Tutorial 9: PYNQ Overlay Tutorial (Part 1)

(Part 1: Use Vitis HLS to Create a Custom IP)

You can find more details about this tutorial in https://pynq.readthedocs.io/en/latest/overlay_design_methodology/overlay_tutorial.html

Objectives

After completing this tutorial, you will be able to:

- Write HLS code to design a hardware module
- Run the simulation in HLS and export the custom IP

High-level synthesis (HLS)

High-level synthesis allows you to design hardware modules with C/C++ language, the C synthesis will translate you design into Verilog/VHDL modules.

From Xilinx document UG902:

> High-level synthesis (HLS) bridges hardware and software domains, providing the following primary benefits:
>
> - Improved productivity for hardware designers
> - Improved system performance for software designers
>
> Using a high-level synthesis design methodology allows you to:
>
> - Develop algorithms at the C-level
> - Verify at the C-level
> - Control the C synthesis process through optimization directives
> - Create multiple implementations from the C source code using optimization directives
> - Create readable and portable C source code

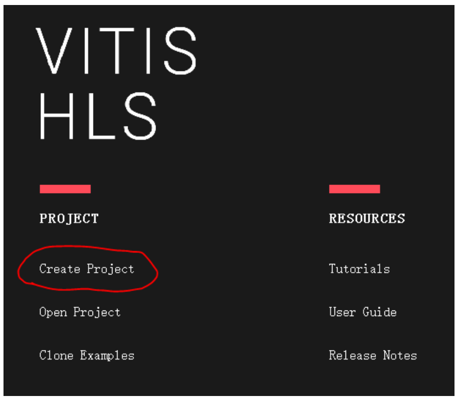
Overlay Introduction

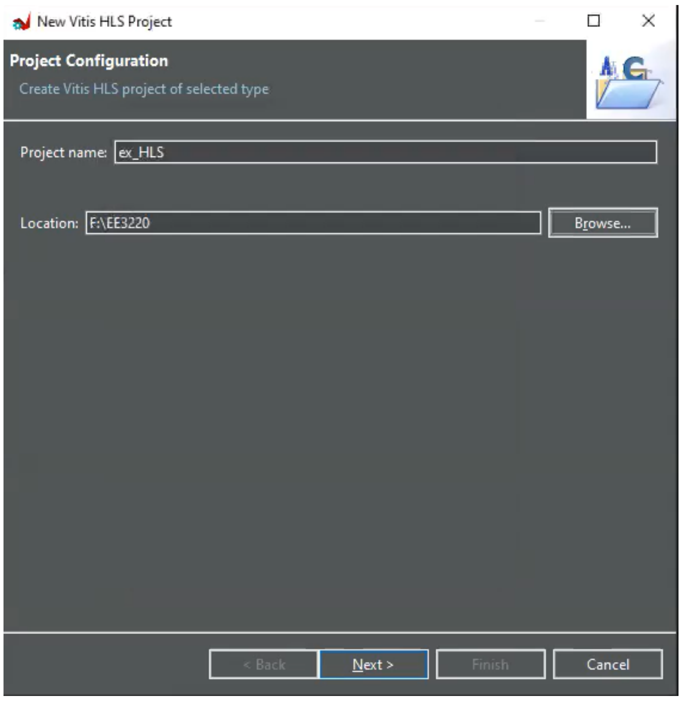
A programmer can download overlays into the Xilinx Programmable Logic, and overlays can provide functionality required by the software application. An *overlay* is a class of Programmable Logic design. Overlays are designed to be configurable, and reusable for broad set of applications. A PYNQ overlay will have a Python interface, allowing a software programmer to use it like any other Python package. [1]

Steps

A. Launch Vitis HLS and create an empty project targeting the Pynq board.

1. Open Vitis HLS 2020.1, click **Create Project** to start the wizard. You will see **New Vitis HLS Project** dialog box. Enter **ex_HLS** in the *Project name* field and provide a suitable **Location**. Click **Next**.





