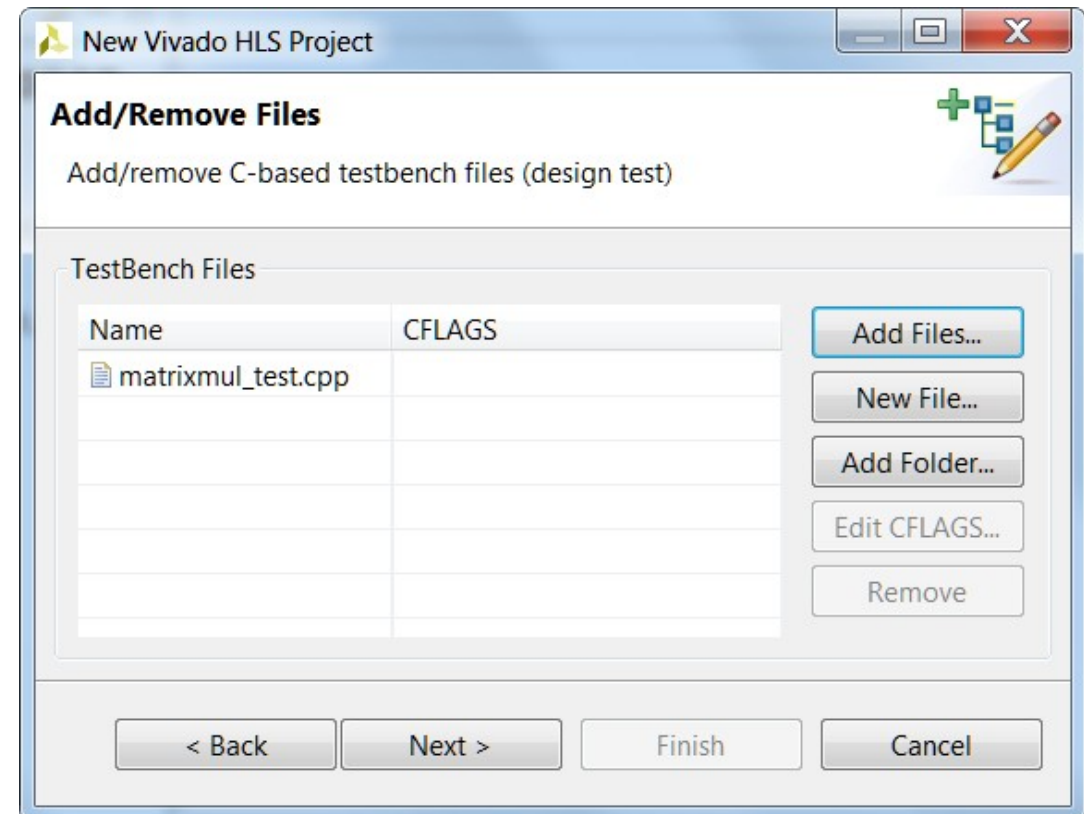
Add Design Source Files

- Add Design Source Files
 - This allows Vivado HLS to determine the top-level design for synthesis, from the test bench and associated files
 - Not required for SystemC designs
- 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
 - Vivado 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)
 - Vivado 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



Test benches I

- The test bench should be in a separate file
- Or excluded from synthesis
 - The Macro `__SYNTHESIS__` can be used to isolate code which will not be synthesized
 - This macro is defined when Vivado HLS parses any code (`-D__SYNTHESIS__`)

```
// test.c
#include <stdio.h>
void test (int d[10]) {
    int acc = 0;
    int i;
    for (i=0;i<10;i++) {
        acc += d[i];
        d[i] = acc;
    }
}
#ifdef __SYNTHESIS__
int main () {
    int d[10], i;
    for (i=0;i<10;i++) {
        d[i] = i;
    }
    test(d);
    for (i=0;i<10;i++) {
        printf("%d %d\n", i, d[i]);
    }
    return 0;
}
#endif
```

Design to be synthesized

Test Bench
Nothing in this ifdef will be read
by Vivado HLS
(will be read by gcc)

Test benches II

➤ Ideal test bench

- Should be self checking
 - RTL verification will re-use the C test bench
- If the test bench is self-checking
 - Allows RTL Verification to be run without a requirement to check the results again
- RTL verification “passes” if the test bench return value is 0 (zero)
 - Actively return a 0 if the simulation passes

```
int main () {  
    // Compare results  
    int ret = system("diff --brief -w test_data/output.dat test_data/output.golden.dat");  
    if (ret != 0) {  
        printf("Test failed !!!\n", ret); return 1;  
    } else {  
        printf("Test passed !\n", ret); return 0;  
    }  
}
```

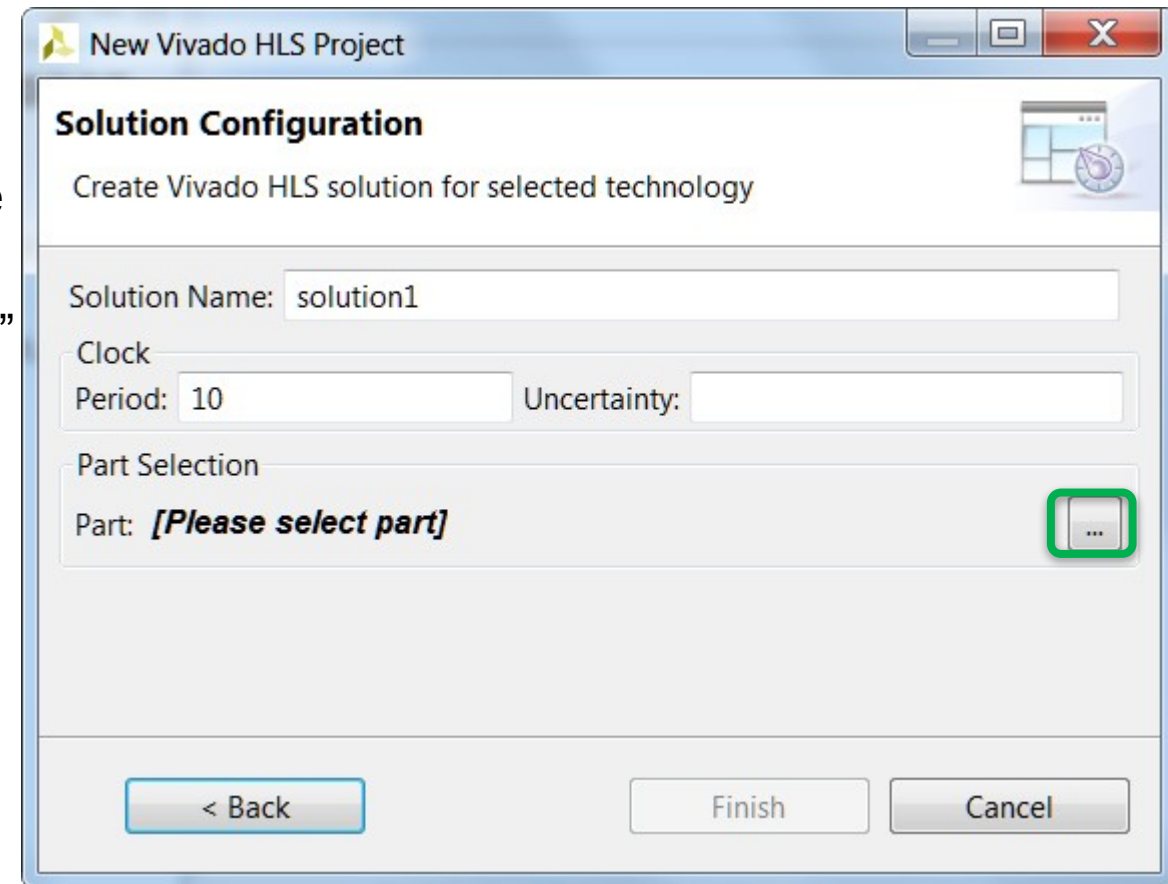
The **-w** option ensures the “newline” does not cause a difference between Windows and Linux files

- Non-synthesizable constructs may be added to a synthesize function if `__SYNTHESIS__` is used

```
#ifndef __SYNTHESIS__  
    image_t *yuv = (image_t *)malloc(sizeof(image_t));  
#else // Workaround malloc() calls w/o changing rest of code  
    image_t _yuv;  
#endif
```

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”
 - Vivado 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



