SMR4110 Photo Album

Joint ICTP-IAEA School on Detector Signal Processing and Machine Learning for Scientific Instrumentation and Reconfigurable Computing

The Abdus Salam International Centre for Theoretical Physics

IAEA International Atomic Energy Agency

unesco United Nations Educational, Scientific and Cultural Organization

PSoC Design Methodology

Cristian Sisterna

Senior Associate, ICTP-MLAB

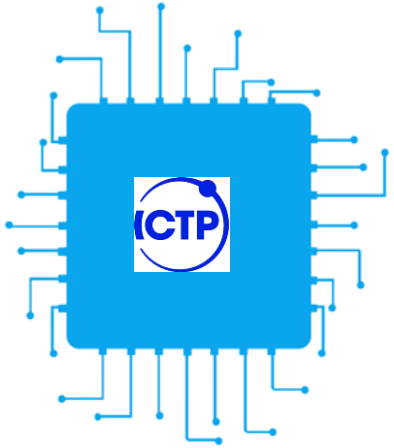
Professor at Universidad Nacional San Juan- Argentina

Agenda

01 PSoC Design Flow

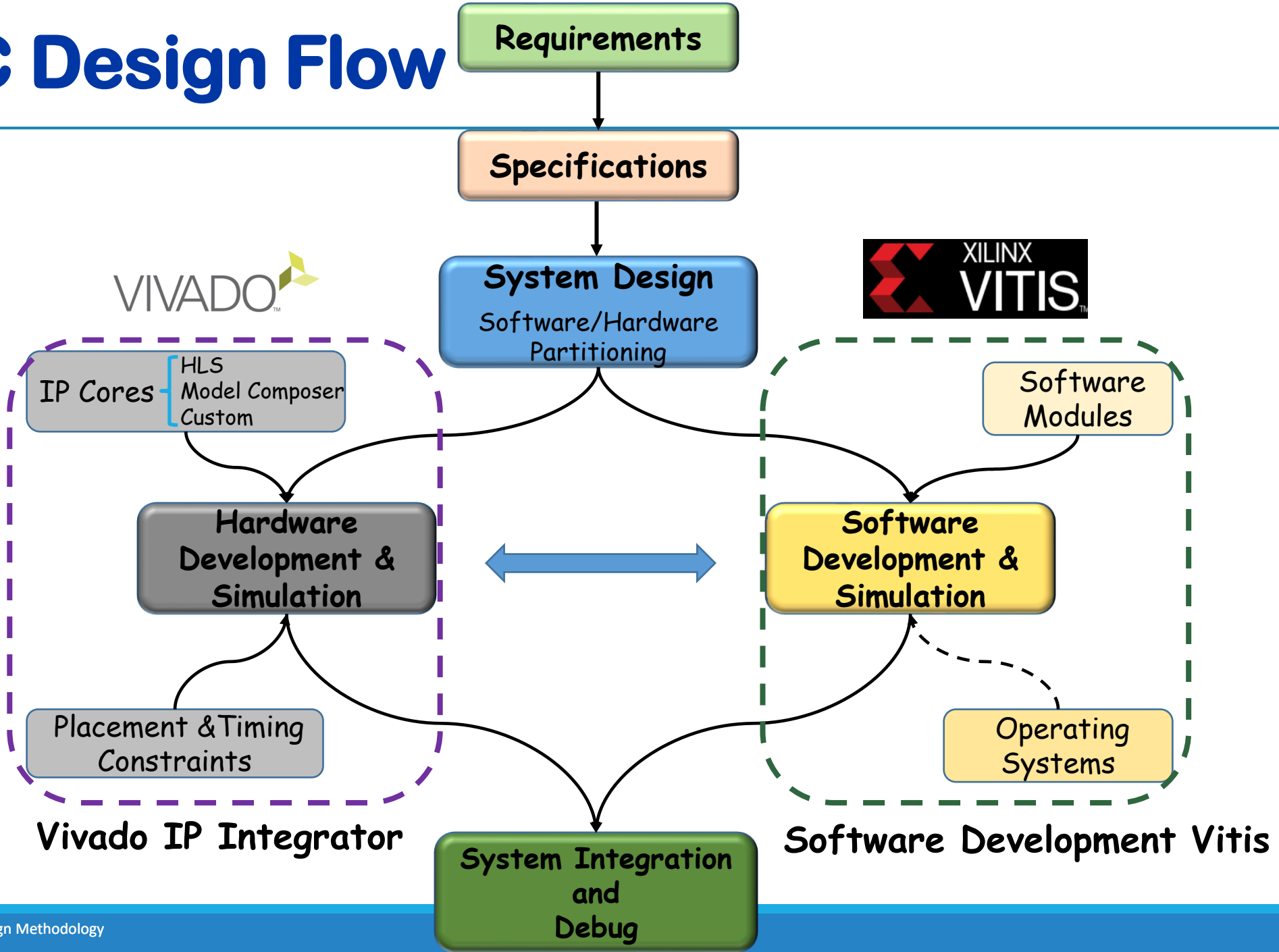
02 PSoC Design Methodoloy

03 TCL – Overview and Usage

