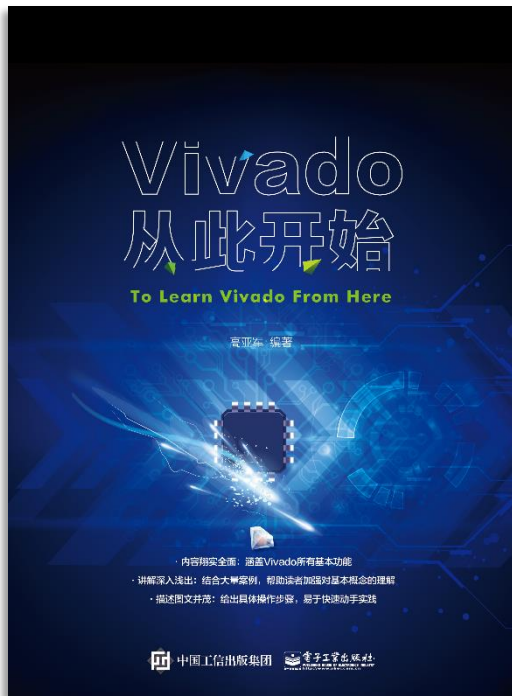


Vivado从此开始 (To Learn Vivado From Here)



本书围绕Vivado四大主题

- 设计流程
- 时序约束
- 时序分析
- Tcl脚本的使用



作者：高亚军（Xilinx战略应用高级工程师）

- 2012年2月，出版《基于FPGA的数字信号处理（第1版）》
- 2012年9月，发布网络视频课程《Vivado入门与提高》
- 2015年7月，出版《基于FPGA的数字信号处理（第2版）》
- 2016年7月，发布网络视频课程《跟Xilinx SAE学HLS》

- ◆ 内容翔实全面：涵盖Vivado所有基本功能
- ◆ 讲解深入浅出：结合大量案例，帮助读者加强对基本概念的理解
- ◆ 描述图文并茂：给出具体操作步骤，易于快速动手实践



XILINX

ALL PROGRAMMABLE™

TCL, Vivado One World

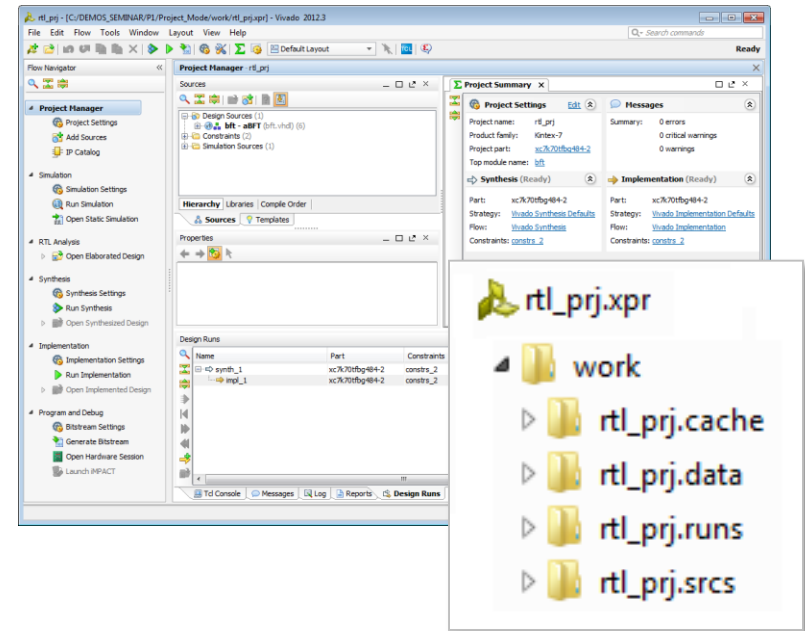
Part 6: Design Flow Management with Tcl in Project Mode

Lauren Gao

Project Mode

➤ Key advantage – many tasks are automated

- Supports push-button flow
- Automatic management
 - Project status
 - HDL sources, constraints, IPs
 - Dependency management
 - Stores implementation results, reports
 - Saves checkpoints (after each step)
- Multiple runs support
 - Multiple strategies