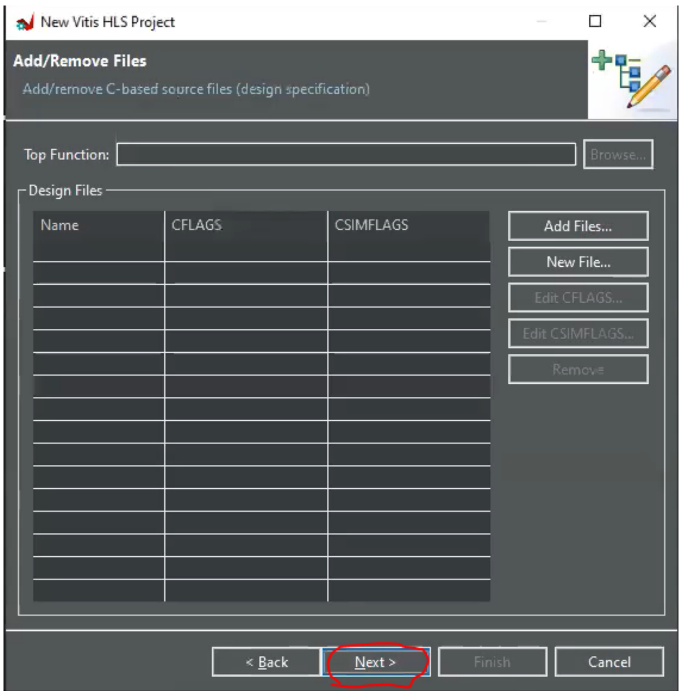
Project Configuration

Create Vitis HLS project of selected type

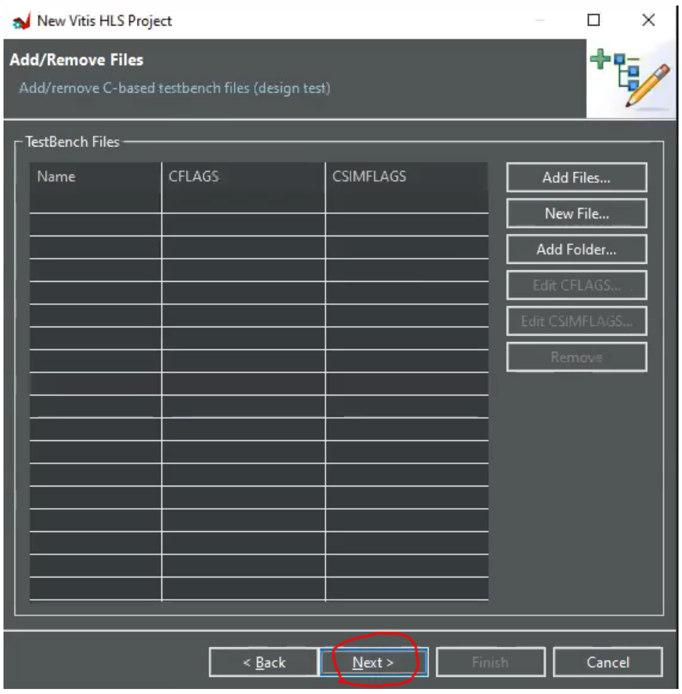
Project name: ex_HLS

Location: F:\EE3220

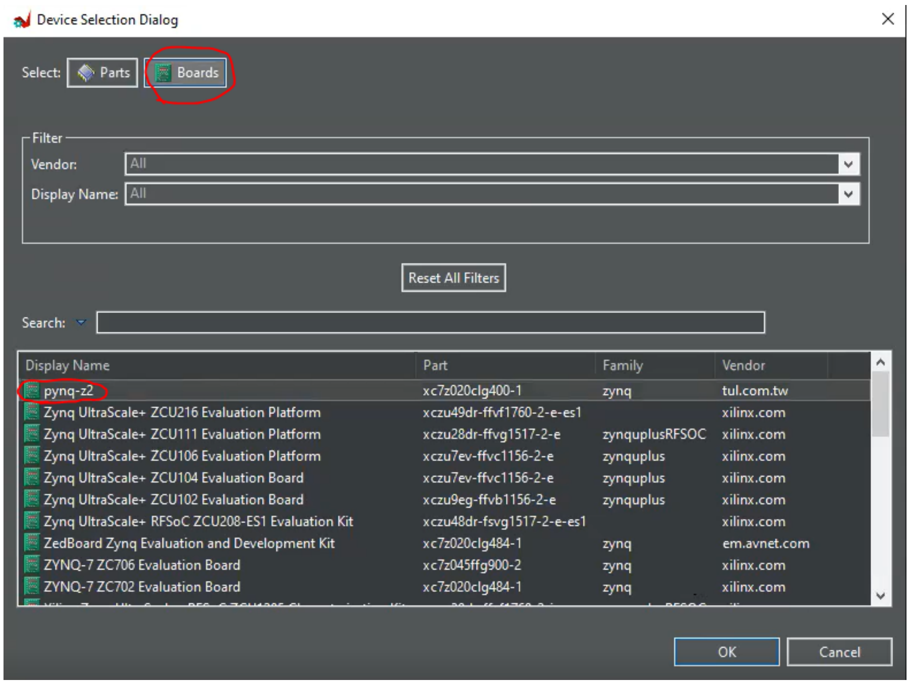
2. In the **Add/Remove Files** dialog box, click **Next**. We will create new **Design Files** for our design later.



In the **Add/Remove Files** dialog box, click **Next**. We will create new **TestBench Files** for our design later.

