Joint ICTP-IAEA
School on FPGA-based SoC
and its Applications for
Nuclear and Related
Instrumentation

# Vivado Design Flow for SoC

**Cristian Sisterna**
Universidad Nacional de San Juan
Argentina

# Why Vivado Design Suite?

Larger FPGAs lead to more difficult design issues

- Users integrating more functionality into the FPGA
  - Use of multiple hard logic objects (block RAMs, GTs, DSP slices, and microprocessors, for example)
- I/O and clock planning critical to FPGA performance
- Higher routing and utilization density
- Complex timing constraints with designs that have multiple clock domains

FPGA designs are now looking like ASIC platform designs

- Assembled from IP cores—commercial or developed in-house
  - Maintaining place and route solutions is very important (this is resolved with the use of partitions)
  - Bottom-up design methodology
- Team design flows becoming a necessity

***Vivado Design Suite** provides solution to all of the above*

# Vivado IDE Solution

## Interactive design and analysis

- Timing analysis, connectivity, resource utilization, timing constraint analysis

## RTL development and analysis

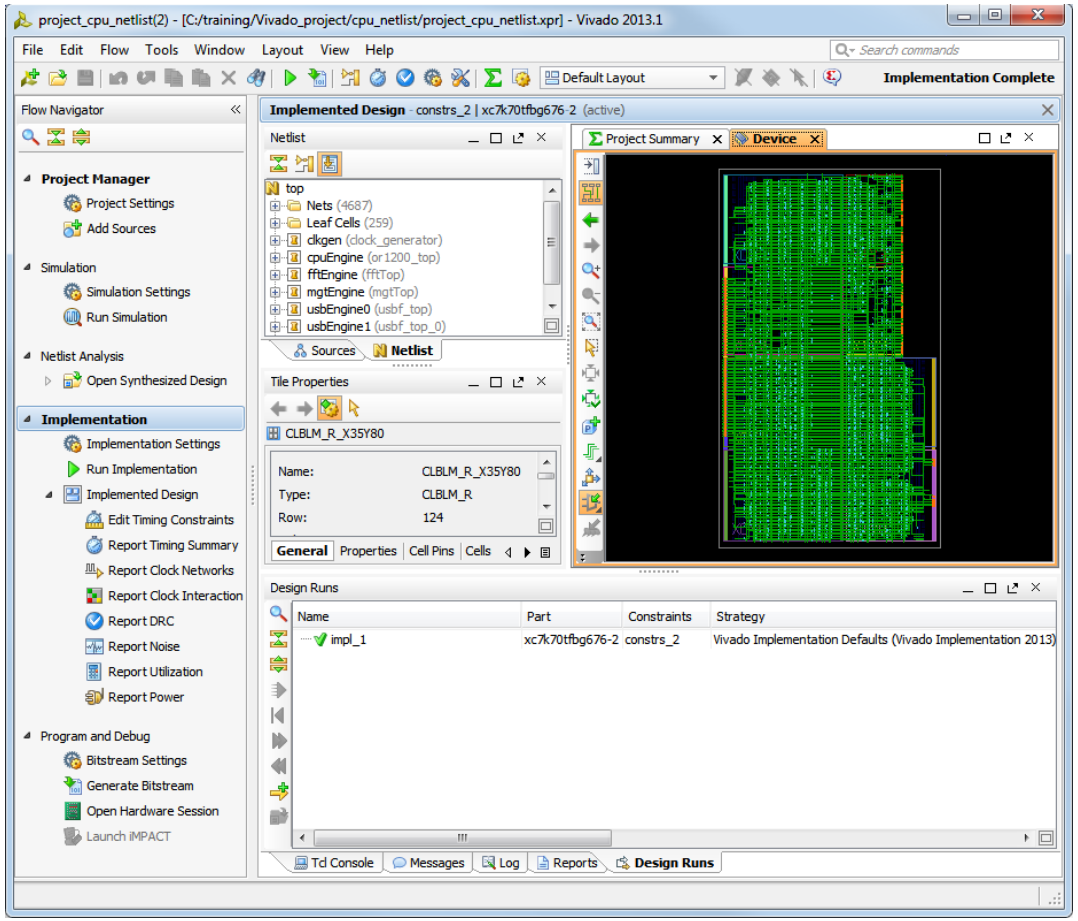
- Elaboration of HDL
- Hierarchical exploration
- Schematic generation

## XSIM simulator integration

- Synthesis, implementation and simulation in one package

## I/O pin planning

- Interactive rule-based I/O assignment

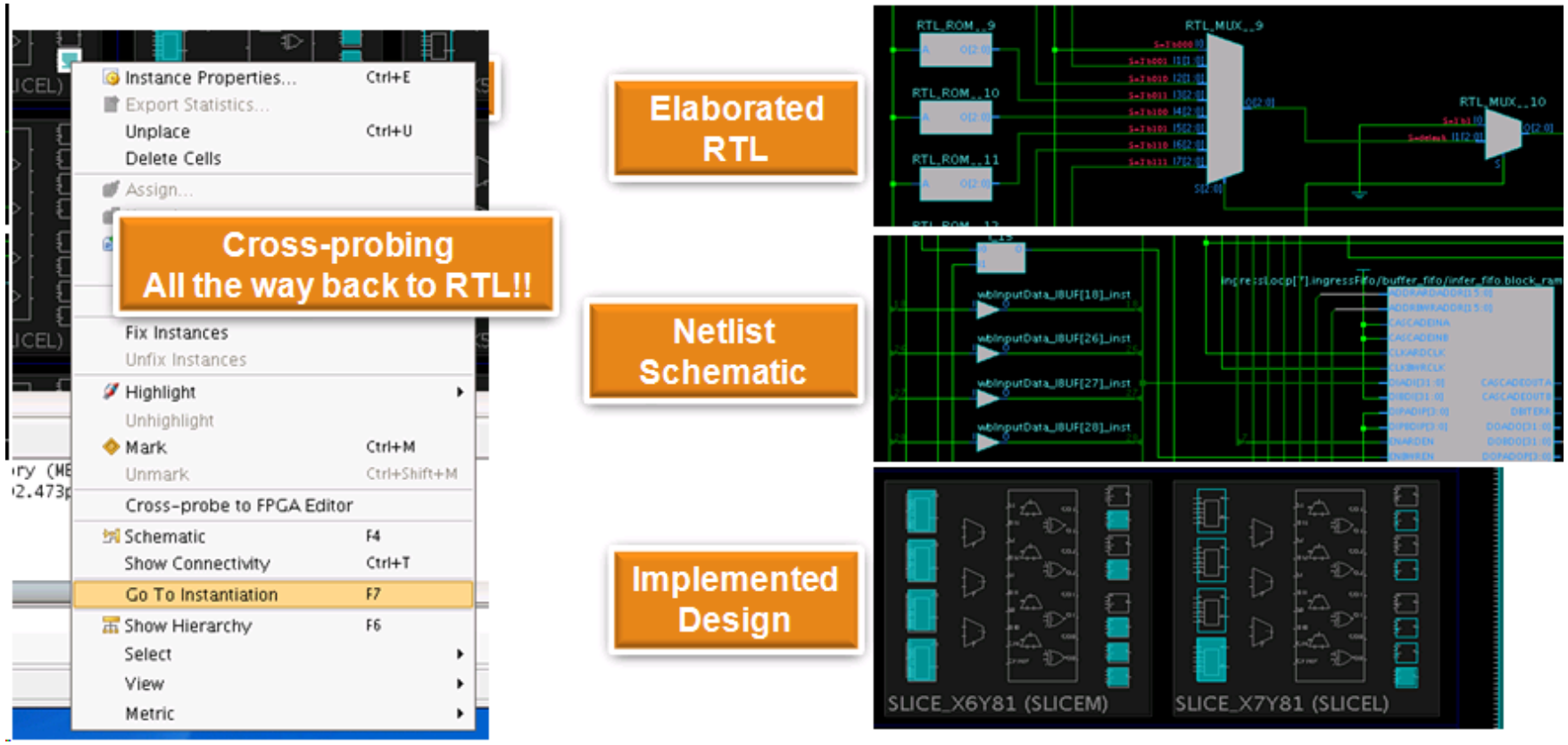



*Hierarchical Design Analysis and Implementation Environment*

# Vivado Visualization Features

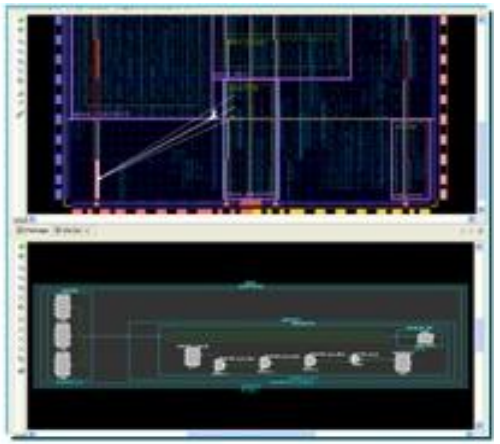
## Visualize and debug a design at any flow stage

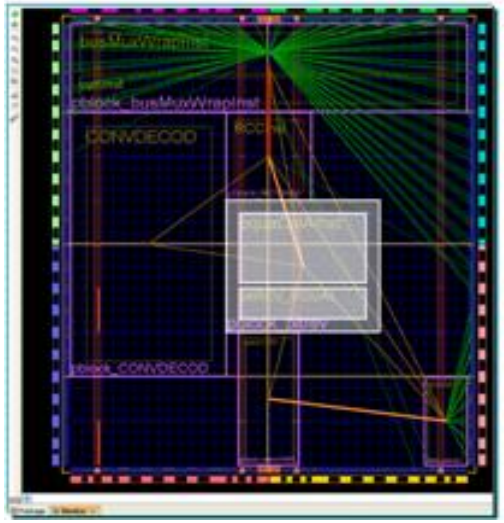
- Cross-probing between netlist/schematic/RTL



# Gain Faster Timing Closure

- **Analyze multiple implementation results**
  - Highlight failing timing paths from post-route timir analysis
  - Quickly identify and constrain critical logic path
- **Connectivity display**
  - I/Os, net bundles, clock domains
- **Hierarchical floorplanning**
  - Guide *place & route* toward better results
- **Utilization estimates**
  - All resource types shown for each Pblock
  - Clocks or carry chains





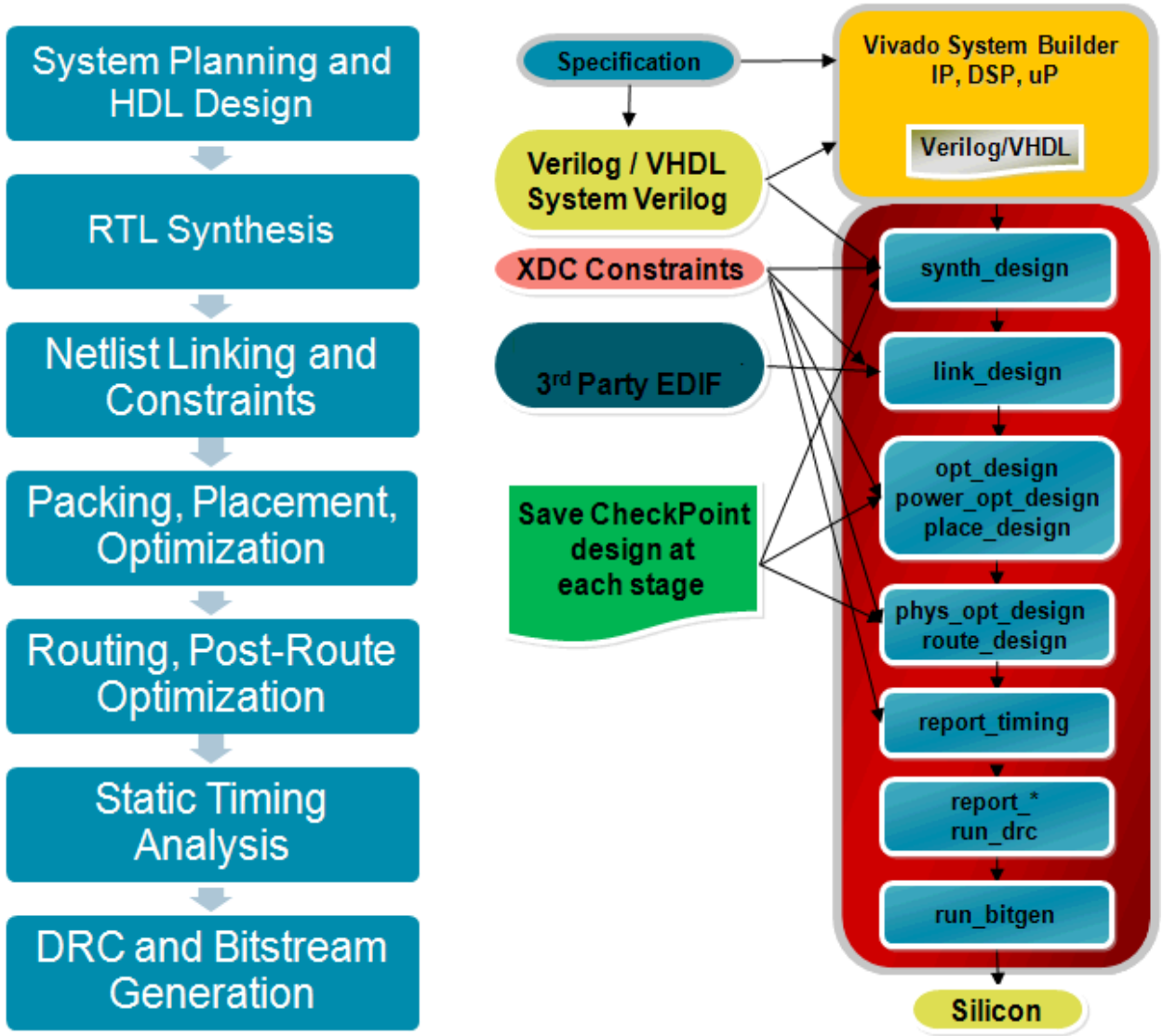
# Tool Command Line (.tcl) Features

- Tcl Console enables the designer to actively query the design netlist
- Full Tcl scripting support in two design flows
  - Project-based design flow provides easy project management by the Vivado IDE
  - Non-project batch design flow enables entire flow to be executed in memory
- *Journal* and *log* files can be used for script construction

# *Vivado Design Suite Introduction*

# Typical vs Vivado Design Flow

- ✓ **Interactive IP plug-n-play environment**
  - ✓ AXI4, IP_XACT
- ✓ **Common constraint language (XDC) throughout flow**
  - ✓ *Apply constraints at any stage*
- ✓ **Reporting at any stage**
  - ✓ Robust Tcl API
- ✓ **Common data model throughout the flow**
  - ✓ "In memory" model improves speed
  - ✓ Generate reports at all stages
- ✓ **Save checkpoint designs at any stage**
  - ✓ Netlist, constraints, place and route results



# Project Data and Directories

All project data is stored in a *project_name* directory containing the following directories

- ***project_name.xpr*** file: Object that is selected to open a project (Vivado IDE project file)
- ***project_name.runs*** directory: Contains all run data (synthesis, implementation)
- ***project_name.srcs*** directory: Contains all imported local HDL source files, netlists, and XDC files
- ***project_name.data*** directory: Stores floorplan and netlist data

# Journal and Log Files

## Journal file (*vivado.jou*)

- Contains just the Tcl commands executed by the Vivado IDE

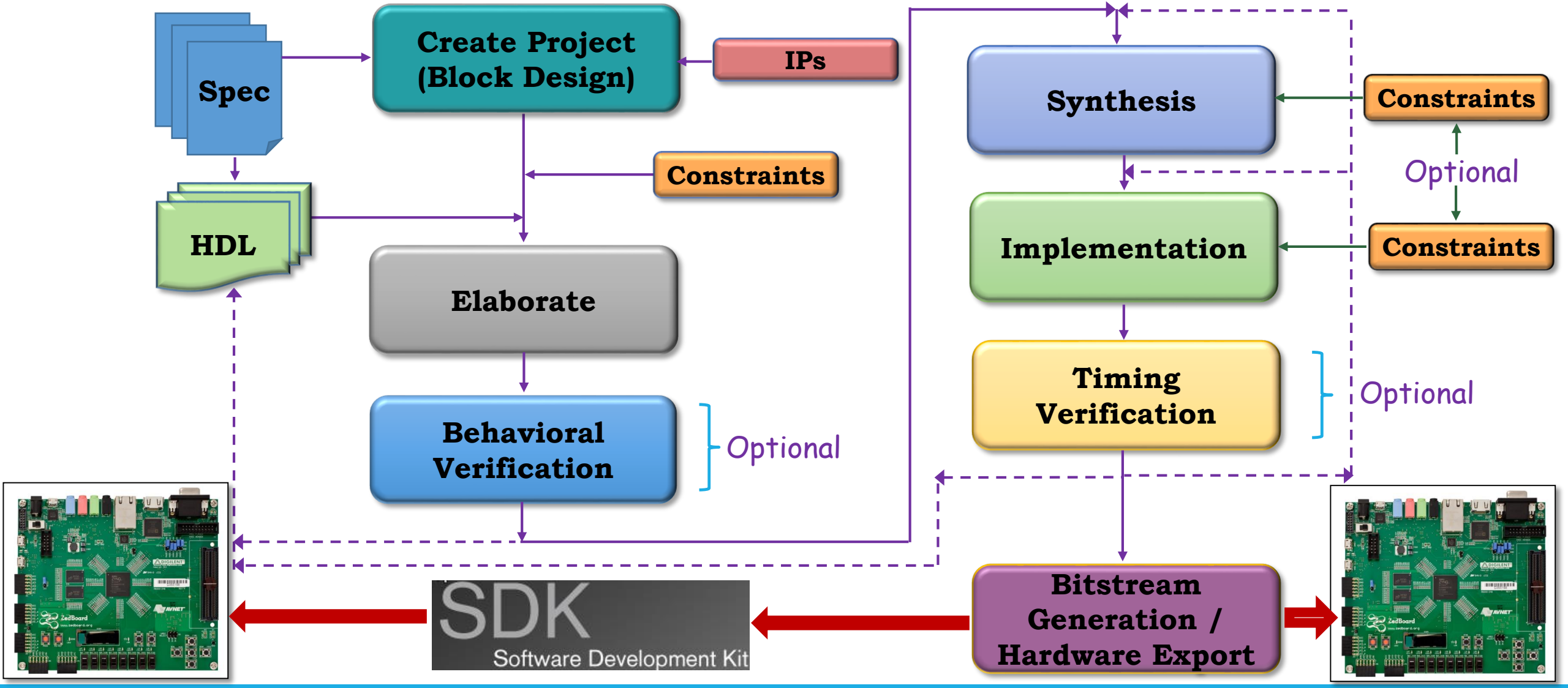
## Log file (*vivado.log*)

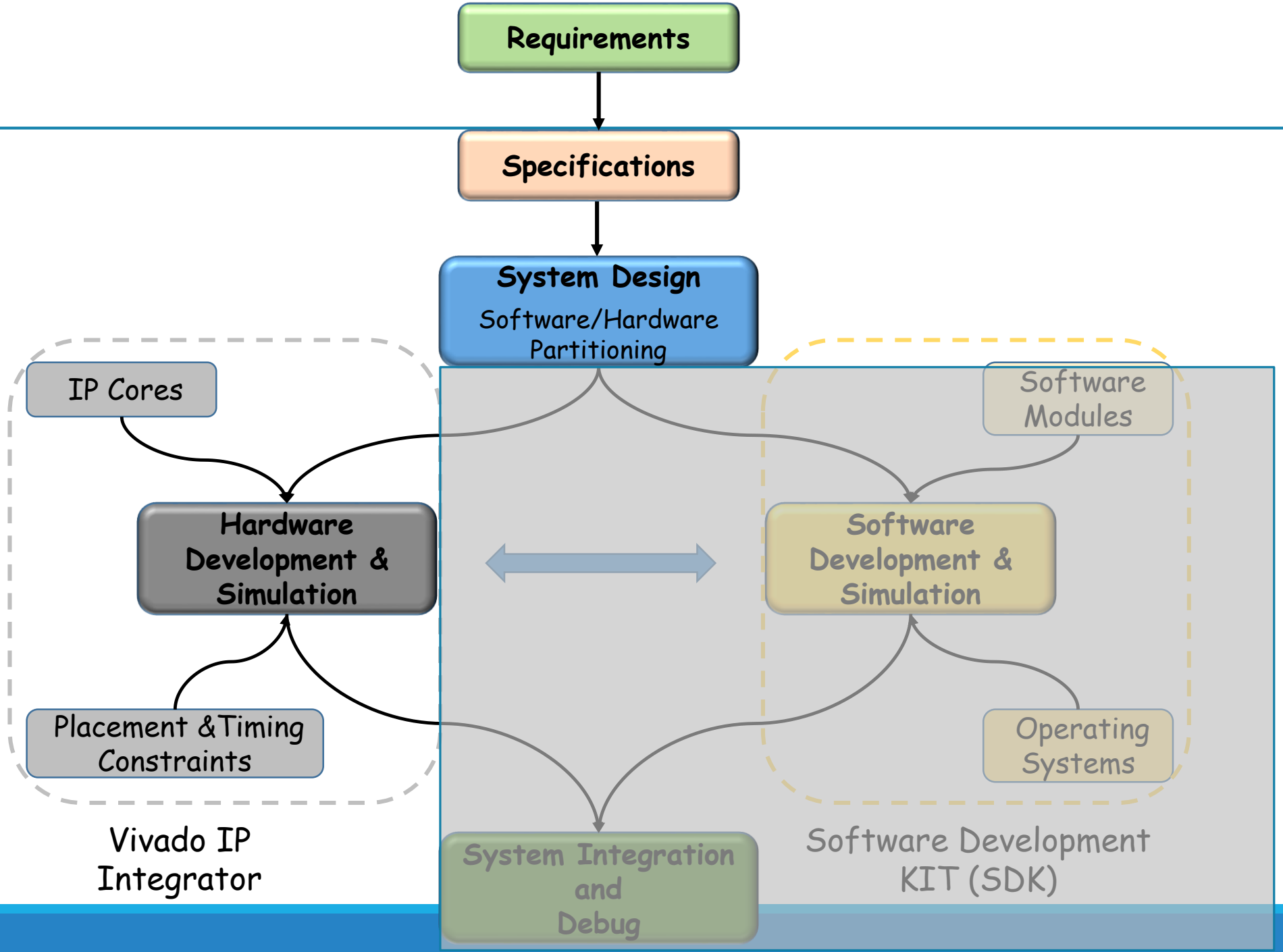
- Contains all messages produced by the Vivado IDE, including Tcl commands and results, info, warning, error messages, etc.

## Location

- Linux: directory where the Vivado IDE is invoked
- Windows via icon: *%APPDATA%\Xilinx\Vivado* or *C:\Users\<user_name>\AppData\Roaming\Xilinx\Vivado*

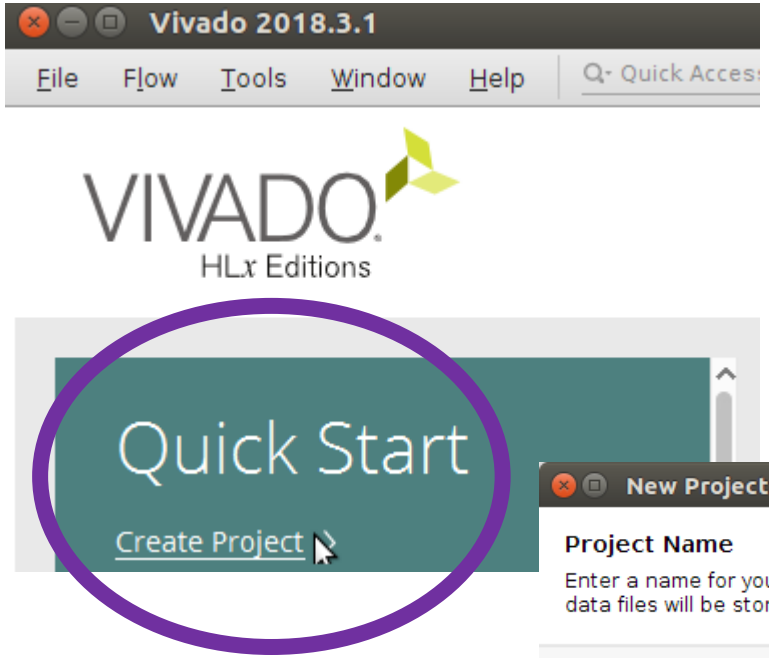
# Embedded System Design – Vivado Flow

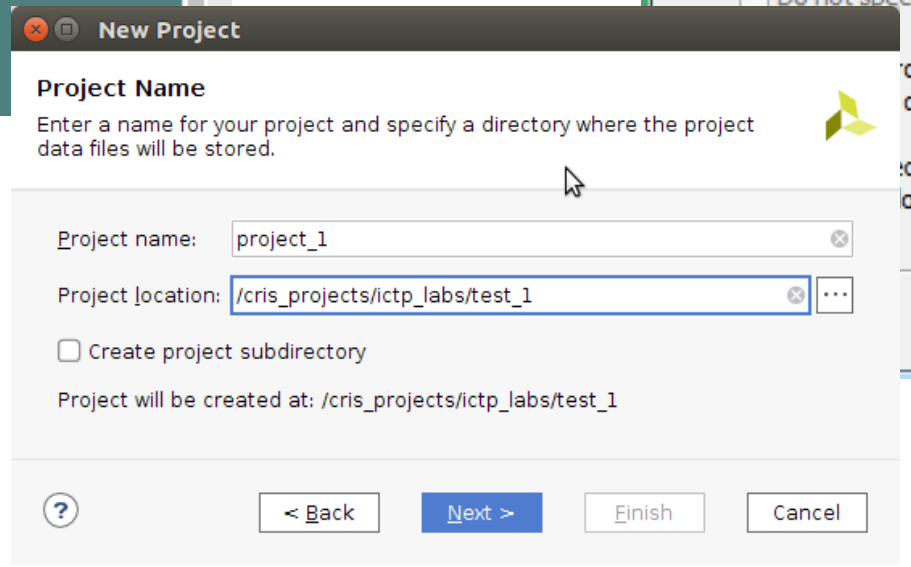


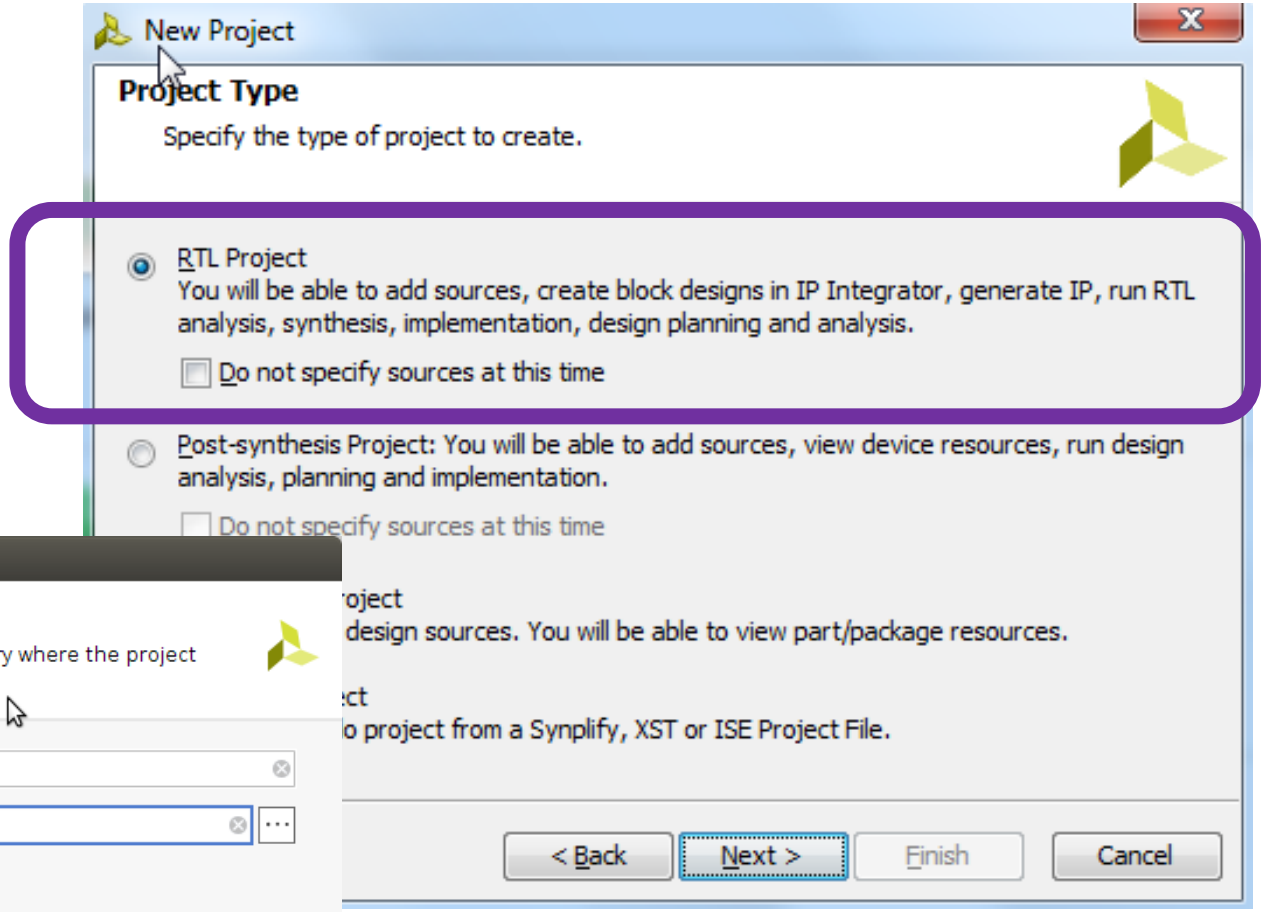

Requirements
Specifications
System Design
Software/Hardware Partitioning
IP Cores
Hardware Development & Simulation
Placement &Timing Constraints
Vivado IP Integrator
Software Modules
Software Development & Simulation
Operating Systems
Software Development KIT (SDK)
System Integration and Debug