Selecting Part and Implementation Engine

➤ Select the target part either through Parts or Boards specify

➤ Select RTL Tools

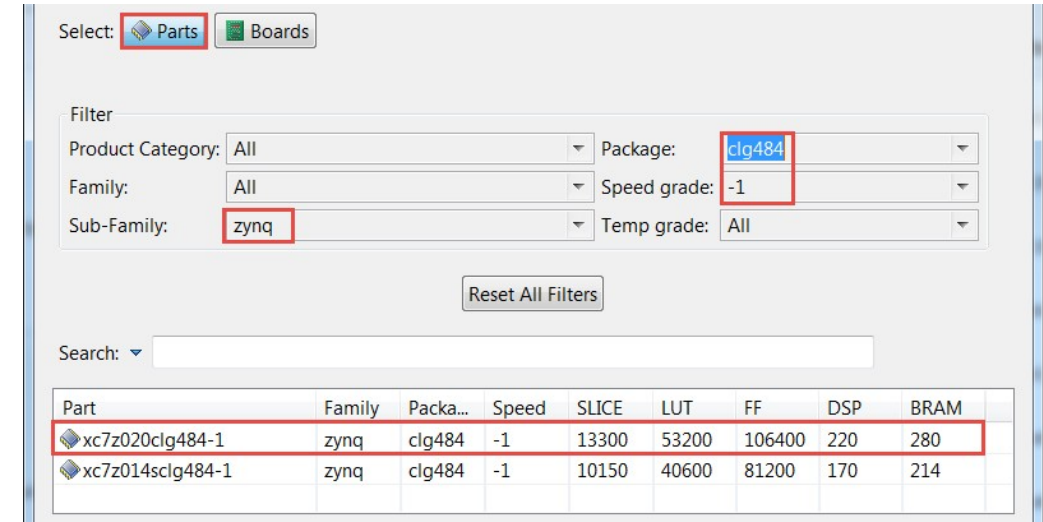
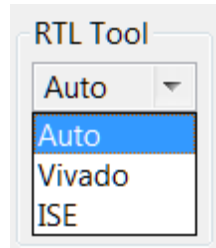
– Auto

- Will select Vivado for 7 Series and Zynq devices
- Will select ISE for Virtex-6 and earlier families

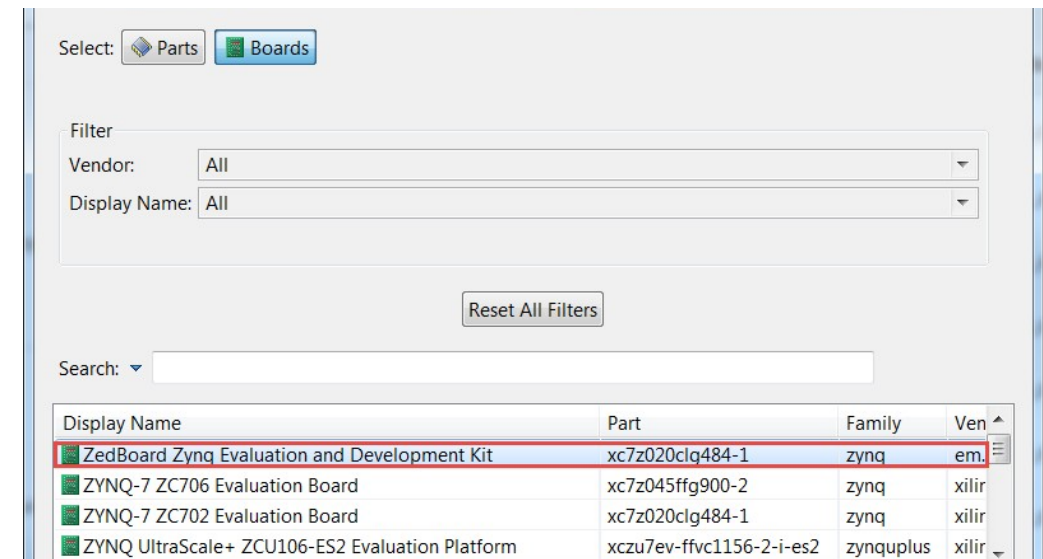
– Vivado

– ISE

- ISE Design Suite must be installed and must be included in the PATH variable

A screenshot of the 'Parts' selection interface in Vivado. The 'Select' tab is active, showing a filter section with dropdowns for Product Category (All), Family (All), Sub-Family (zynq), Package (clg484), Speed grade (-1), and Temp grade (All). A 'Reset All Filters' button is below the filters. A search bar is present. Below the search bar is a table of parts.

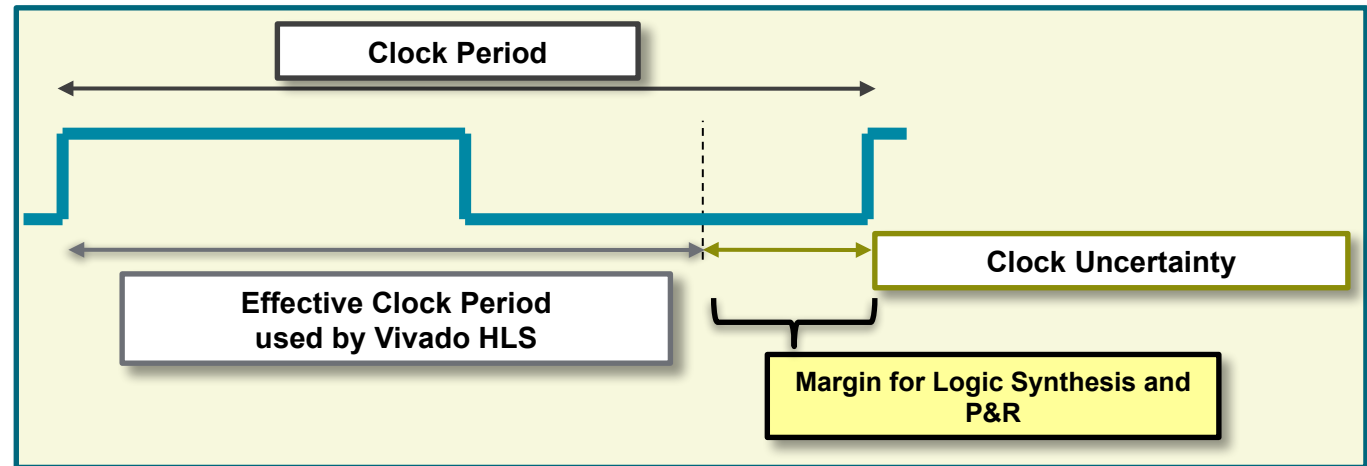
Part	Family	Packa...	Speed	SLICE	LUT	FF	DSP	BRAM
xc7z020clg484-1	zynq	clg484	-1	13300	53200	106400	220	280
xc7z014sclg484-1	zynq	clg484	-1	10150	40600	81200	170	214

A screenshot of the 'Boards' selection interface in Vivado. The 'Select' tab is active, showing a filter section with dropdowns for Vendor (All) and Display Name (All). A 'Reset All Filters' button is below the filters. A search bar is present. Below the search bar is a table of boards.