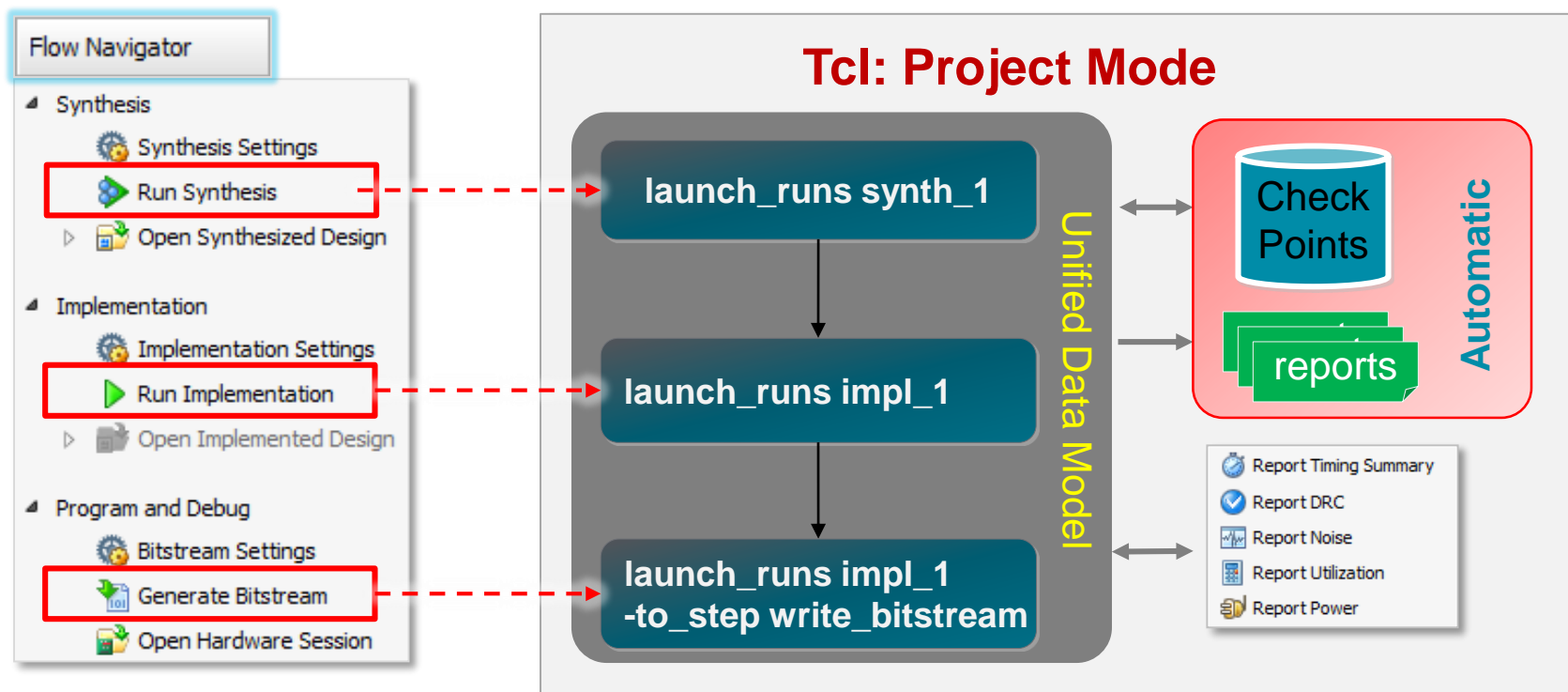


➤ Controlled via GUI and/or Tcl

➤ New Vivado Design Suite users: Good starting point

Project Mode: GUI → Tcl

➤ Key flow steps



Preparing Design Files for Vivado Project

➤ Four types of design files

- RTL design files
 - VHDL (.vhd), Verilog (.v), System Verilog (.sv)
- Testbench files for simulation
 - VHDL (.vhd), Verilog (.v), System Verilog (.sv)
- XDC files for design constraints
 - .xdc, support both project and module design constraints
- IP files
 - .xci, IPs have been generated by Vivado Manage IP

➤ It's better to create different file folder to store the design files accordingly

src

sim

xdc

IP

Create Project

➤ Tcl command: create_project

- Three arguments must be specified
 - Part, Project name, work directory
- Three file sets **sources_1**, **sim_1** and **constrs_1** are created in the same time
- Two design runs **synth_1** and **impl_1** are also built.