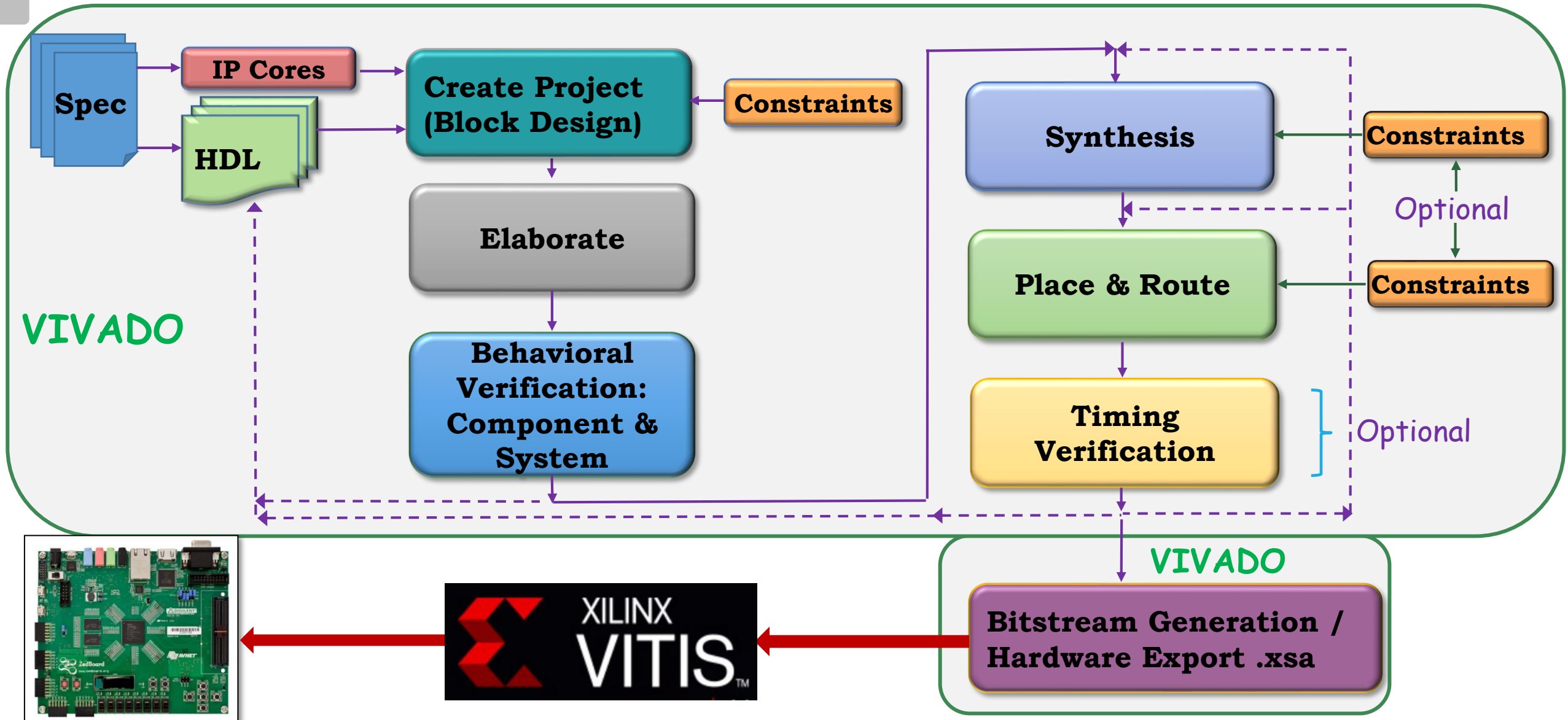


PSoC Design Methodology

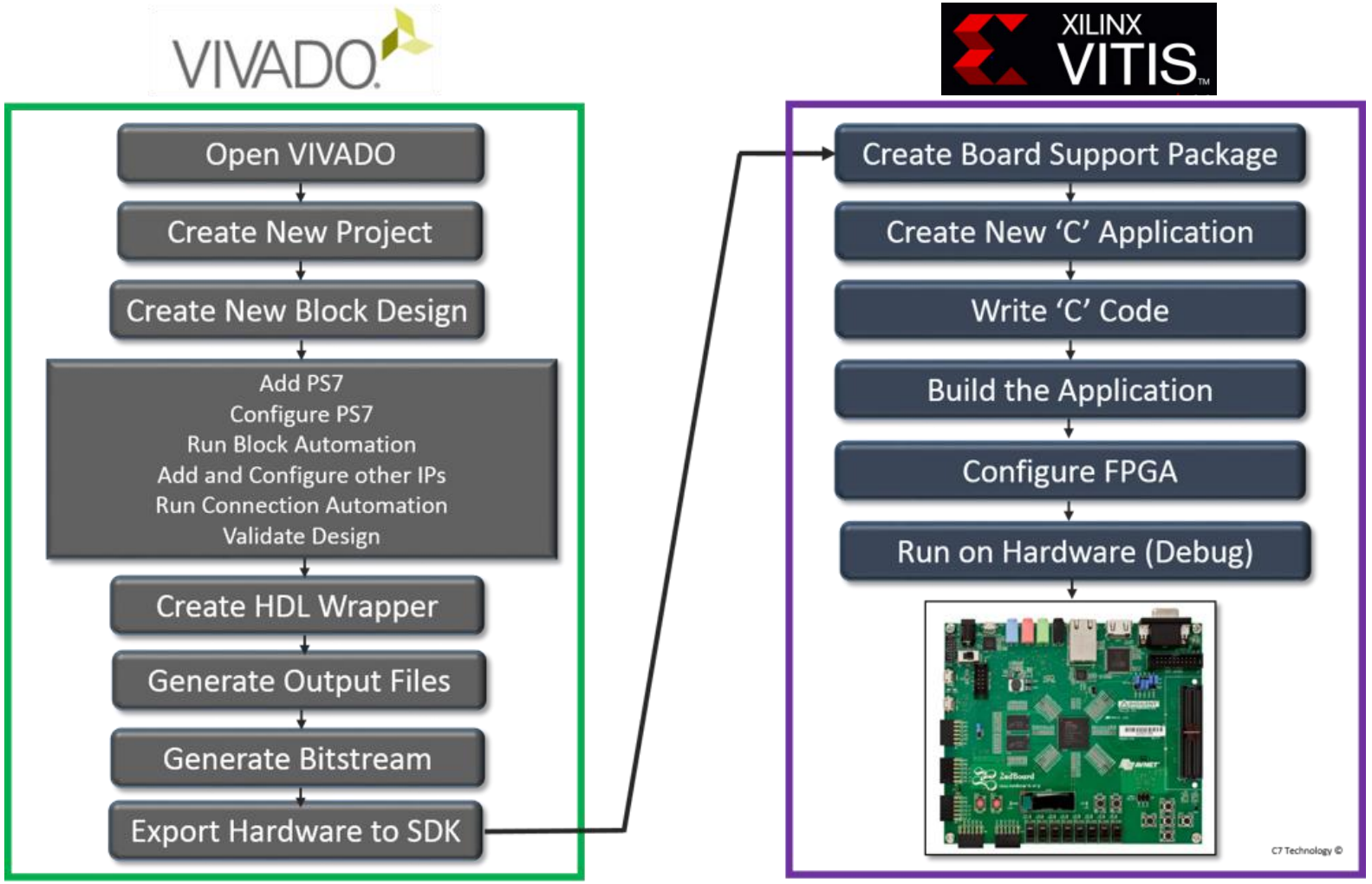
PSoC Design Flow



PSoC Design – PL Design Flow



PSoC Design – Vivado-Vitis Design Flow



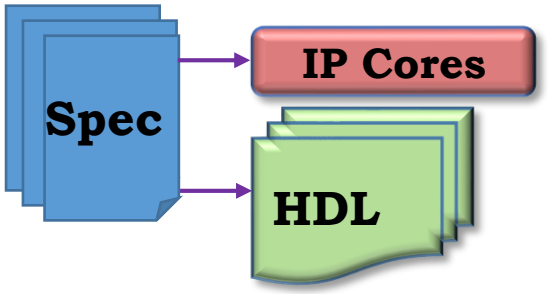
PL Design Specification

- ✓ Written specifications (spec for short) for the design to be done
- ✓ Spec can specify:
  - ✓ Functionality
  - ✓ Timing
  - ✓ Interfaces
  - ✓ Power

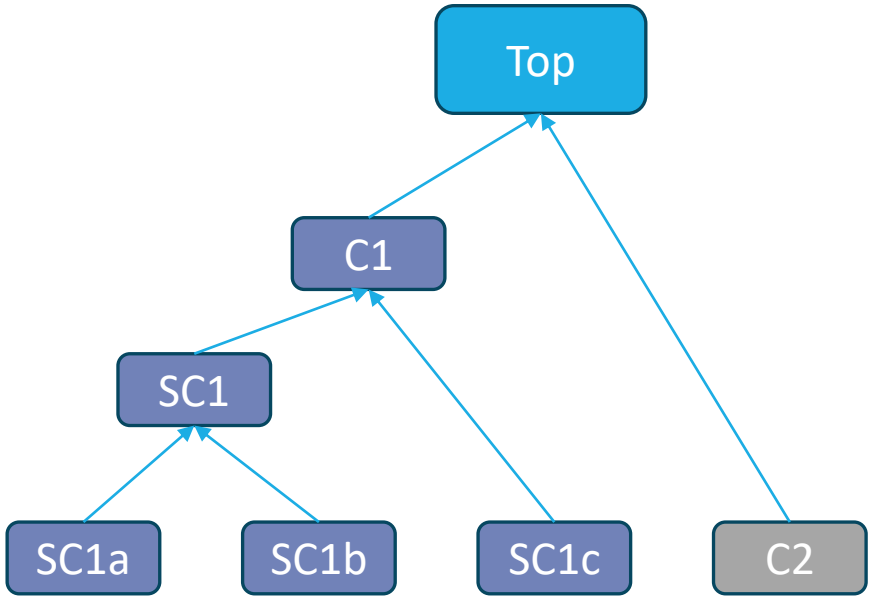
PL Design Partition

- ✓ Divide and conquer strategy
- ✓ Complex design is progressively partitioned into smaller and simpler functional units. This is known as