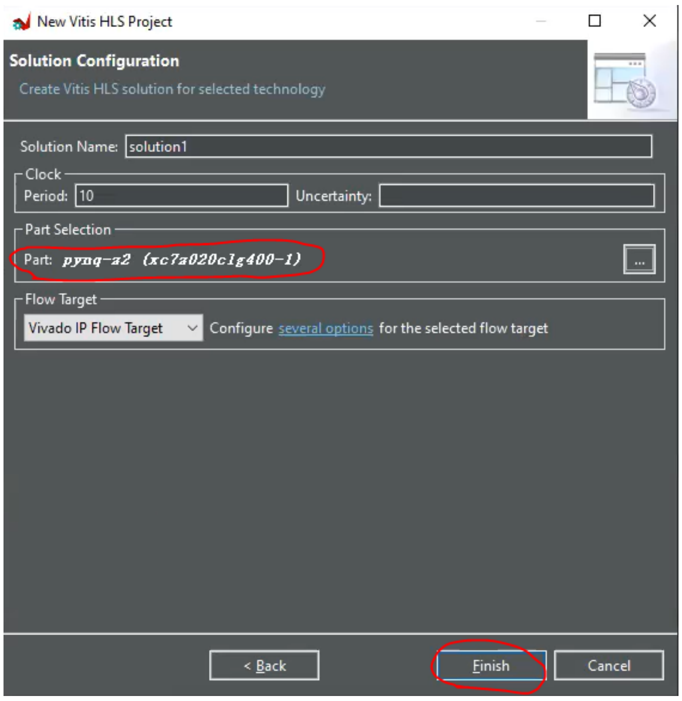


3. In **Solution Configuration**, choose **Boards** Selection and click **pynq-z2**, then click **ok**.



Display Name	Part	Family	Vendor
[✓ pynq-z2]	xc7z020clg400-1	zynq	tul.com.tw
Zynq UltraScale+ ZCU216 Evaluation Platform	xczu49dr-ffvf1760-2-e-es1		xilinx.com
Zynq UltraScale+ ZCU111 Evaluation Platform	xczu28dr-ffvg1517-2-e	zynquplusRFSOC	xilinx.com
Zynq UltraScale+ ZCU106 Evaluation Platform	xczu7ev-ffvc1156-2-e	zynquplus	xilinx.com
Zynq UltraScale+ ZCU104 Evaluation Board	xczu7ev-ffvc1156-2-e	zynquplus	xilinx.com
Zynq UltraScale+ ZCU102 Evaluation Board	xczu9eg-ffvb1156-2-e	zynquplus	xilinx.com
Zynq UltraScale+ RFSoC ZCU208-ES1 Evaluation Kit	xczu48dr-fsvg1517-2-e-es1		xilinx.com
ZedBoard Zynq Evaluation and Development Kit	xc7z020clg484-1	zynq	em.avnet.com
ZYNQ-7 ZC706 Evaluation Board	xc7z045ffg900-2	zynq	xilinx.com
ZYNQ-7 ZC702 Evaluation Board	xc7z020clg484-1	zynq	xilinx.com

Click **Finish**.



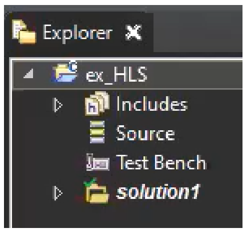
Solution Name: solution1

Clock
Period: 10 Uncertainty:

Part Selection
Part: *pynq-z2 (xc7z020clg400-1)*

Flow Target
Vivado IP Flow Target — Configure several options for the selected flow target

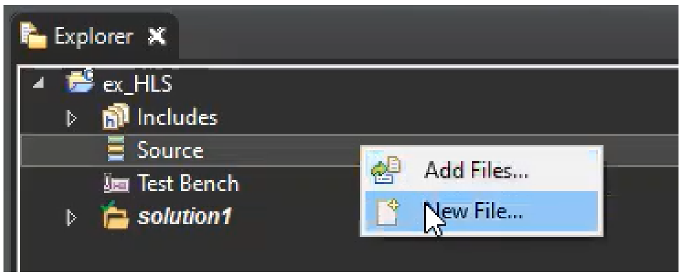
You will see a new empty HLS project with the following hierarchy. The **Source** contains your design files and your testbench files are in **Test Bench**.



B. Create new HLS files

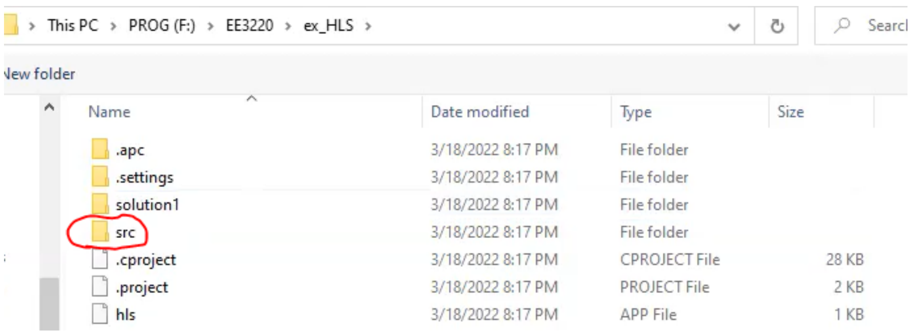
We are going to finish a simple design. The hardware module calculates the sum of two Integer inputs.

1. In the **Explorer**, right click **Source**, and select **New file...**.



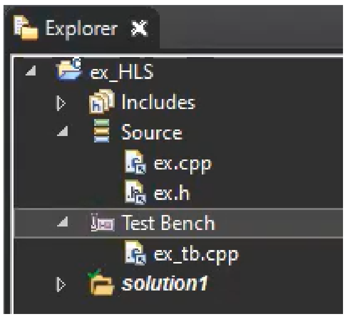
Create a new file

2. We recommend you **create a new folder** named **src** and put all your files inside the folder. Create a new file named **ex.cpp**. Click **Save**. Repeat this step and add another file **ex.h**.