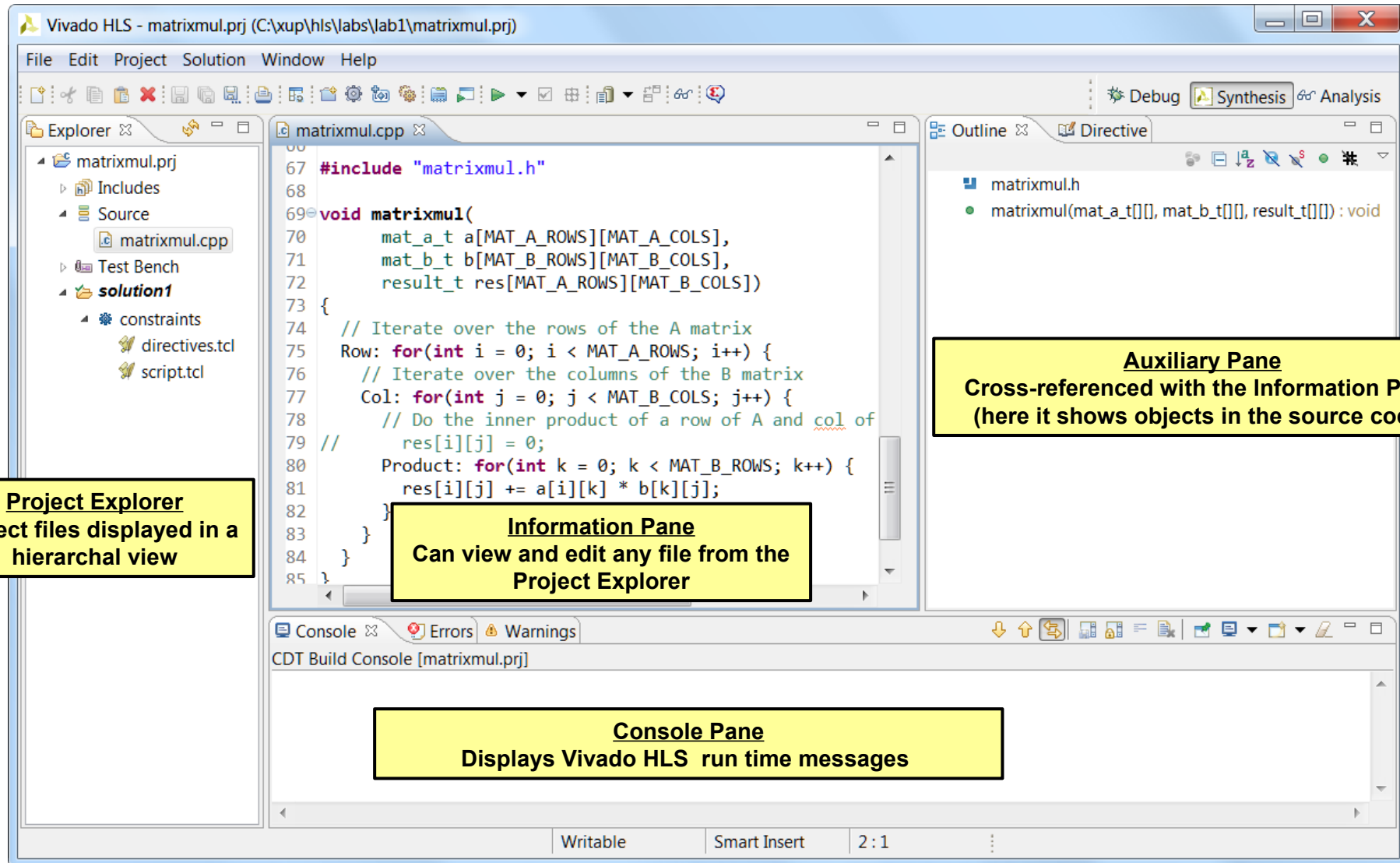
Display Name	Part	Family	Ven
ZedBoard Zynq Evaluation and Development Kit	xc7z020clg484-1	zynq	em
ZYNQ-7 ZC706 Evaluation Board	xc7z045ffg900-2	zynq	xilir
ZYNQ-7 ZC702 Evaluation Board	xc7z020clg484-1	zynq	xilir
ZYNQ UltraScale+ ZCU106-ES2 Evaluation Platform	xczu7ev-ffvc1156-2-i-es2	zynqplus	xilir

Clock Specification

- Clock frequency must be specified
 - Only 1 clock can be specified for C/C++ functions
 - SystemC can define multiple clocks
- 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



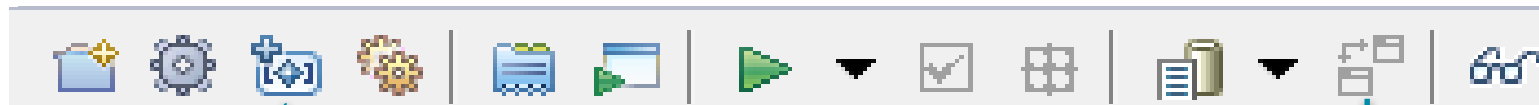
A Vivado HLS Project



Vivado 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



Create a new Project

Change Project Settings

Create a new Solution

Change Solution Settings

Run C Simulation

Open Analysis Viewer

Compare Reports

Open Reports

Export RTL

Run C/RTL Cosimulation

Run C Synthesis

Files: Views, Edits & Information

The screenshot displays the Vivado HLS IDE interface for a project named 'matrixmul.prj'. The main editor window shows the source file 'matrixmul.cpp' with the following code:

```
67 #include "matrixmul.h"
68
69 void matrixmul(
70     mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
71     mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
72     result_t res[MAT_A_ROWS][MAT_B_COLS])
73 {
74     // Iterate over the rows of the A matrix
75     Row: for(int i = 0; i < MAT_A_ROWS; i++) {
76         // Iterate over the columns of the B matrix
77         Col: for(int j = 0; j < MAT_B_COLS; j++) {
78             // Do the inner product of a row of A and col of
79             //
80             res[i][j] = 0;
81             Product: for(int k = 0; k < MAT_B_ROWS; k++) {
82                 res[i][j] += a[i][k] * b[k][j];
83             }
84         }
85     }
```

Annotations and callouts:

- Open file and it will display in the information pane:** An arrow points from the 'matrixmul.cpp' file in the Explorer pane to the main editor window.
- The Auxiliary pane is context sensitive with respect to the information pane:** An arrow points from the 'matrixmul.h' file in the Outline pane to the main editor window.
- Here it displays elements in the code which can have directives specified on them:** This callout points to the 'Col:' and 'Product:' labels in the code, which are used for loop directives.

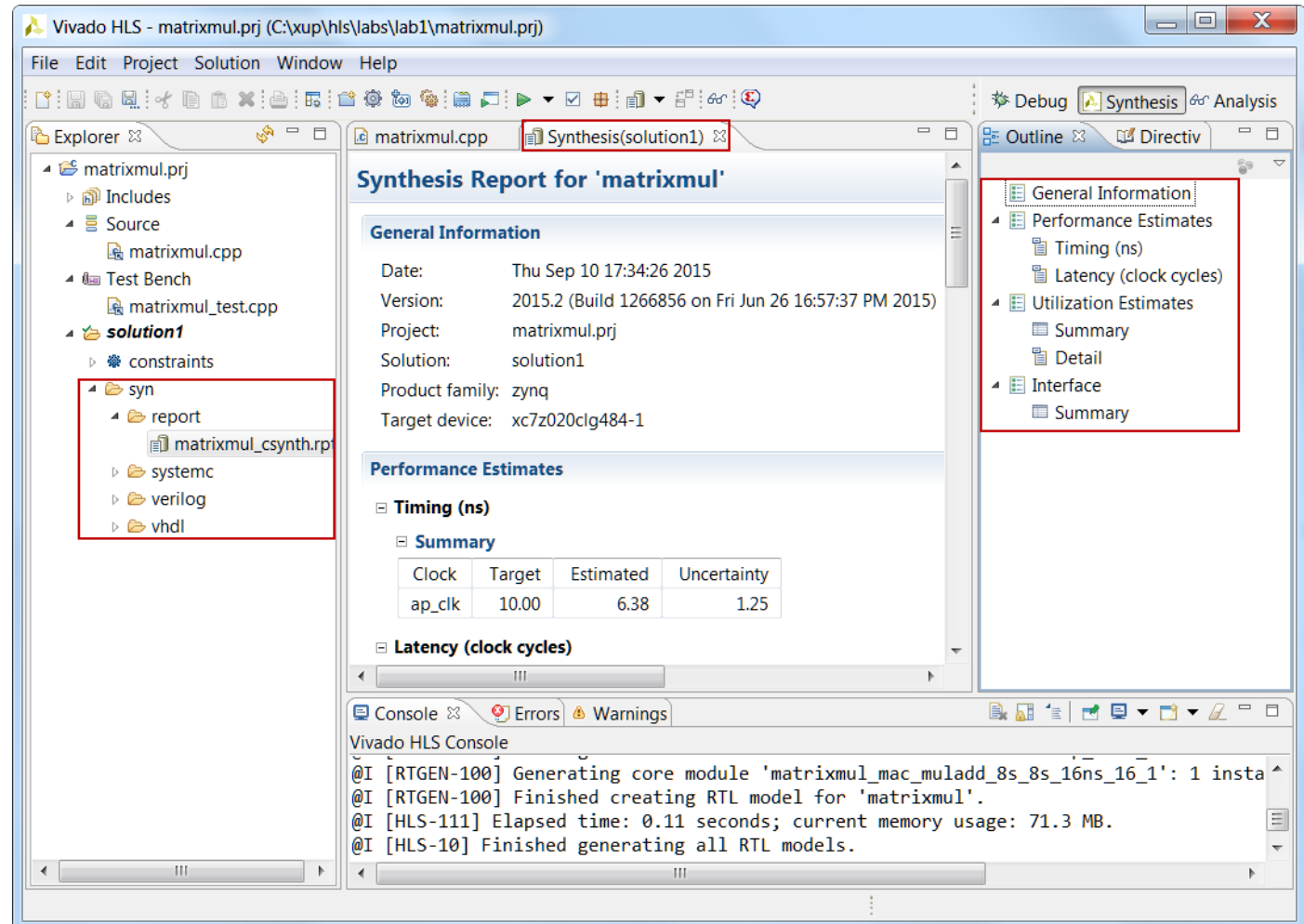
The interface includes a menu bar (File, Edit, Project, Solution, Window, Help), a toolbar, and several panes: Explorer, Editor, Outline, Directive, Console, Errors, and Warnings. The status bar at the bottom shows 'Writable', 'Smart Insert', and '2:1'.