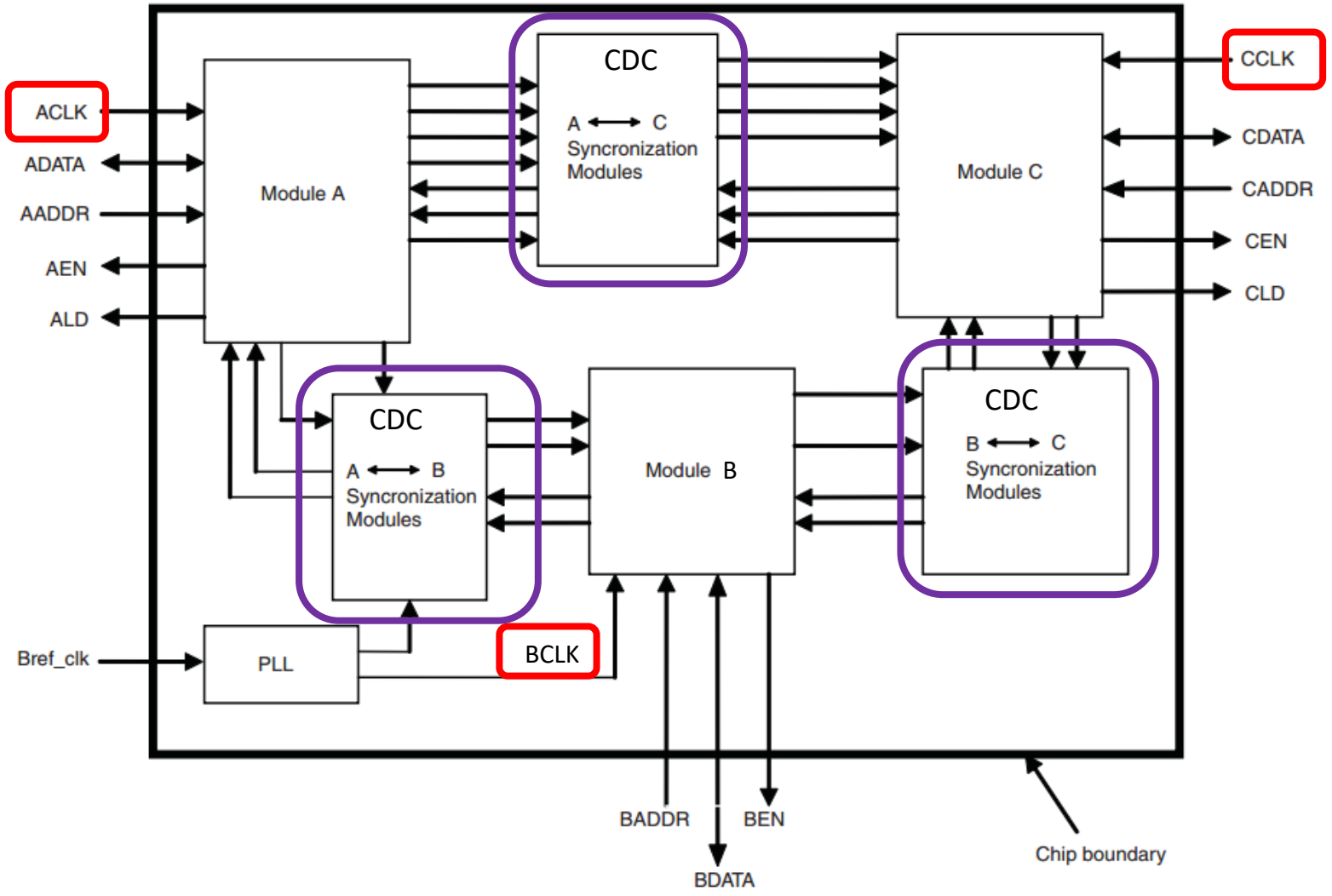
Top-Down Design or Hierarchical Design



- ✓ Behavioral model for each functional unit are written
  - ✓ It has its own synthesis results
  - ✓ It has it own functional test bench
  - ✓ Some cases it has its own place and route and timing constraints



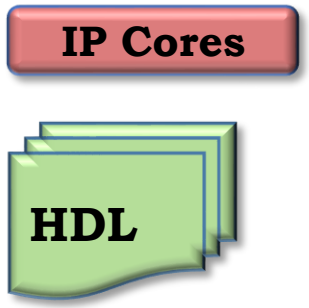
PL Design Partition - Clocking



CDC: Clock Domain Crossing

PL Design Partition

- ✓ **Partition the design at functional boundaries.**
- ✓ **Minimize the I/O connections between different partitions.**
- ✓ **Register all inputs and outputs of each block.** This makes logic synchronous and avoids glitches and avoids any delay penalty on signals that cross between partitions. Registering I/Os typically eliminates the need to specify timing requirements for signals that connect between different blocks.
- ✓ **Do not use "glue logic" or connection logic between hierarchical blocks**. When you preserve hierarchy boundaries, glue logic is not merged with hierarchical blocks. Your synthesis software may optimize glue logic separately, which can degrade synthesis results and is not efficient when used with the LogicLock design methodology.
- ✓ Remember that logic is not synthesized or optimized across partition boundaries, which means any constant values (signals set to GND, for example) will not be propagated across partitions**.**



PL Design Partition

- **Do not use tri-state signals or bidirectional ports on hierarchical boundaries.** If you use boundary tri-states in a lower-level block, synthesis pushes the tri-states through the hierarchy to the top-level to take advantage of the tri-state drivers on the output pins of Xilinx devices.
- **Limit clocks to one per block. Partitioning the design into clock domains makes synthesis and timing analysis easier.**
- **Partitioning the design into clock domains makes synthesis and timing analysis easier.**
- **Place state machines in separate blocks to speed optimization and provide greater encoding control.**
- **Separate timing-critical functions from non-timing-critical functions.**
- **Limit the critical timing path to one hierarchical block.** You can group the logic from several design blocks to ensure the critical path resides in one block

PL HDL Components

- **Behavioral modeling** describes the functionality of a component (design), that is, what the component will do
  - A behavioral prototype of a component can be quicly created
  - Its functionality verified
  - Synthesized, optimized and mapped, to a specific technology