# *Vivado Flow Practical Steps*

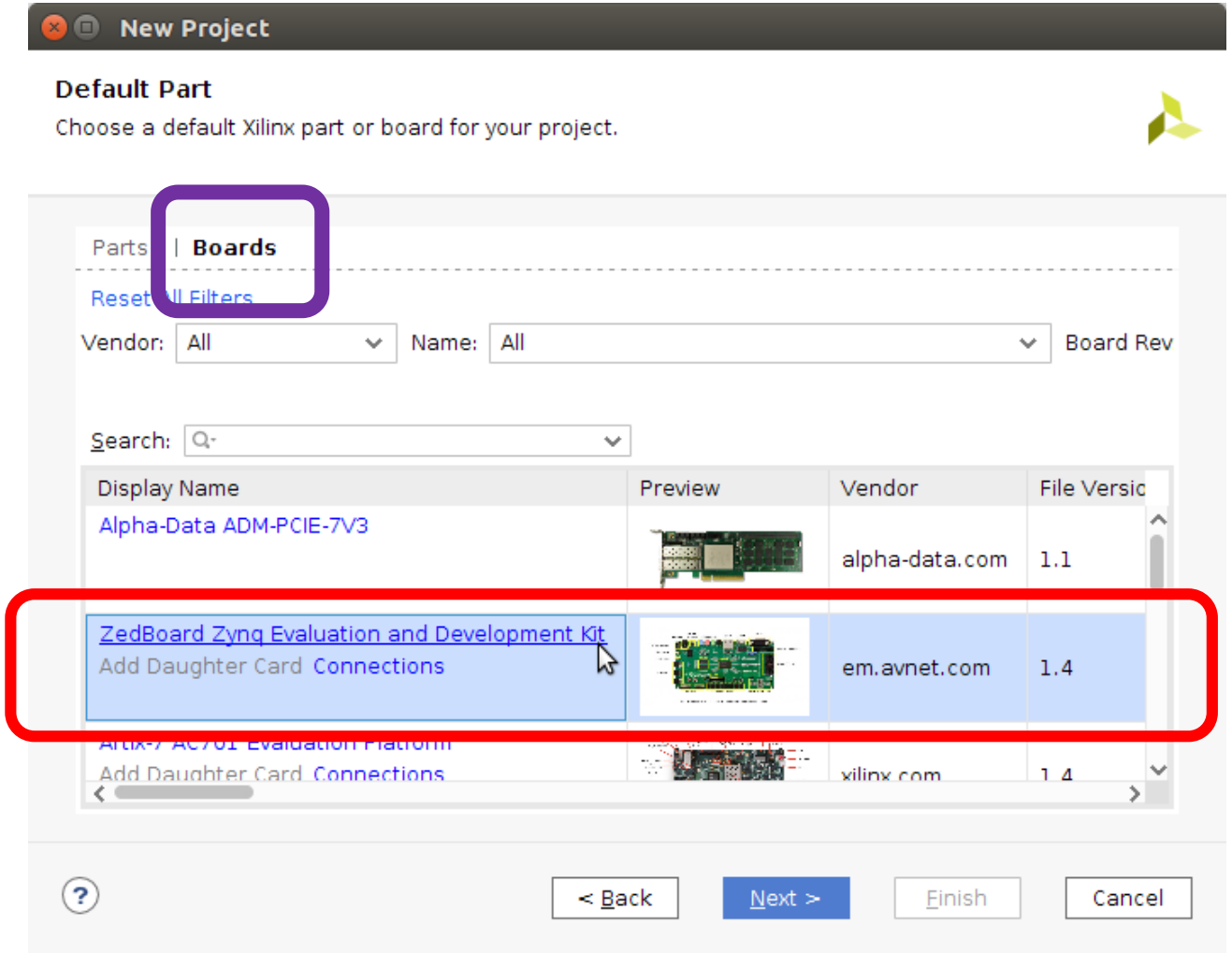
# Creating a Project

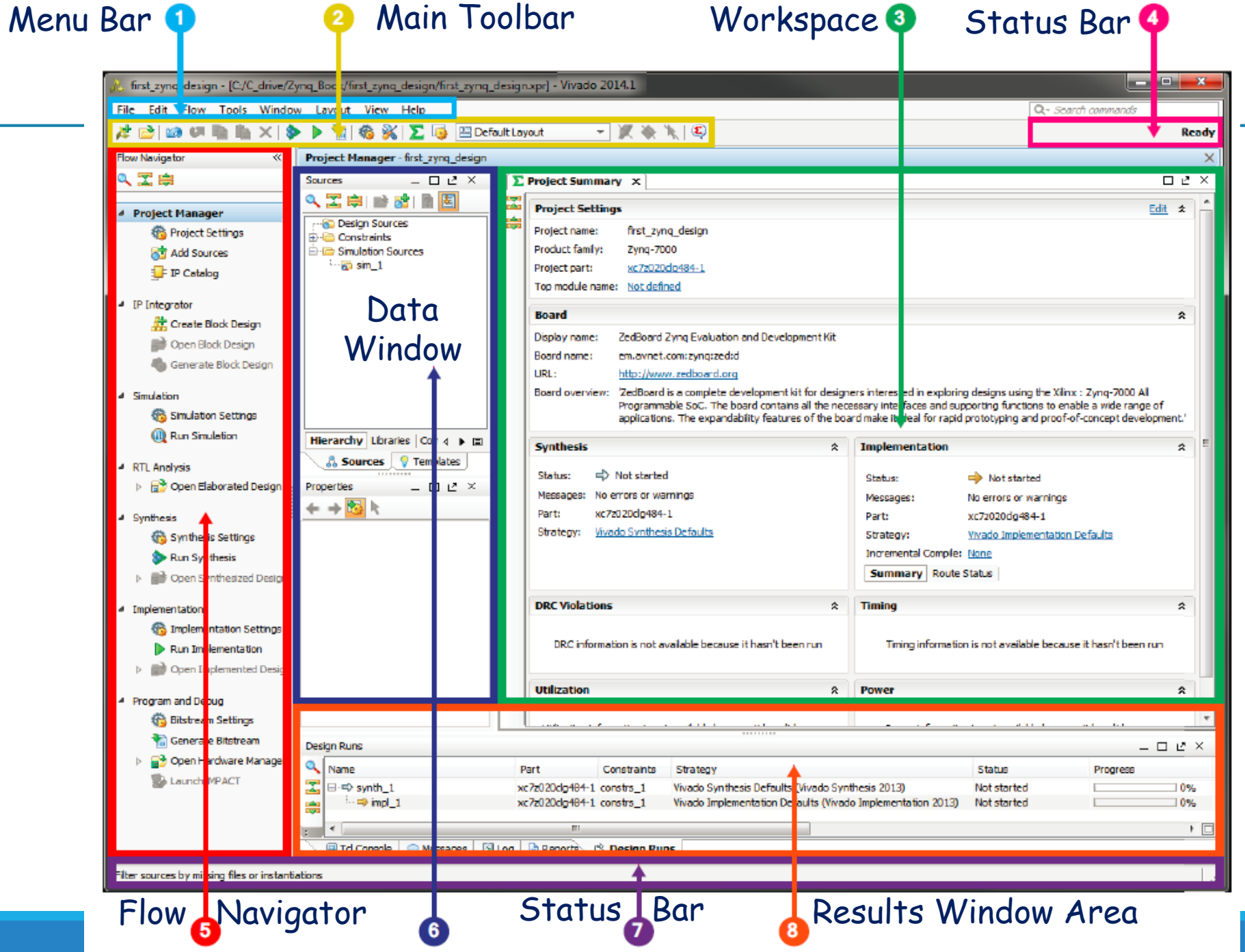


# Creating a Project

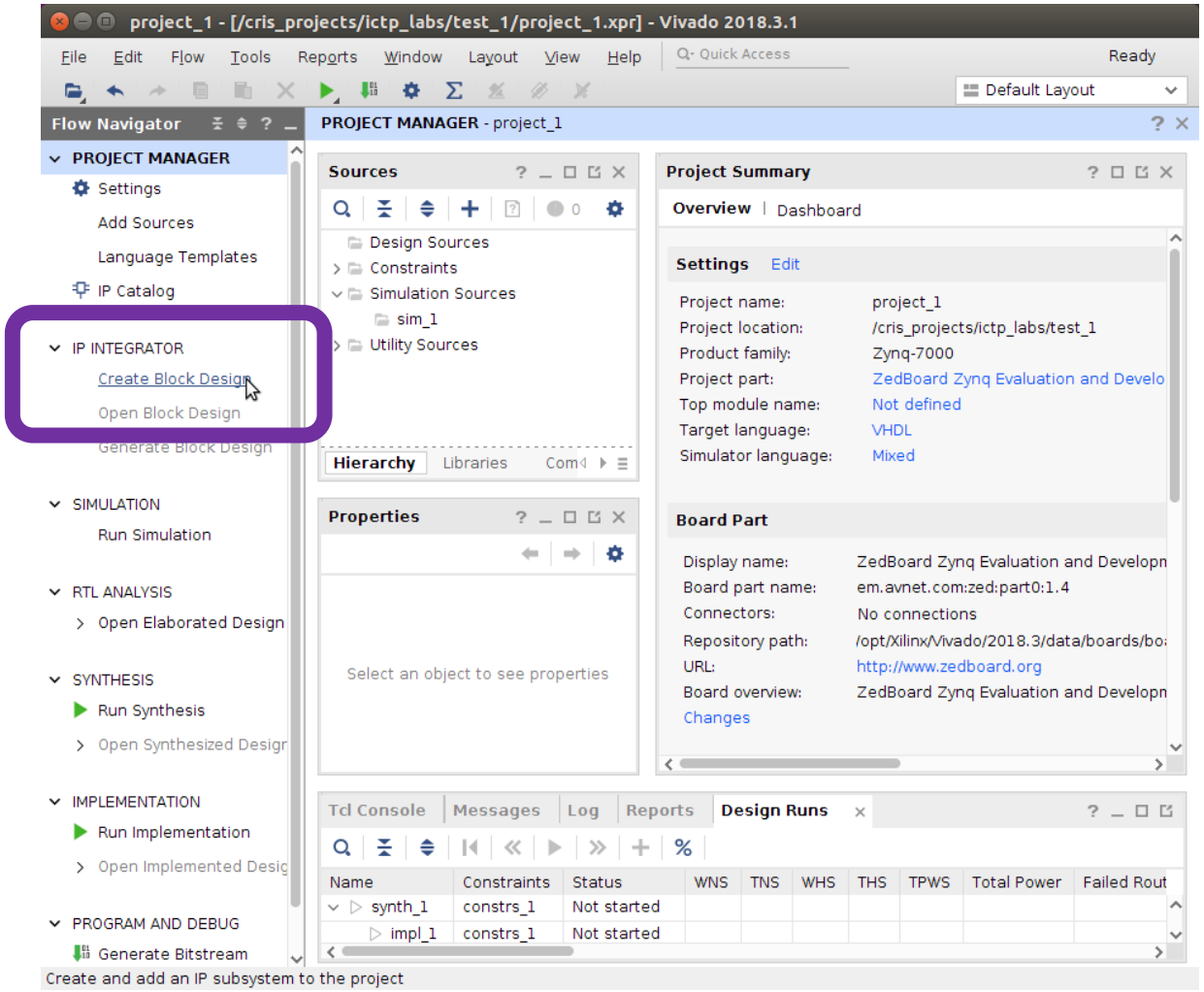


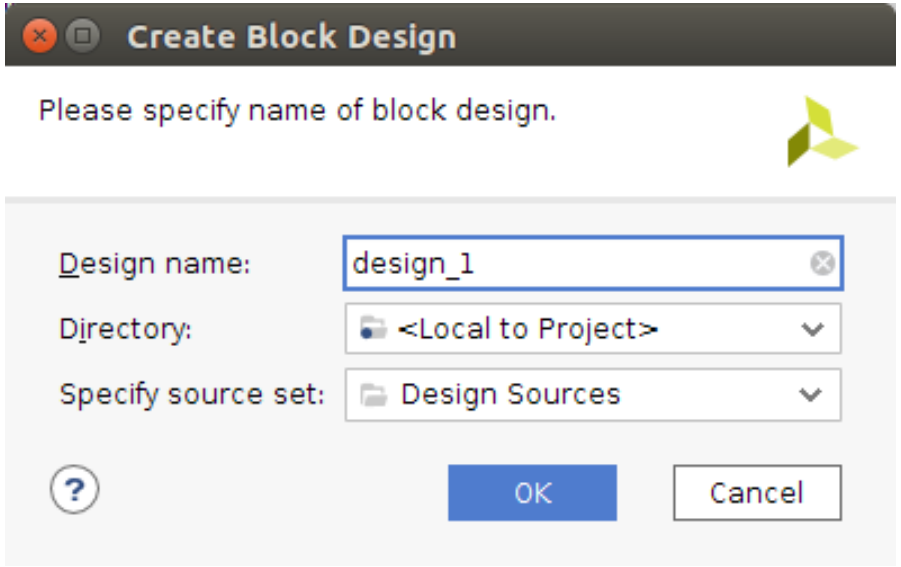

Menu Bar 1
2 Main Toolbar
Workspace 3
Status Bar 4
Data Window
Flow Navigator 5
6
Status Bar 7
Results Window Area 8
Vivado 2014.1
File Edit Flow Tools Window Layout View Help
Default Layout
Ready
Flow Navigator
Project Manager
Project Settings
Add Sources
IP Catalog
IP Integrator
Create Block Design
Open Block Design
Generate Block Design
Simulation
Simulation Settings
Run Simulation
RTL Analysis
Open Elaborated Design
Synthesis
Synthesis Settings
Run Synthesis
Implementation
Implementation Settings
Run Implementation
Program and Debug
Bitstream Settings
Generate Bitstream
Open Hardware Manager
Sources
Design Sources
Constraints
Simulation Sources
sim_1
Hierarchy
Libraries
Sources
Templates
Properties
Project Summary
Project Settings
Edit
Project name: first_zynq_design
Product family: Zynq-7000
Project part: xc7z020clg484-1
Top module name: Not defined
Board
Display name: ZedBoard Zynq Evaluation and Development Kit
Board name: em.avnet.com:zynq:zed:d
URL: http://www.zedboard.org
Synthesis
Status: Not started
Messages: No errors or warnings
Part: xc7z020clg484-1
Strategy: Vivado Synthesis Defaults
Implementation
Status: Not started
Messages: No errors or warnings
Part: xc7z020clg484-1
Strategy: Vivado Implementation Defaults
Incremental Compile: None
Summary
Route Status
DRC Violations
DRC information is not available because it hasn't been run
Timing
Timing information is not available because it hasn't been run
Utilization
Power
Design Runs
Name
Part
Constraints
Strategy
Status
Progress
synth_1
impl_1
Not started
Filter sources by missing files or instantiations

# Main Components of the Project Navigator

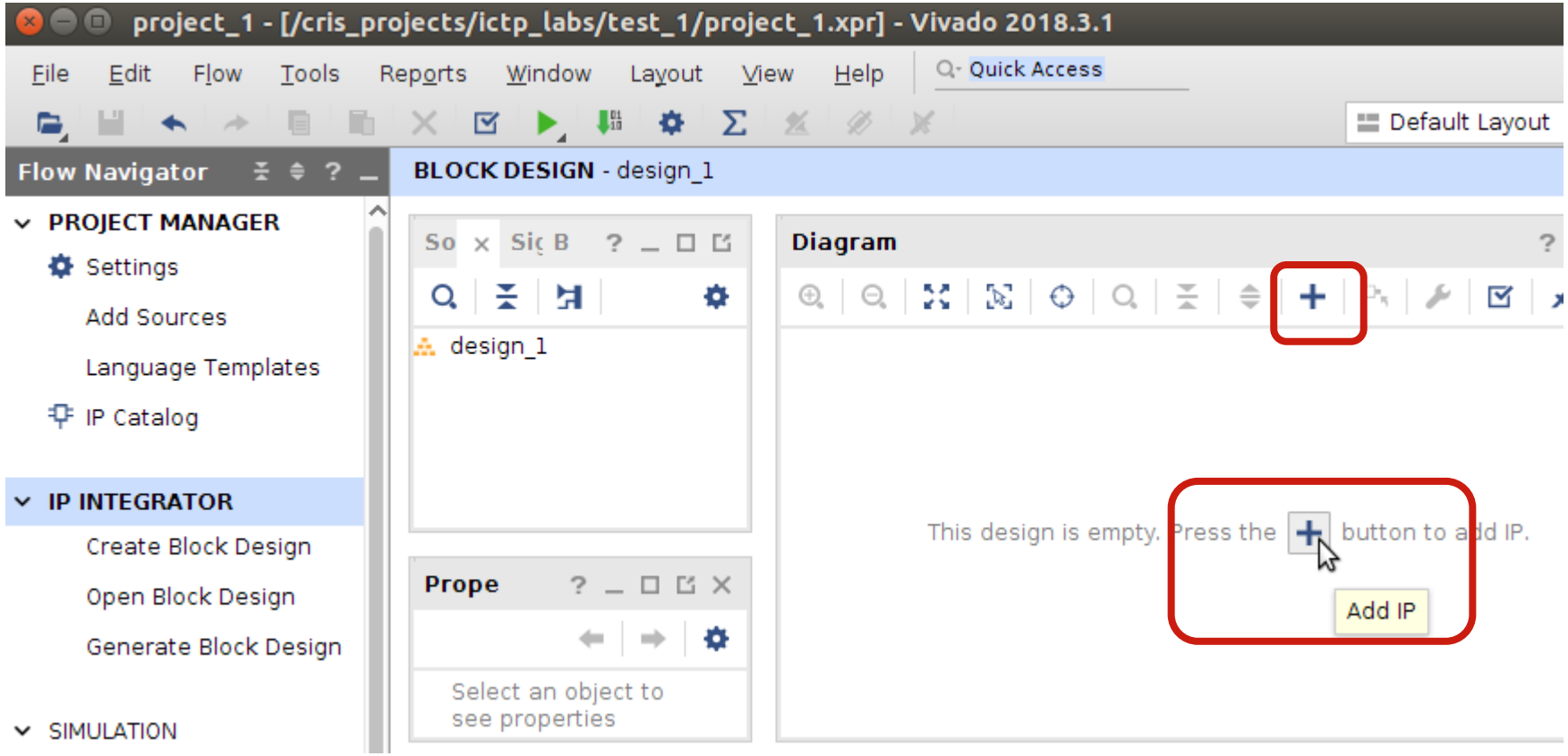
1. **Menu Bar**: Vivado IDE commands
2. **Main Toolbar**: Access to the most commonly used Vivado IDE commands
3. **Workspace**: area for schematic panel, device panel, package panel, text editor panel.
4. **Project Status Bar**: displays the status of the currently active design
5. **Flow Navigator**: provide easy access to the tools and commands necessary to guide the design from start to finish.
6. **Data Window Pane**: by default displays information that relates to design data and sources, such as Property Window, Netlist Window, and Source Window
7. **Status Bar**: displays information about menu bar and toolbar commands; task progresses
8. **Results Window Area**: there are a set of windows, such as Messages, showing message for each process, Tcl Console, Tcl commands of each activity, Reports, reports generated throughout the design flow, Desing Runs, display the different run for the current project

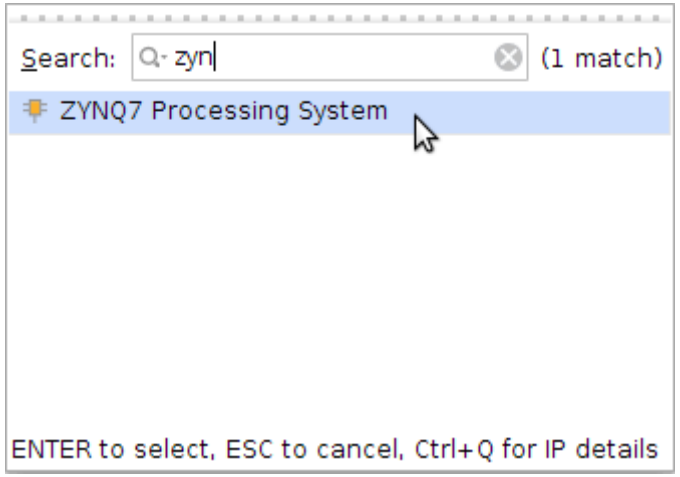
# Create a Block design



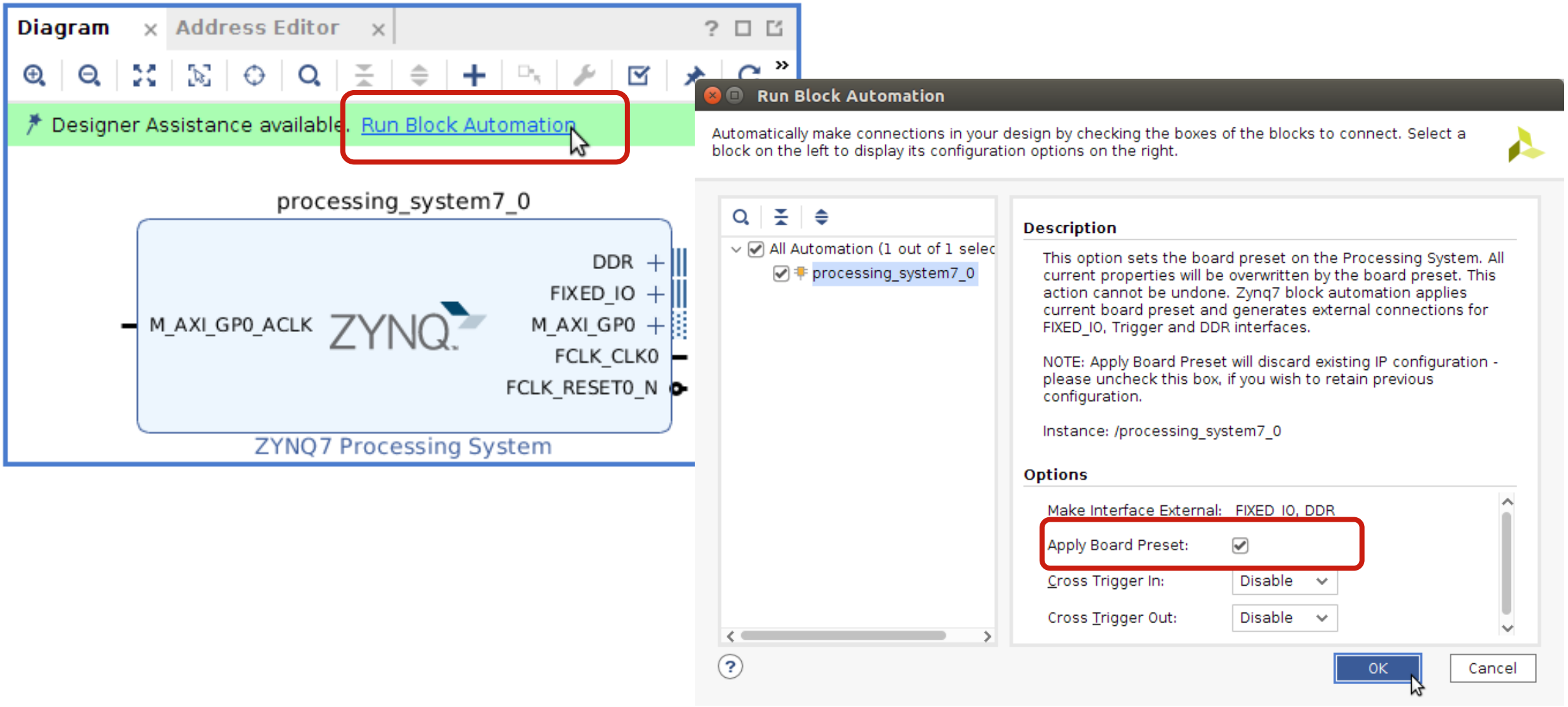
# Adding IP Modules to the Design Canvas



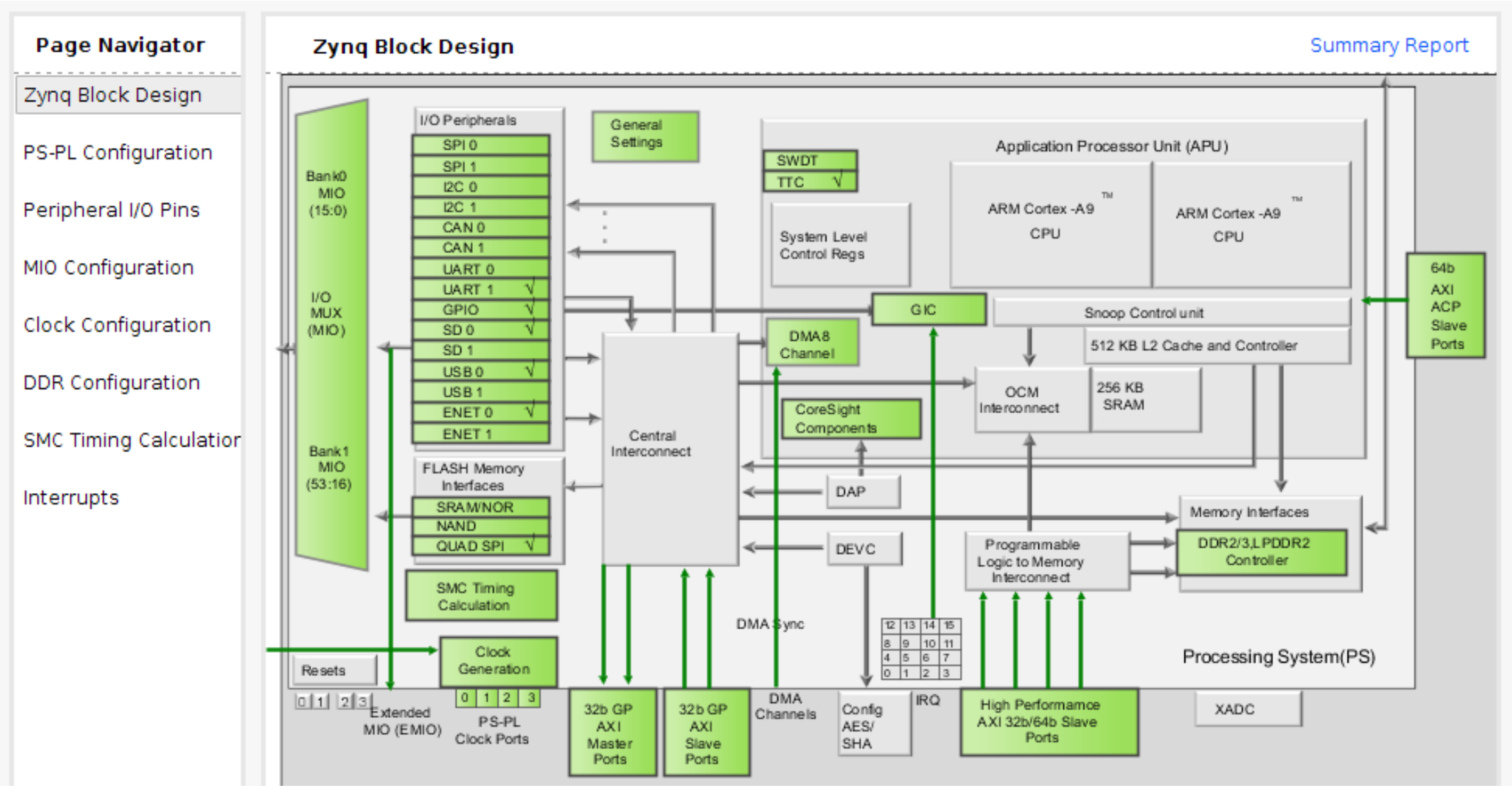




To add multiple IP to the Block Design, you can highlight the additional desired IP (**Ctrl+Click**) and press the **Enter** key.