Name	Date modified	Type	Size
.apc	3/18/2022 8:17 PM	File folder	
.settings	3/18/2022 8:17 PM	File folder	
solution1	3/18/2022 8:17 PM	File folder	
src	3/18/2022 8:17 PM	File folder	
.cproject	3/18/2022 8:17 PM	CPROJECT File	28 KB
.project	3/18/2022 8:17 PM	PROJECT File	2 KB
hls	3/18/2022 8:17 PM	APP File	1 KB

3. Similarly, right click **Test Bench**, and select **New file...**. Create a new testbech file named **ex_tb.cpp**. You can put the **ex_tb.cpp** file in the **src** folder as well. Now you will see the below hierarchy.



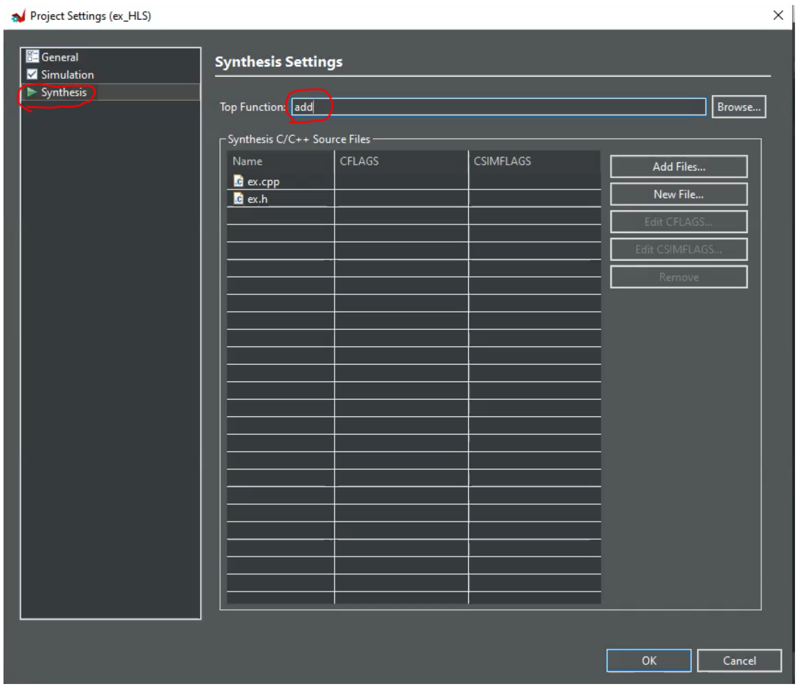
4. For this simple example we are going to design the **IP** that adds two 32-bit integers together. It would be better for you if you have basic concept about C/C++. Put the following code in the design file **ex.cpp**, and the full code for the accelerator is:

```
#include "ex.h"

void add(int a, int b, int& c)
{
#pragma HLS INTERFACE s_axilite port=return
#pragma HLS INTERFACE s_axilite port=a
#pragma HLS INTERFACE s_axilite port=b
#pragma HLS INTERFACE s_axilite port=c

    c = a + b;
}
```

In **ex.h**, put following code

```
#ifndef __EX__
#define __EX__

void add(int a, int b, int& c);

#endif
```

5. In the code, `#pragma` provides optimizations and techniques to direct Vitis HLS to produce a micro-architecture that satisfies the desired performance and area goals. The `INTERFACE` defines how RTL ports are created. Our simple design uses an **AXI-4-Lite interface** so that the design can be controlled by a CPU.
6. From menu bar, **Project->Project Settings...**, select **Synthesis**, fill in **Top function** as **add**


Project Settings (ex_HLS)

General
Simulation
Synthesis

Synthesis Settings

Top Function: add

Synthesis C/C++ Source Files

Name	CFLAGS	CSIMFLAGS
ex.cpp		
ex.h		

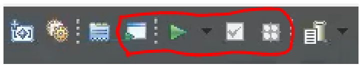
7. Put following code in the testbench file **ex_tb.cpp**:

```
#include <iostream>
#include "ex.h"

using namespace std;

int main()
{
    int a;
    int b;
    int gc;
    int c;
    for (a = -100; a < 100; a++)
    {
        for (b = -100; b <= 100; b++)
        {
            gc = a + b;
            add(a,b,c);
            if (gc != c)
            {
                cout << "Error when a=" << a << ", b=" << b;
                return 1;
            }
        }
    }
    return 0;
}
```