➤ Other project properties

- Target language: VHDL, Verilog
 - `set_property target_language VHDL [current_project]`
- Source management mode: All①, DisplayOnly②, None③
 - `set_property source_mgmt_mode DisplayOnly [current_project]`

①	Automatic Update and Compile Order
②	<input checked="" type="checkbox"/> Automatic Update, Manual Compile Order
③	No Update, Manual Compile Order

```
create_project waveprj G:/Vivado/wavegen/waveprj \  
-part xc7k325tffg900-2 😊  
create_project -name waveprj -dir G:/Vivado/wavegen/waveprj \  
-part xc7k325tffg900-2
```

Add Design Files to Current Project

➤ Add design source files

- `add_files -fileset sources_1 ./src`
- `update_compile_order -fileset sources_1`

➤ Add simulation source files

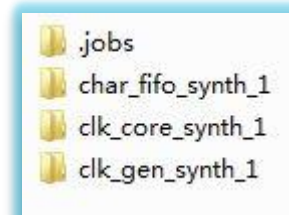
- `add_files -fileset sim_1 ./sim`
- `update_compile_order -fileset sim_1`

➤ Add constraints files

- `add_files -fileset constrs_1 ./xdc`

➤ Add existing IPs

- `add_files [glob ./ip/*/*.xci]`
- `update_compile_order -fileset sources_1`



Synthesis Settings

➤ Define expected properties for synthesis

- Set the value of FLATTEN_HIERARCHY
- Set the module as OOC

➤ Create new synthesis run

```
set_property STEPS.SYNTH_DESIGN.ARGS.FLATTEN_HIERARCHY rebuilt \  
[get_runs synth_1]
```

```
create_fileset -blockset -define_from dac_spi dac_spi
```

```
create_run synth_2 -flow {Vivado Synthesis 2014} \  
-strategy {Vivado Synthesis Defaults}
```


Launch Synthesis and Check Synthesis Result

- Launch synthesis
- Check synthesis result

```
launch_runs synth_1  
wait_on_run synth_1
```

```
open_run synth_1  
get_timing_paths -slack_lesser_than 0 -quiet
```

Setting Configuration Memory Property

- Bit file properties must be set after synthesis
 - For example: CONFIGRATE
- Configuration memory such as SPI flash must be set after synthesis and before implementation
 - Take SPI flash as an example
 - SPI_BUSWIDTH
 - Config_mode

```
set_property BITSTREAM.CONFIG.CONFIGRATE 66 [get_designs synth_2]  
set_property CONFIG_VOLTAGE 1.8 [get_designs synth_2]  
set_property CFGBVS GND [get_designs synth_2]
```

```
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [get_designs synth_2]  
set_property config_mode SPIx4 [current_design]
```

Launch Implementation

➤ Define expected properties for implementation

- Select the desired strategy
- Enable or Disable intermediate flow

➤ Launch implementation

```
set_property strategy Performance_Explore [get_runs impl_1]
```

```
set_property STEPS.PHYS_OPT_DESIGN.IS_ENABLED true [get_runs impl_4]  
set_property STEPS.OPT_DESIGN.ARGS.DIRECTIVE AddRemap \  
[get_runs impl_4]
```

```
launch_runs impl_1  
wait_on_run impl_1
```