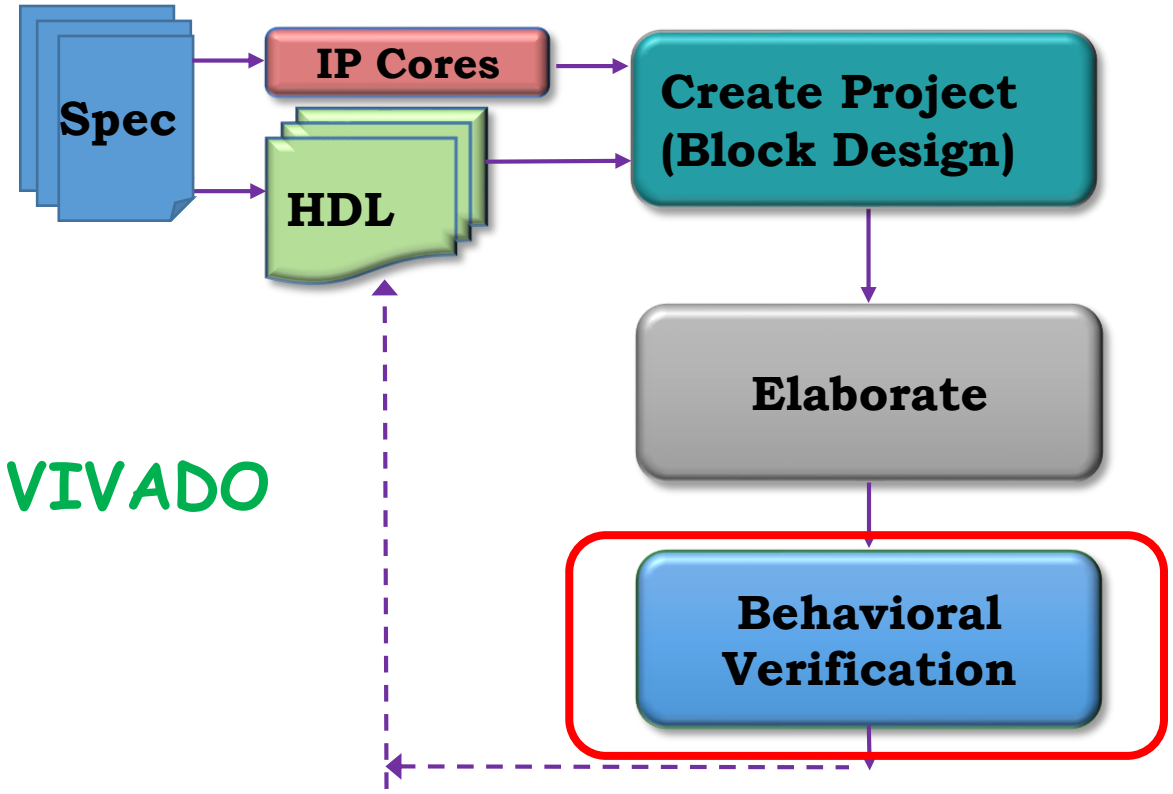
- **Structural Modeling** connect components to create a specific functionality.
  - Architectural partitioning forms a structural model, but the functional components are modeled behaviorally

PL Functional Verification – Block Level

- In general, a design should be partitioned along functional lines into smaller functional units, each having a common clock domain, and each of which is to be verified separately.

✓ The verification process is threefold:

- ✓ Development of a test plan
- ✓ Development of test-benches
- ✓ Execution of the simulations



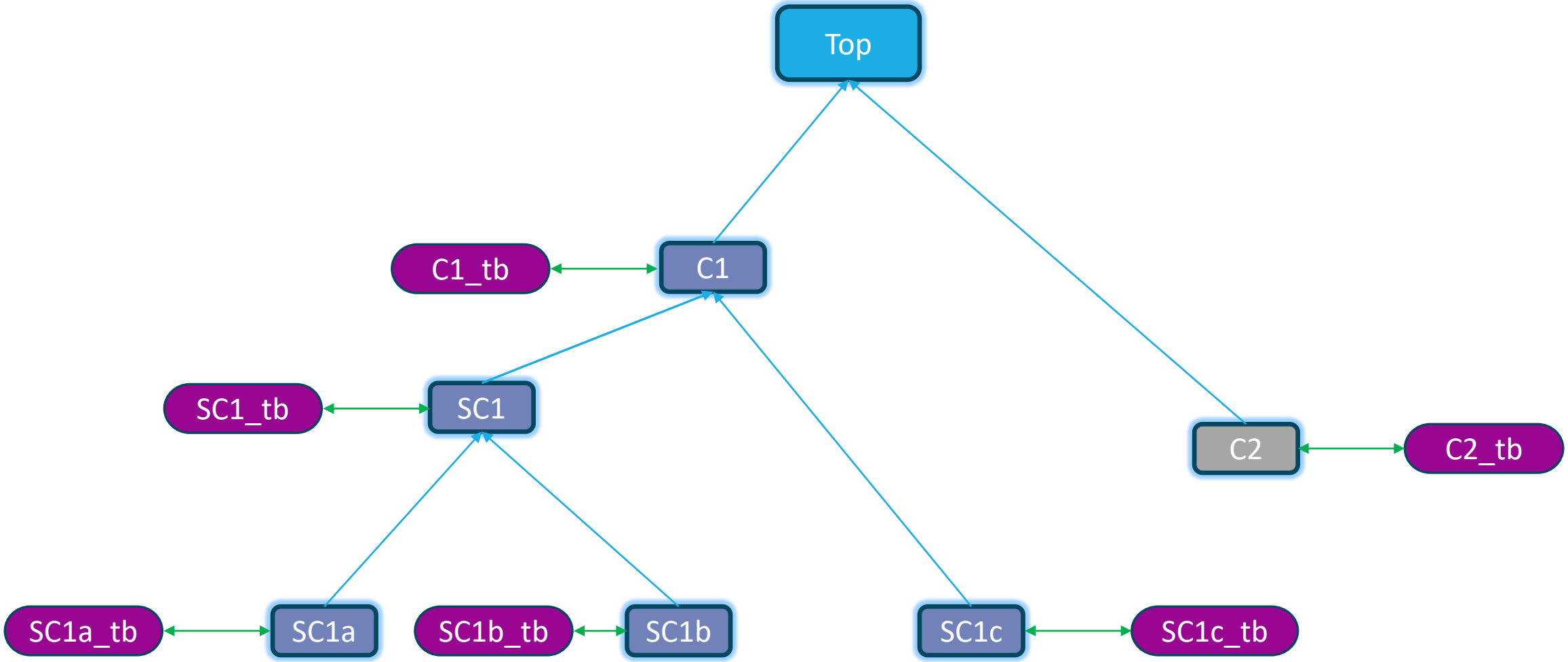
PL Functional Verification – Block Level

- ✓ Development of a **test plan:** Specify ***what*** functional features are to be tested and ***how*** they are to be tested
  - ✓For example, the test plan might specify that an exhaustive simulation of its behavior will verify the instruction set of an ALU.
  - ✓Test plans for sequential machines must be more elaborate to ensure a high level of confidence in the design.
  - ✓A test plan identifies the stimulus generators, response monitors, and the "gold" response against which the model will be tested.
  - ✓***Your grade, and your company's future, will depend on the care that you take in developing and executing your test plan.***

PL Functional Verification – Block Level

- **Test bench development**
  - A Test Bench is a VHDL module in which the Unit Under Test (UUT) has to be instantiated together with pattern generators that are to be applied to the inputs of the component during simulation.
  - Note: If a design is formed as an architecture of multiple modules, each must be verified separately, beginning with the lowest level of the design hierarchy, then the integrated design must be tested to verify that the modules interact correctly
- **Test bench development Test bench execution**
  - The test bench is ***exercised*** according to a ***test plan***, and the response is verified against the original specification for the design