# Processing System (PS)



# Configuring the PS



# Configuring the PS

Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation | Presets | IP Location | Import XPS Settings

**Page Navigator**

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculatic
- Interrupts

**PS-PL Configuration**

Search:

| Name | Select | Description |
|---|---|---|
| General | | |
| AXI Non Secure Enablement | 0 | Enable AXI Non Secure |
| GP Slave AXI Interface | | |
| S AXI GP0 interface | ☑ | Enables General purpo |
| S AXI GP1 interface | ☐ | Enables General purpo |
| HP Slave AXI Interface | | |
| S AXI HP0 interface | ☐ | Enables AXI high perfor |
| S AXI HP1 interface | ☐ | Enables AXI high perfor |
| S AXI HP2 interface | ☐ | Enables AXI high perfor |
| S AXI HP3 interface | ☐ | Enables AXI high perfor |
| ACP Slave AXI Interface | | |
| DMA Controller | | |
| PS-PL Cross Trigger interface | ☐ | Enables PL cross trigge |

Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation | Presets | IP Location | Import XPS Settings

**Page Navigator**

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculatic
- Interrupts

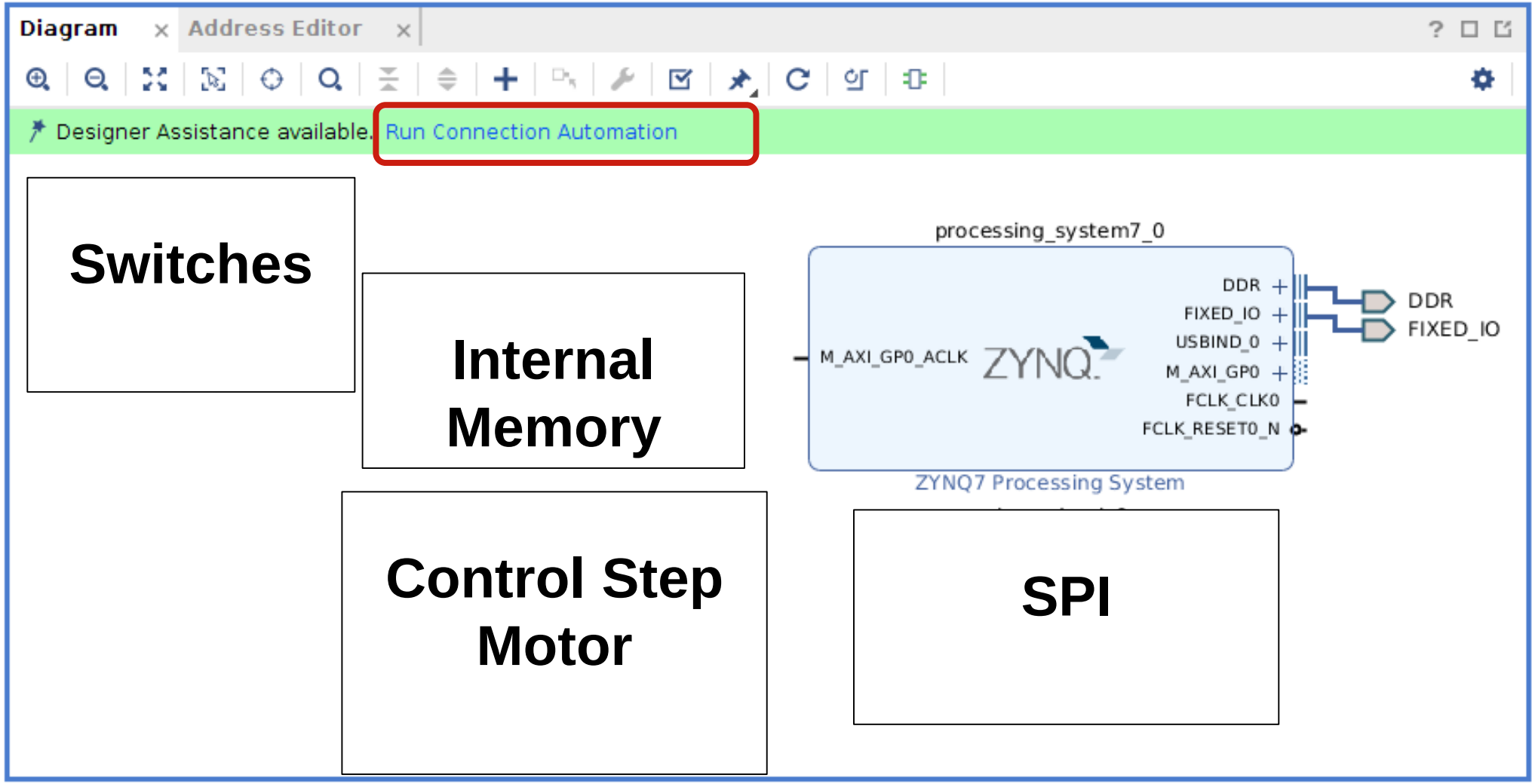
**Peripheral I/O Pins** Summary Report

Search:

| Peripherals | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ☐ SPI 1 | | | | | SPI1 | | | | | | | | | | | | SPI1 | | | | |
| ☐ UART 0 | | | | | UART0 | | | | UART0 | | | | UART0 | | | | UART0 | | | | UART0 |
| ☑ UART 1 | | | UART1 | | | | UART1 | | | | UART1 | | | | UART1 | | | | UART1 | | |
| ☑ I2C 0 | | | | | I2C0 | | | | I2C0 | | | | I2C0 | | | | I2C0 | | | | I2C0 |
| ☐ I2C 1 | | | I2C1 | | | | I2C1 | | | | I2C1 | | | | I2C1 | | | | I2C1 | | |
| ☑ CAN 0 | | | | | CAN0 | | | | CAN0 | | | | CAN0 | | | | CAN0 | | | | CAN0 |
| ☐ CAN 1 | | | CAN1 | | | | CAN1 | | | | CAN1 | | | | CAN1 | | | | CAN1 | | |
| ☐ TTC0 | | | | | | | | | | | | | | | | | | | | | |

OK | Cancel

# Making Up the System



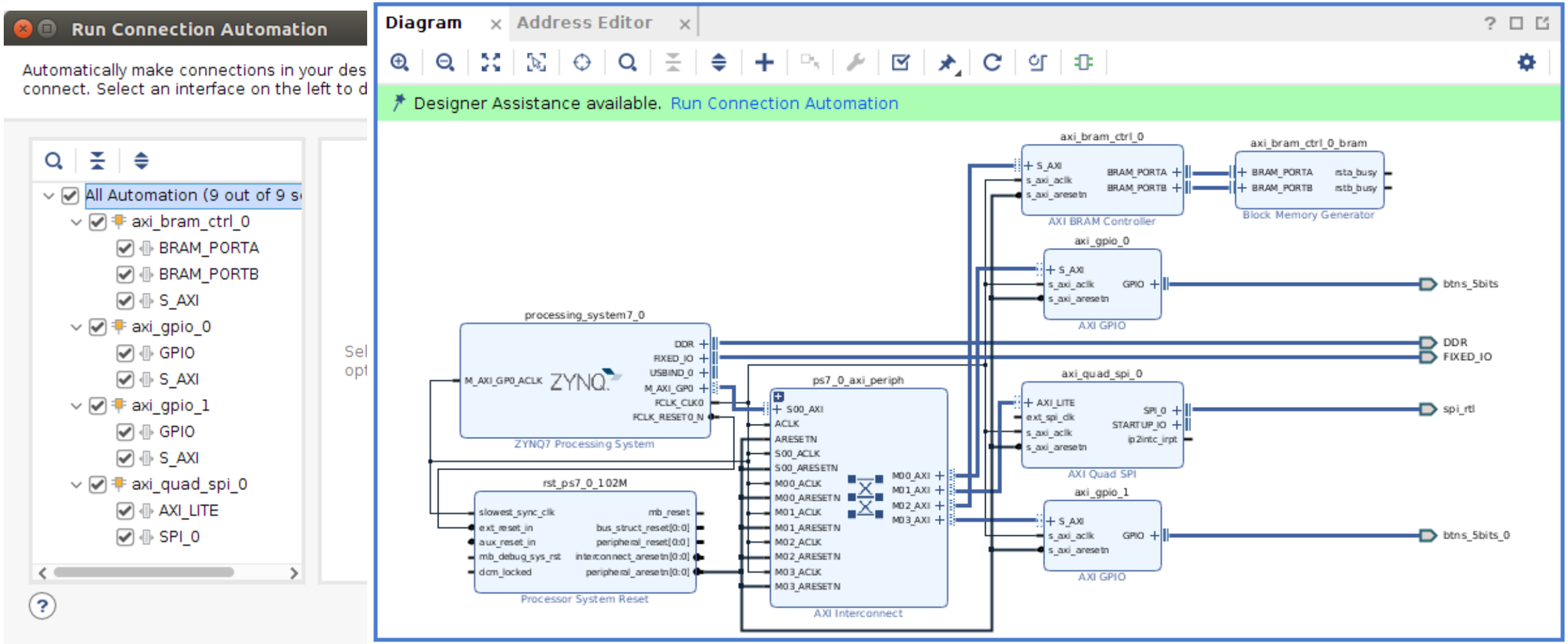

**Switches**

**Internal Memory**

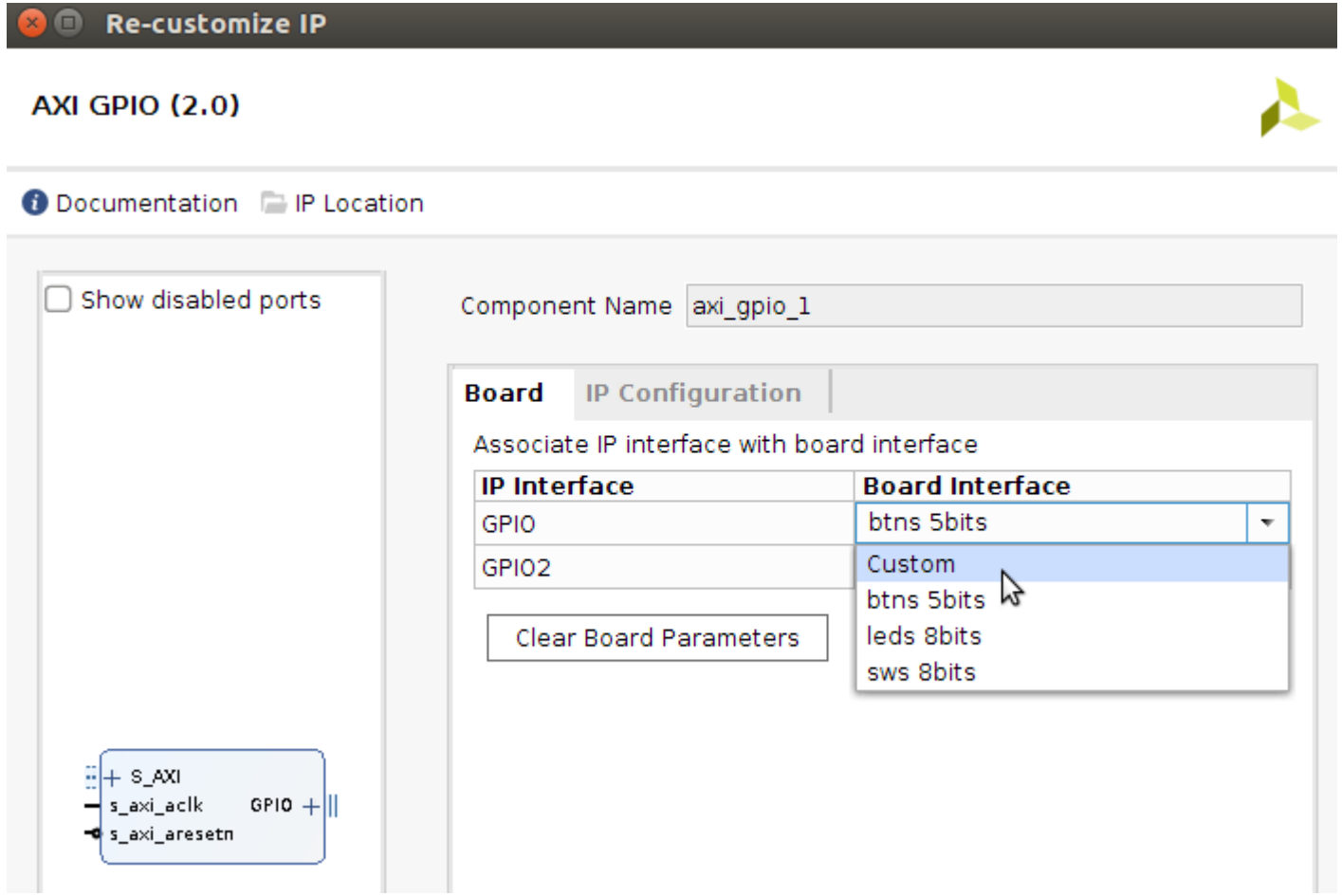
**Control Step Motor**

**SPI**

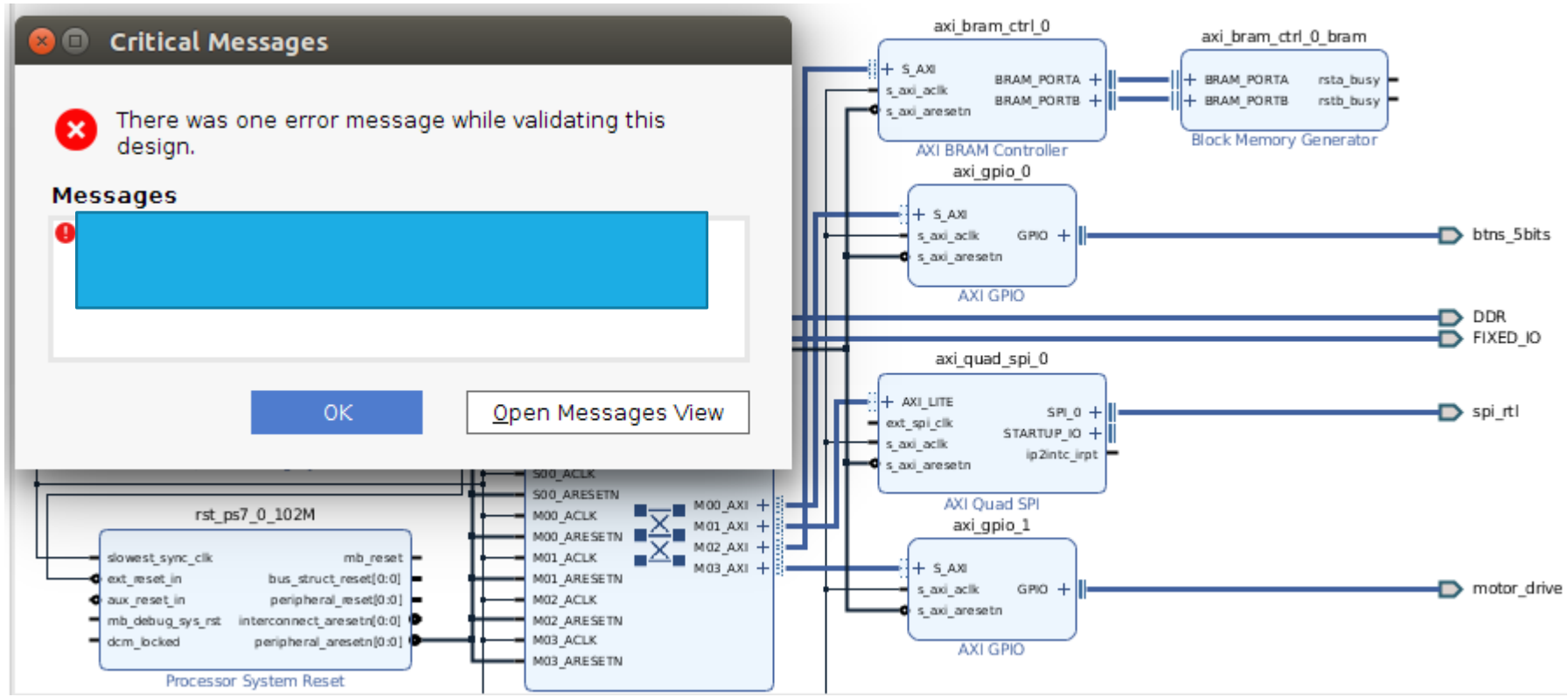
# Running Connection Automation



# Custom GPIO


Re-customize IP
AXI GPIO (2.0)
Documentation
IP Location
Show disabled ports
Component Name
axi_gpio_1
Board
IP Configuration
Associate IP interface with board interface
IP Interface
Board Interface
GPIO
btns 5bits
GPIO2
Custom
btns 5bits
leds 8bits
sws 8bits
Clear Board Parameters
S_AXI
s_axi_aclk
GPIO
s_axi_aresetn

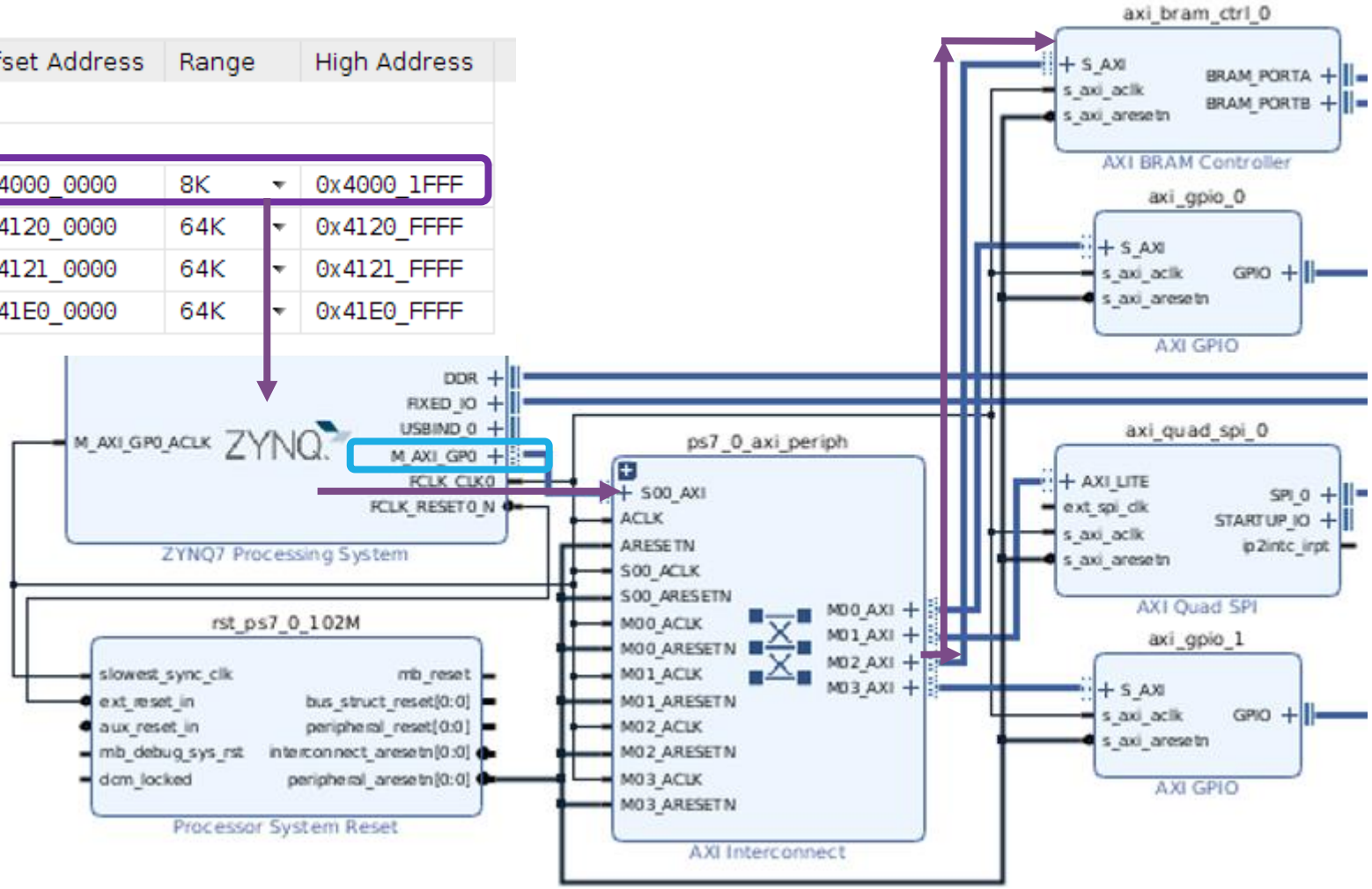
# DRC (Desing Rule Check) Design Validation



# Memory Map

Diagram | Address Editor

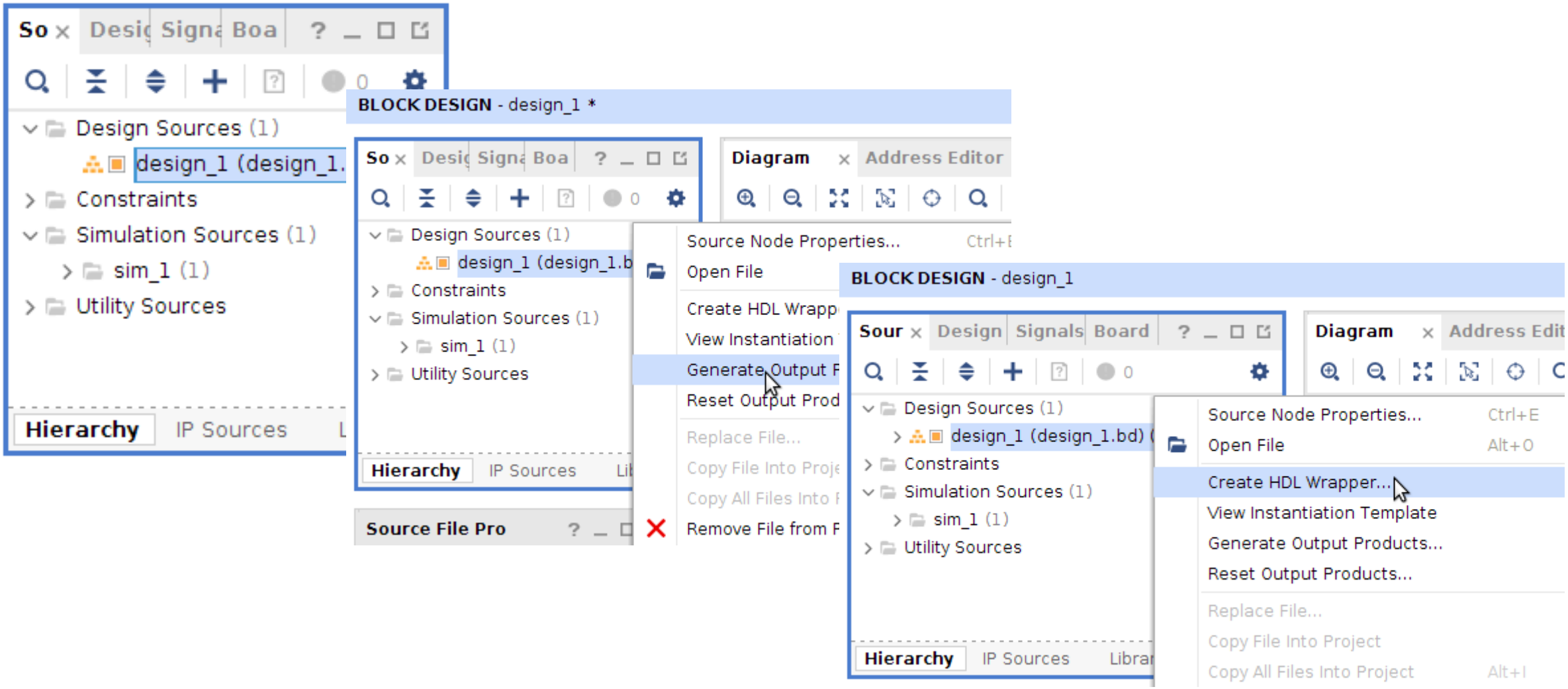
| Cell | Slave Interface | Base Name | Offset Address | Range | High Address |
|---|---|---|---|---|---|
| processing_system7_0 | | | | | |
| Data (32 address bits : 0x40000000 [ 1G ]) | | | | | |
| axi_bram_ctrl_0 | S_AXI | Mem0 | 0x4000_0000 | 8K | 0x4000_1FFF |
| axi_gpio_0 | S_AXI | Reg | 0x4120_0000 | 64K | 0x4120_FFFF |
| axi_gpio_1 | S_AXI | Reg | 0x4121_0000 | 64K | 0x4121_FFFF |
| axi_quad_spi_0 | AXI_LITE | Reg | 0x41E0_0000 | 64K | 0x41E0_FFFF |



# System Level Address Map

| Address Range | CPUs and ACP | AXI_HP | Other Bus Masters[1] | Notes |
|---|---|---|---|---|
| 0000_0000 to 0003_FFFF[2] | OCM | OCM | OCM | Address not filtered by SCU and OCM is mapped low |
| | DDR | OCM | OCM | Address filtered by SCU and OCM is mapped low |
| | DDR | | | Address filtered by SCU and OCM is not mapped low |
| | | | | Address not filtered by SCU and OCM is not mapped low |
| 0004_0000 to 0007_FFFF | DDR | | | Address filtered by SCU |
| | | | | Address not filtered by SCU |
| 0008_0000 to 000F_FFFF | DDR | DDR | DDR | Address filtered by SCU |
| | | DDR | DDR | Address not filtered by SCU[3] |
| 0010_0000 to 3FFF_FFFF | DDR | DDR | DDR | Accessible to all interconnect masters |
| 4000_0000 to 7FFF_FFFF | PL | | PL | General Purpose Port #0 to the PL, M_AXI_GP0 |
| 8000_0000 to BFFF_FFFF | PL | | PL | General Purpose Port #1 to the PL, M_AXI_GP1 |
| E000_0000 to E02F_FFFF | IOP | | IOP | I/O Peripheral registers, see Table 4-6 |
| E100_0000 to E5FF_FFFF | SMC | | SMC | SMC Memories, see Table 4-5 |
| F800_0000 to F800_0BFF | SLCR | | SLCR | SLCR registers, see Table 4-3 |
| F800_1000 to F880_FFFF | PS | | PS | PS System registers, see Table 4-7 |
| F890_0000 to F8F0_2FFF | CPU | | | CPU Private registers, see Table 4-4 |
| FC00_0000 to FDFF_FFFF[4] | Quad-SPI | | Quad-SPI | Quad-SPI linear address for linear mode |
| FFFC_0000 to FFFF_FFFF[2] | OCM | OCM | OCM | OCM is mapped high |
| | | | | OCM is not mapped high |

# Getting the System Ready to be Implemented



# Export Hardware Design to SDK

Software development is performed with the Xilinx Software Development Kit tool (SDK)

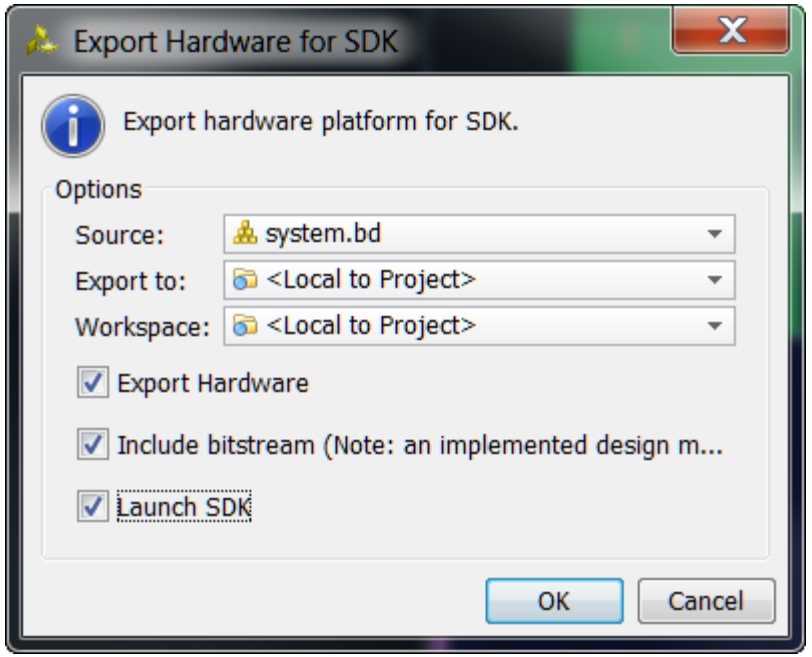
The design must be opened if a bitstream of the design is generated