Generate Bit Files and Memory Configuration File

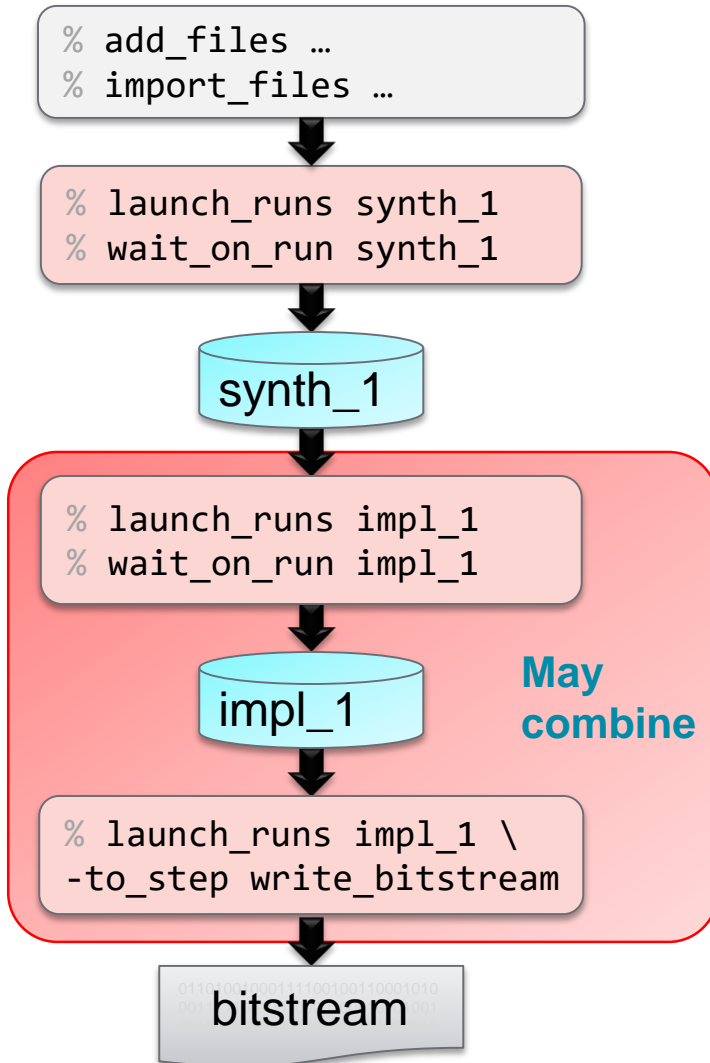
- Generate bit file
- Combine implementation with bit stream generation
- Generate memory configuration file

```
launch_runs impl_1 -to_step write_bitstream  
wait_on_run impl_1
```

```
set_property STEPS.WRITE_BITSTREAM.ARGS.BIN_FILE true \  
[get_runs impl_1]  
launch_runs impl_1 -to_step write_bitstream  
wait_on_run impl_1
```

```
write_cfgmem -force -format MCS -interface SPIx4 -loadbit \  
"up 0x0 G:/Vivado/wavegen/waveprj/waveprj.runs/impl_2/wave_gen.bit" \  
G:/Vivado/wavegen/waveprj/waveprj.runs/impl_2/wave_gen_impl_2
```

Scripted Project Flow: Overview



*Add sources: automatically managed
(VHDL, Verilog, IP, XDC, ...)*

Launch synthesis run and wait

synth_1:

- Post-synthesis design = DCP
- Reports

Launch implementation run and wait

impl_1:

- Implemented design = post-routed DCP
- Reports

Generate bitstream

**Recall: Design snapshots and reports
automatically generated**

Project Mode


Custom Design Analysis

➤ Project mode – direct access to design database


- Post-synthesis
- Post-implementation

➤ Need to “open a design” first

`open_run synth_1`

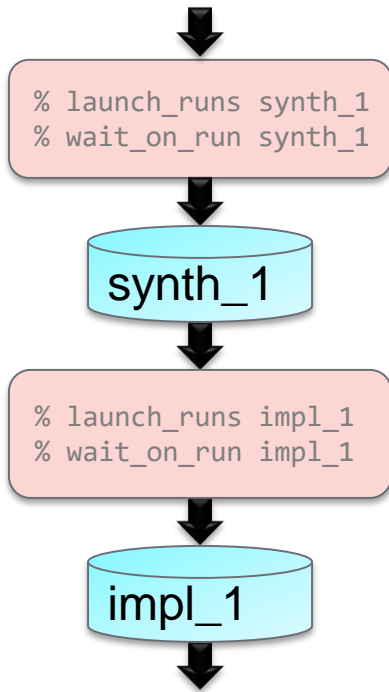
 Open Synthesized Design

`open_run impl_1`

 Open Implemented Design

➤ Available tools

- Reporting: `report_timing, report_utilization, ...`
- Constraining: `create_clock, set_max_delay, ...`
- Netlist exploration: `get_cells, get_nets, get_property, ...`



Project Mode

Custom Design Analysis

➤ Example: `report_timing` using post-implementation design

```
Vivado% report_timing
```

```
ERROR: [Common 17-53] User Exception: No open design. Please open a design before executing this command.
```

```
Vivado% get_runs
```

```
synth_1 impl_1
```

```
Vivado% open_run impl_1
```

```
INFO: [Netlist 29-17] Analyzing 290 Unisim elements for replacement
```

```
. . .
```

```
open_run: Time (s): cpu = 00:00:29 ; elapsed = 00:00:29 . Memory (MB): peak = 4957.078 ; gain = 40.016
```

```
Vivado% report_timing
```

```
INFO: [Timing 38-91] UpdateTimingParams: Speed grade: -2, Delay Type: max, Constraints type: SDC.
```