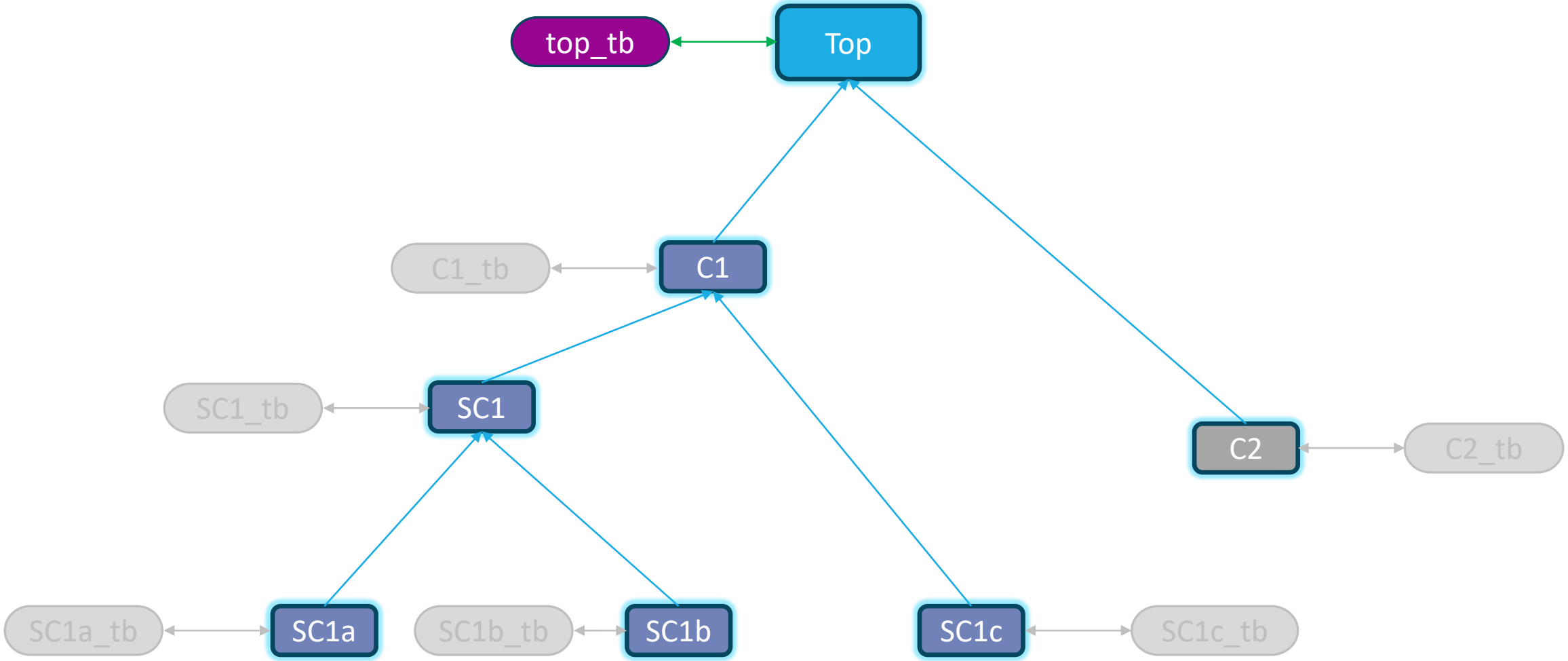
PL Functional Verification – Block Level



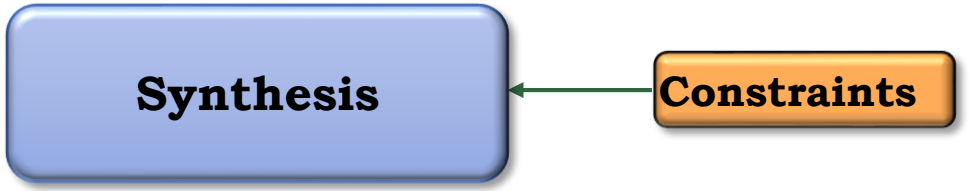
PL Design Integration – System Level Verification

- After each of the functional sub components has been verified to have correct functionality, the architecture must be integrated and verified to have the correct overall functionality
  - A separate *test plan for the system* is developed at the beginning of this step.
  - This requires development of a separate testbench whose stimulus generators exercise the input/output functionality of the top-level module, monitor port and bus activity across module boundaries, and observe state activity in any embedded state machines.
  - This step in the design flow is crucial and must be executed thoroughly to ensure that the design that is being signed off for synthesis is correct.

PL Design Integration – System Level Verification

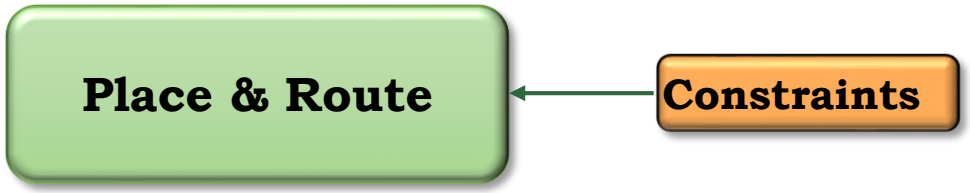


PL Synthesis Constraints – Post-Synthesis Simulation



- A *synthesis* tool is used to translate from 'software' (VHDL) to 'hardware': logic gates, flip-flops, memory, etc.
- A synthesis tool removes redundant logic and seeks to satisfy the requirements regarding the area of the logic needed to implement the functionality and the performance (speed) specifications
- Post-Synthesis simulation is, in general, **optional**, but it is advisable in case of using specific synthesis attributes.

PL Place & Route



- ✓ The logic generated by the synthesis tool is a netlist, commonly known as EDIF netlist, that is take by the Place and Route tool to scatter the logic in the FPGA's resources.
- ✓ P&R tool has different effort levels, which can be used in case the final result does not meet the needed requirements.
- ✓ In complex design:
  - ✓ P&R could take several hours to accomplish its task.
  - ✓ Some ***floor planning*** may be needed (constraints).