The Block design must be open before the design can be exported

An XML description of the hardware is imported in the SDK tool

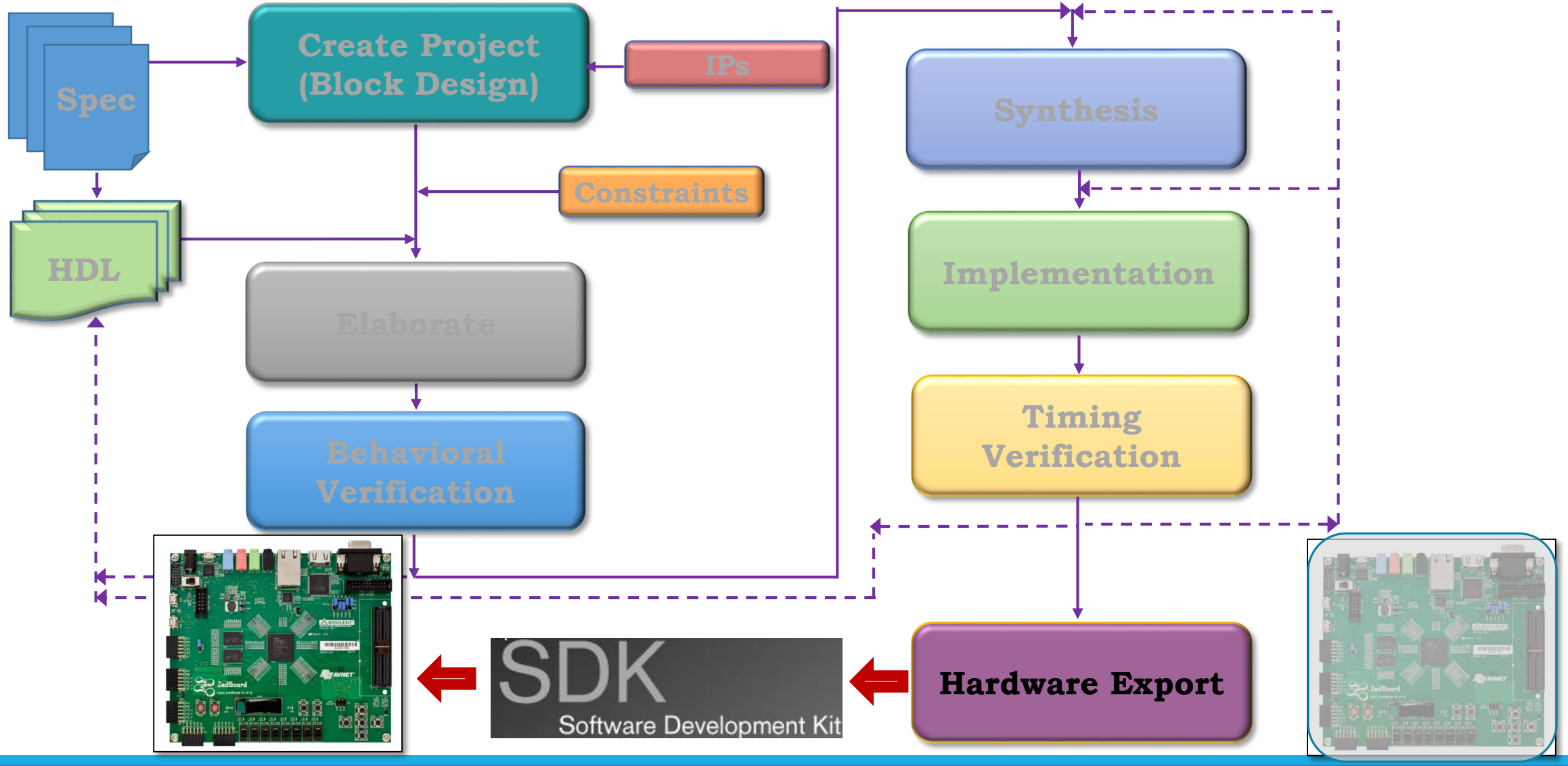
- The hardware platform is built on this description
- Only one hardware platform for an SDK project

The SDK tool will then associate user software projects to hardware



# Software Development Kit (SDK)

# Embedded System Design – Vivado-SDK Flow



# Embedded System Design – Vivado-SDK Flow

VIVADO

Open VIVADO
↓
Create New Project
↓
Create New Block Design
↓
Add PS7
Configure PS7
Run Block Automation
Add and Configure other IPs
Run Connection Automation
Validate Design
↓
Create HDL Wrapper
↓
Generate Output Files
↓
Generate Bitstream
↓
Export Hardware to SDK

SDK
Software Development Kit

Create Board Support Package
↓
Create New 'C' Application
↓
Write 'C' Code
↓
Build the Application
↓
Configure FPGA
↓
Run on Hardware (Debug)
↓

C7 Technology ©

# Embedded System Tools: Software

Eclipse IDE-based Software Development Kit (SDK)

- Board support package creation : LibGen
- GNU software development tools
- C/C++ compiler for the ARM Cortex-A9 processor (gcc)
- Debugger for the ARM Cortex-A9 processor (gdb)

Board support packages (BSPs)

- Stand-alone BSP
  - Free basic device drivers and utilities from Xilinx
  - NOT an RTOS

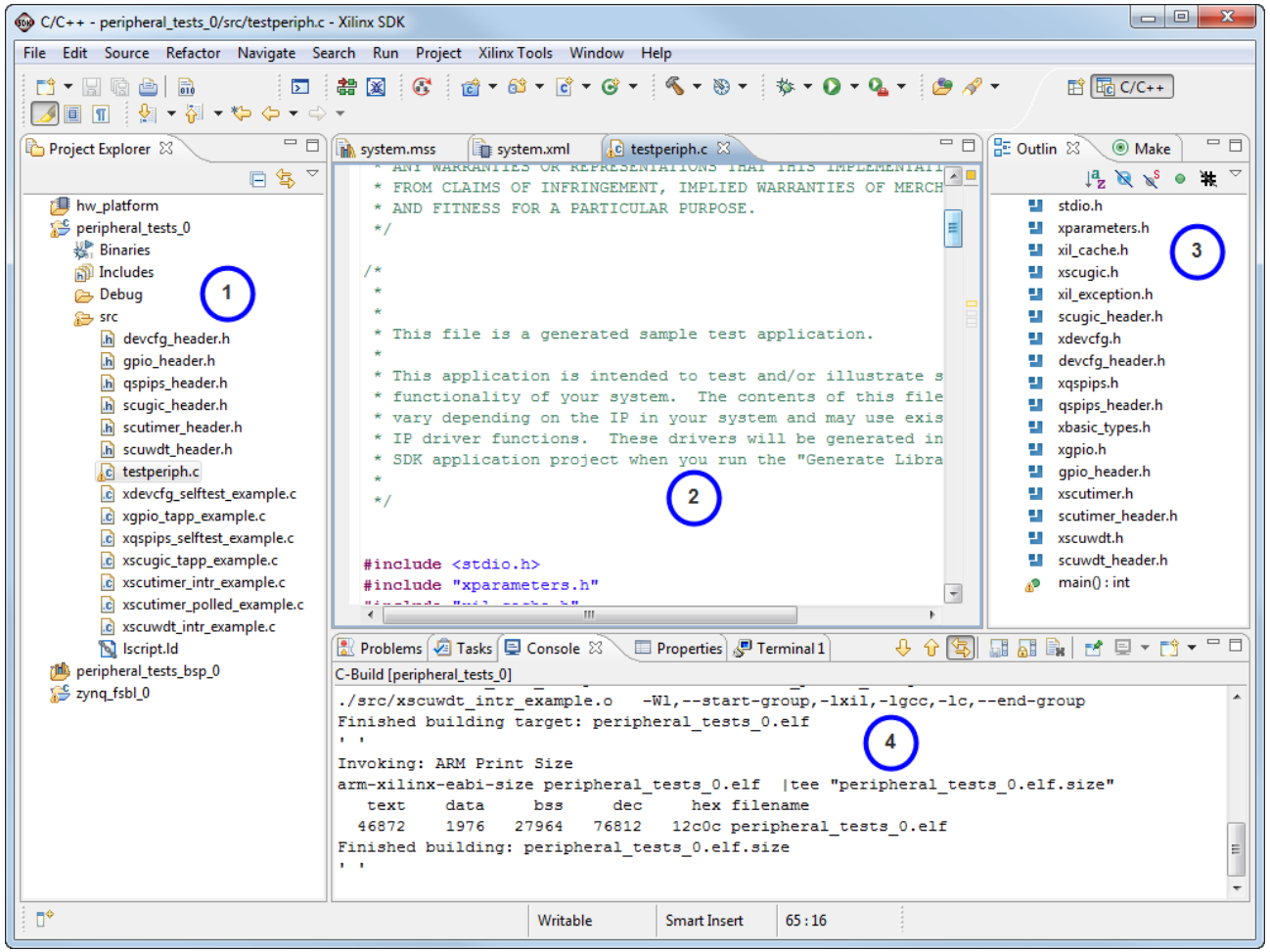
# SDK Workbench Views

① C/C++ project outline displays the elements of a project with file decorators (icons) for easy identification

② C/C++ editor for integrated software creation

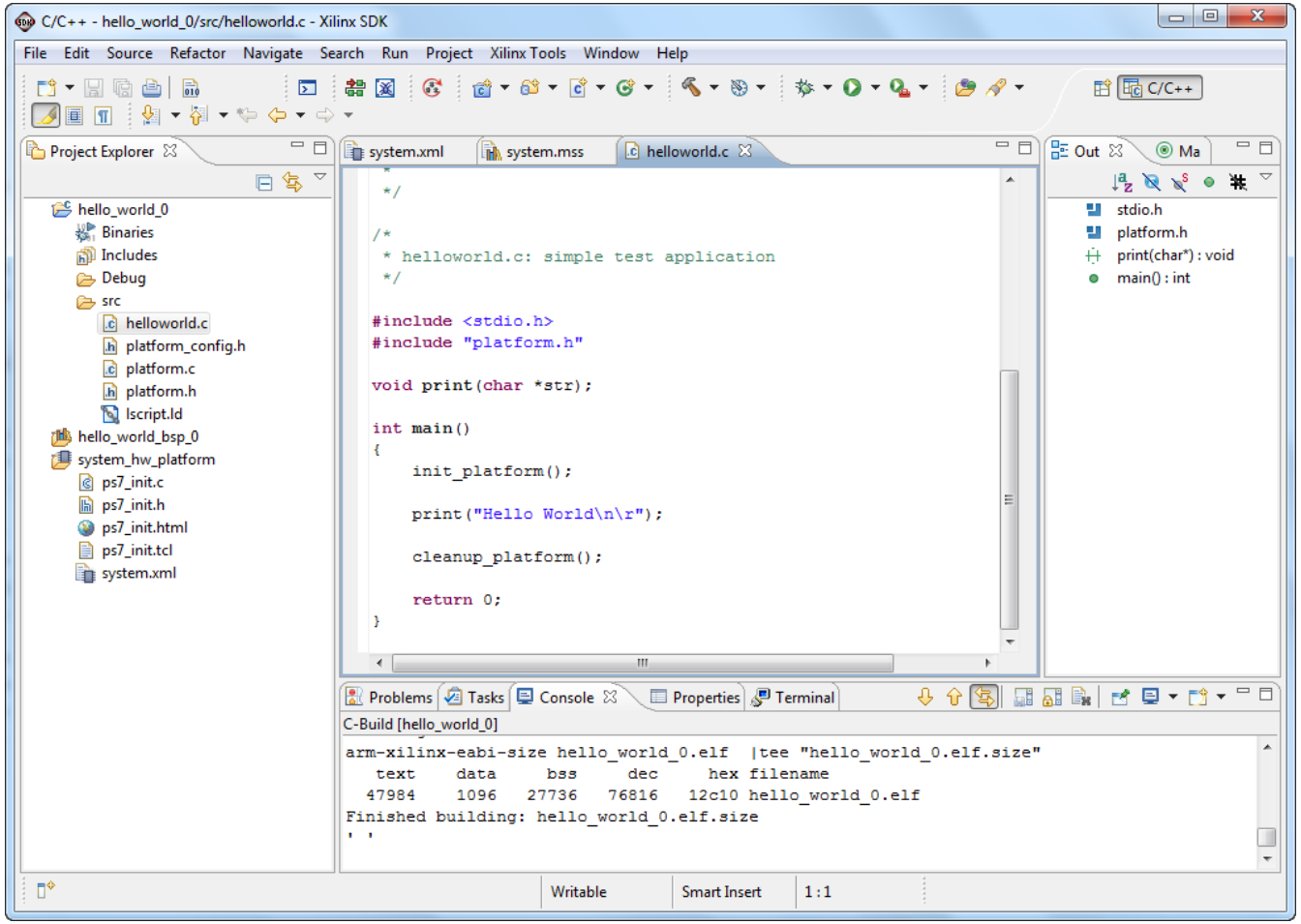
③ Code outline displays elements of the software file under development with file decorators (icons) for easy identification

④ Problems, Console, Properties views list output information associated with the software development flow



# Build Software Application in SDK

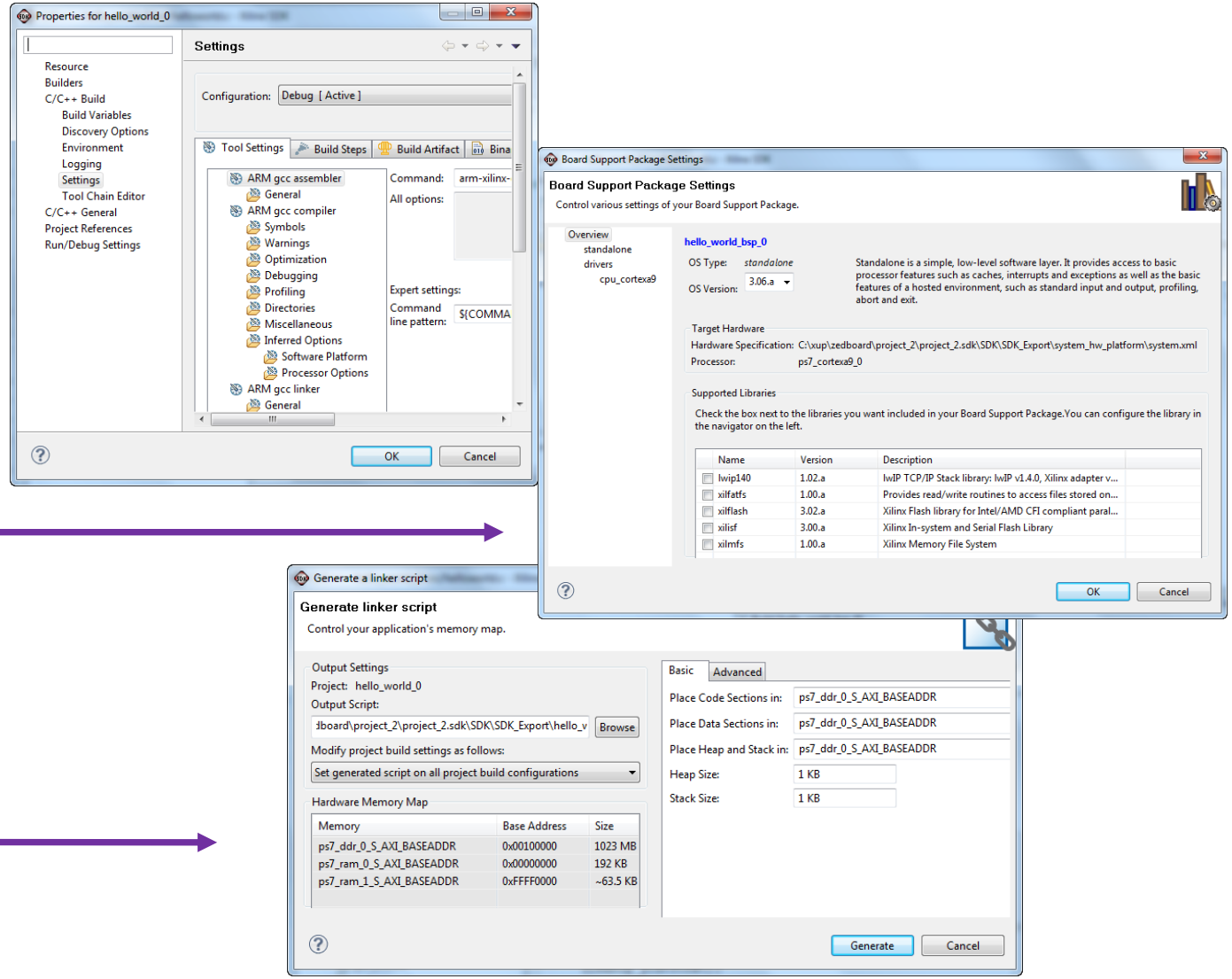
- Create software platform
  - System software, board support package
  - LibGen program
- Create software application
- Optionally, create linker script
- Build project
  - Compile, assemble, link output file *<app_project>.elf*



# Software Management Settings

**Software is managed in three major areas**

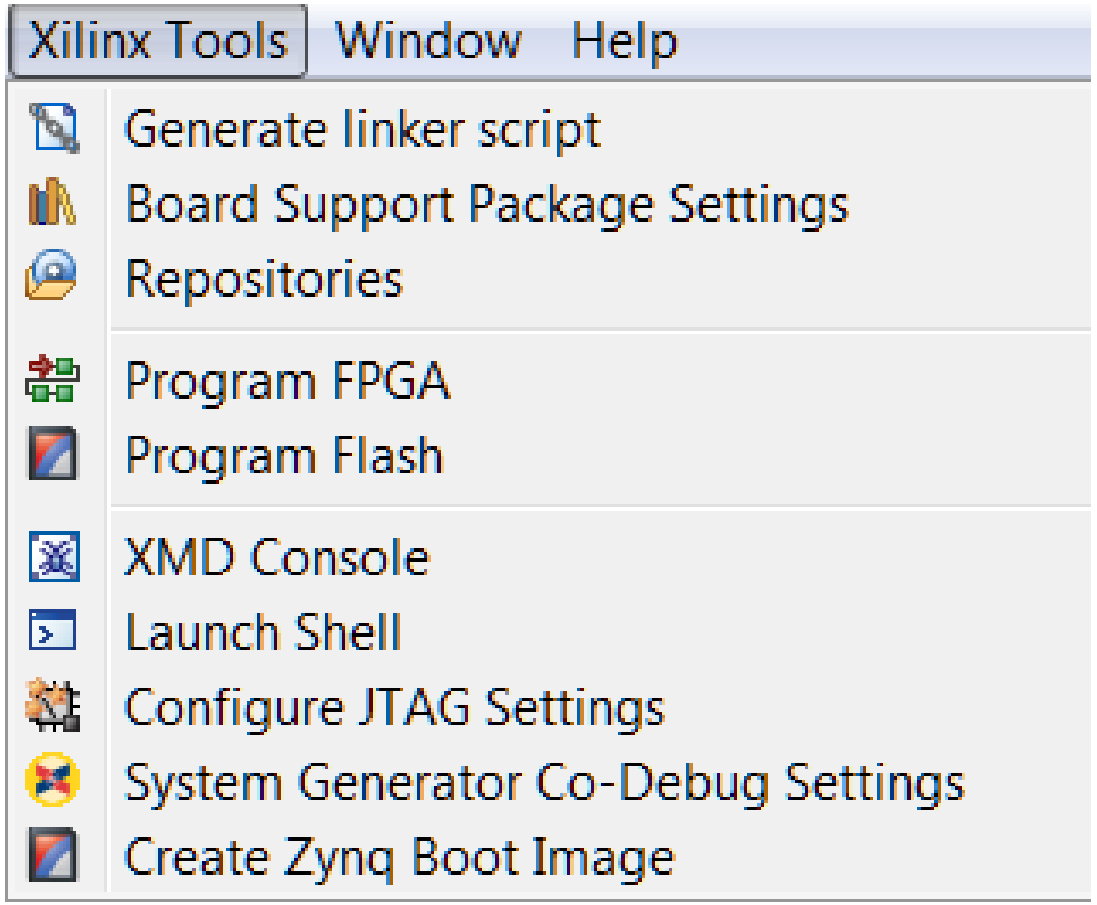
- Compiler/Linker Options
  - Application program
- Software Platform Settings
  - Board support package
- Linker Script Generation
  - Assigning software to memory resources



# Integrated Xilinx Tools in the SDK

Xilinx additions to the Eclipse IDE

- BSP Settings
- Software Repositories
- Generate Linker Script
- Program the programmable logic
  - Bitstream must be available
- Create Zynq Boot Image
- Program Flash Memory
- Launch XMD Console
- Launch Shell
- Configure JTAG Settings
- SysGen Co-Debug Settings



# 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 the design data*

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

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)

The **Vivado** tools write a journal file called ***vivado.jou*** into the directory from which **Vivado** was launched. The journal is a record of the Tcl commands run during the session.

Thus, they can be used as a starting point to create a new Tcl script

# 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
```

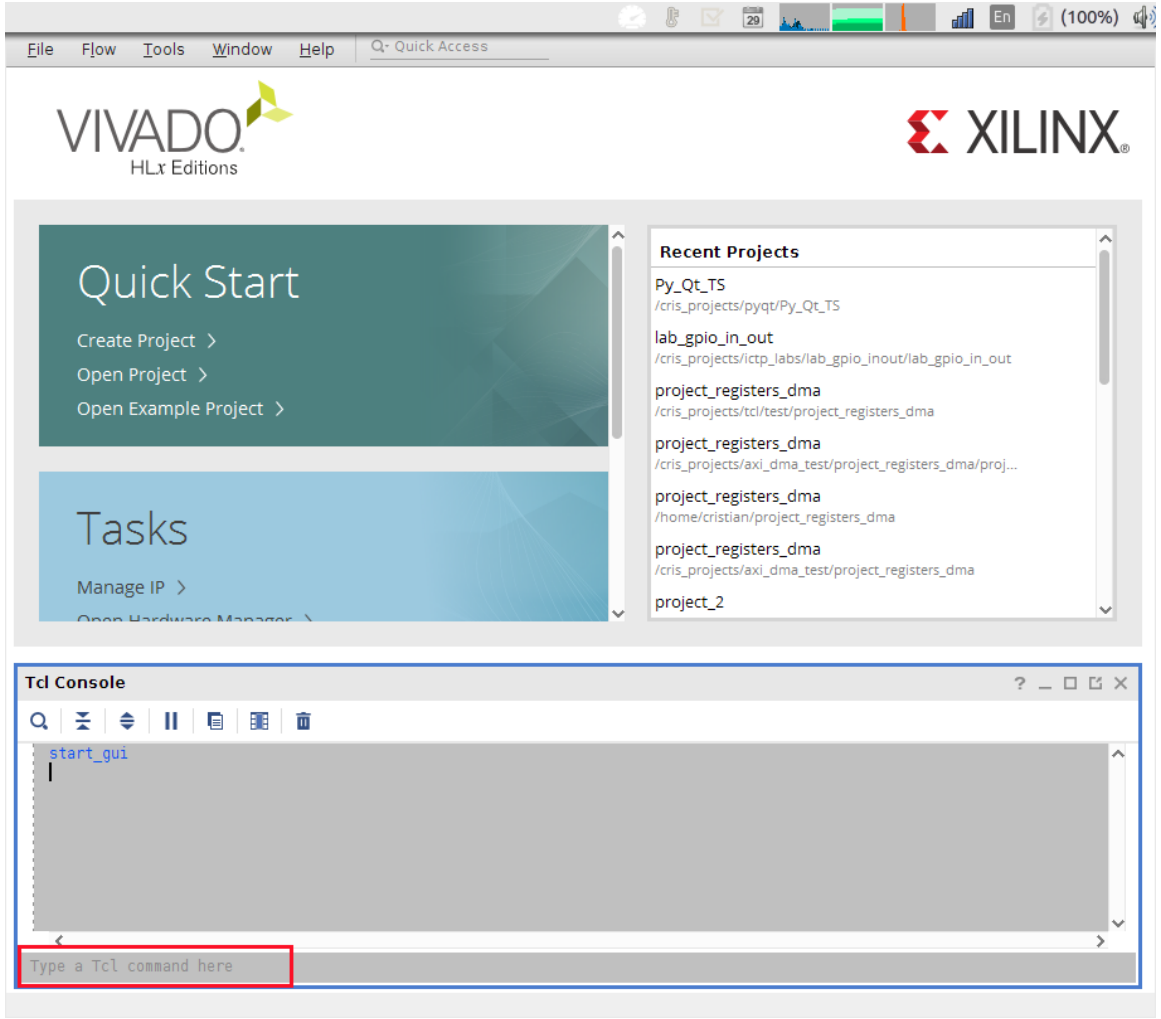
# How to run a provided .tcl script

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

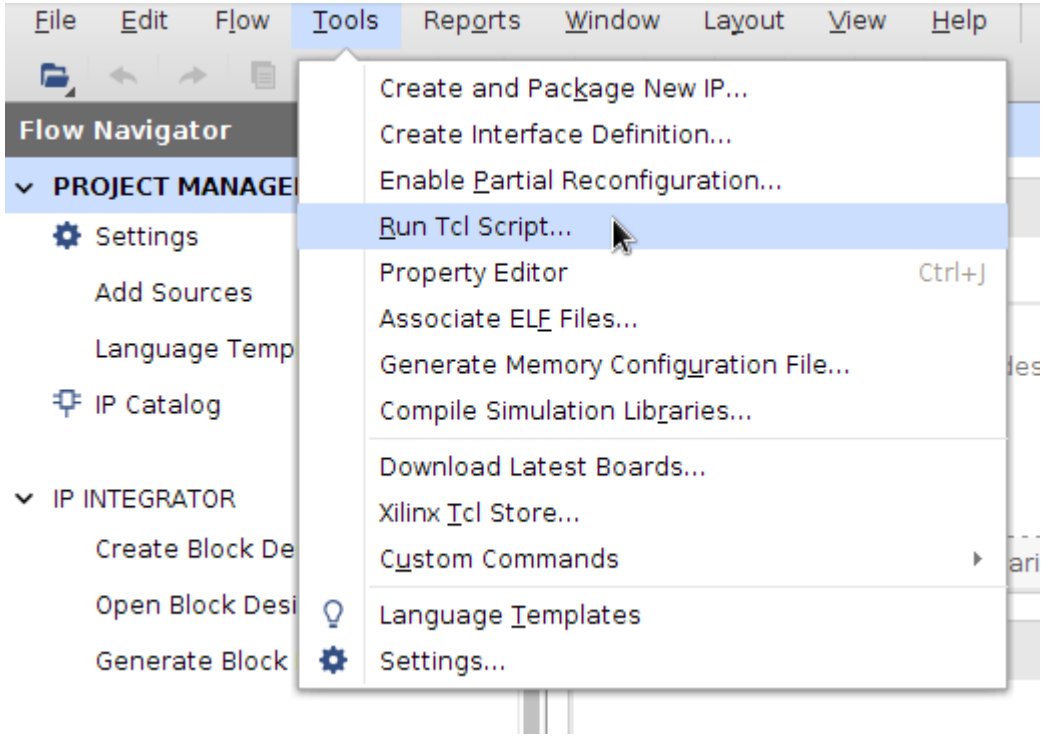
# Method 1: Run .tcl in Vivado TCL Console

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 some times 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

1. In W10 you can start the Vivado TCL Shell by doing Start-> All apps->Vivado 2019.1 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 installed Vivado, the Tcl interpreter should be installed*
3. Execute TCL script: You can either execute using "tclsh helloworld.tcl" or "./helloworld.tcl".
   - *$ tclsh helloworld.tcl*
   - *Hello World!*

   *( or )*
   - *$ chmod u+x helloworld.tcl*
   - *$ ./helloworld.tcl*
   - *Hello World!l.*

# 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 single execution

# TCL Docs

Vivado Design Suite TCL Command Reference Guide

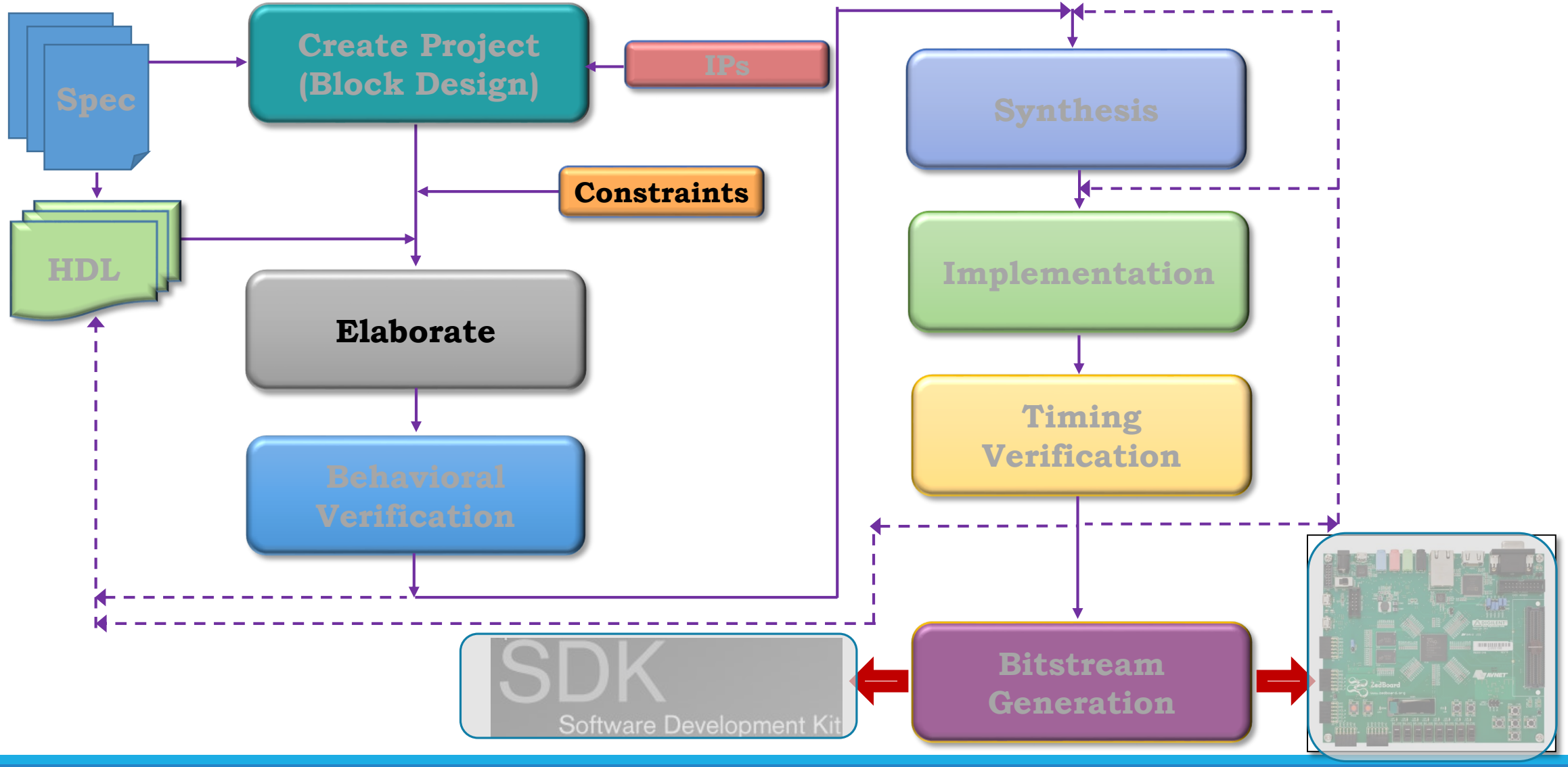
Vivado Design Suite User Guide - Using TCL Scripting

TCL Tutorial (up to Chapter 14 for Vivado appl

# Apendix

# *Vivado Design Suite*
# *Elaboration Process*

# Embedded System Design – Vivado Flow



# Elaboration

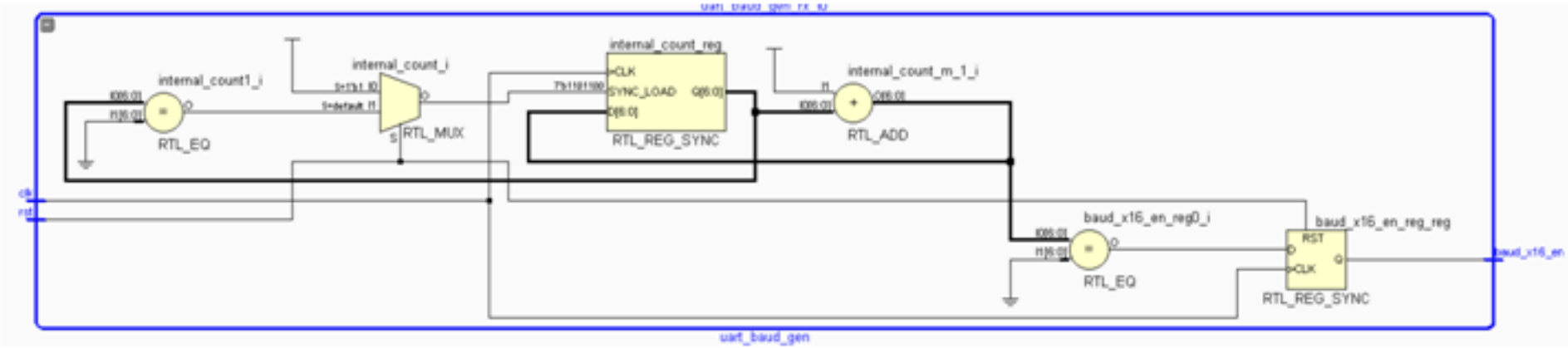
- Elaboration is the RTL optimization to an FPGA technology
- Vivado IDE allows designers to import and manage RTL sources
  - Verilog, System Verilog, VHDL, NGC, or testbenches
- Create and modify sources with the RTL Editor
  - Cross-selection between all the views
- Sources view
  - Hierarchy view: Display the modules in the design by hierarchy
  - Libraries view: Display sources by category

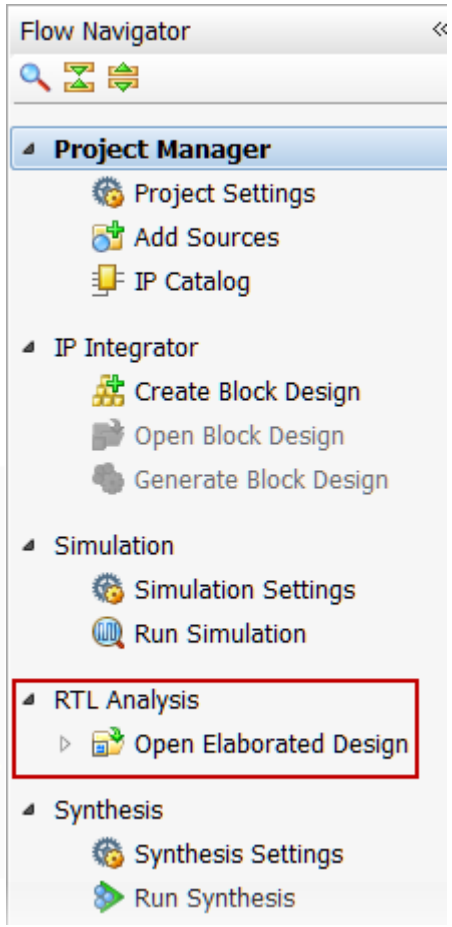
# Elaborated Design

Accessed through the Flow Navigator by selecting Open Elaborated Design

Representation of the design before synthesis

- Interconnected netlist of hierarchical and generic technology cells
  - Instances of modules/entities
  - Generic technology representations of hardware components
    - AND, OR, buffer, multiplexers, adders, comparators, etc…



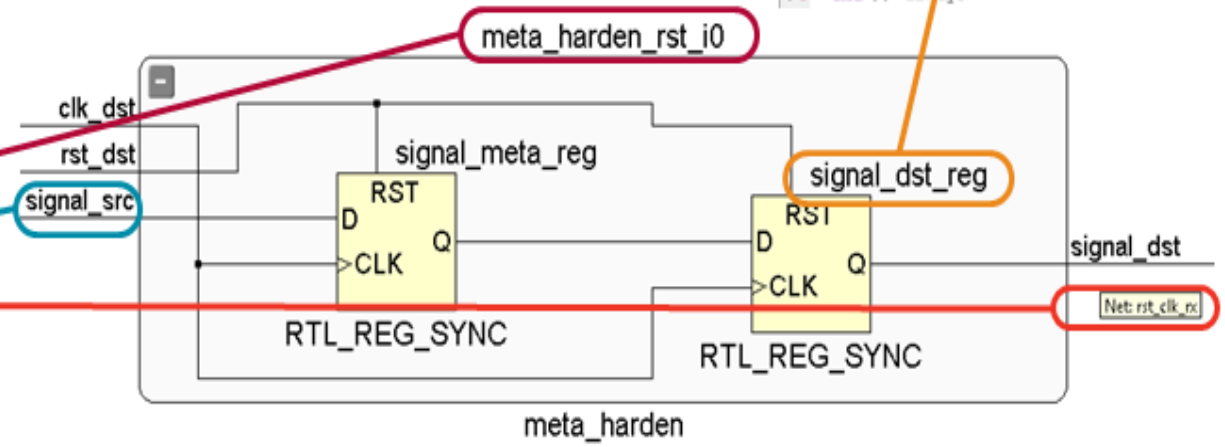
# Object Names in Elaborated Design

Object names are extracted from RTL

- Instance and pin names of hierarchical objects
- Inferred flip-flops from underlying reg/signal/logic
  - Suffix _reg is added
- Nets from underlying reg/signal/logic when it makes sense