Outline

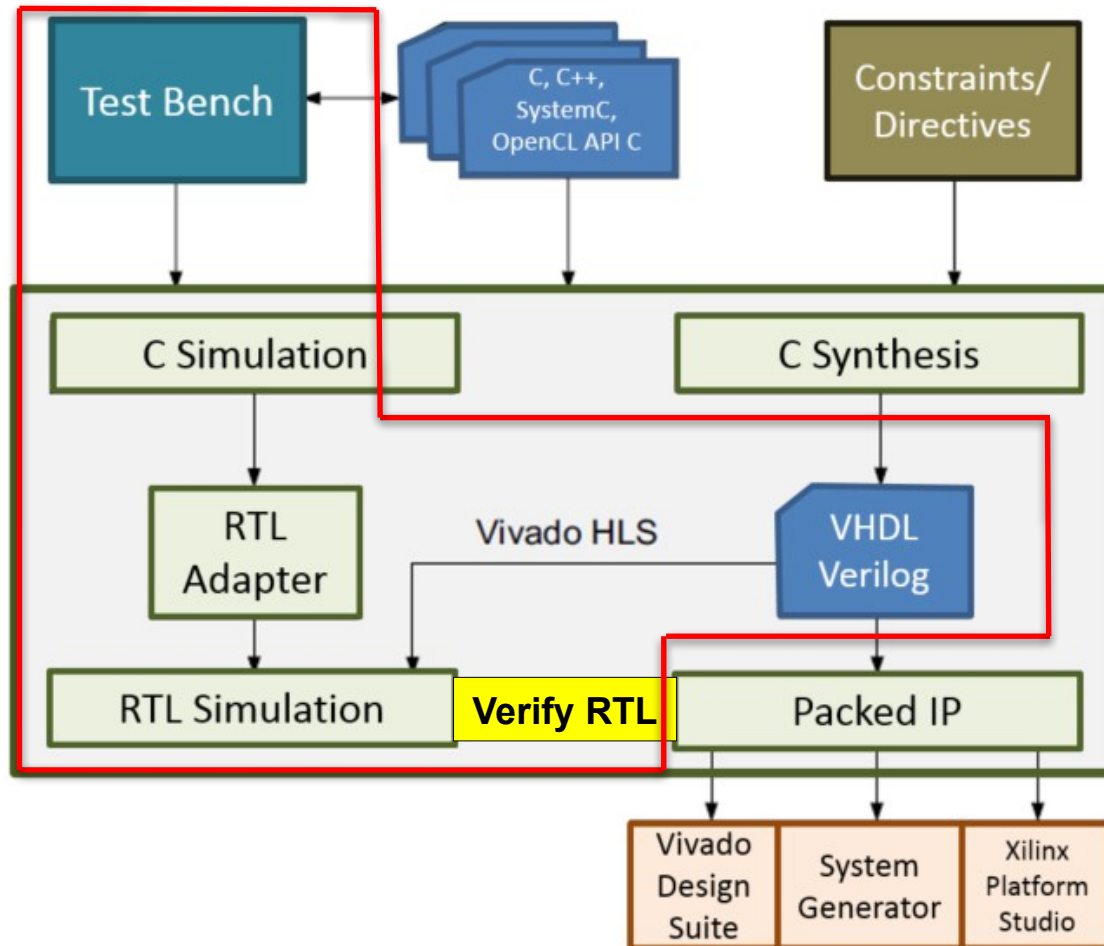
- Invoking Vivado HLS
- Project Creation using Vivado HLS
- *Synthesis to IPXACT Flow*
- Design Analysis
- Other Ways to use Vivado HLS
- Summary

Synthesis

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



Vivado HLS : RTL Verification



RTL output in Verilog and VHDL

Automatic re-use of the C-level test bench

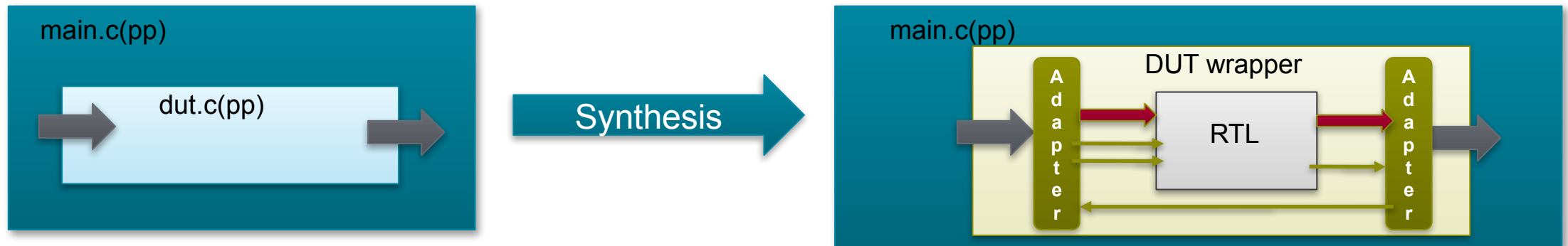
RTL verification can be executed from within Vivado HLS

Support for Xilinx simulators (XSim and ISim) and 3rd party HDL simulators in automated flow

RTL Verification: Under-the-Hood

➤ RTL Co-Simulation

- Vivado HLS provides RTL verification
- Creates the wrappers and adapters to re-use the C test bench



- Prior to synthesis
 - Test bench
 - Top-level C function

- After synthesis
 - Test bench
 - Wrapper created by Vivado HLS
 - Adapters created by Vivado HLS
 - RTL output from Vivado HLS
 - Verilog or VHDL

There is no HDL test bench created

RTL Verification Support

➤ Vivado HLS RTL Output

- Vivado HLS outputs RTL in SystemC, Verilog and VHDL
 - The SystemC output is at the RT Level
 - The input is not transformed to SystemC at the ESL

➤ RTL Verification with SystemC

- The SystemC RTL output can be used to verify the design without the need for a HDL simulator and license

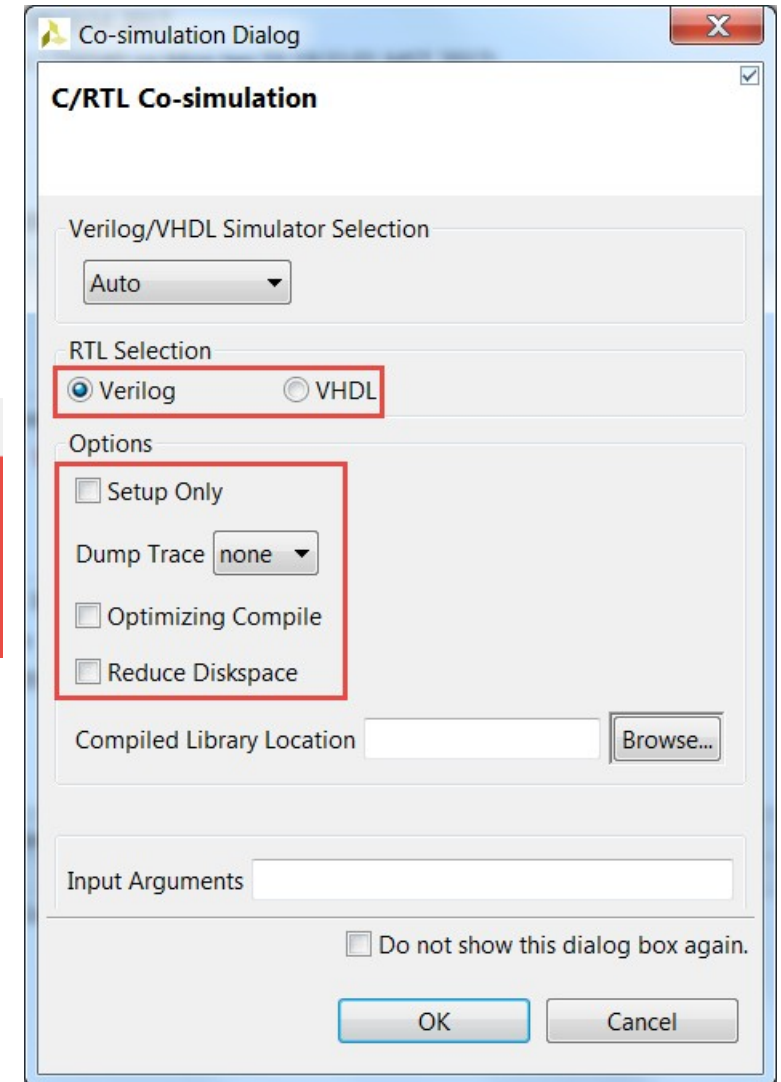
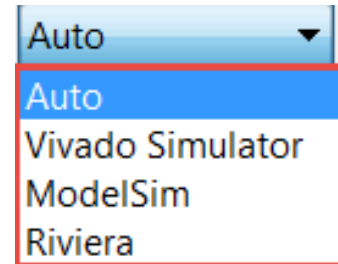
➤ HDL Simulation Support