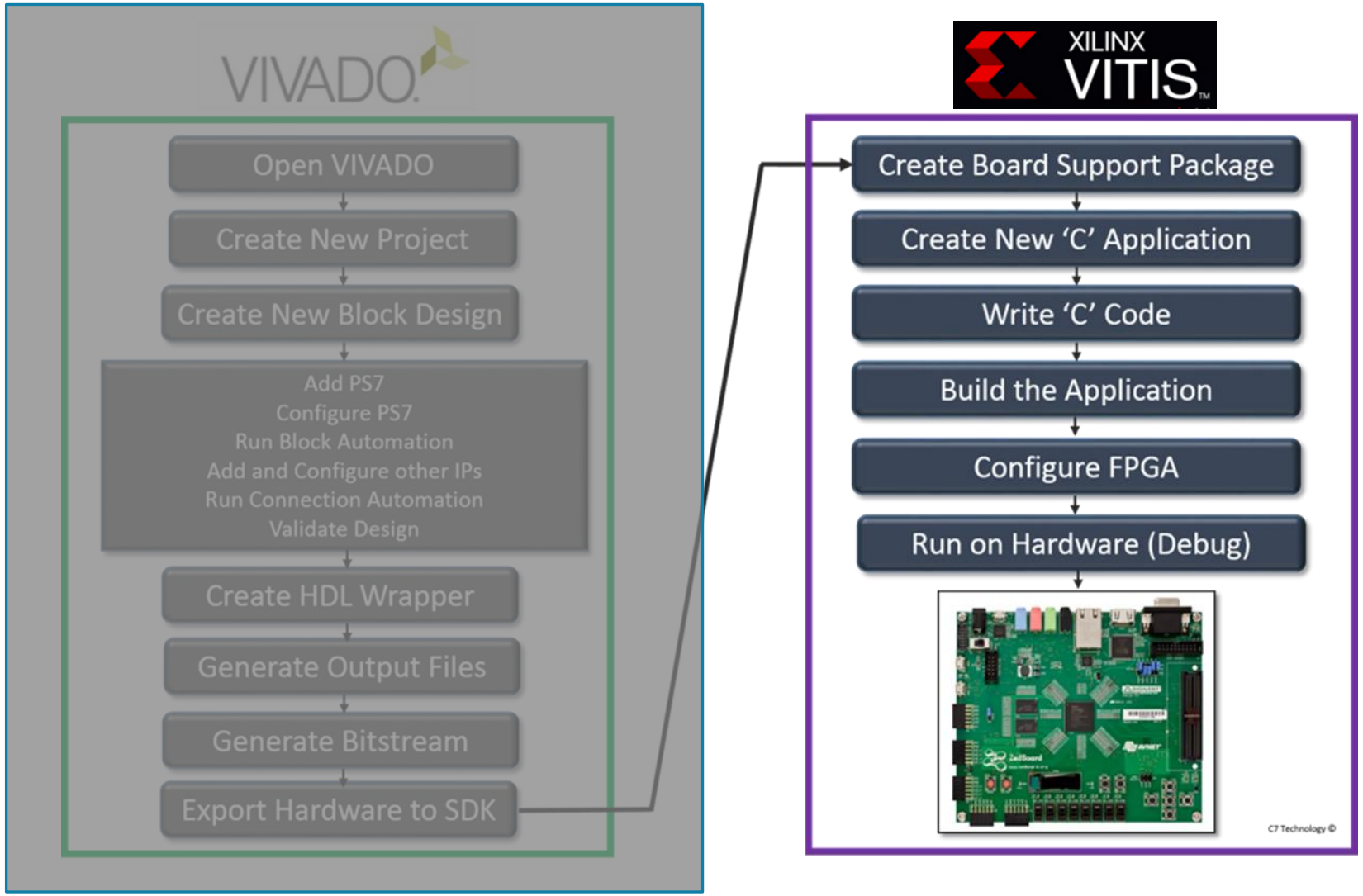
PL Post Place & Route Simulation

- ✓ The simulation test (test bench) not only test the logical/behavioural functionality but also the timing of the whole system:
  - ✓ Routing and logic delay are taking into consideration when executing this simulation
  - ✓ Each delay is well known after the P&R
  - ✓ Hold-time and Set-up time violations can be found out in this simulation as well as any glitch

Export Hardware Platform (.xsa file)

- Once the bitstream is generated, export the hardware platform as an ***XSA*** (Xilinx Shell Architecture) file.
- This file encapsulates all the hardware information, including the bitstream, address map, and peripheral details, which will be used by Vitis.

Vitis Flow Design

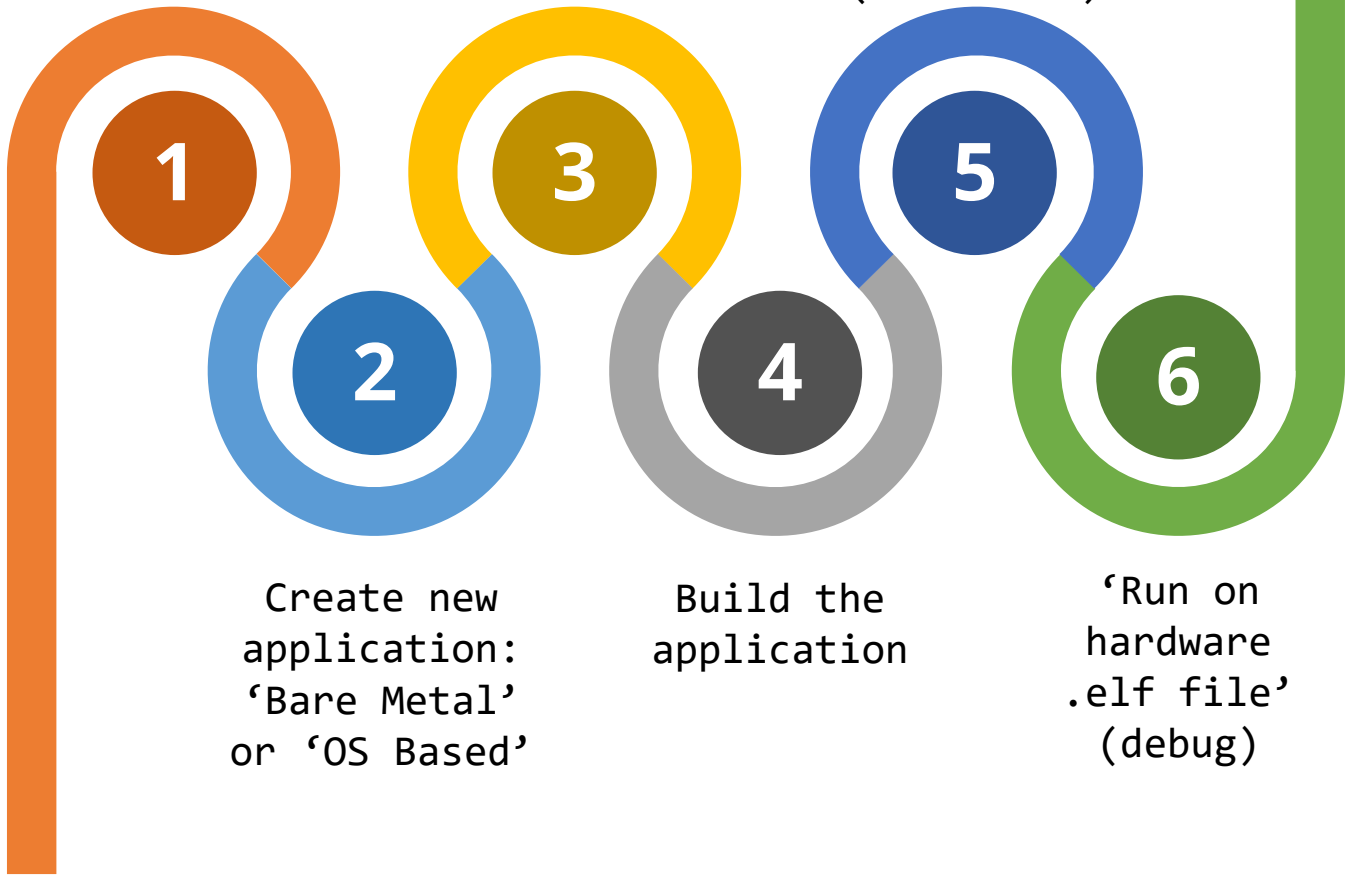


The **AMD Vitis** environment provides a comprehensive design flow for ***Zynq SoCs***.

The **Vitis flow** emphasizes a software-centric approach, allowing developers to create and write an application in bare-metal 'C' code or run an Operating System, such as FreeRTOS, PetaLinux, etc.

From the Vitis application can easily access to the logic on the PL or any peripheral in the PS.

Vitis Design Flow



XILINX VITIS

Vitis Design Flow

The **Vitis** environment takes over from the **XSA** file to facilitate software development.

Create a Platform Project: Import the XSA file generated from Vivado to create a Vitis platform project. This platform serves as the base for your software applications, providing information about the hardware components and their addresses. You can choose to create platforms for:

- **Standalone (Bare-metal) applications:** For simple, direct control of hardware.
- **FreeRTOS applications:** For real-time operating system based designs.
- **Linux applications:** If you plan to run a full Linux OS on the Zynq's PS (Petalinux).

generation).

Create an Application Project: Based on the platform, create a new application project. This is where you write your C/C++ code.

- You can select from various templates (e.g., "Hello World", memory tests) or start with an empty application.
- For accelerated applications, you can define "kernels" (functions intended to run on the PL) which will be compiled and implemented in the FPGA fabric.