```
39  reg          signal_meta;
40  reg          signal_dst;
41
42  always @(posedge clk_dst)
43  begin
44    if (rst_dst)
45    begin
46      signal_meta <= 1'b0;
47      signal_dst  <= 1'b0;
48    end
49    else // if rst_dst
50    begin
51      signal_meta <= signal_src;
52      signal_dst  <= signal_meta;
53    end // if rst
54  end // always
```

```
96   meta_harden meta_harden_rst_i0 (
97     .clk_dst      (clk_rx),
98     .rst_dst      (1'b0),
99     .signal_src   (rst_i),
100    .signal_dst   (rst_clk_rx)
101  );
```



# Elaboration and Analysis

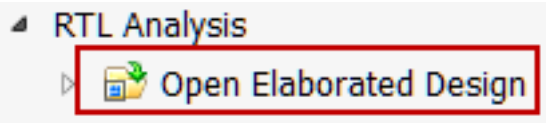
In a RTL based design, elaboration is the first step

Click on the Open Elaborated Design under RTL Analysis to

- Compile the RTL source files and load the RTL netlist for interactive analysis

You can check RTL structure, syntax, and logic definitions

Analysis and reporting capabilities include:

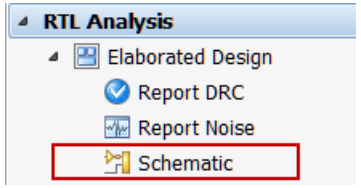


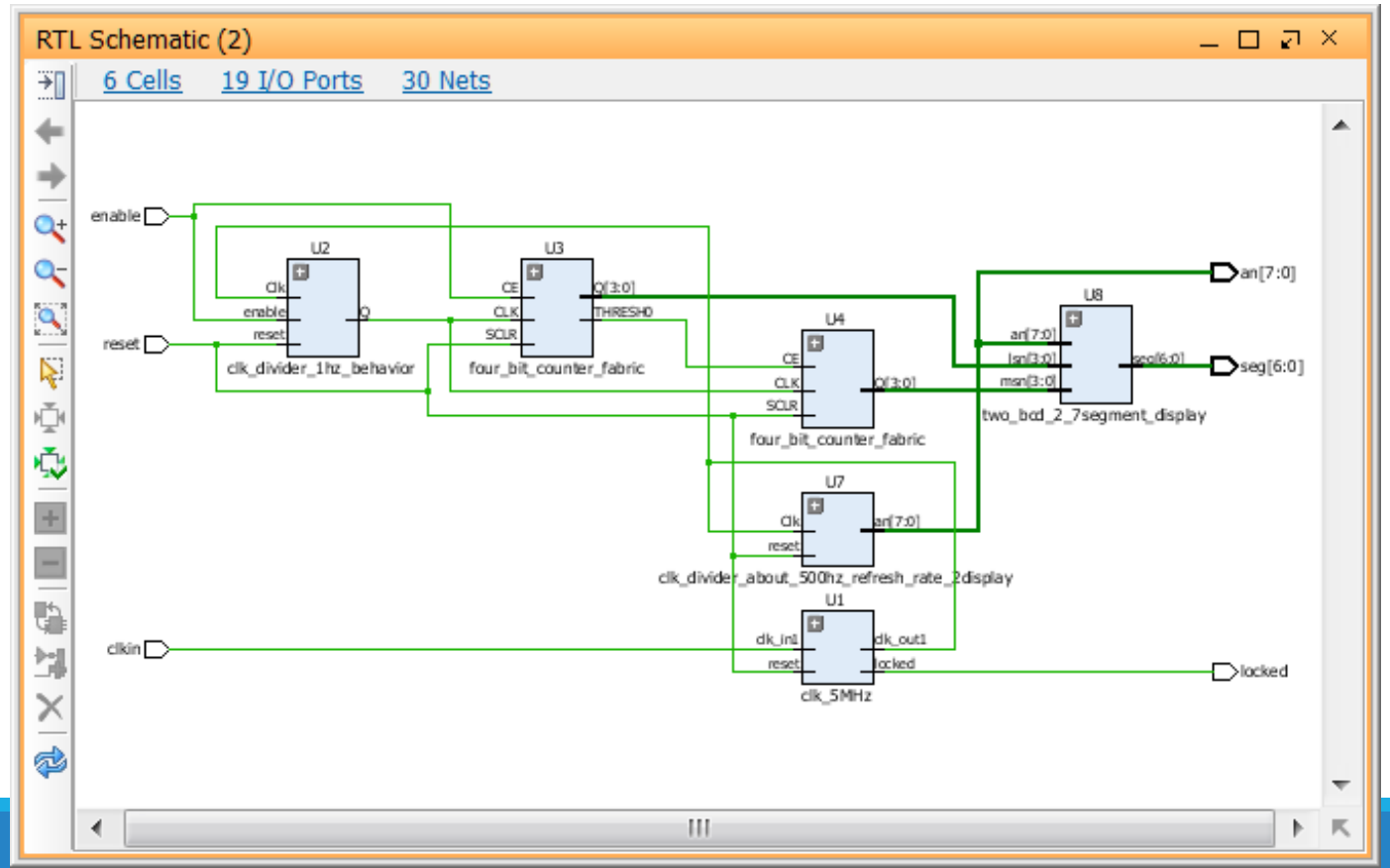

- RTL compilation validation and syntax checking
- Netlist and schematic exploration
- Design rule checks
- Early I/O pin planning using an RTL port list
- Ability to select an object in one view and cross probe to the object in other views, including instantiations and logic definitions within the RTL source files

# Schematic View of an Elaborated Design

When Schematic is clicked under the Elaborated Design, the schematic is opened showing the hierarchical blocks

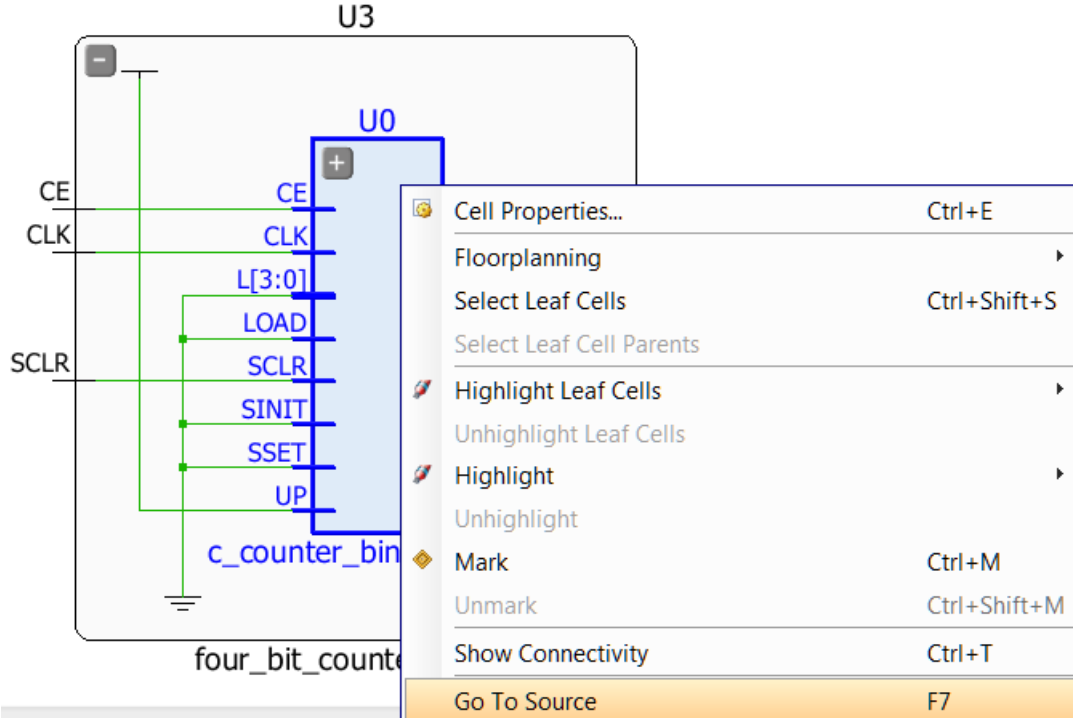
- Note that no IO buffers are inferred at this stage



# Cross Probing

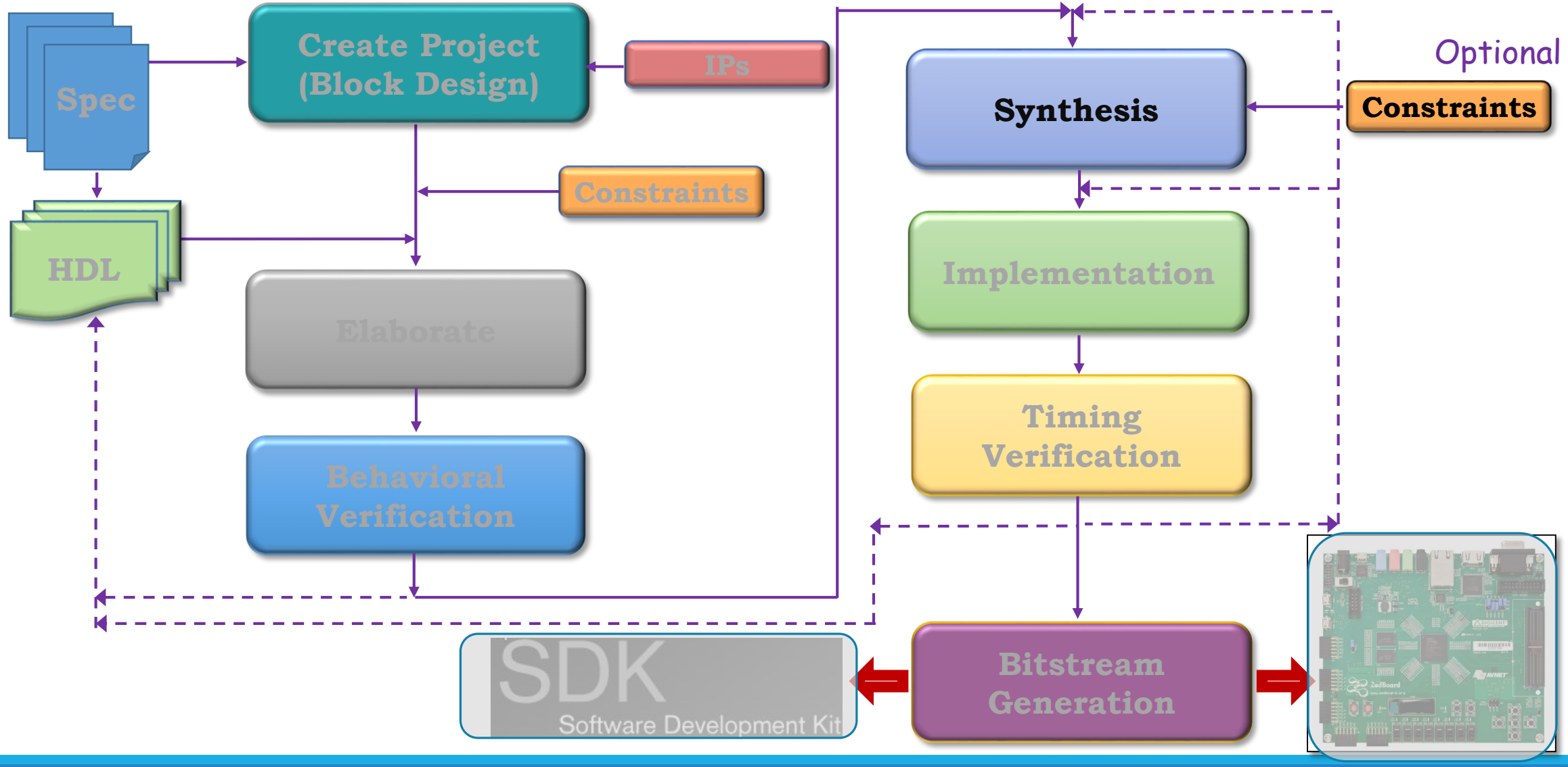
Select an object in the schematic, right-click, and select Go To Source to view where the object is defined in the source file




```
114 BEGIN
115   U0 : c_counter_binary_v12_0
116     GENERIC MAP (
117       C_IMPLEMENTATION => 0,
118       C_VERBOSITY => 0,
119       C_XDEVICEFAMILY => "artix7",
120       C_WIDTH => 4,
121       C_HAS_CE => 1,
122       C_HAS_SCLR => 1,
123       C_RESTRICT_COUNT => 1,
124       C_COUNT_TO => "1001",
125       C_COUNT_BY => "1",
126       C_COUNT_MODE => 0,
127       C_THRESH0_VALUE => "1001",
128       C_CE_OVERRIDES_SYNC => 0,
129       C_HAS_THRESH0 => 1,
130       C_HAS_LOAD => 0,
131       C_LOAD_LOW => 0,
132       C_LATENCY => 1,
133       C_FB_LATENCY => 0,
134       C_AINIT_VAL => "0",
135       C_SINIT_VAL => "0",
136       C_SCLR_OVERRIDES_SSET => 1,
137       C_HAS_SSET => 0,
138       C_HAS_SINIT => 0
139     )
```

# *Vivado Design Suite*
## *Synthesis Process*

# Embedded System Design – Vivado Flow

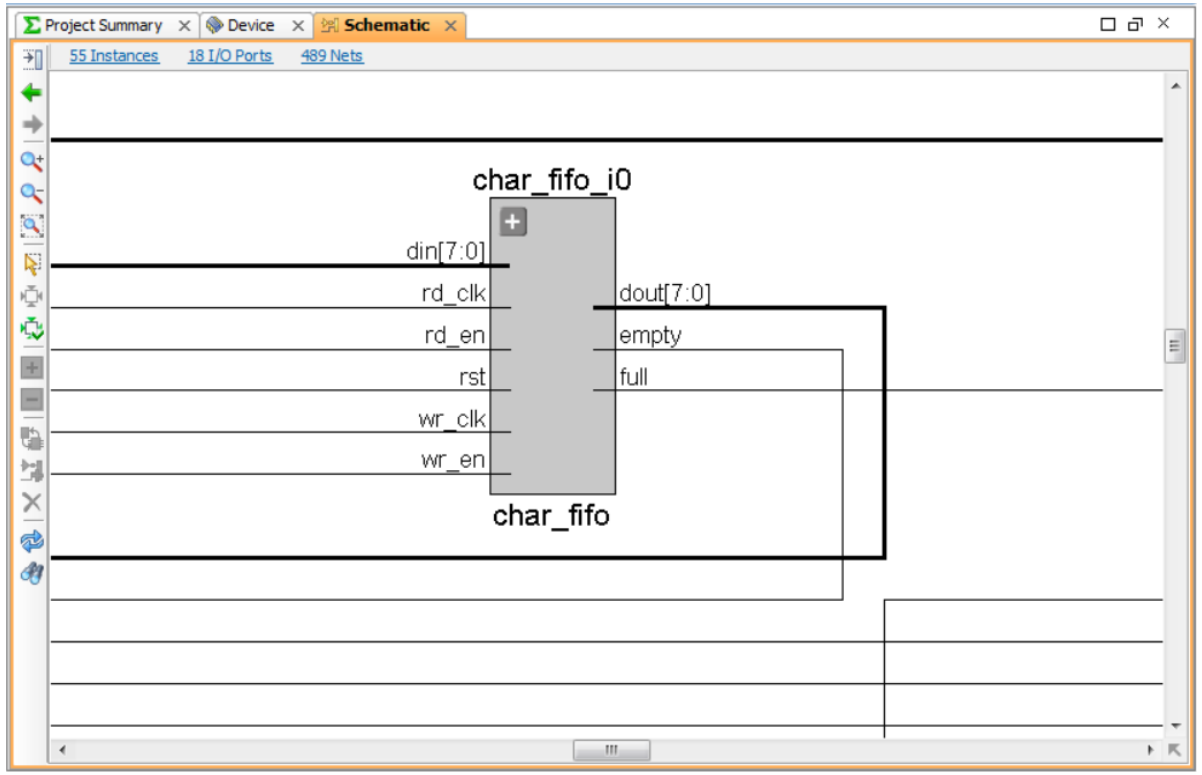


# Vivado IDE Synthesis

- **Applicable only for RTL (HDL) design flows**
  - EDIF is black boxed and linked after synthesis
- **Synthesis tool uses XDC constraints to drive synthesis optimization**
  - Design must first be synthesized without timing constraints for constraints editor usage
  - XDC file must exist
- **Synthesis settings provide access to additional options**

# Logic Optimization and Mapping to Device Primitives

Synthesis of an RTL design not only optimizes the gate-level design but also maps the netlist to Xilinx primitives (sometimes called technology mapping)



# Synthesized Design

**Accessed through the Flow Navigator by selecting Open Synthesized Design**

**Representation of the design after synthesis**

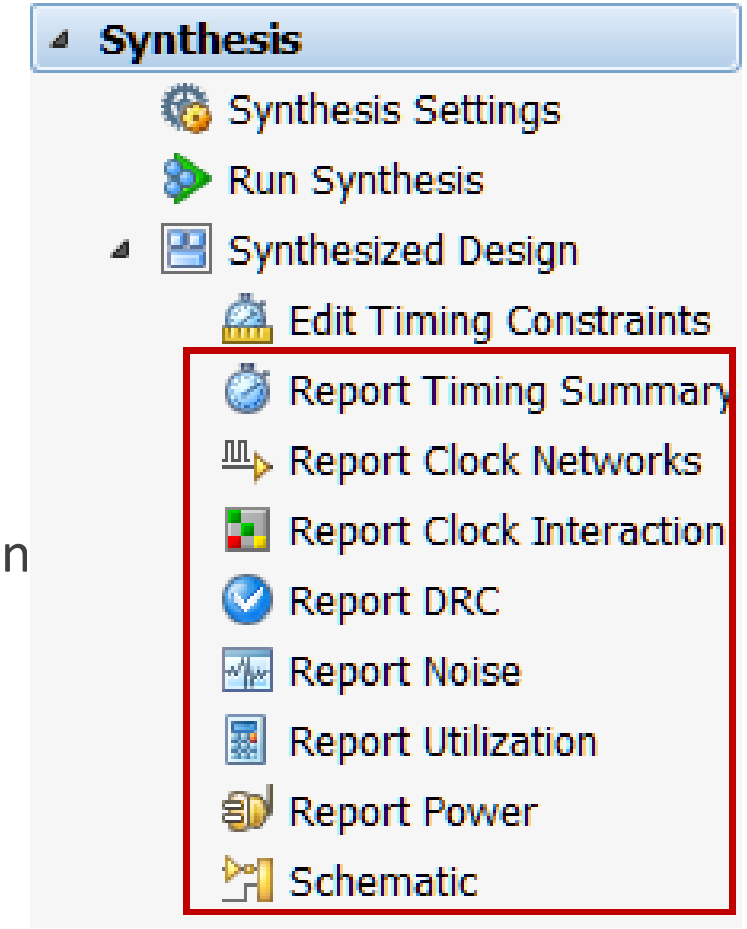
- Interconnected netlist of hierarchical and basic elements (BELs)
  - Instances of modules/entities
  - Basic elements
    - LUTs, flip-flops, carry chain elements, wide MUXes
    - Block RAMs, DSP cells
    - Clocking elements (BUFG, BUFR, MMCM, …)
    - I/O elements (IBUF, OBUF, I/O flip-flops)

**Object names are the same as names in the elaborated netlist when possible**

# Commands Available After Synthesis

Flow Navigator is optimized to provide quick access to the options most frequently used after synthesis

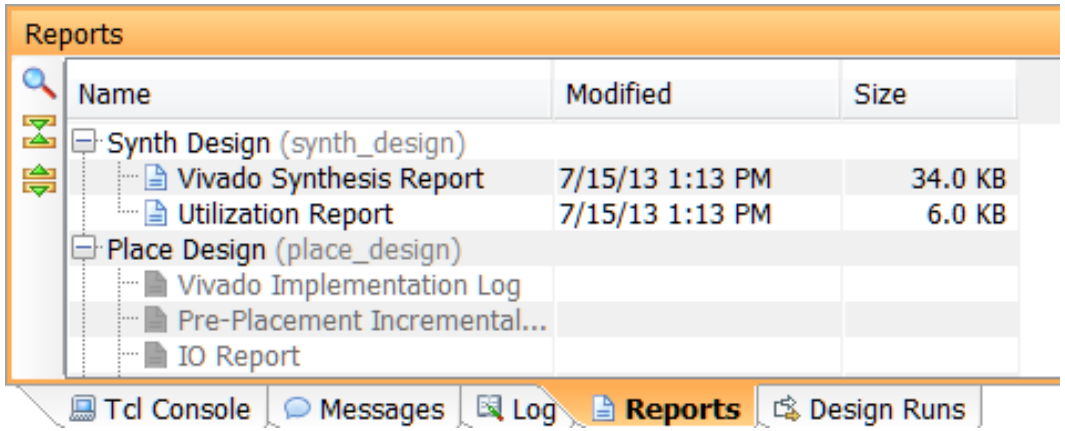
- Report Timing Summary: Generate a default timing report
- Report Clock Networks: Generates a clock tree for the design
- Report Clock Interaction: Verifies constraint coverage on paths between clock domains
- Report DRC: Performs design rule check on the entire design
- Report Noise: Performs an SSO analysis of output and bidirectional pins in the design
- Report Utilization: Generates a graphical version of the Utilization Report
- Report Power: Detailed power analysis reports that can be customized for the power supply and application environment
- Schematic: Opens the Schematic viewer



# Synthesis Reports

While the Flow Navigator points to the most important reports, the Reports tab contains several other useful reports

- Vivado Synthesis Report shows
  - HDL files synthesized, synthesis progress, timing constraints read, and RTL primitives from the RTL design
  - Timing optimization goals, technology mapping, removed pins/ports, and final cell usage (technology-mapped cell usage)
- Utilization Report shows
  - Technology-mapped cell usage in an easy-to-read tabular format



# Synthesis Utilization Report

Reports slice logic, memory, DSP slice, IO, clocking, and other resources used by the design