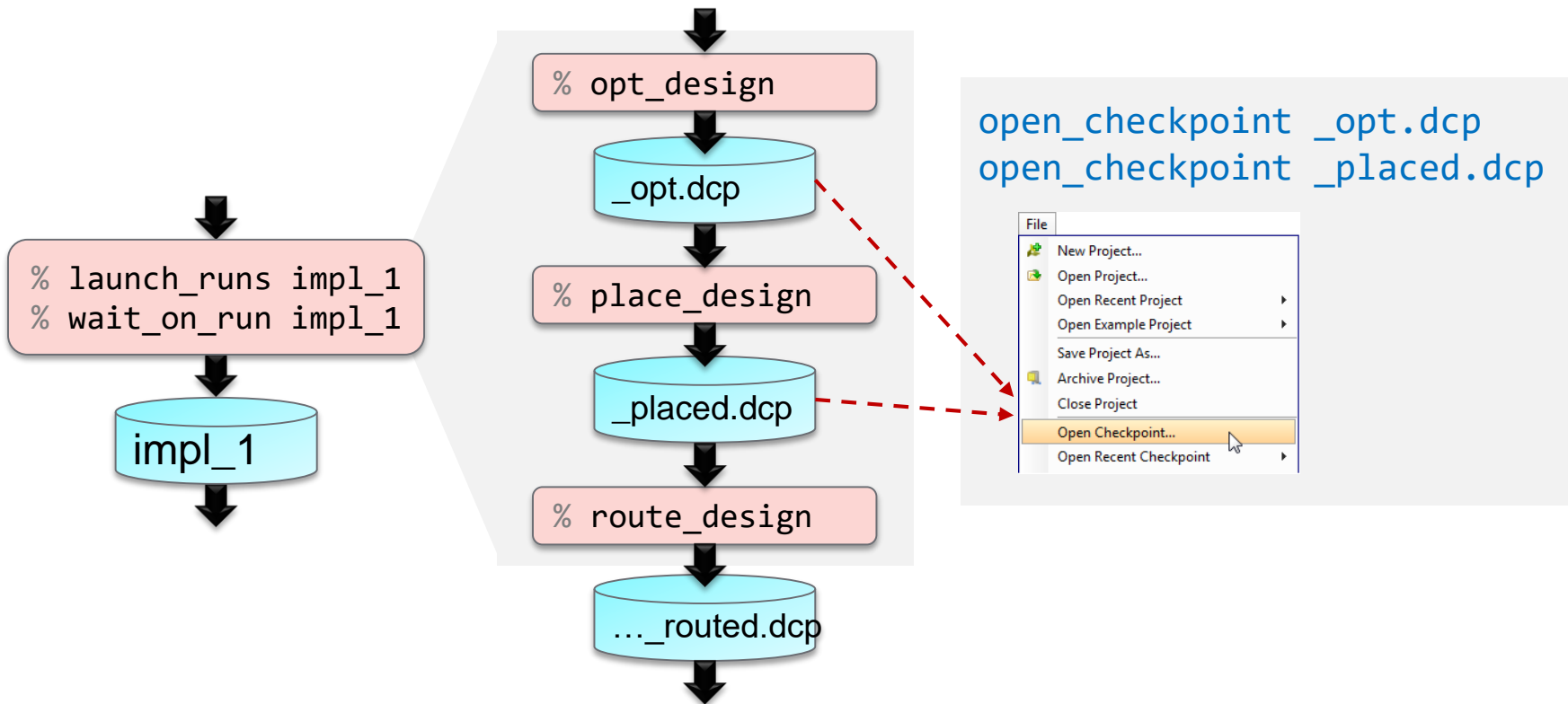
```
. . .
```

Project Mode

Access to Intermediate Implementation Results

► `lunch_run impl_1` – a set of fine-granularity commands

- In project mode: DCP stored after each command
- Open intermediate results = open checkpoint



Summary

➤ Pros

- High degree of automation
 - Flow: a single Tcl command – multiple flow steps
 - Source and result management
 - Multiple strategies: easy run

➤ Cons

- Flow
 - Reduced flexibility to work with intermediate implementation results
 - Not all features are easily accessible
(ex: re-entrant routing)
- File structure: fixed file organization
 - Revision control – many project files

Demo