Vitis Design Flow

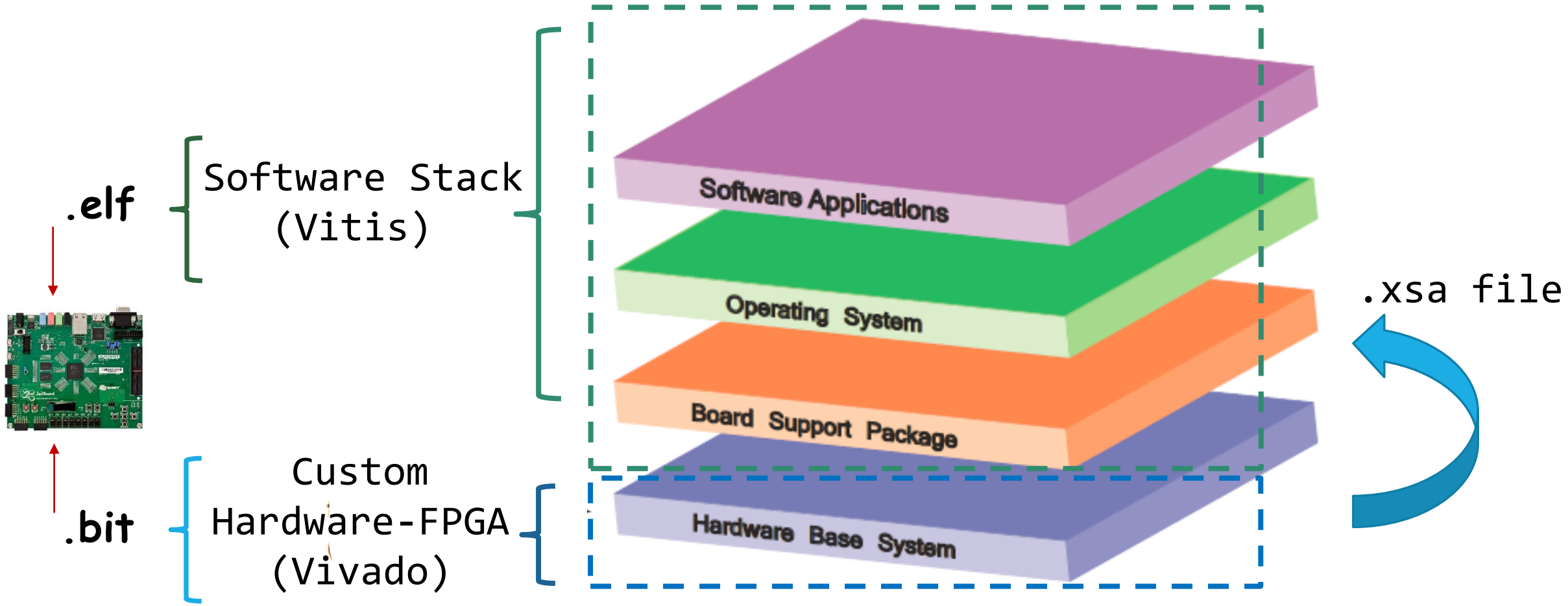
Write Software Application: Host Application (for PS): This is the C/C++ code that runs on the ARM processor(s) of the Zynq. It interacts with the PL via AXI interfaces.

Build the Project: Vitis compiles your software application and links it with the necessary libraries and the hardware platform. For accelerated designs, this also involves building the PL kernels and generating the final bitstream if it includes custom hardware.

Generate Boot Image: Once your application is stable, generate a boot image (e.g., BOOT.BIN) that includes the First Stage Boot Loader (FSBL), the bitstream, and your application. This image is used to boot the Zynq device

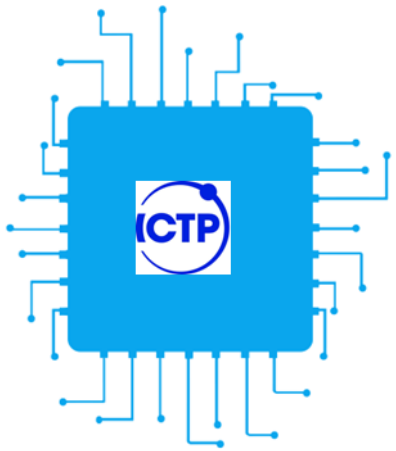
Program and Run on Hardware: Load the boot image onto your Zynq board (e.g., via SD card, QSPI flash, or JTAG) and run your application.

Vitis Design Flow - .elf .bit Download



Vitis - Running the App. - .elf .bit Download

- ✓ First of all, configure the PL part of the Zynq, using the .bit file
- ✓ Download the .elf file to the PS (processor) part of the Zynq.
- ✓ Execute the C/Linux application
- ✓ Check the result into either:
  - ✓ Terminal
  - ✓ Console
  - ✓ 3rd party terminal emulator



Basics of TCL in Vivado

Tool Command Language

TCL , is *an interpreted programming language* with variables, procedures , and control structures, to interface to *a variety of design tools and to design data.*

It has been an industry standard language since early 90s’

AMD-Xilinx adopted TCL for the Vivado Design Suite

Tool Command Language (cont)

TCL in Vivado enables the designer to:

- Create a project
- Target a SoPC device/board
- Create a block design
- Include IP Cores
- Configure PS, IP Cores, etc.
- Run synthesis
- Run implementation
- Modify P&R options
- Customize reports
- Program SoPC

Tool Command Language (cont)

```
#-----------------------------------------------------------
# Vivado v2018.3.1 (64-bit)
# SW Build 2489853 on Tue Mar 26 04:18:30 MDT 2019
# IP Build 2486929 on Tue Mar 26 06:44:21 MDT 2019
# Start of session at: Wed May 22 20:07:21 2019
# Process ID: 19219
# Current directory: /cris_projects
# Command line: vivado
# Log file: /cris_projects/vivado.log
# Journal file: /cris_projects/vivado.jou
#-----------------------------------------------------------
start_gui
create_project project_1 /cris_projects/ZedBoard/borrar/hw -part
xc7z020clg484-1
set_property board_part em.avnet.com:zed:part0:1.4
[current_project]
set_property target_language VHDL [current_project]
create_bd_design "design_1"
update_compile_order -fileset sources_1
startgroup
create_bd_cell -type ip -vlnv
xilinx.com:ip:processing_system7:5.5 processing_system7_0
endgroup
apply_bd_automation -rule
xilinx.com:bd_rule:processing_system7 -config {make_external
"FIXED_IO, DDR" apply_board_preset "1" Master "Disable" Slave
"Disable" }  [get_bd_cells processing_system7_0]
generate_target all [get_files
/cris_projects/ZedBoard/borrar/hw/project_1.srcs/sources_
1/bd/design_1/design_1.bd]
startgroup
```