```
Copyright 1986-1999, 2001-2013 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------
| Tool Version : Vivado v.2013.2 (win64) Build 272601 Sat Jun 15
| Date         : Mon Jul 15 13:13:54 2013
| Host         :  running 64-bit Service Pack 1  (build 7601)
| Command      : report_utilization -file two_digits_counter_on_
| Design       : two_digits_counter_on_2_7segment_display
| Device       : xc7a100t
| Design State : Synthesized
------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
2. Memory
3. DSP
4. IO and GTX Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------
```

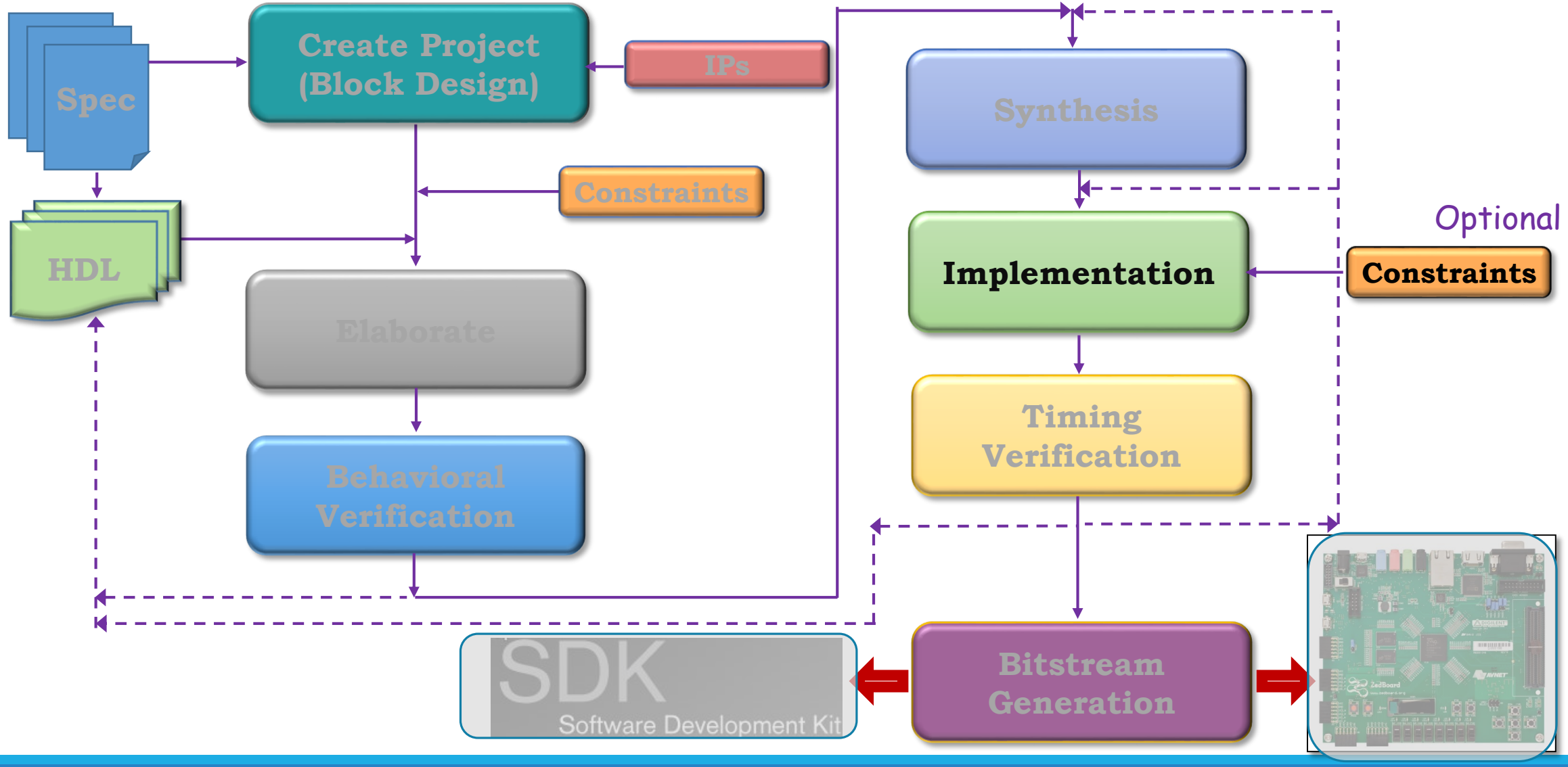
```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Loced | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   73 |     0 |     63400 |  0.11 |
|   LUT as Logic          |   73 |     0 |     63400 |  0.11 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |   50 |     0 |    126800 |  0.03 |
|   Register as Flip Flop |   50 |     0 |    126800 |  0.03 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    4 |     0 |     31700 |  0.01 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

```
5. Clocking
-----------

+-------------+------+-------+-----------+-------+
|  Site Type  | Used | Loced | Available | Util% |
+-------------+------+-------+-----------+-------+
| BUFGCTRL    |    2 |     0 |        32 |  6.25 |
| BUFIO       |    0 |     0 |        24 |  0.00 |
| MMCME2_ADV  |    1 |     0 |         6 | 16.66 |
| PLLE2_ADV   |    0 |     0 |         6 |  0.00 |
| BUFMRCE     |    0 |     0 |        12 |  0.00 |
| BUFHCE      |    0 |     0 |        96 |  0.00 |
| BUFR        |    0 |     0 |        24 |  0.00 |
+-------------+------+-------+-----------+-------+
```

```
4. IO and GTX Specific
----------------------

+-----------------------------+------+-------+-----------+-------+
|          Site Type          | Used | Loced | Available | Util% |
+-----------------------------+------+-------+-----------+-------+
| Bonded IOB                  |   19 |     0 |       210 |  9.04 |
| Bonded IPADs                |    0 |     0 |         2 |  0.00 |
| IBUFGDS                     |    0 |     0 |       202 |  0.00 |
| IDELAYCTRL                  |    0 |     0 |         6 |  0.00 |
| IN_FIFO                     |    0 |     0 |        24 |  0.00 |
| OUT_FIFO                    |    0 |     0 |        24 |  0.00 |
| PHASER_REF                  |    0 |     0 |         6 |  0.00 |
| PHY_CONTROL                 |    0 |     0 |         6 |  0.00 |
| PHASER_OUT/PHASER_OUT_PHY   |    0 |     0 |        24 |  0.00 |
| PHASER_IN/PHASER_IN_PHY     |    0 |     0 |        24 |  0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY |    0 |     0 |       300 |  0.00 |
| ODELAYE2/ODELAYE2_FINEDELAY |    0 |     0 |         0 |  0.00 |
| IBUFDS_GTE2                 |    0 |     0 |         4 |  0.00 |
| ILOGIC                      |    0 |     0 |       210 |  0.00 |
| OLOGIC                      |    0 |     0 |       210 |  0.00 |
+-----------------------------+------+-------+-----------+-------+
```

# *Vivado Design Suite*
# *Implementacion Process*

# Embedded System Design – Vivado Flow



# Vivado Implementation Sub-Processes

Vivado Design Suite Implementation process transform a logical netlist (generated by the synthesis tool) into a placed and routed design ready for bitstream generation

- **Opt design**
  - Optimizes the logical design to make it easier to fit onto the target FPGA
- **Place design**
  - Places the design onto the FPGA's logic cells
- **Route design**
  - Routing of connections between the FPGA's cells

# Using Design Constraints for Guiding Implementation

There are two types of design constraints, *physical constraints* and *timing constraints*.

**Physical Constraints**: define a relationship between logic design objects and device resources

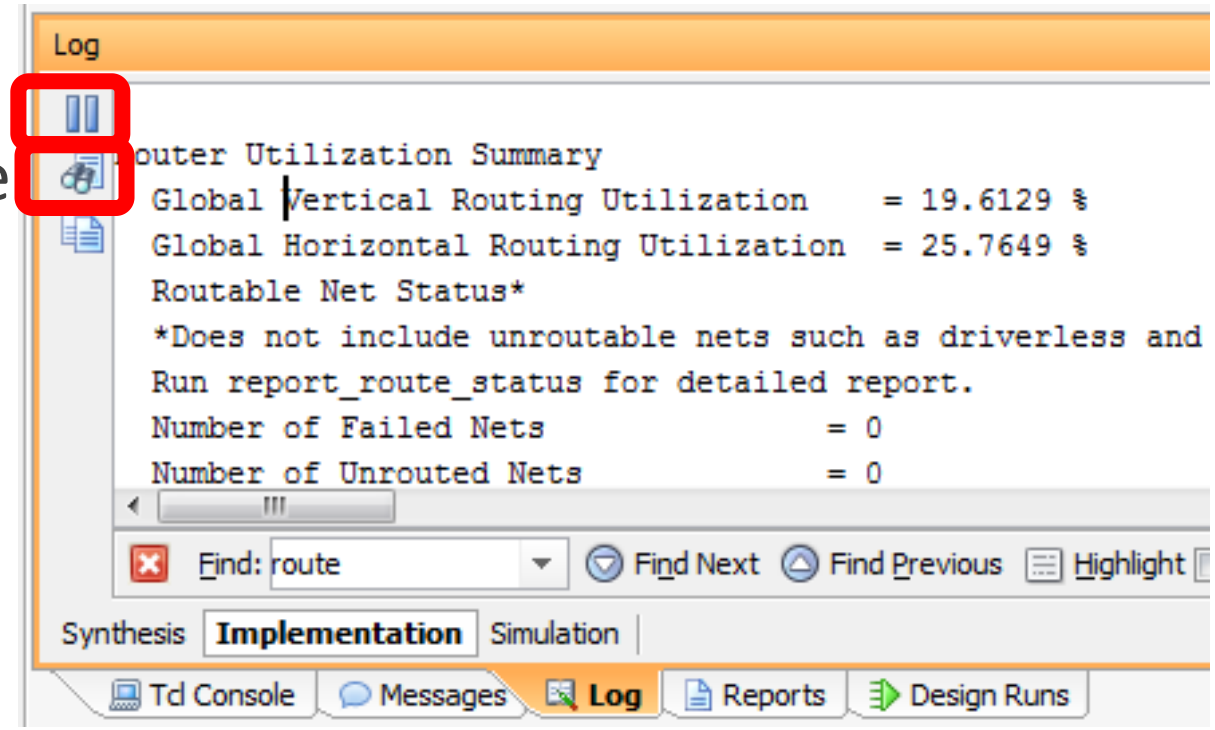
- Package pin placement
- Absolute or relative placement of cells:
  - Block RAM
  - DSP
  - LUTs
  - Filp-Flops
- Floorplanning constraints that assign cells to general regions of an FPGA

**Timing Constraints**: define the frequency requirements for the design. Without timing constraints, Vivado Design Suite optimizes the design solely for wire length and routing congestion and makes no effort to asses or improve design performance

# Implementation Log Messages

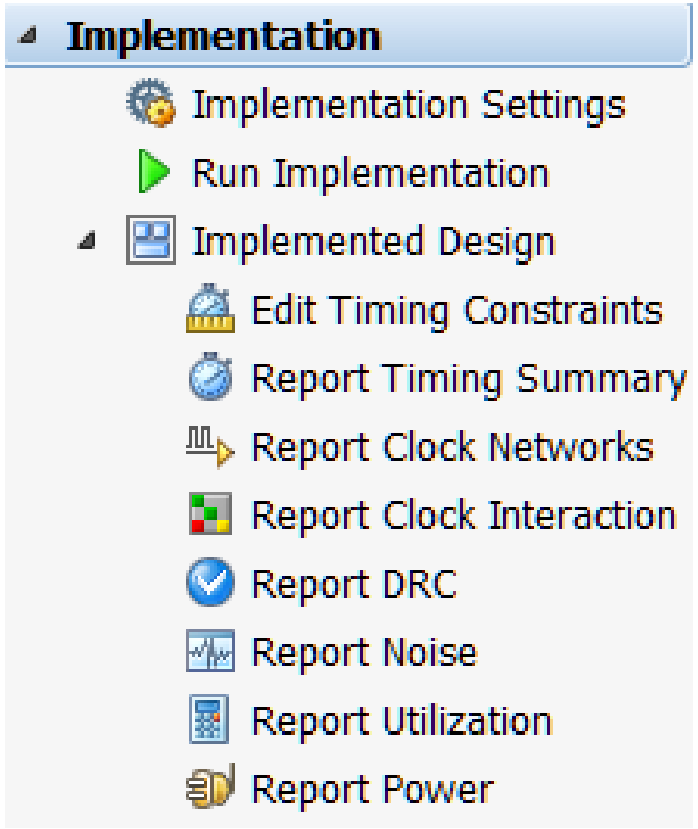
## Viewing the Log in the Log Window

The Log window opens in the Vivado IDE after you launch a run. It shows the standard output messages. It also displays details about the progress of each individual implementation process, such as *place_design* and *route_design.*

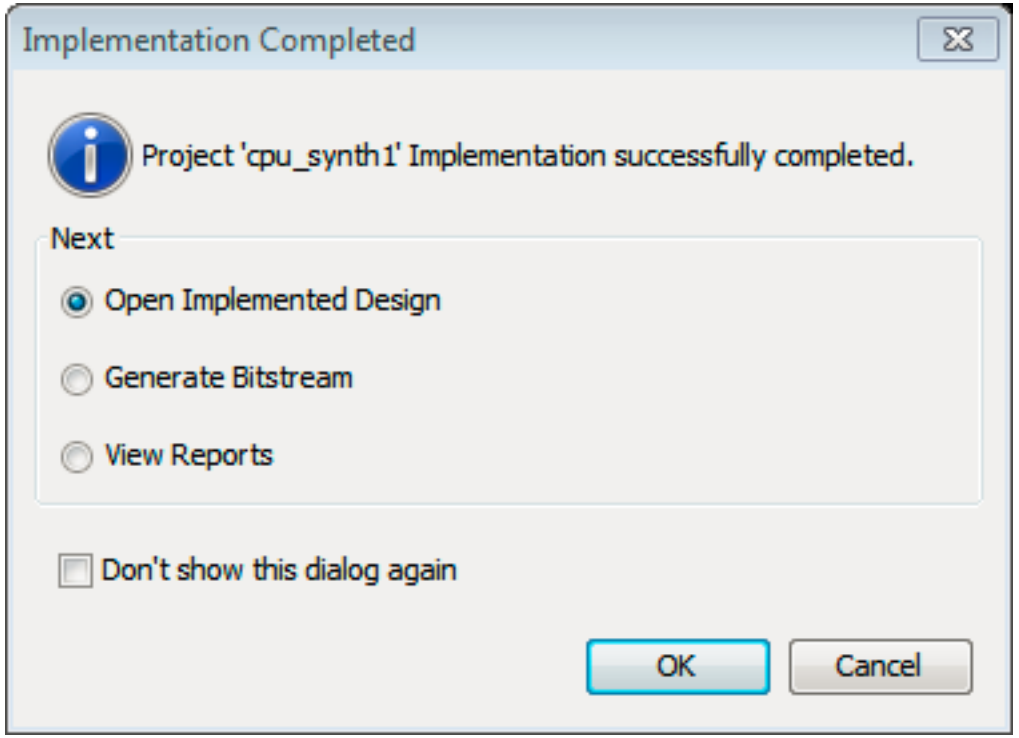


# After Implementation

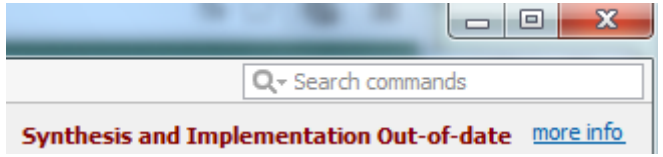
- Sources and Netlist tabs do not change
- Now as each resources is selected, it will show the exact placement of the resource on the die
- Timing results have to be generated with the Report Timing Summary
- As each path is selected, the placement of the logic and its connections is shown in the Device view
- This is the cross-probing feature that helps with static timing analysis

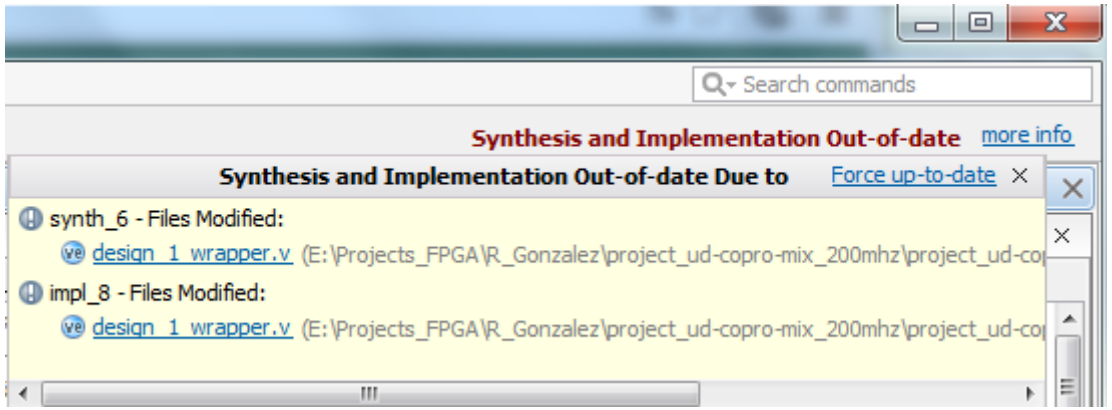


# After Completing Implementation

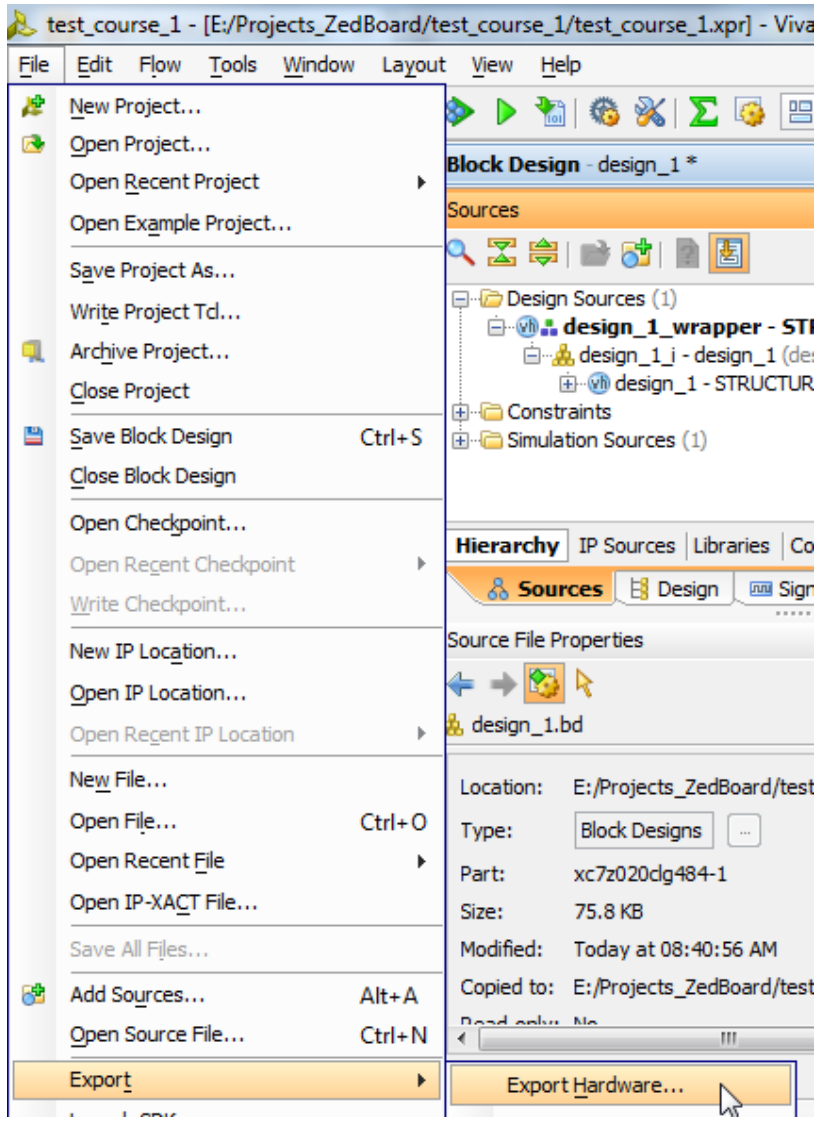


# Implementation Out-of-Date Message



# Exporting a Hardware Description



# Export Hardware Design to SDK

Software development is performed with the Xilinx Software Development Kit tool (SDK)

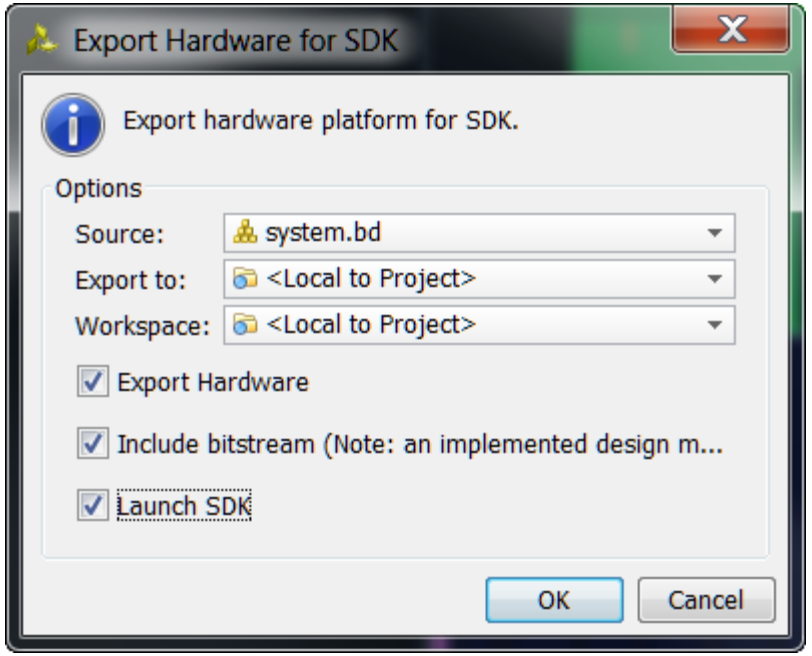
The design must be opened if a bitstream of the design is generated

The Block design must be open before the design can be exported

An XML description of the hardware is imported in the SDK tool

- The hardware platform is built on this description
- Only one hardware platform for an SDK project

The SDK tool will then associate user software projects to hardware



# Exporting IP Integrator Design to SDK – Main Files

| File | Description |
|---|---|
| system.xml | This file opens by default when you launch SDK and displays the address map of your system |
| ps7_init.c<br>s7_init.h | The ps7_init.c and ps7_init.h files contain the initialization code for the Zynq Processing System and initialization settings for DDR, clocks, PLLs and MIOs. SDK uses these settings when initializing the PS so applications can run on top of the PS. |
| ps7_init.tcl | This is the Tcl version of the ***init*** file |
| ps7_init.html | This ***init*** file describes the initialization data. |

# *Vivado Design Suite*

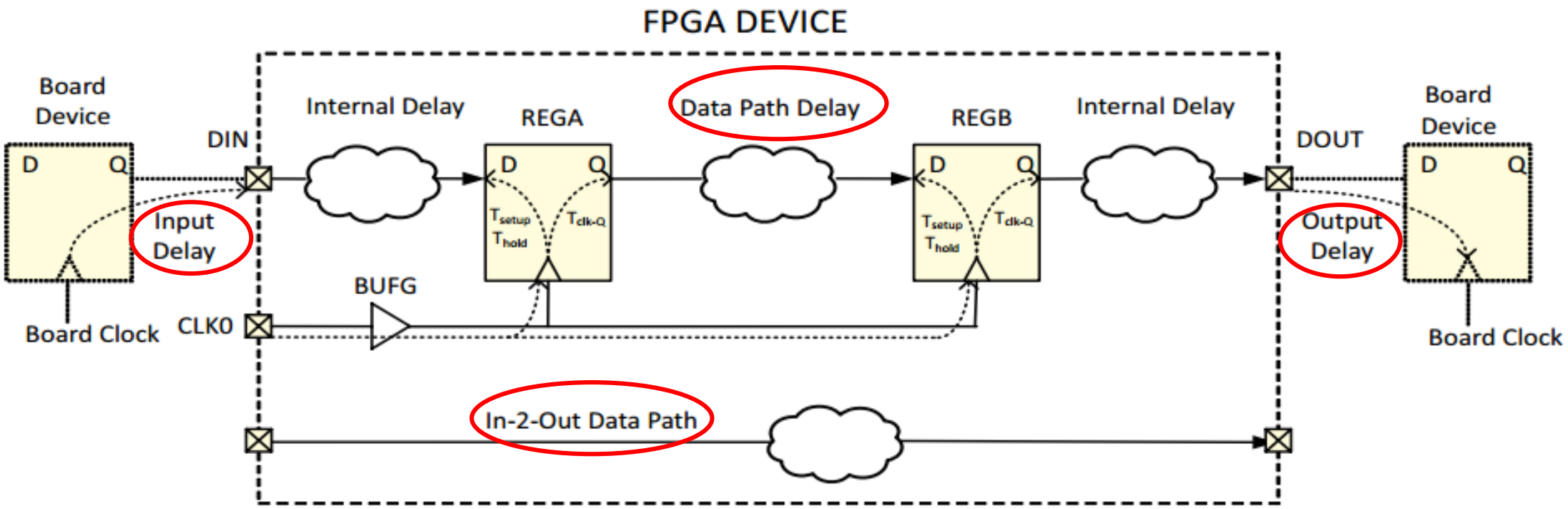
## *Basic Static Timing Constraints*

# Basic Timing Constraints

There are three basic timing constraints applicable to a sequential machine

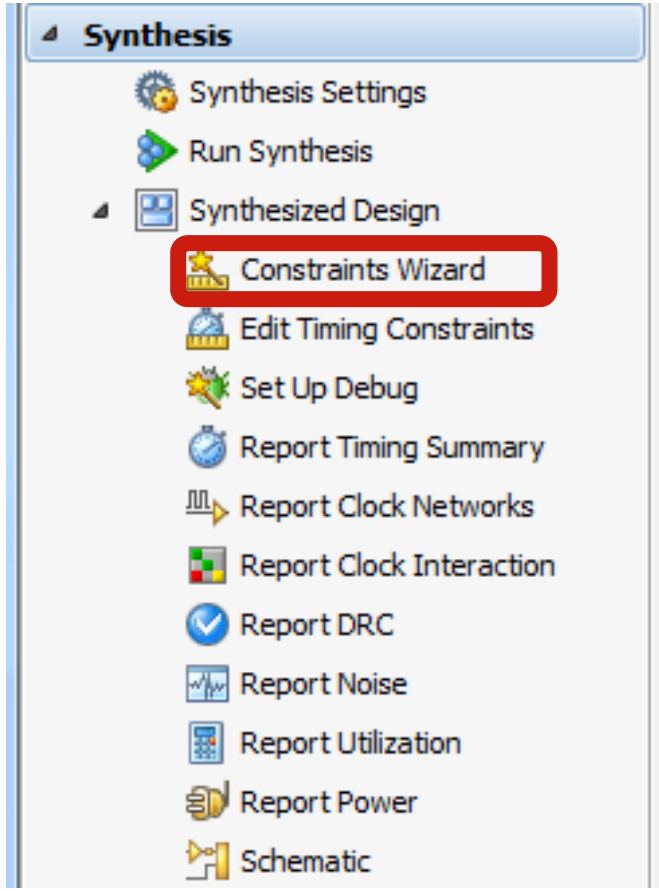
- Period
  - Paths between synchronous elements clocked by the reference clock net
    - Synchronous elements include flip-flops, latches, synchronous RAM, and DSP slices
  - Use `create_clock` to create the constraint
- Input Delay
  - Paths between input pin and synchronous elements
  - Use `set_input_delay` to create the constraint
- Output delay
  - Paths between synchronous elements and output pin
  - Use `set_output_delay` to create the constraint