- Vivado HLS supports HDL simulators on both Windows & Linux
- The 3rd party simulator executable must be in OS search path

Simulator	Linux	Windows	SUSE	Ubuntu
XSim (Vivado Simulator)	Supported	Supported	Supported	Supported
ISim (ISE Simulator)	Supported	Supported	Supported	Supported
ModelSim	Supported	Supported	Supported	N/A
Synopsys VCS	Supported	N/A	Supported	N/A
Cadence NCSim	Supported	N/A	Supported	N/A
Riviera	Supported	Supported	Supported	N/A

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
- 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

The screenshot displays the Vivado HLS interface for a project named 'matrixmul.prj'. The Explorer pane on the left shows the project structure, with the 'sim' directory highlighted. The main window shows the 'Cosimulation Report for 'matrixmul'' with a table of results. The console at the bottom shows the simulation output, including the start of C post-checking and the final 'Test passed' message.

Cosimulation Report for 'matrixmul'

Result

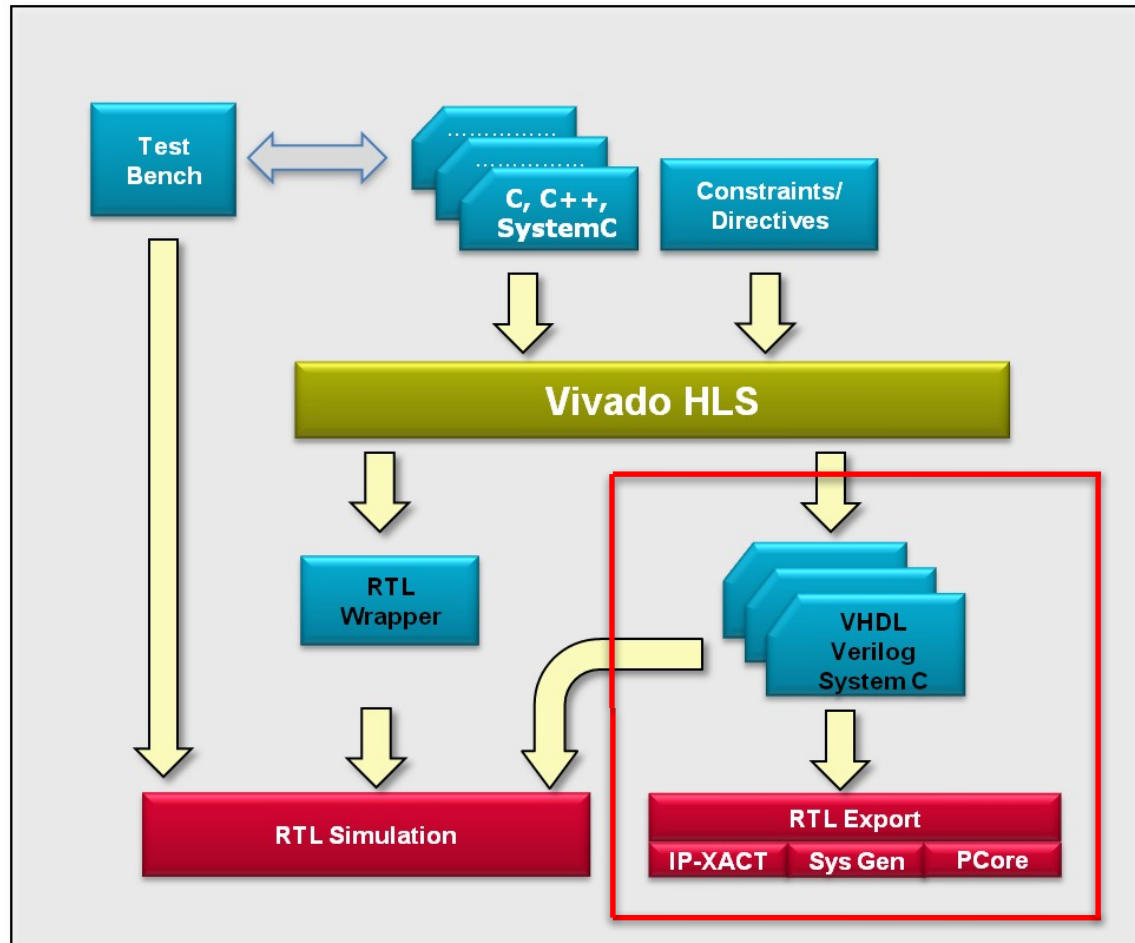
RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	106	106	106	107	107	107

Export the report(.html) using the [Export Wizard](#)

Vivado HLS Console

```
@I [SIM-316] Starting C post checking ...
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.
@I [SIM-1000] *** C/RTL co-simulation finished: PASS ***
@I [LIC-101] Checked in feature [VIVADO_HLS]
```

Vivado HLS : RTL Export



RTL output in Verilog, VHDL and SystemC

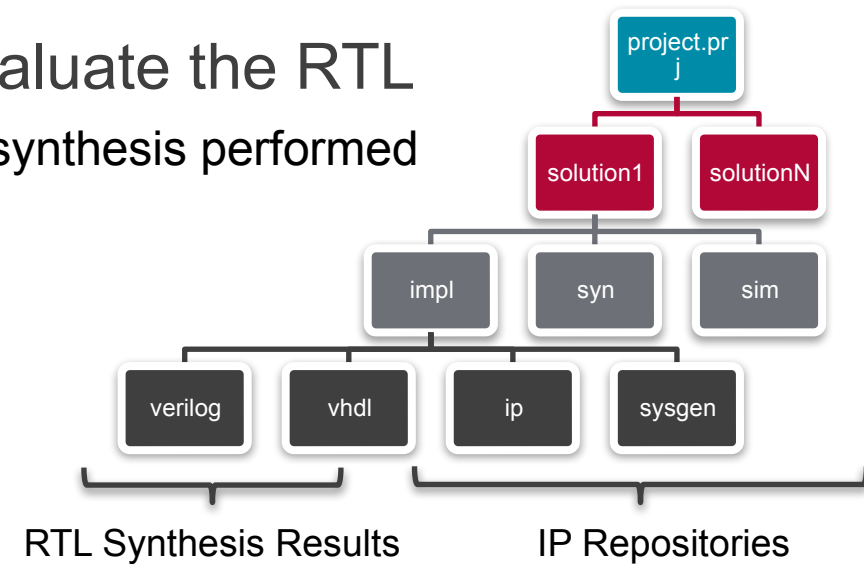
Scripts created for RTL synthesis tools

RTL Export to IP-XACT, SysGen, and Pcore formats

IP-XACT and SysGen => Vivado HLS for 7 Series and Zynq families
PCore => Only Vivado HLS Standalone for all families

RTL Export: Synthesis

- RTL Synthesis can be performed to evaluate the RTL
 - IP-XACT and System Generator formats: Vivado synthesis performed
 - Pcore format: ISE synthesis is performed