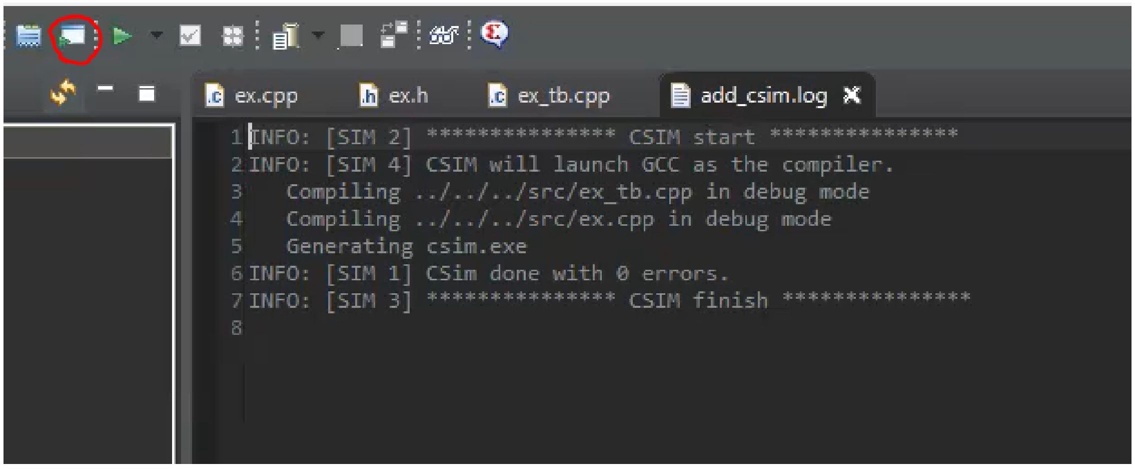
7. In HLS, all design files and testbench files are written in high-level languages such as C++. In Xilinx HLS, the hardware module is defined as a C++ function. The testbench is the main function that will call your designed function. The results are compared with the golden standard. If the main function returns 0, the design is correct, otherwise the simulation fails.

C. Run C Simulation, C Synthesis, C/RTL Cosimulation, and export RTL design

At the **top** of HLS tool, find four buttons, they are **C Simulation, C Synthesis, C/RTL Cosimulation**, and **export RTL design**, respectively.