# Timing Paths Example

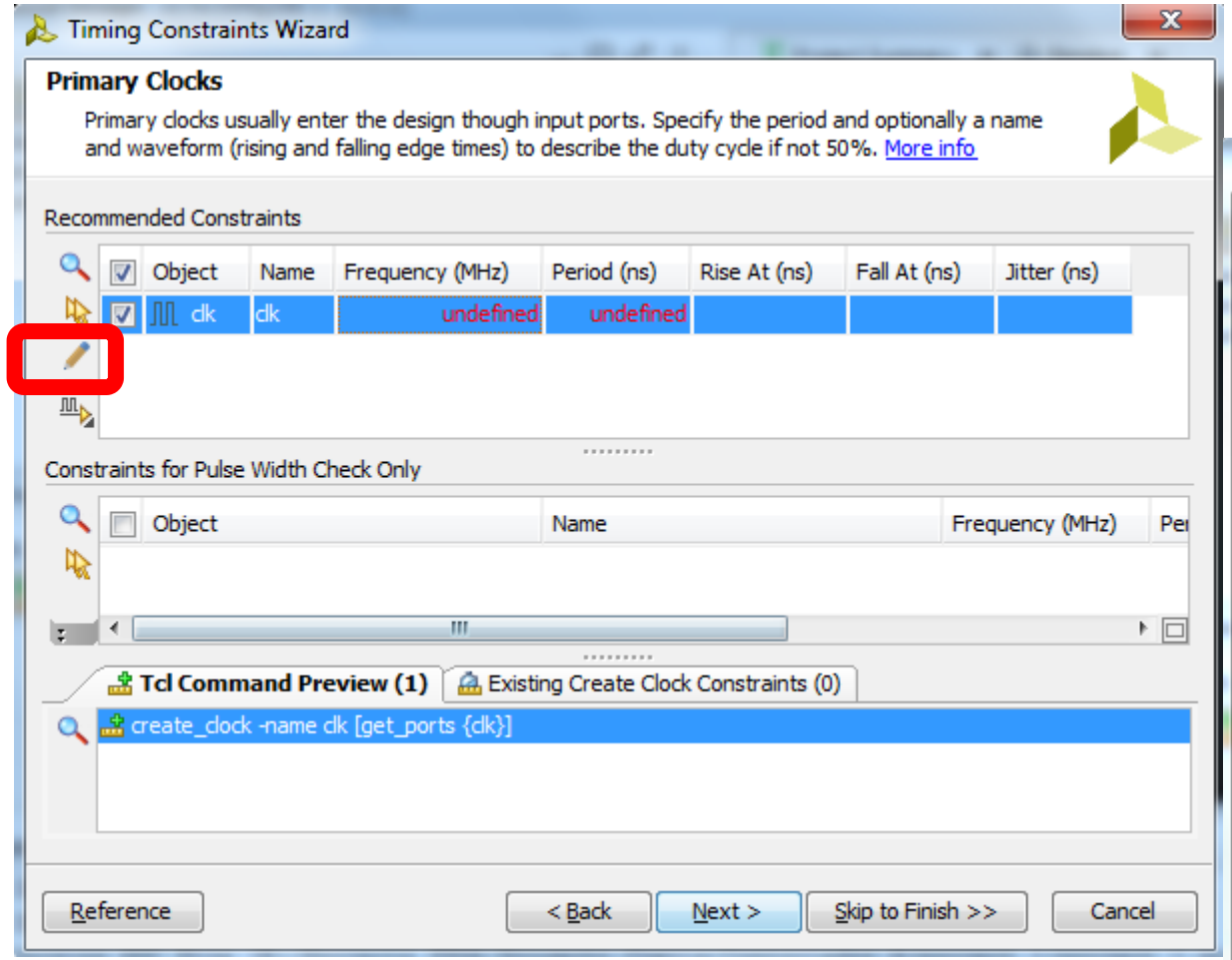


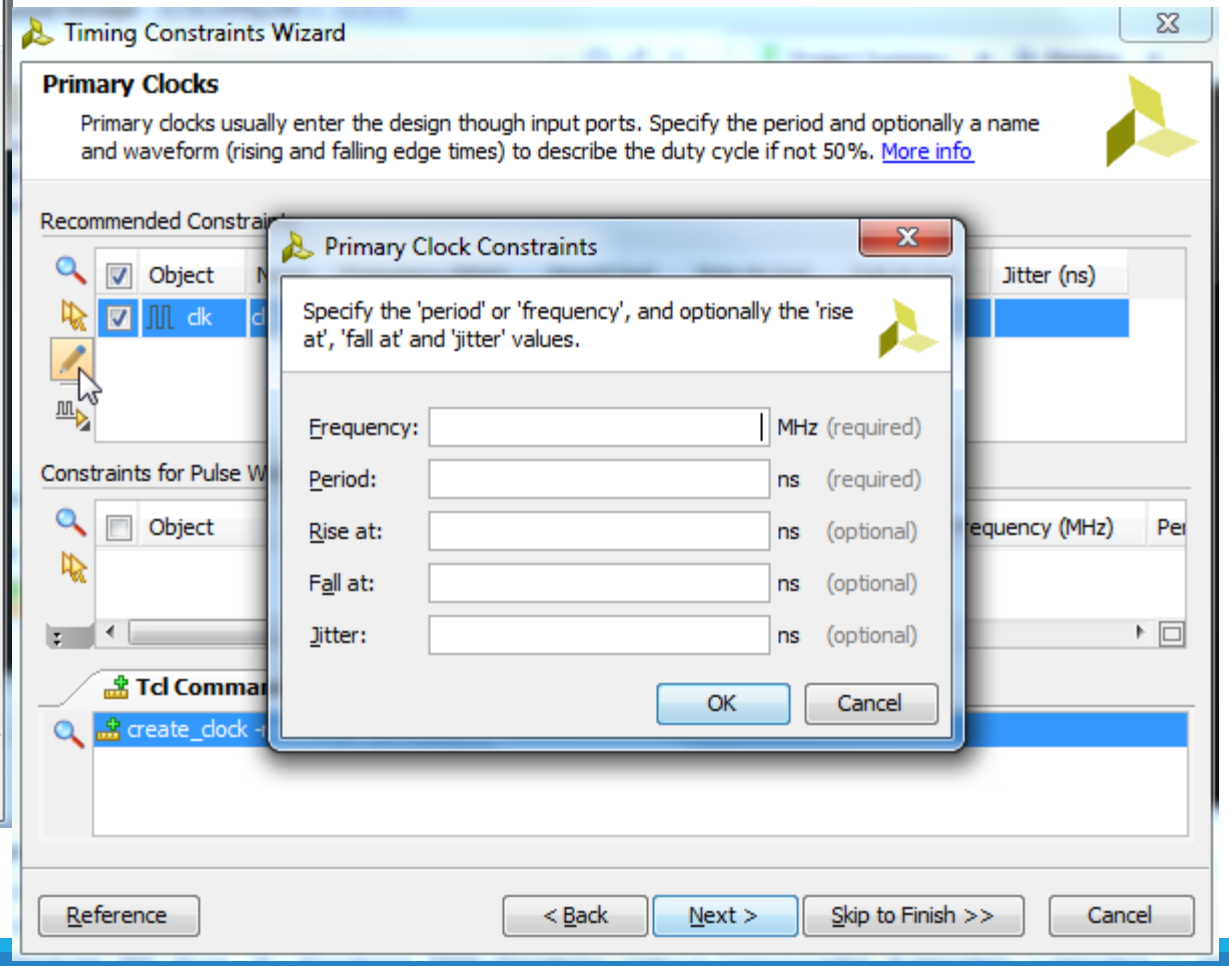
# Creating Basic Timing Constraints in Vivado IDE

1. Run Synthesis
2. Open the synthesized design
3. Invoke constraints editor

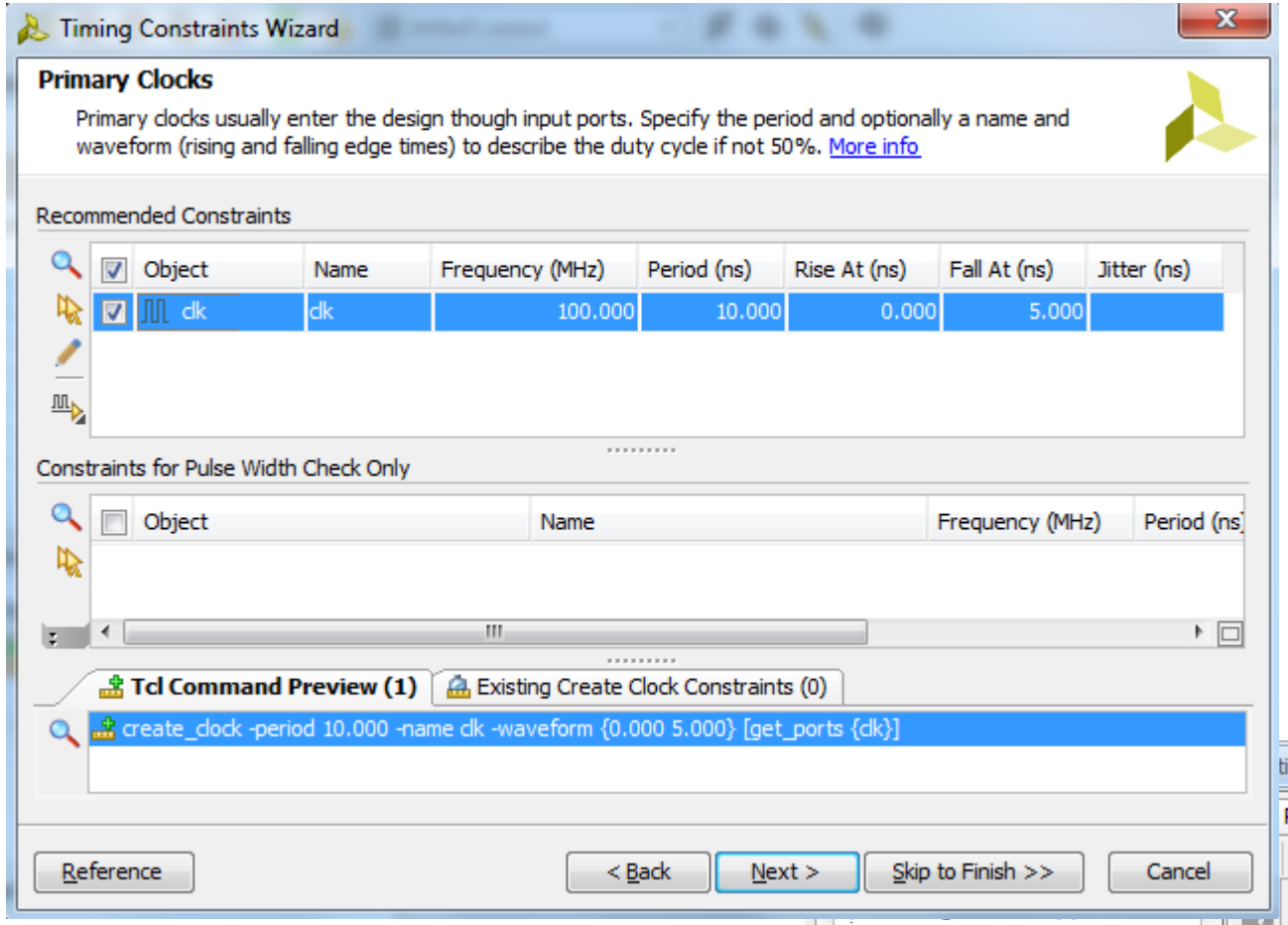


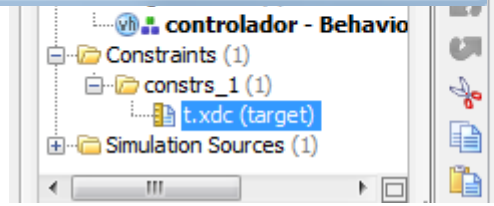
# Clock Constraint Setting

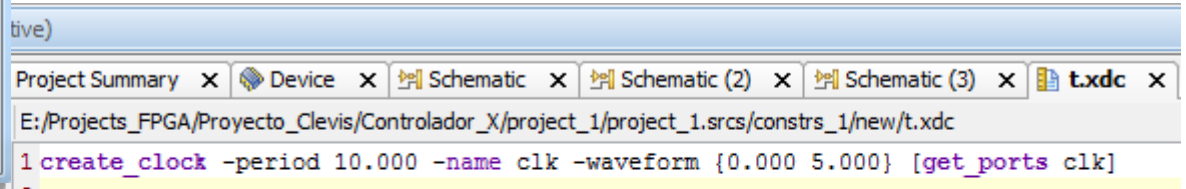


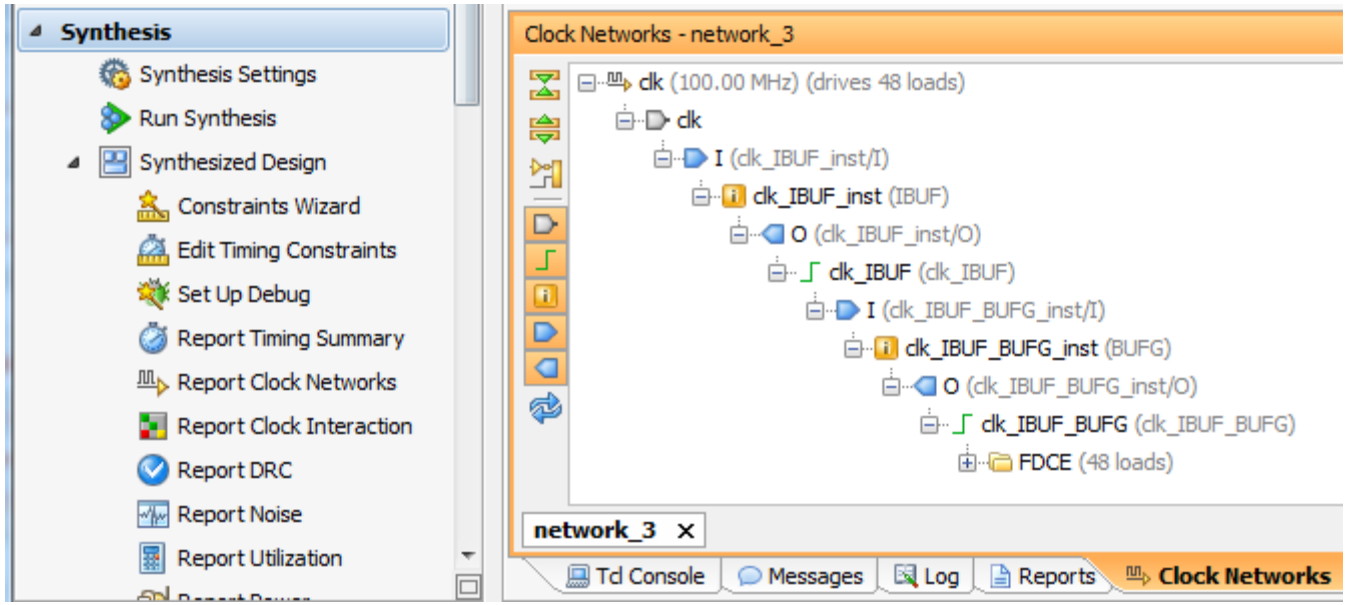
# Clock Constraint Setting



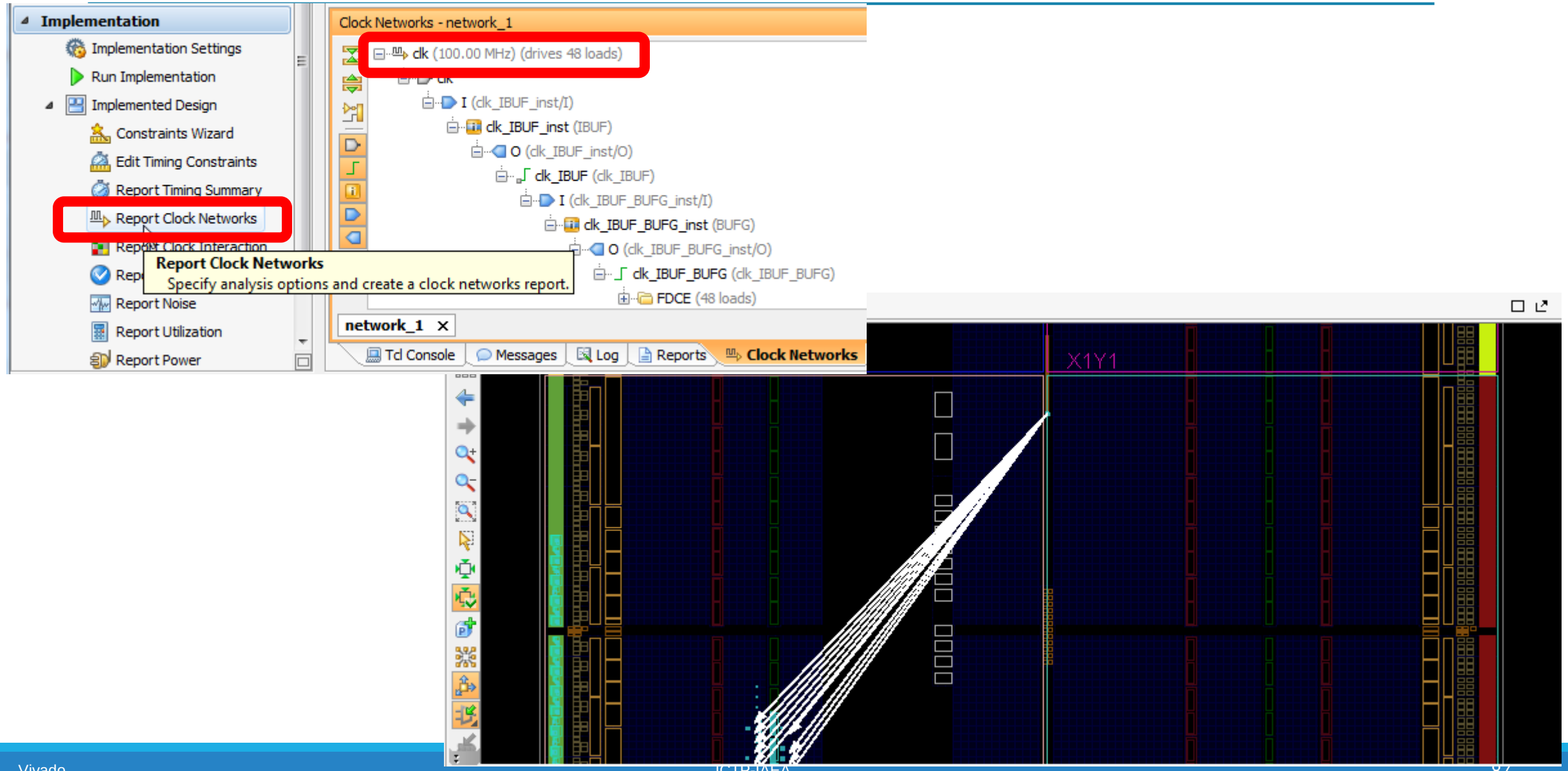



# Clock Network Report



# Clock Network Report and Visualization

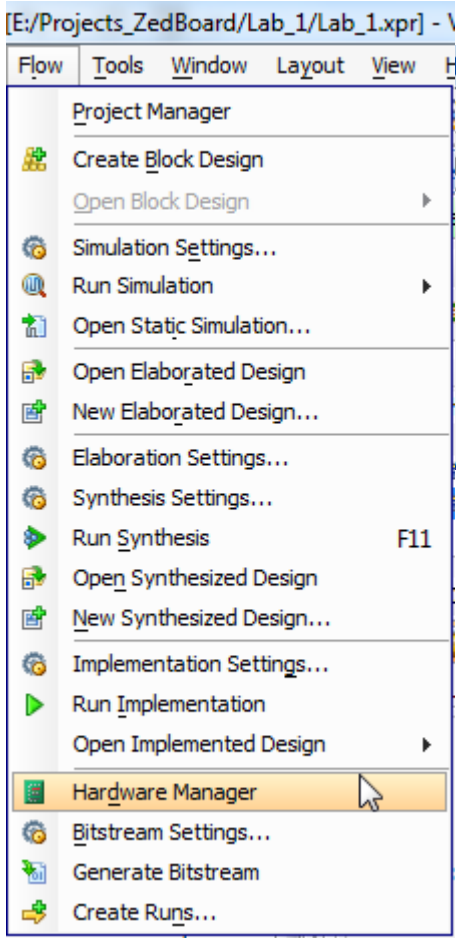


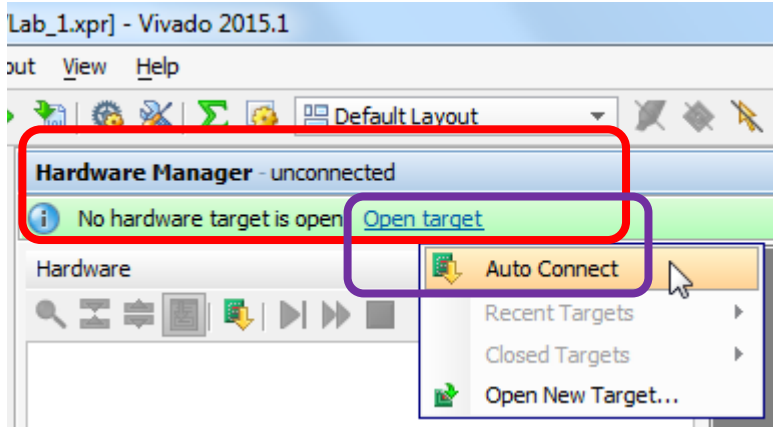
# *Vivado Design Suite*

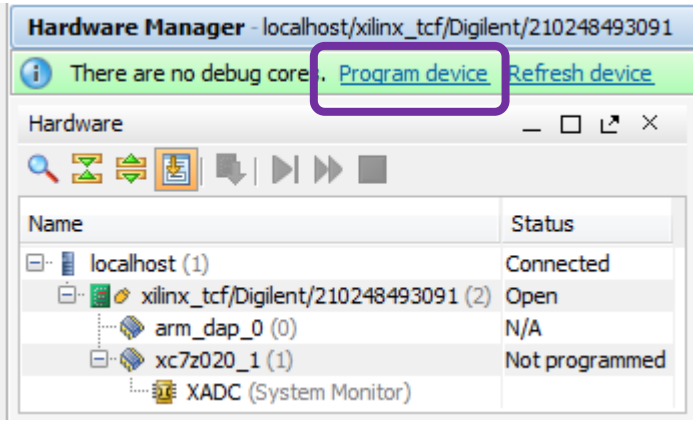
## *Generate Bit Stream Process*

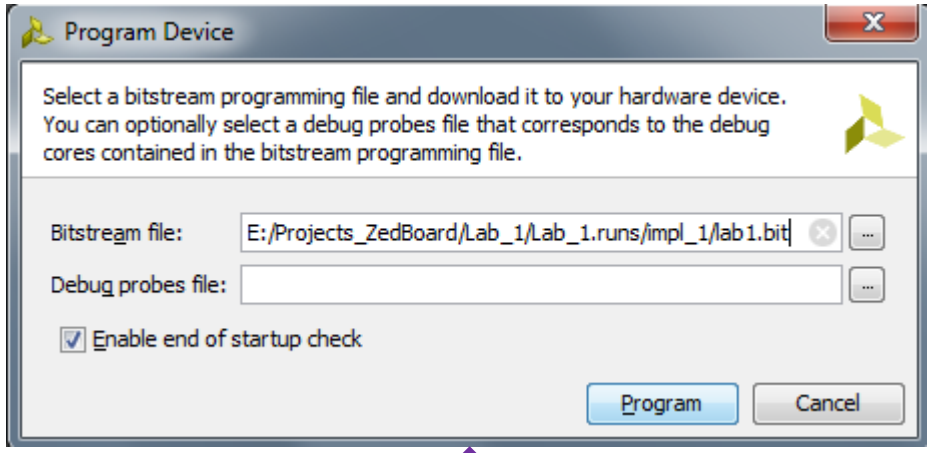
## *Configuring FPGA Process*

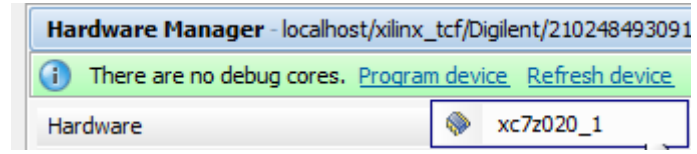
# Steps to Configure *only* the PL

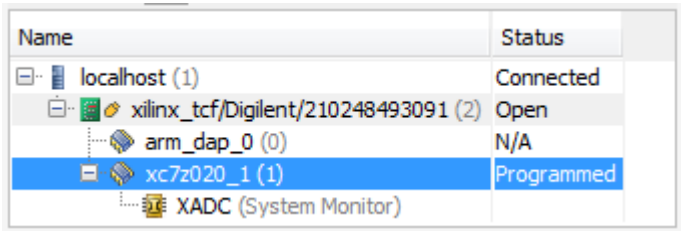



Blue "Done" LED

# Clocking Resources: MMCM and PLL

Up to 24 CMTs per device

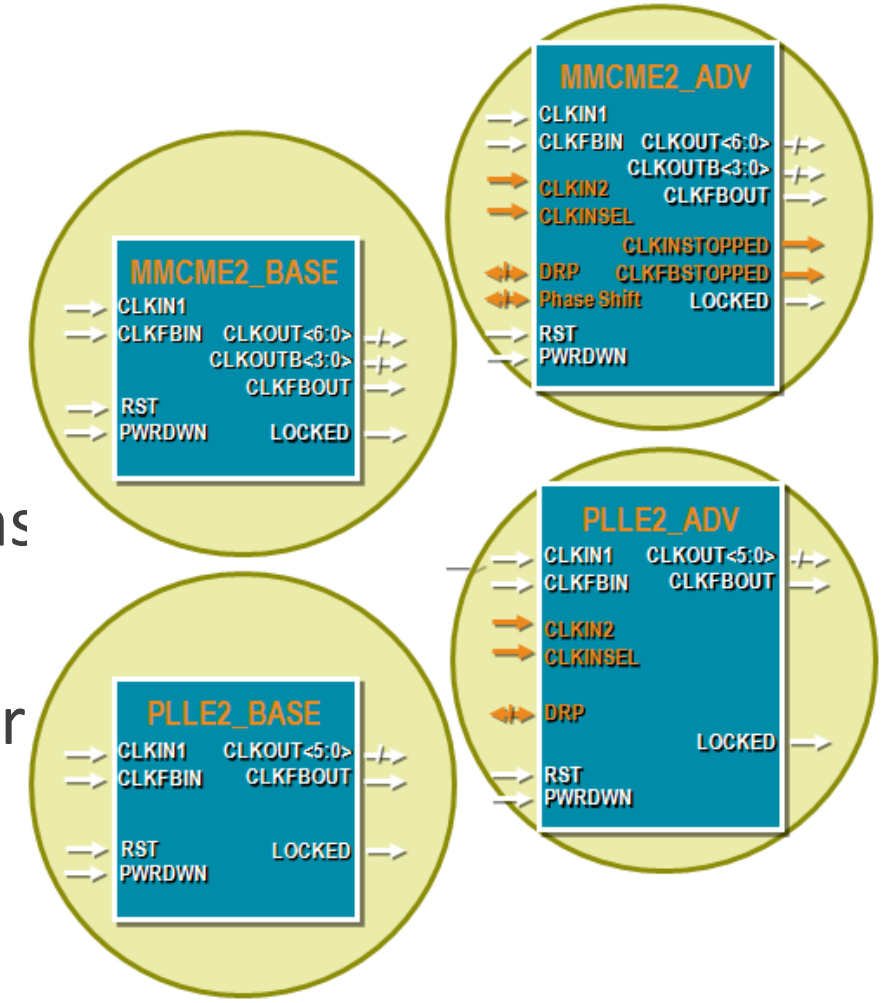
One MMCM and one PLL per CMT

Two software primitives (instantiation)

- *_BASE has only the basic ports
- *_ADV provides access to all ports

PLL is primarily intended for use with the I/O phas
for high-speed memory controllers

The MMCM is the primary clock resource for user



# Inference

Clock networks are represented by nets in your RTL design

- The mapping of an RTL net to a clock network is managed by using the appropriate clock buffer to generate that net

Certain resources can be inferred

- A primary input net (with or without an IBUF instantiated) will be mapped to a global clock if it drives the clock inputs of clocked resources
  - The BUFG will be inferred
- BUFH drivers will be inferred whenever a global clock (driven by a BUFG) is required in a clock region
  - BUFHs for each region required will be inferred

BUFIO, BUFR, and BUFMR cannot be inferred

- Instantiating these buffers tells the tools that you want to use the corresponding clock networks

PLLs and MMCMs cannot be inferred

# Instantiation

All clocking resources can be directly instantiated in your RTL code

- Simulation models exist for all resources
- Refer to the Library Guide for HDL Designs
- Use the Language Templates (  ) tab

PLLs and MMCMs have many inputs and outputs, as well as many attributes

- Optimal dividers for obtaining the desired characteristics may be hard to derive
- The Clocking Wizard via the IP Catalog
  - Only *_ADV available

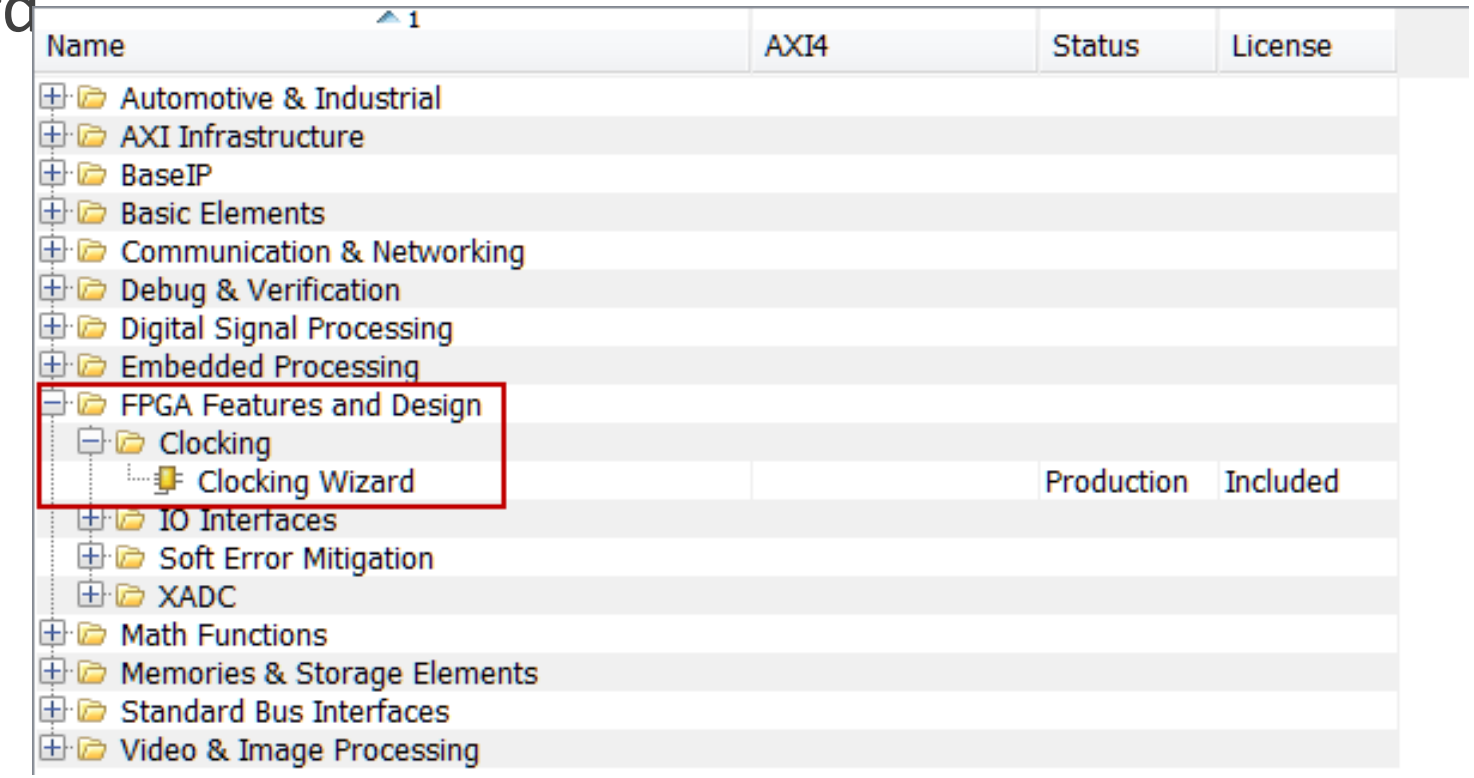
# Invoking Clocking Wizard

Click on the IP Catalog

Expand FPGA Features and Design > Clocking

Double-click on Clocking Wizard

The Clocking Wizard walks you through the generation of complete clocking subsystems