- RTL synthesis results are not included with the IP package
 - Evaluate step is provided to give confidence
 - Timing will be as estimate (or better)
 - Area will be as estimated (or better)
 - Final RTL IP is synthesized with the rest of the RTL design

RTL Export: IP Repositories

- IP can be imported into other Xilinx tools

Project Directory
Top-level project directory
(there must be one)

Solution directories

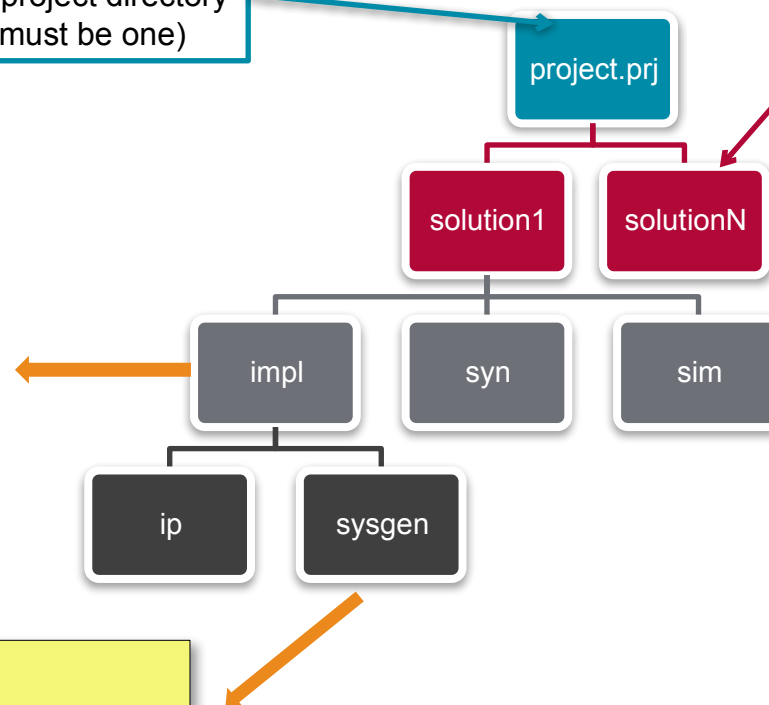
There can be multiple solutions for each project. Each solution is a different implementation of the same (project) source code

In Vivado :

1. Project Manager > IP Catalog
2. Add IP to import this block
3. Browse to the zip file inside "ip"

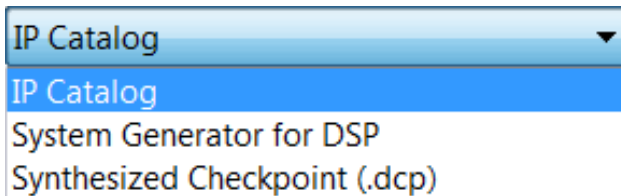
In System Generator :

1. Use XilinxBlockAdd
2. Select Vivado_HLS block type
3. Browse to the solution directory

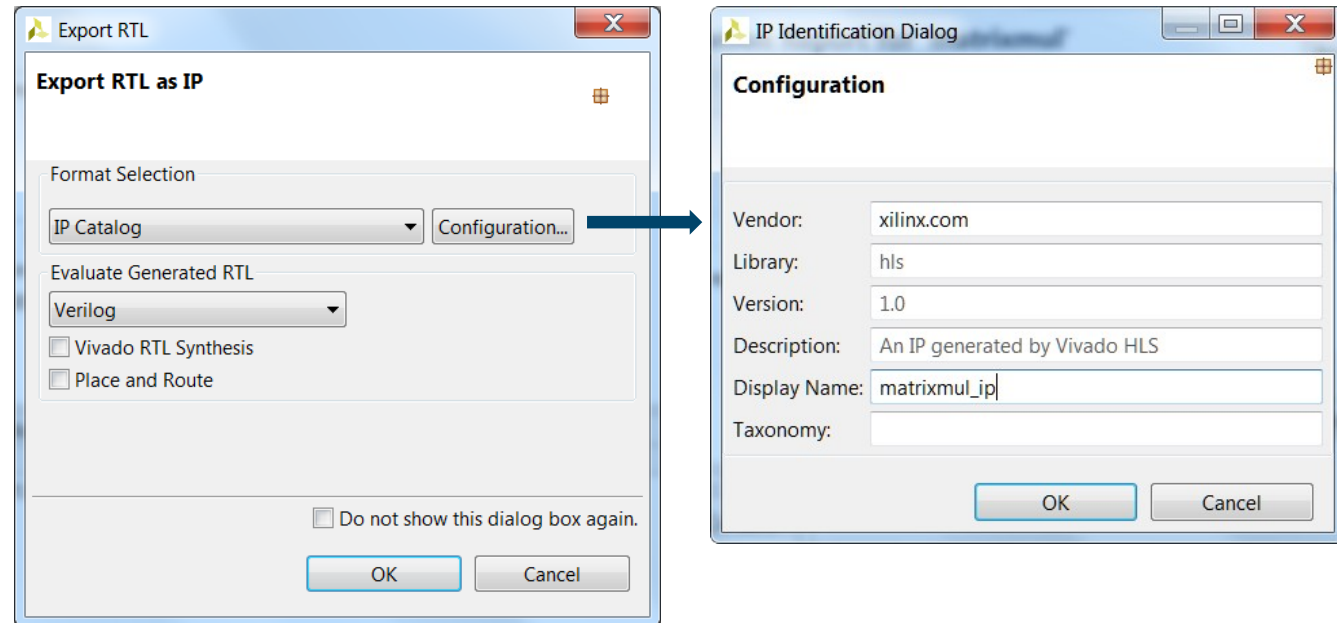


RTL Export for Implementation

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



- Optionally, configure the output
- Select the desired language
- Optionally, click on Vivado RTL Synthesis and Place and Route options for invoking implementation tools from within Vivado 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

➤ Report

- A report is created and opened automatically

```
Implementation tool: Xilinx Vivado v.2017.4
Project:            matrixmul.prj
Solution:           solution1
Device target:      xc7z020clg484-1
Report date:        Tue Feb 13 10:29:28 -0800 2018
```

```
==== Post-Implementation Resource usage ===
```

```
SLICE:             10
LUT:               34
FF:               27
DSP:              1
BRAM:             0
SRL:              0
```

```
==== Final timing ===
```

```
CP required:       10.000
CP achieved post-synthesis: 3.019
CP achieved post-implementation: 3.728
Timing met
```

```
INFO: [Common 17-206] Exiting Vivado at Tue Feb 13 10:29:28 2018...
Finished export RTL.
```

Export Report for 'matrixmul'

General Information

Report date: Tue Feb 13 10:29:28 -0800 2018
Project: matrixmul.prj
Solution: solution1
Device target: xc7z020clg484-1
Implementation tool: Xilinx Vivado v.2017.4

Resource Usage

	VHDL
SLICE	10
LUT	34
FF	27
DSP	1
BRAM	0
SRL	0

Final Timing

	VHDL
CP required	10.000
CP achieved post-synthesis	3.019
CP achieved post-implementation	3.728

Timing met

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  **solution1**

➤ No report will be created

➤ Observe the console

- No packing, routing phases

Starting export RTL ...

`C:/Xilinx/Vivado/2017.4/bin/vivado_hls.bat C:/xup/hls/labs/lab1/matrixmul.prj/solution1/export.tcl`

INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2017.4/bin/unwrapped/win64.o/vivado_hls.exe'

INFO: [HLS 200-10] For user 'parimalp' on host 'xsjparimalp31' (Windows NT_amd64 version 6.1) on T

INFO: [HLS 200-10] In directory 'C:/xup/hls/labs/lab1'

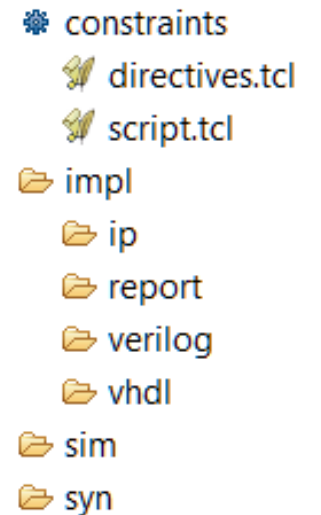
INFO: [HLS 200-10] Opening project 'C:/xup/hls/labs/lab1/matrixmul.prj'.

INFO: [HLS 200-10] Opening solution 'C:/xup/hls/labs/lab1/matrixmul.prj/solution1'.

INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.

INFO: [HLS 200-10] Setting target device to 'xc7z020clg484-1'

INFO: [IMPL 213-8] Exporting RTL as a Vivado IP.



Outline

- Invoking Vivado HLS
- Project Creation using Vivado HLS
- Synthesis to IPXACT Flow
- *Design Analysis*
- Other Ways to use Vivado HLS
- Summary

Analysis Perspective

- Perspective for design analysis
 - Allows interactive analysis

The screenshot shows the Vivado HLS interface in the Analysis perspective. A red arrow points to the 'Analysis' tab in the top toolbar. The 'Module Hierarchy' window displays a tree of modules and a table of resource usage.

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	6	1	182	334	3959	3960	none
dct_2d	4	1	128	248	3668	3669	none
dct_1d	1	1	55	104	209	210	none

The 'Performance Profile' window shows a table of performance metrics for the 'dct' module and its sub-blocks.

	Pipelined	Latency	Initiation Interval	Iteration Late...	Trip count
dct	-	3959	3960	-	-
RD_Loop_Row	no	144	-	18	8
RD_Loop_Col	no	16	-	2	8
WR_Loop_Row	no	144	-	18	8
WR_Loop_Col	no	16	-	2	8

The 'Performance View' window shows a Gantt chart of scheduled operations for the 'dct' module. The operations are color-coded: yellow for loops and green for modules.

Module Hierarchy
Hierarchical Summary and Navigation

Performance Profile
Latency and Interval summary for this block

Performance View
Scheduled operations.

Loops : shown in Yellow are expandable and collapsible

Modules: shown in Green open the view on sub-blocks

Performance Analysis

The screenshot displays the Vivado Performance Analysis tool interface. The main window, titled "Performance - dct_1d", shows a hierarchical view of the design. The "Current Module" is set to "dct > dct_2d > dct_1d". The "Hierarchical Navigation" pane on the left lists operations and control states, including "i_21_read(wire_re)", "i_2_read(wire_re)", "[~]DCT_Outer_Loop", "exitcond(icmp)", "k_1(+)", "[+]DCT_Inner_Loop", "tmp_1(+)", "p_addr3(+)", and "node_60(write)". The "Loop Hierarchy" pane shows a tree structure of loops, with a red arrow pointing to the "[~]DCT_Outer_Loop" node. The "Scheduled States" pane shows a list of states, including "c0", "c1", "c2", "c3", "c4", and "c5". A yellow box labeled "Operations, loops and functions" is positioned over the left pane, and another yellow box labeled "Scheduled States" is positioned over the bottom pane. A red arrow points from the "[~]DCT_Outer_Loop" node in the Loop Hierarchy pane to the "C Source" window on the right. The "C Source" window shows the C code for the design, with the line "dst[k] = DESCALE(tmp, CONST_BITS);" highlighted in yellow. A yellow box labeled "Select operations and right-click to cross reference with the C source and HDL" is positioned over the C Source window, with a red arrow pointing to the highlighted line.

Performance - dct_1d

Hierarchical Navigation

Current Module : **dct** > **dct_2d** > **dct_1d**

Operation\Control State	c0	c1	c2	c3	c4	c5
i_21_read(wire_re)						
i_2_read(wire_re)						
[~]DCT_Outer_Loop						
exitcond(icmp)						
k_1(+)						
[+]DCT_Inner_Loop						
tmp_1(+)						
p_addr3(+)						
node_60(write)						

Loop Hierarchy

Operations, loops and functions

Scheduled States

C Source

File: C:\Vivado_HLS_Tutorial\Design_Analysis\lab1\dct.cp

```
1 |
2 #include "dct.h"
3
4 void dct_1d(dct_data_t src[DCT_SIZE], dct_data_t dst[DCT_SIZE]) {
5 {
6     unsigned int k, n;
7     int tmp;
8     const dct_data_t dct_coeff_table[DCT_SIZE][DCT_SIZE];
9     #include "dct_coeff_table.txt"
10 };
11
12 DCT_Outer_Loop:
13 for (k = 0; k < DCT_SIZE; k++) {
14     for (n = 0; n < DCT_SIZE; n++) {
15         tmp = src[k][n] * dct_coeff_table[k][n];
16         dst[k] = DESCALE(tmp, CONST_BITS);
17     }
18 }
19
20 }
21 }
22
23 void dct_2d(dct_data_t in_block[DCT_SIZE][DCT_SIZE],
24             dct_data_t out_block[DCT_SIZE][DCT_SIZE]) {
```

Resources Analysis

Resource Sharing - dct_1d

Current Module : dct > dct_2d > dct_1d

Resource\Control St	C0	C1	C2	C3	C4	C5	C6	C7	C8
[-]Memory Ports									
src		read	read	read	read				
src		read	read	read	read				
dct_coeff_tab		read							
dct_coeff_tab			read						
dct_coeff_tab			read						
dct_coeff_tab			read						
dct_coeff_tab				read					
dct_coeff_tab				read					
dct_coeff_tab				read					
dst									
[+]I/O Ports									
[+]Expressions									
[+]Instances									

Performance | Resource Sharing

Resource View
Scheduled operations associated with resource: anything on the same row shares the same resource

```

4 void dct_1d(dct_data_t src[DCT_SIZE], dct_data_t dst[DCT_SIZE]) {
5 {
6   unsigned int k, n;
7   int tmp;
8   const dct_data_t dct_coeff_table[DCT_SIZE][DCT_SIZE] = {
9 #include "dct_coeff_table.txt"
10 };
11
12 DCT_Outer_Loop:
13   for (k = 0; k < DCT_SIZE; k++) {
14     DCT_Inner_Loop:
15       for (n = 0, tmp = 0; n < DCT_SIZE; n++) {
16         int coeff = (int)dct_coeff_table[k][n];
17         tmp += src[n] * coeff;
18       }
19     }
20   }
21 }
    
```

Performance Profile | Resource Profile

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth
dct	36	8	617	584				
I/O Ports(2)					32			
Instances(3)	18	8	606	582				
Memories(9)	18	0	0	144				18
buf_2d_in_0_U	2	0	0	16				2
buf_2d_in_1_U	2	0	0	16				2
buf_2d_in_2_U	2	0	0	16				2
buf_2d_in_3_U	2	0	0	16				2
buf_2d_in_4_U	2	0	0	16				2
buf_2d_in_5_U	2	0	0	16				2
buf_2d_in_6_U	2	0	0	16				2
buf_2d_in_7_U	2	0	0	16				2
buf_2d_out_U	2	0	0	16				2
Expressions(1)	0	0	0	2	1	1	0	
Registers(11)			11	11				
FIFO(0)	0	0	0	0				0
Multiplexers(0)	0	0	0	0				0

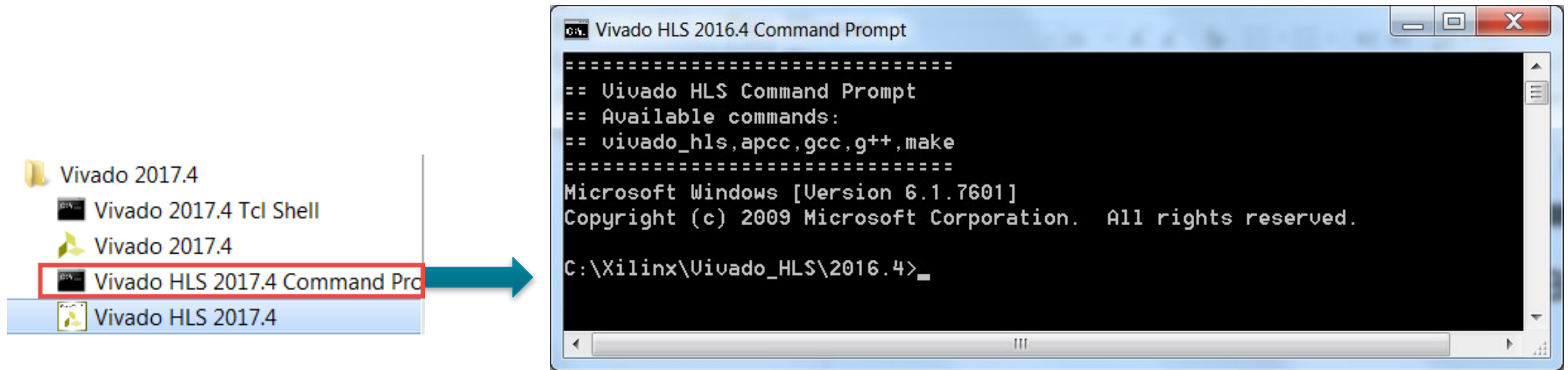
Resource Profile
Resource summary for this block