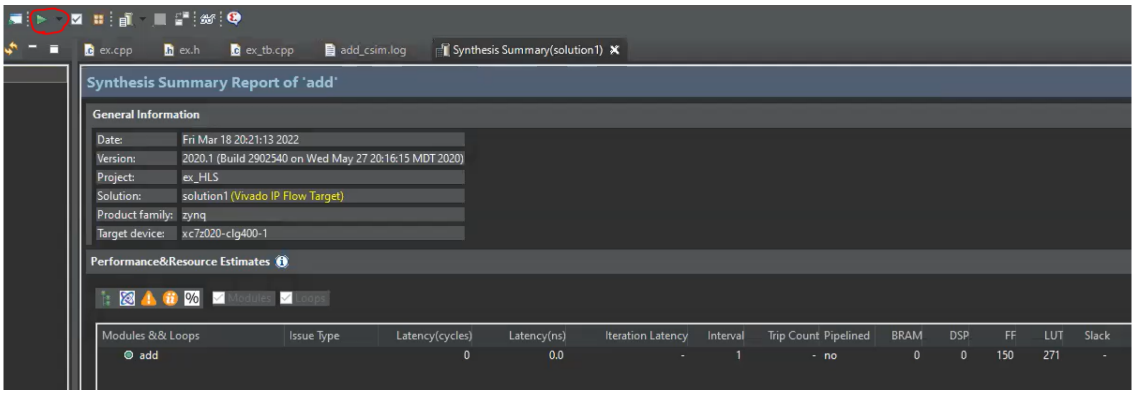


1. Click **Run C Simulation**， then click **OK** in the new dialog. It will compile your testbench file in pure software level and do the functional simulation. If everything is correct, you will get the result with 0 error.



```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../src/ex_tb.cpp in debug mode
   Compiling ../../../src/ex.cpp in debug mode
   Generating csim.exe
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

2. Click **C Synthesis**, wait until it finishes. This step translate your C++ design file into Verilog/VHDL. After it finishes, you will get a report about estimated latencty and resource consumption of your design.



Synthesis Summary Report of 'add'

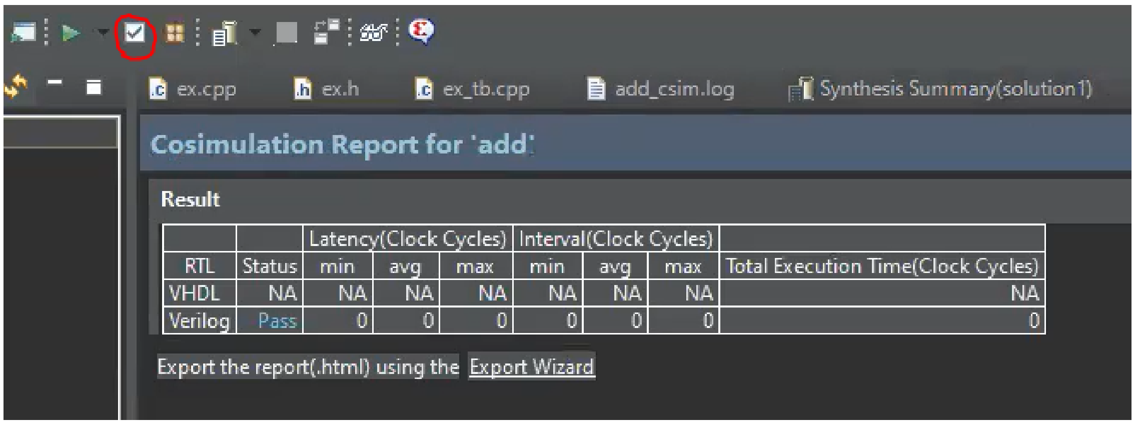
General Information

Date:	Fri Mar 18 20:21:13 2022
Version:	2020.1 (Build 2902540 on Wed May 27 20:16:15 MDT 2020)
Project:	ex_HLS
Solution:	solution1 (Vivado IP Flow Target)
Product family:	zynq
Target device:	xc7z020-clg400-1

Performance&Resource Estimates

Modules && Loops	Issue Type	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	Slack
● add		0	0.0	-	1	-	no	0	0	150	271	-

3. Click **Run C/RTL Cosimulation**, here we use default settings, click **OK** in the new dialog. This step translate your testbench file and does the RTL simulation with XSim. Based on the coverage ratio of your testbench, it takes several minutes. And you will get a report when it finishes.



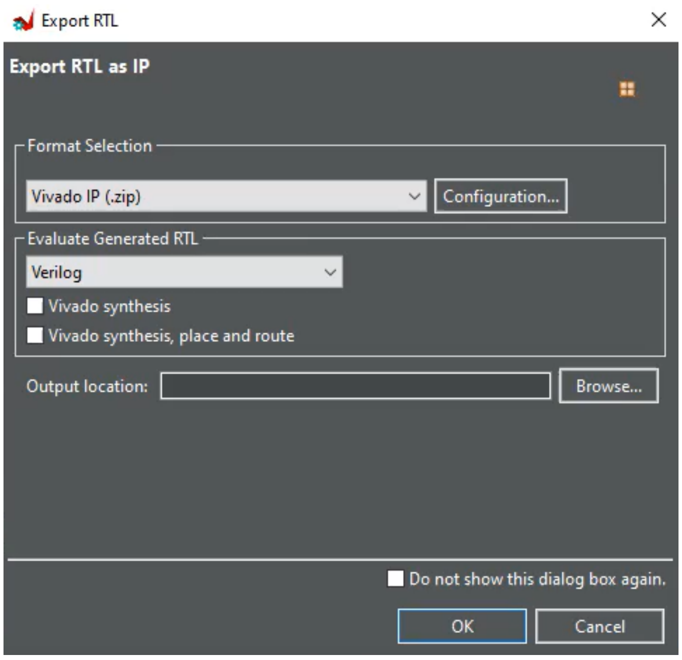
Cosimulation Report for 'add'

Result

		Latency(Clock Cycles)			Interval(Clock Cycles)			
RTL	Status	min	avg	max	min	avg	max	Total Execution Time(Clock Cycles)
VHDL	NA	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	0	0	0	0	0	0	0

Export the report(.html) using the Export Wizard

4. At last, click **Export RTL**. We select Verilog and click **OK** in the new dialog. Wait until it finishes.



Format Selection: Vivado IP (.zip)

Evaluate Generated RTL: Verilog

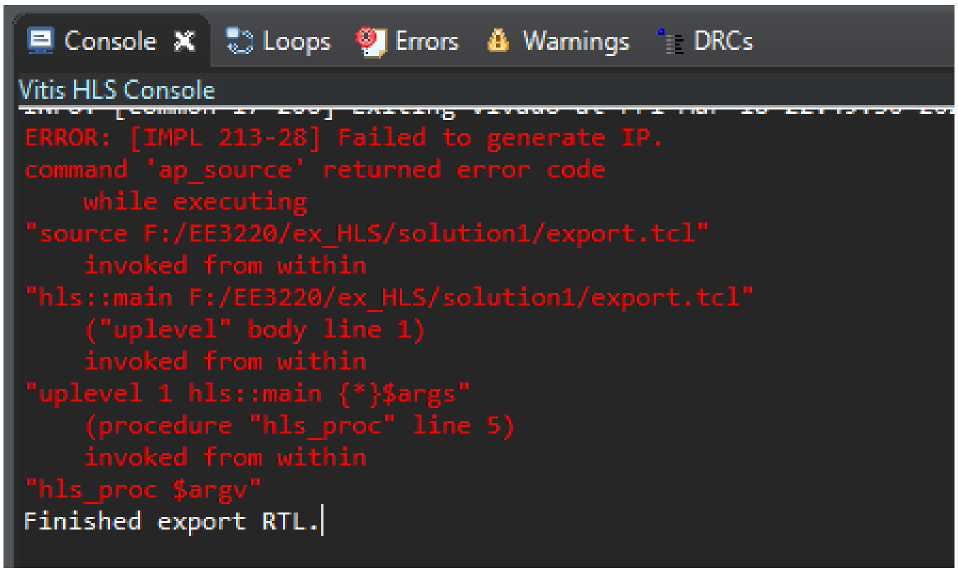
☐ Vivado synthesis

☐ Vivado synthesis, place and route

Output location:

☐ Do not show this dialog box again.

You may face the error:



```
ERROR: [IMPL 213-28] Failed to generate IP.
command 'ap_source' returned error code
    while executing
"source F:/EE3220/ex_HLS/solution1/export.tcl"
    invoked from within
"hls::main F:/EE3220/ex_HLS/solution1/export.tcl"
    ("uplevel" body line 1)
    invoked from within
"uplevel 1 hls::main {*}$args"
    (procedure "hls_proc" line 5)
    invoked from within
"hls_proc $argv"
Finished export RTL.
```

We provide the solution:

You can refer to https://support.xilinx.com/s/article/76960?language=en_US for detail.

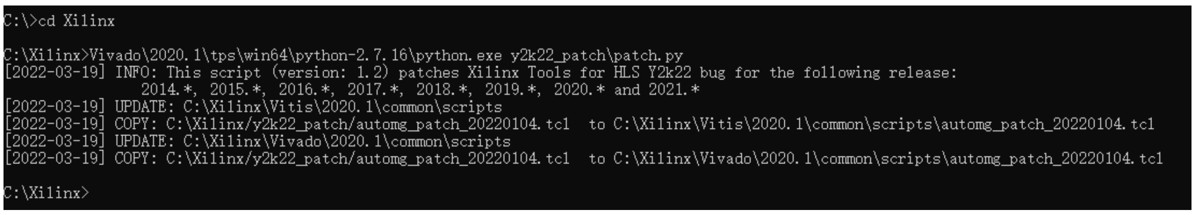
For example, we use **Vitis HLS 2020.1** on **Windows**:

Please download the **y2k22_patch-1.2.zip** patch at the bottom of https://support.xilinx.com/s/article/76960?language=en_US , then decompress and move the patch under the **installation root location**, e.g., C:\Xilinx\y2k22_patch.

Then you need enter into the **installation root location** (e.g., C:\Xilinx) in the **Command Prompt** and input the following command line:

`vivado\2020.1\tps\win64\python-2.7.16\python.exe y2k22_patch\patch.py`

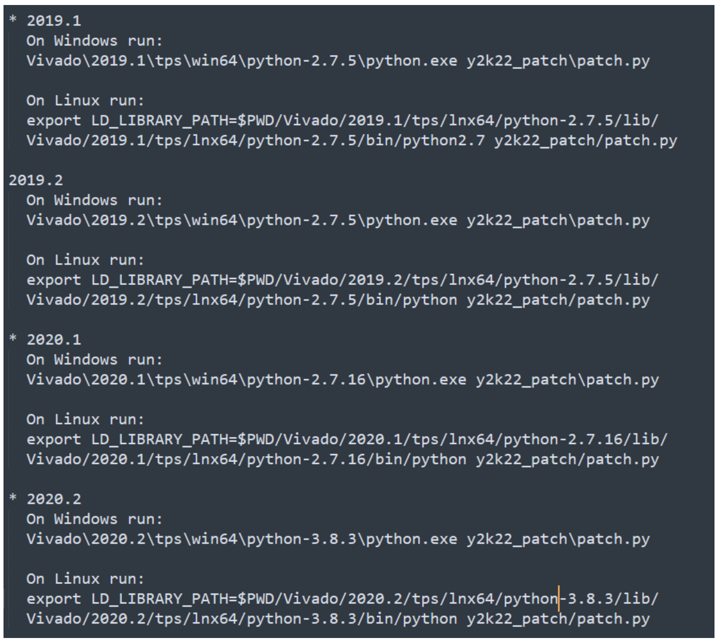
We can see the following output:



```
C:\>cd Xilinx

C:\Xilinx>Vivado\2020.1\tps\win64\python-2.7.16\python.exe y2k22_patch\patch.py
[2022-03-19] INFO: This script (version: 1.2) patches Xilinx Tools for HLS Y2k22 bug for the following release:
              2014.*, 2015.*, 2016.*, 2017.*, 2018.*, 2019.*, 2020.* and 2021.*
[2022-03-19] UPDATE: C:\Xilinx\Vitis\2020.1\common\scripts
[2022-03-19] COPY: C:\Xilinx/y2k22_patch/automg_patch_20220104.tcl  to C:\Xilinx\Vitis\2020.1\common\scripts\automg_patch_20220104.tcl
[2022-03-19] UPDATE: C:\Xilinx\Vivado\2020.1\common\scripts
[2022-03-19] COPY: C:\Xilinx/y2k22_patch/automg_patch_20220104.tcl  to C:\Xilinx\Vivado\2020.1\common\scripts\automg_patch_20220104.tcl