# The Clocking Wizard: Clocking Options

Select Primitives to be used

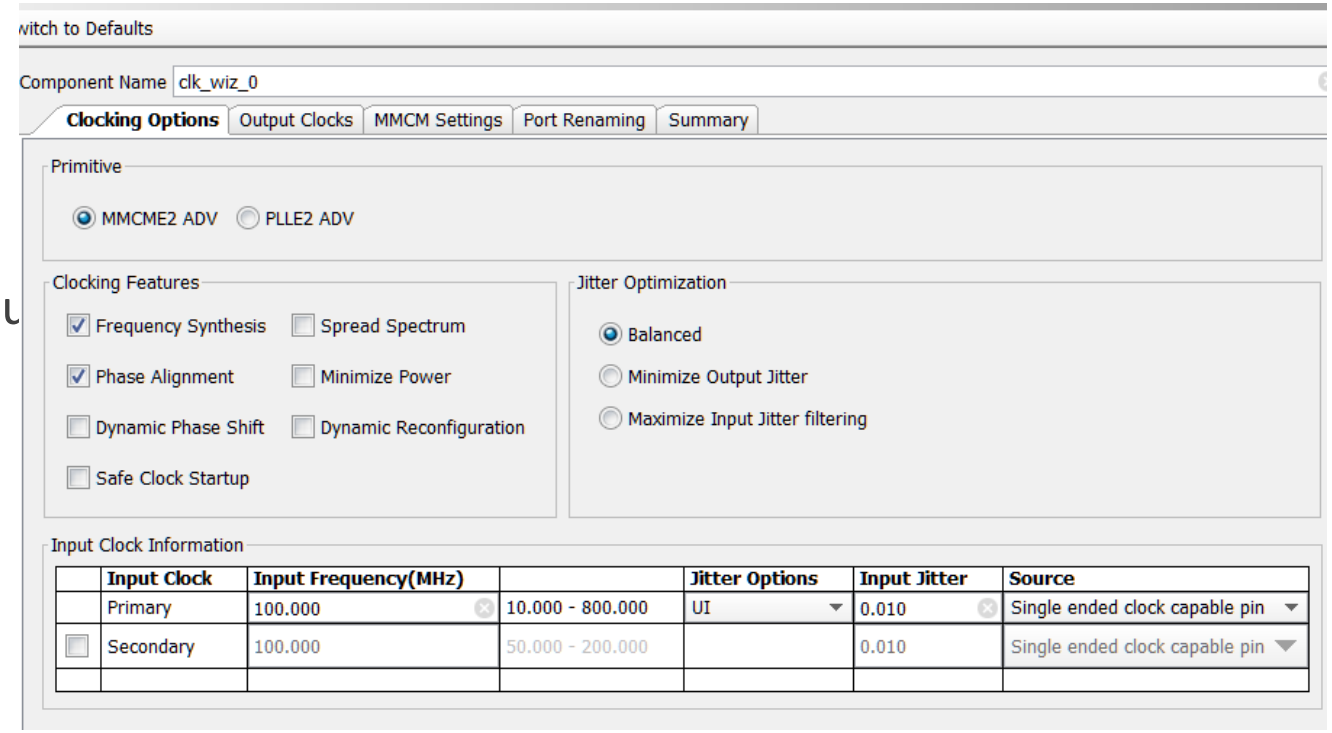
- MMCME2_ADV
- PLLE2_ADV

Specify the primary input frequency and source type

- Optionally, select and specify secondary inpu

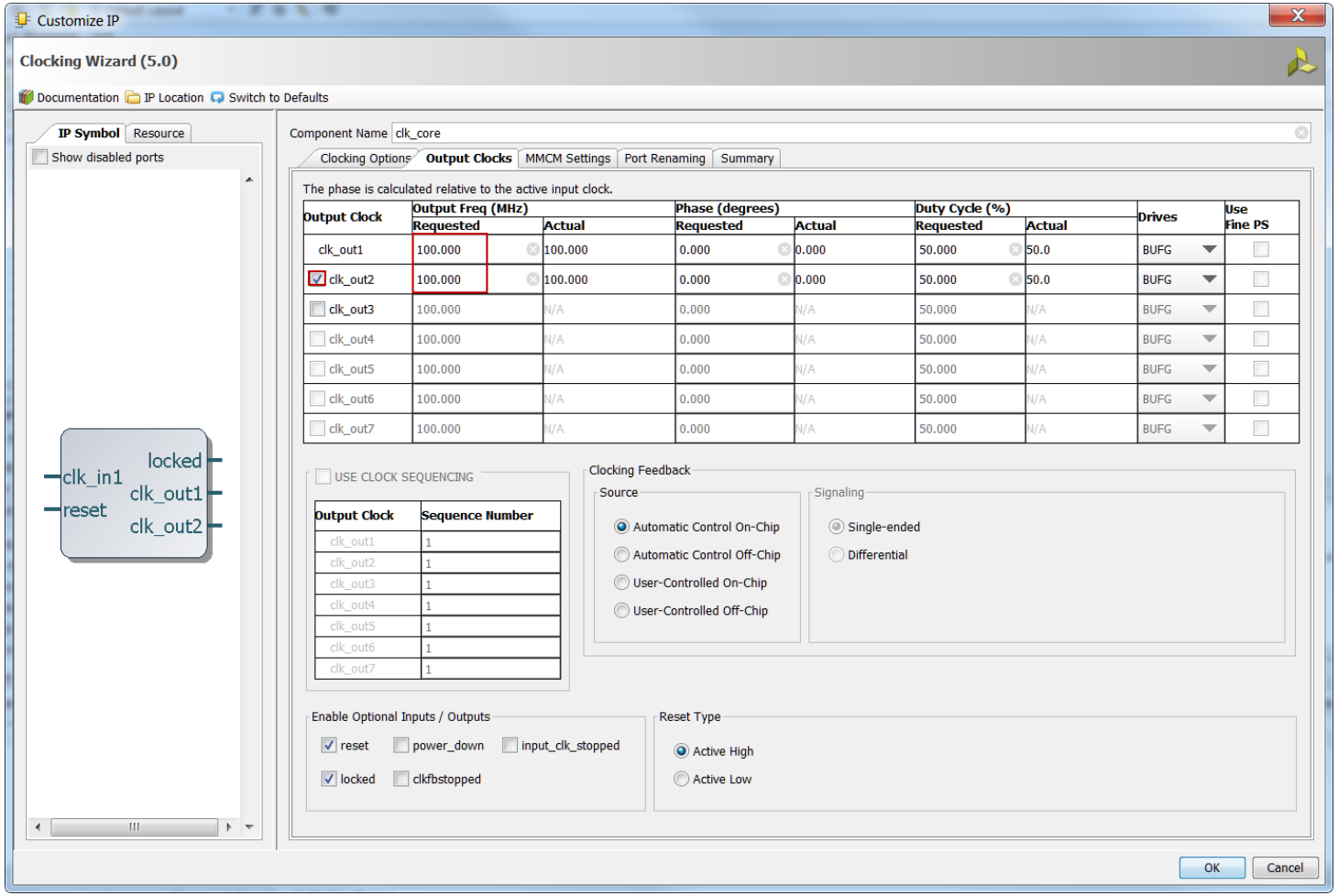
Select clocking features

- Frequency synthesis
- Phase alignment
- Dynamic phase shift
- …



# The Clocking Wizard: Output Clocks

- Select the desired number of output clocks
- Set the desired output frequencies
- Select optional ports



# The Clocking Wizard: Port Renaming

Change input/output port names

Change optional port names

Clocking Options | Output Clocks | MMCM Settings | **Port Renaming** | Summary

Input Clock

| Input Clock | Port Name | Freq (MHz) | Input Jitter (UI) |
|---|---|---|---|
| Primary | clk_in1 | 100.000 | 0.010 |

Output Clock

VCO Freq = 1000.000 MHz

| Output Clock | Port Name | Output Freq (MHz) | Phase (degrees) | Duty Cycle (%) |
|---|---|---|---|---|
| clk_out1 | clk_out1 | 100.000 | 0.000 | 50.0 |
| clk_out2 | clk_out2 | 100.000 | 0.000 | 50.0 |

Optional Port Names

| Other Pins | Port Name |
|---|---|
| reset | reset |
| locked | locked |

# The Clocking Wizard: Summary

Shows the input, output frequencies

Other attributes depending on the selections made

Clocking Options | Output Clocks | MMCM Settings | Port Renaming | **Summary**

| Attribute | Value |
|---|---|
| Input Clock (MHz) | 100.000 |
| Phase Shift | None |
| Divide Counter | 1 |
| Mult Counter | 10.000 |
| CLKOUT0 Divider | 10.000 |
| CLKOUT1 Divider | 10 |
| CLKOUT2 Divider | OFF |
| CLKOUT3 Divider | OFF |
| CLKOUT4 Divider | OFF |
| CLKOUT5 Divider | OFF |
| CLKOUT6 Divider | OFF |

The Resource tab on the left provides summary of type and number of resources used

IP Symbol | **Resource**

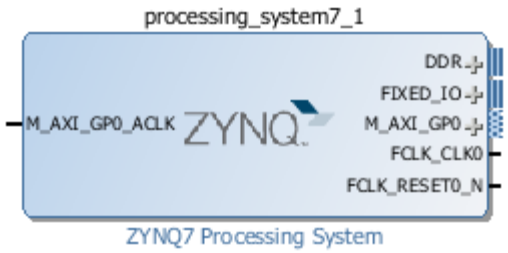
1 MMCME2

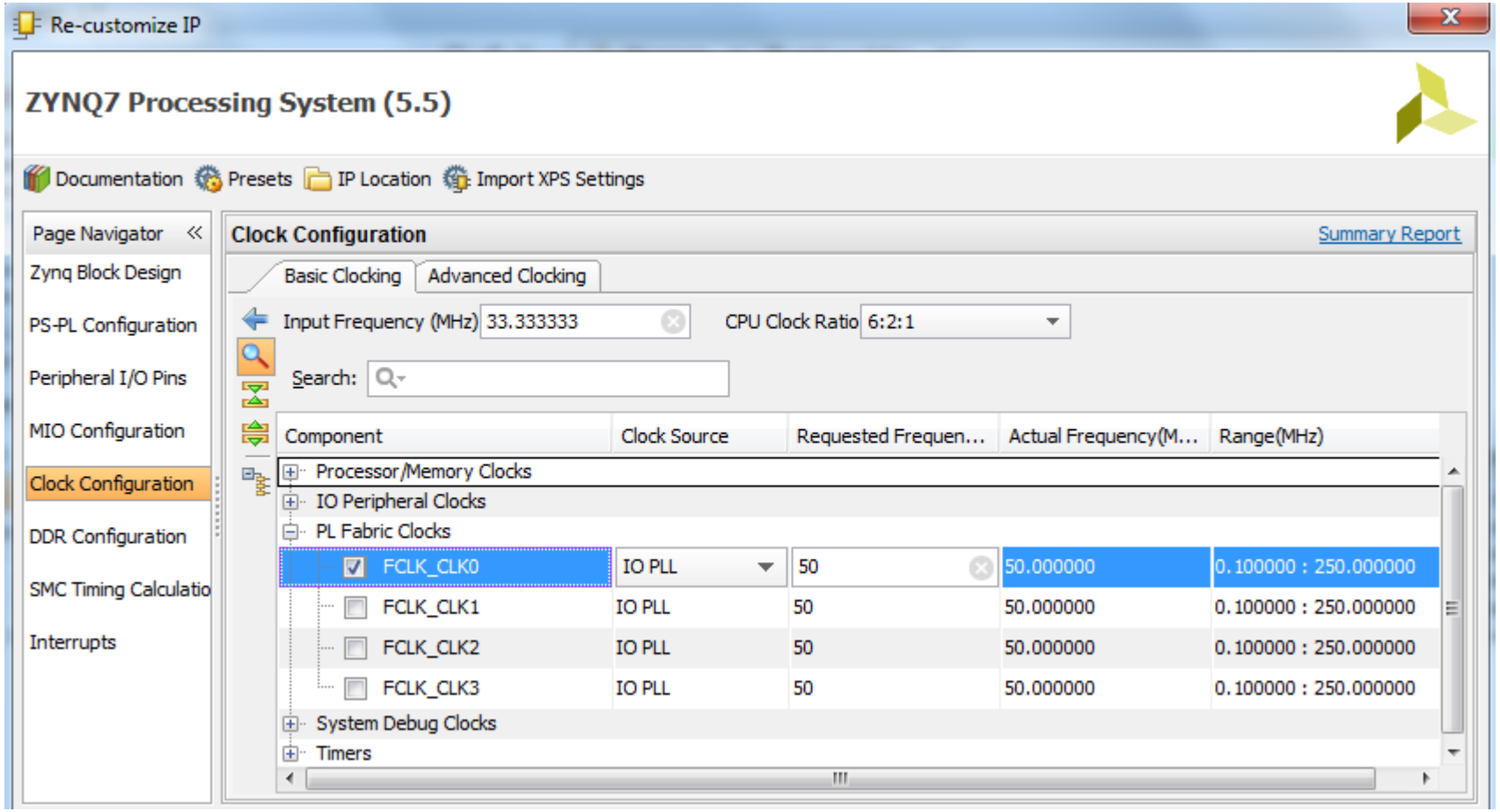
1 IBUFG

3 BUFG

# Reset and Clock Topology

# Enabling Clock for PL



Re-customize IP

## ZYNQ7 Processing System (5.5)

Documentation | Presets | IP Location | Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

**PS-PL Configuration** — Summary Report

Search:

| Name | Select | Description |
|---|---|---|
| General | | |
| UART0 Baud Rate | 115200 | Baud rate is generated with internally fixed UART Ref Clocl |
| UART1 Baud Rate | 115200 | Baud rate is generated with internally fixed UART Ref Clocl |
| PL AXI idle Port | ☐ | Enables idle AXI signal to the PS used to indicate that there |
| DDR ARB bypass Port | ☐ | Enables DDR urgent/arb signal used to signal a critical mem |
| PS-PL Debug interface | ☐ | Enables PL debug signals to PS and vice-versa |
| FTM Trace data interface | ☐ | Enables FTM Trace AXI stream interface used to capture d |
| FTM Trace buffer | 0 | Generates a FIFO to hold trace data |
| FTM Data edge detector | 0 | Stores trace data in the FIFO when the data changes as m |
| FTM Trace buffer FIFO size | 128 | FTM Trace buffer FIFO size |
| FTM Trace buffer clock delay | 12 | Number of clock cycles interval for a trace data output fror |
| Include ACP transaction checker | ☐ | Enables ACP transaction checker. |
| Trace data/control signal pipeline width | 8 | Enables configurable number of pipeline stages on the TRA |
| Power-on reset(POR) 4k timer | ☐ | Enables power-on reset(POR) 4k timer. By default, 64k tim |
| Processor event interface | ☐ | Enables event bus which provides a low-latency and direct |
| Address Editor | | |
| Enable Clock Triggers | | |
| Enable Clock Resets | | |
| FCLK_RESET0_N | ☑ | Enables general purpose reset signal 0 for PL logic |
| FCLK_RESET1_N | ☐ | Enables general purpose reset signal 1 for PL logic |
| FCLK_RESET2_N | ☐ | Enables general purpose reset signal 2 for PL logic |
| FCLK_RESET3_N | ☐ | Enables general purpose reset signal 3 for PL logic |

OK | Cancel

# SDK Compilers

# GNU Tools: GCC

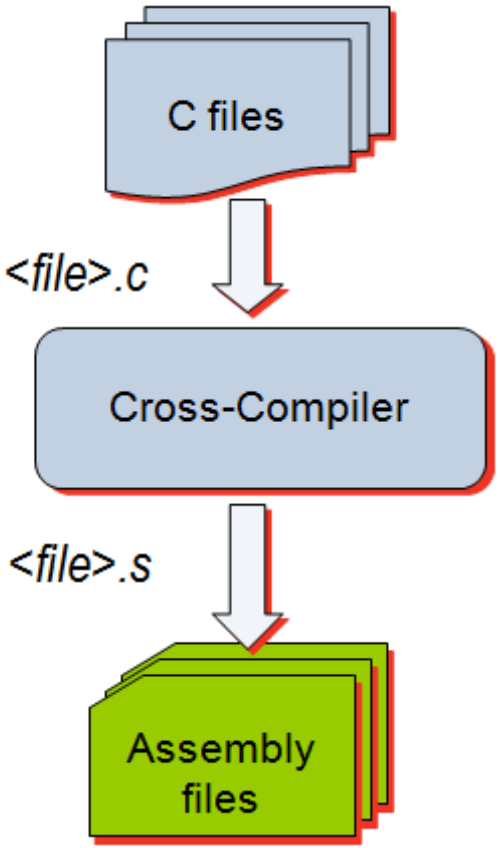
GCC translates C source code into assembly language

GCC also functions as the user interface, passing options to GNU assembler and to the GNU linker, calling the assembler and the linker with the appropriate parameters

Supported cross-compilers

ARM processor compiler

- GNU GCC (arm-xilinx-eabi-gcc)
- GNU Linux GCC (arm-xilinx-linux-eabi-gcc)



# GNU Tools: AS

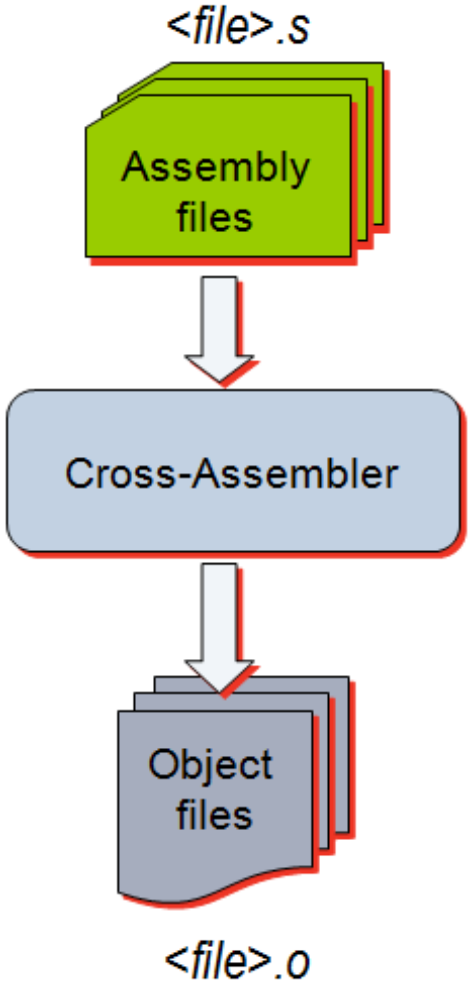
Input: assembly language files
- File extension: .s

Output: object code
- File extension: .o

Contains
- Assembled piece of code
- Constant data
- External references
- Debugging information

Typically, the compiler automatically calls the assembler

Use the -Wa switch if the source files are assembly only and use gcc



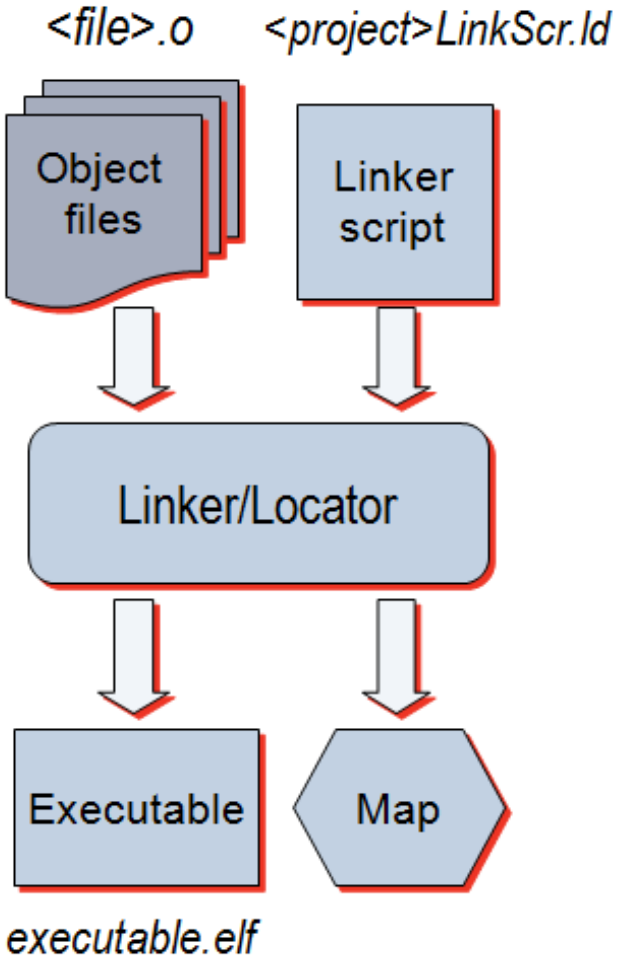
# GNU Tools: Linker (LD)

Inputs

- Several object files
- Archived object files (library)
- Linker script (**.ld*)

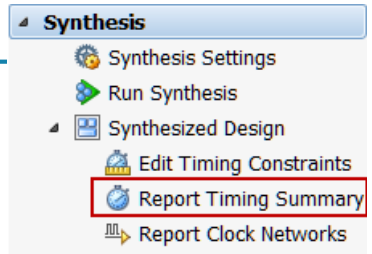
Outputs

- Executable image (ELF)
- Map file



# Timing Reports

# Report Timing Summary




## Tcl command: report_timing_summary

```
report_timing_summary -delay_type max -report_unconstrained -check_timing_verbose -max_paths 10 -input_pins -name timing_1
```

## Vivado IDE

## Options tab

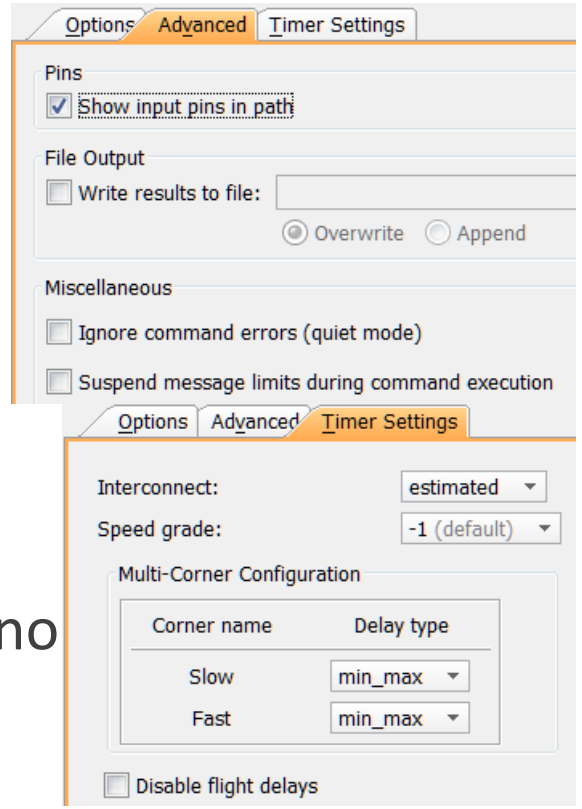
- Maximum number of paths

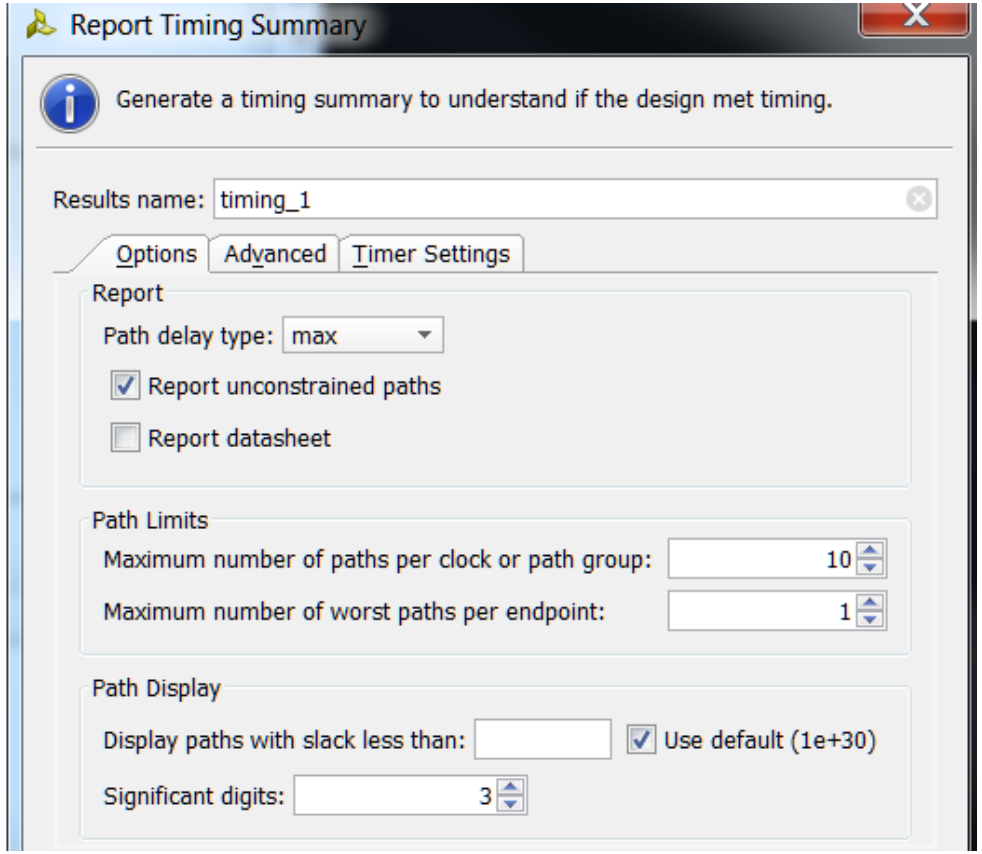
## Advanced tab

- Write to a file

## Timer Settings

- Interconnect delay can be igno
- Flight delays can be disabled



# Report Timing Summary

## Design Timing Summary

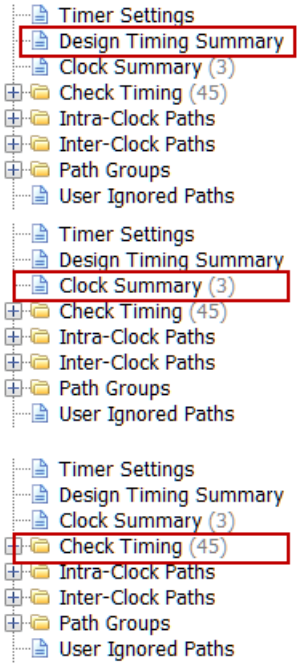
- WNS, TNS, total number of endpoints are of interest

## Clock Summary

- Primary and derived clocks

## Check Timing

- Number of unconstrained internal endpoints




Design Timing Summary

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 1.826 ns | Worst Hold Slack (WHS): | NA | Worst Pulse Width Slack (WPWS): | 3.000 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | NA | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | NA | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 102 | Total Number of Endpoints: | NA | Total Number of Endpoints: | 45 |

Clock Summary

| Name | Waveform | Period (ns) | Frequency (MHz) |
|---|---|---|---|
| clkin | {0.000 5.000} | 10.000 | 100.000 |
| clk_out1_clk_5MHz | {0.000 100.000} | 200.000 | 5.000 |
| clkfbout_clk_5MHz | {0.000 25.000} | 50.000 | 20.000 |

Check Timing

| Timing Check | Count |
|---|---|
| unconstrained_internal_endpoints | 26 |
| no_clock | 10 |
| no_output_delay | 9 |
| no_input_delay | 0 |
| multiple_clock | 0 |
| generated_clocks | 0 |
| loops | 0 |
| partial_input_delay | 0 |
| partial_output_delay | 0 |