Outline

- Invoking Vivado HLS
- Project Creation using Vivado HLS
- Synthesis to IPXACT Flow
- Design Analysis
- *Other Ways to use Vivado HLS*
- Summary

Command Line Interface: Batch Mode

- Vivado HLS can also be run in batch mode
 - Opening the Command Line Interface (CLI) will give a shell



- Supports the commands required to run Vivado HLS & pre-synthesis verification (gcc, g++, apcc, make)

Using Vivado HLS CLI

➤ Invoke Vivado HLS in interactive mode

- Type Tcl commands one at a time

```
> vivado_hls -i
```

➤ Execute Vivado HLS using a Tcl batch file

- Allows multiple runs to be scripted and automated

```
> vivado_hls -f run_aesl.tcl
```

➤ Open an existing project in the GUI

- For analysis, further work or to modify it

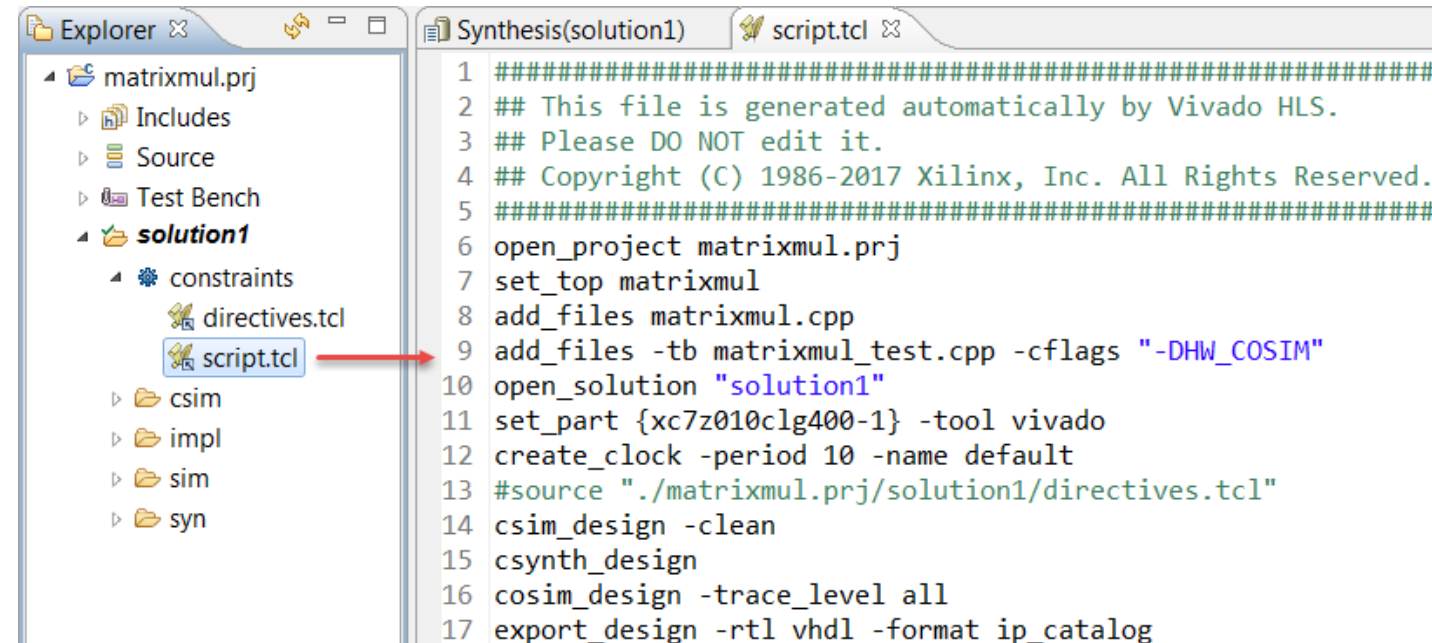
```
> vivado_hls -p my.prj
```

➤ Use the "Run" button in the GUI

```
> vivado_hls
```


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
 - Uncomment the commands before running the Tcl script

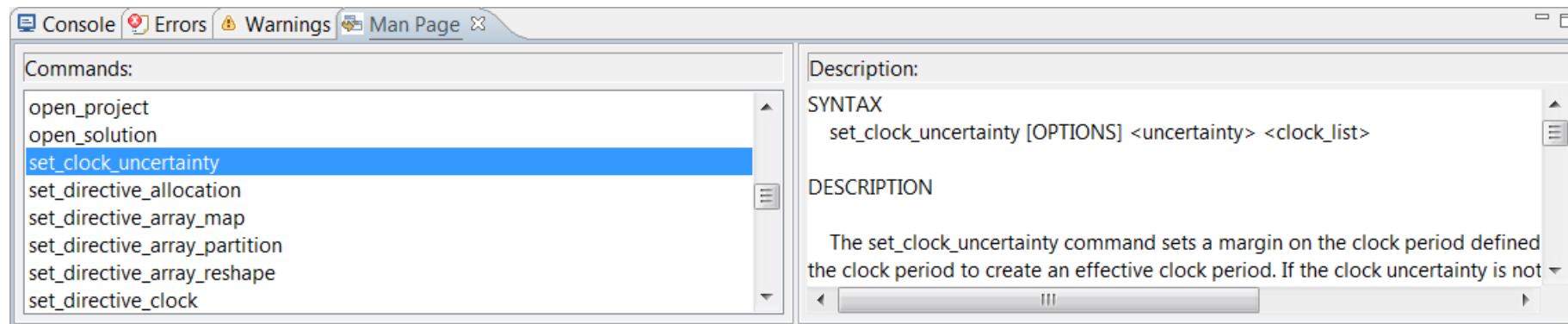


The screenshot shows the Vivado IDE interface. On the left, the 'Explorer' pane displays the project structure for 'matrixmul.prj'. The 'solution1' folder is expanded, showing subfolders 'constraints', 'csim', 'impl', 'sim', and 'syn'. The 'constraints' folder is further expanded, showing 'directives.tcl' and 'script.tcl'. A red arrow points from 'script.tcl' to the right pane. The right pane shows the contents of 'script.tcl', which is a Tcl script generated by Vivado 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, sourcing directives, and running synthesis and simulation.

```
1 #####
2 ## This file is generated automatically by Vivado HLS.
3 ## Please DO NOT edit it.
4 ## Copyright (C) 1986-2017 Xilinx, Inc. All Rights Reserved.
5 #####
6 open_project matrixmul.prj
7 set_top matrixmul
8 add_files matrixmul.cpp
9 add_files -tb matrixmul_test.cpp -cflags "-DHW_COSIM"
10 open_solution "solution1"
11 set_part {xc7z010clg400-1} -tool vivado
12 create_clock -period 10 -name default
13 #source "../matrixmul.prj/solution1/directives.tcl"
14 csim_design -clean
15 csynth_design
16 cosim_design -trace_level all
17 export_design -rtl vhdl -format ip_catalog
```

Help

- Help is always available
 - The Help Menu
 - Opens User Guide, Reference Guide and Man Pages



- In interactive mode

```
Vivado_hls> help add_files  
  
SYNOPSIS  
  add_files [OPTIONS] <src_files>  
Etc...
```

Auto-Complete all commands using the tab key

Outline

- Invoking Vivado HLS
- Project Creation using Vivado HLS
- Synthesis to IPXACT Flow
- Design Analysis
- Other Ways to use Vivado HLS
- *Summary*

Summary

- Vivado HLS can be run under Windows 7/8.1, Red Hat Linux, SUSE OS, and Ubuntu
- Vivado HLS can be invoked through GUI and command line in Windows OS, and command line in Linux
- Vivado HLS project creation wizard involves
 - Defining project name and location
 - Adding design files
 - Specifying testbench files
 - Selecting clock and technology
- The top-level module in testbench is `main()` whereas top-level module in the design is the function to be synthesized

Summary

- Vivado HLS project directory consists of
 - *.prj project file
 - Multiple solutions directories
 - Each solution directory may contain
 - impl, synth, and sim directories
 - The impl directory consists of ip, verilog, and vhdl folders
 - The synth directory consists of reports, vhdl, and verilog folders
 - The sim directory consists of testbench and simulation files