How to run a provided .tcl script

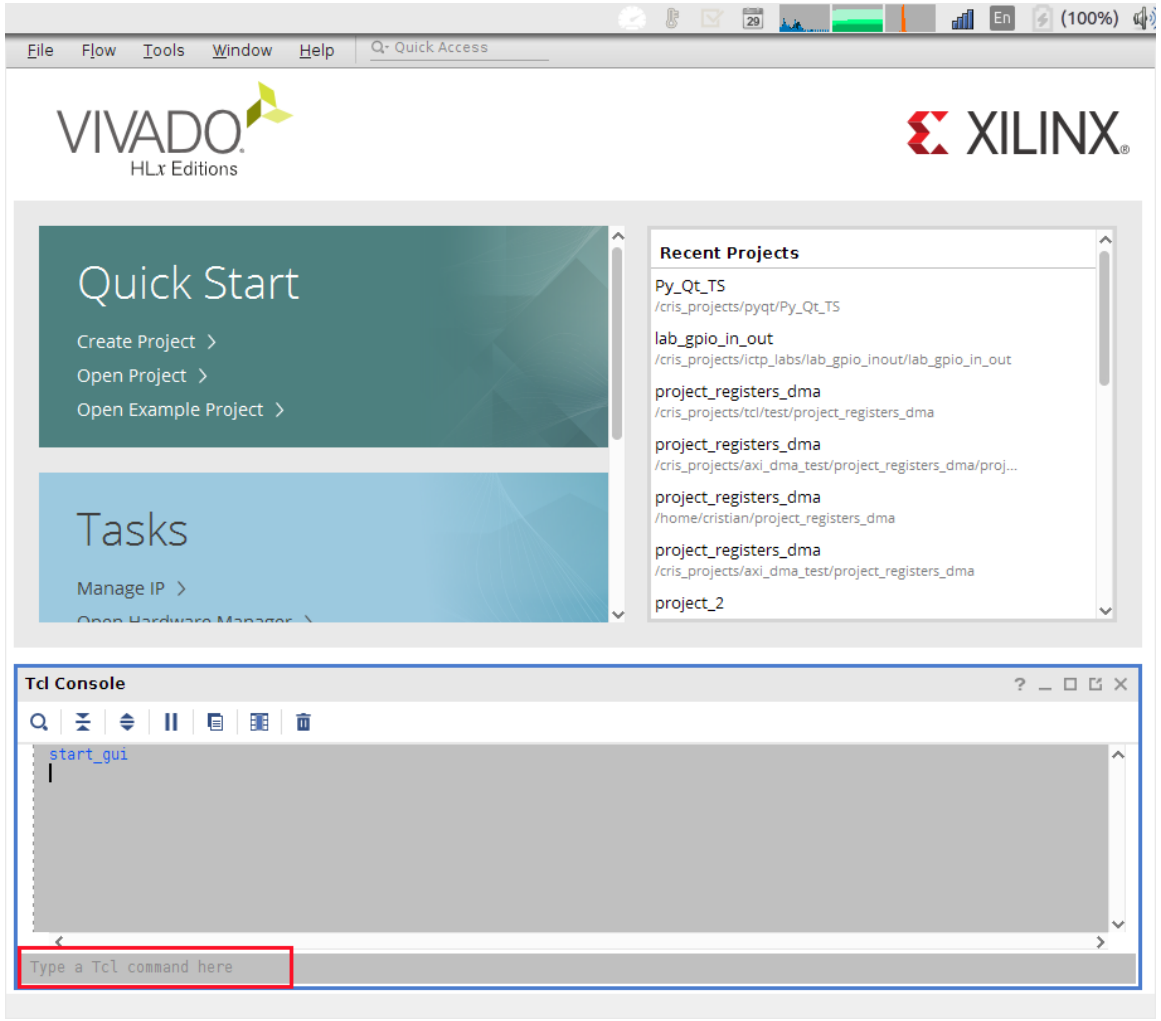
- Methot 1: Through Vivado TCL console
- Method 2: Through Command Line

Method 1: Run .tcl in Vivado TCL Console

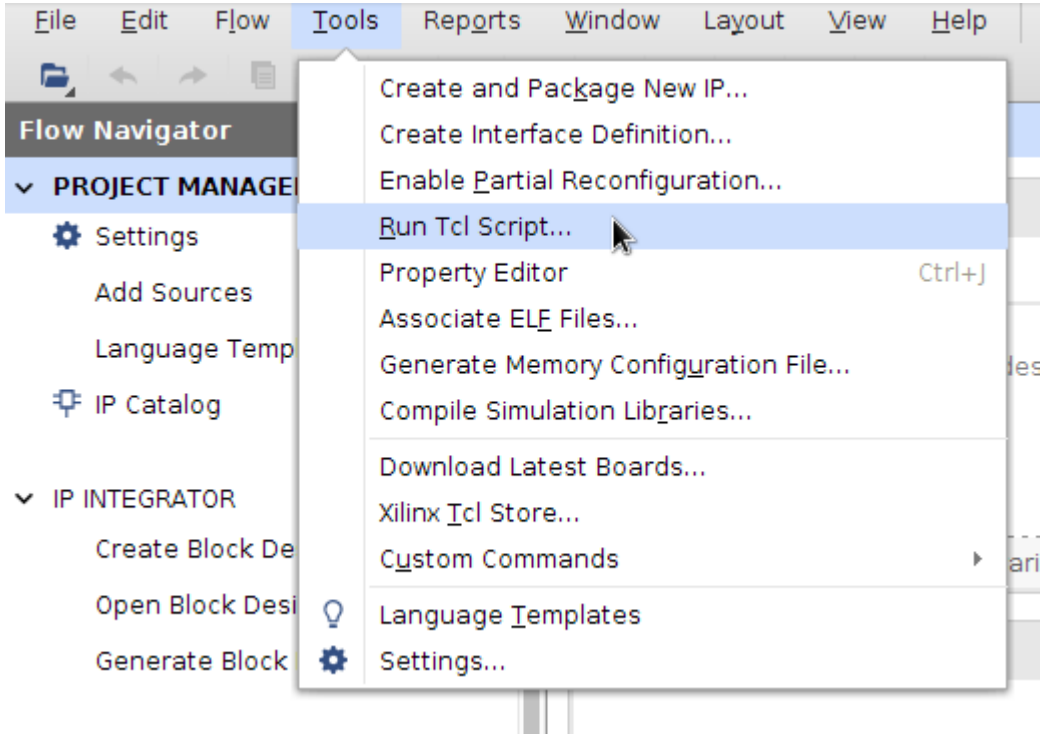
1. Start *Vivado Design Suite*. You can see a *tcl* console on the left bottom of *Vivado Design Suite*
2. Click on the title *'type a tcl command here'* (button left of the screen)
3. Go to the folder location where the tcl script resides (use 'cd', 'pwd')
4. Once the directory has been changed, you can use the *'ls'* command to list the files in the current directory. Check that the .tcl is in there.
5. Run the .tcl script by using the following command:

   source <filename>.tcl
6. The processes defined in the .tcl file will be executed. It could take sometimes to execute a .tcl file (depending on the defined processes)

Vivado TCL Console



Vivado TCL Option in the GUI



Method 2: Run .tcl through Command Line W10/11

1. In W10 you can start the Vivado TCL Shell by doing:

   Start-> All apps->Vivado 20xx.x Tcl Shell

2. A small command line window should come up
3. Go to the folder location where the tcl script resides (use ‘cd’, ‘pwd’)
4. Once the directory has been changed, you can use the *‘dir’* command to list the files in the current directory. Check that the **.tcl** is in there.
5. Run the **.tcl** script by using the following command:

   source <filename>.tcl

6. The processes defined in the .tcl file will be executed. It could take some times to execute a .tcl file (depending on the defined processes)

Run .tcl in Linux

1. Make sure TCL interpreter is installed:

```
$whereis tclsh
tclsh: /usr/bin/tclsh /usr/bin/tclsh8.4 /usr/share/man/man1/tclsh.1.gz
```

2. In case you don't have the tcl interpreter installed, do the following:

```
$ sudo apt-get install tcl8.4
```

Note: if you have already installed Vivado, the Tcl interpreter should be installed

3. Execute TCL script:

```
$ tclsh helloworld.tcl
```

(or)

```
$ chmod u+x helloworld.tcl
$ ./helloworld.tcl
```

Is there any Need to Learn TCL ?

It is purely based on your objectives.

If you want *to automate some basic processes* in creating design , **it is the best choice** as we can export a ***tcl*** script to another computer *and create an exact replica of the project* with same configurations, ip integrations in just a single execution.

Xilinx TCL Docs

Vivado Design Suite TCL Command Reference Guide

Vivado Design Suite User Guide - Using TCL Scripting

TCL Tutorial (up to Chapter 14 for Vivado appl)