C:\Xilinx>
```

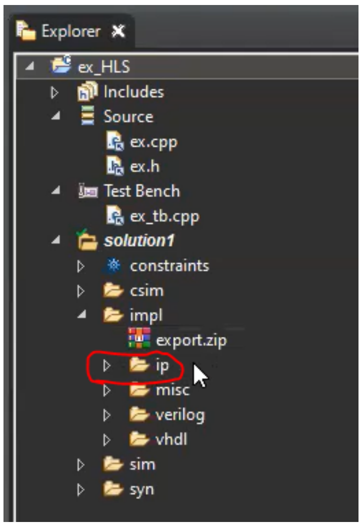
The patch process requires Python.

If you install other version of Vivado and Vitis (including HLS) versions 2014.x through 2021.2, the decompression of **y2k22_patch-1.2.zip** patch under the **installation root location**, Python and entering into the **installation root location** for inputting the specific command line are also necessary. Please open the **README** file in the **y2k22_patch** file folder, and follow the instructions for the **specific version** that you are applying the patch to. For example, you can find the proper command line from the following figure in the **README** file mentioned above:

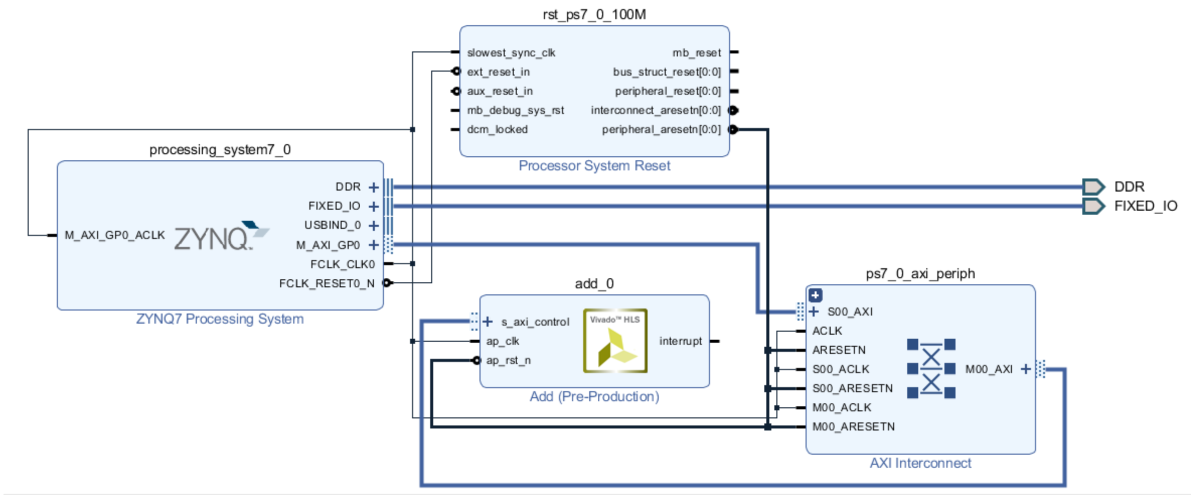


```
* 2019.1
  On Windows run:
  Vivado\2019.1\tps\win64\python-2.7.5\python.exe y2k22_patch\patch.py

  On Linux run:
  export LD_LIBRARY_PATH=$PWD/Vivado/2019.1/tps/lnx64/python-2.7.5/lib/
  Vivado/2019.1/tps/lnx64/python-2.7.5/bin/python2.7 y2k22_patch/patch.py

2019.2
  On Windows run:
  Vivado\2019.2\tps\win64\python-2.7.5\python.exe y2k22_patch\patch.py

  On Linux run:
  export LD_LIBRARY_PATH=$PWD/Vivado/2019.2/tps/lnx64/python-2.7.5/lib/
  Vivado/2019.2/tps/lnx64/python-2.7.5/bin/python y2k22_patch/patch.py

* 2020.1
  On Windows run:
  Vivado\2020.1\tps\win64\python-2.7.16\python.exe y2k22_patch\patch.py

  On Linux run:
  export LD_LIBRARY_PATH=$PWD/Vivado/2020.1/tps/lnx64/python-2.7.16/lib/
  Vivado/2020.1/tps/lnx64/python-2.7.16/bin/python y2k22_patch/patch.py

* 2020.2
  On Windows run:
  Vivado\2020.2\tps\win64\python-3.8.3\python.exe y2k22_patch\patch.py

  On Linux run:
  export LD_LIBRARY_PATH=$PWD/Vivado/2020.2/tps/lnx64/python-3.8.3/lib/
  Vivado/2020.2/tps/lnx64/python-3.8.3/bin/python y2k22_patch/patch.py
```

We click **Export RTL**. We select Verilog and click **OK** in the new dialog again. Wait until it finishes. We can check the generated IP.



Save your project folder. In the next tutorial, we will build the SoC system with this simple IP and test on the Pynq board. And you are expected to build a system in the following figure, with Zynq PS, Processor System Reset, the Add IP built by Vitis HLS in this tutorial, and AXI Interconnect.



In the figure above, we can see how the created IP block is integrated to the overall SoC. We will elaboate more details in the next tutorial 10.

Reference

[1] https://pynq.readthedocs.io/en/latest/overlay_design_methodology.html

[2] https://pynq.readthedocs.io/en/latest/overlay_design_methodology/overlay_tutorial.html

[3] https://support.xilinx.com/s/article/76960?language=en_US

End