# 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


# Vivado Implementation

Lauren Gao

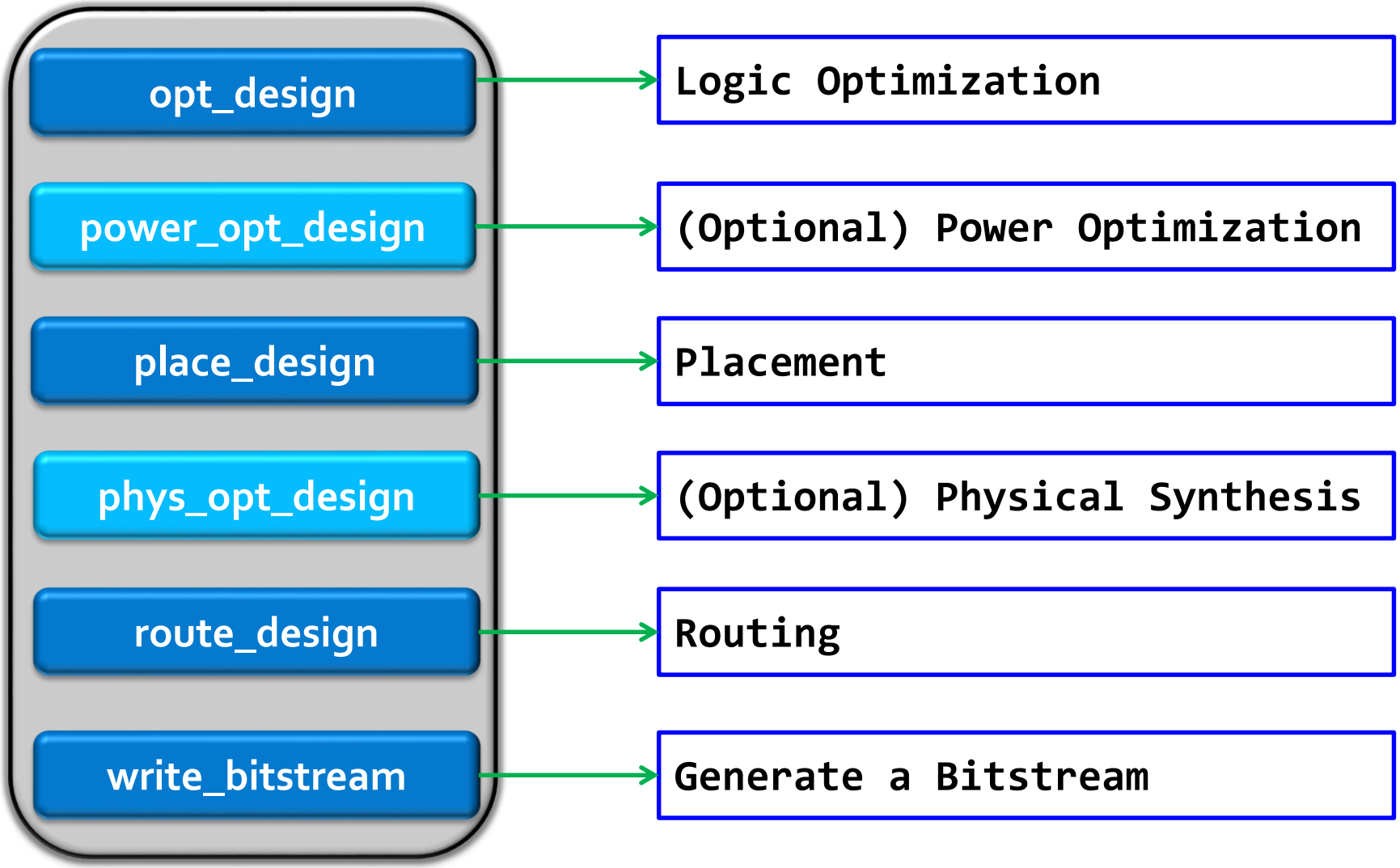
# Agenda

- New Feature: Directive
- Implementation Strategy
- Run Implementation in Project-Mode and Non-Project Mode

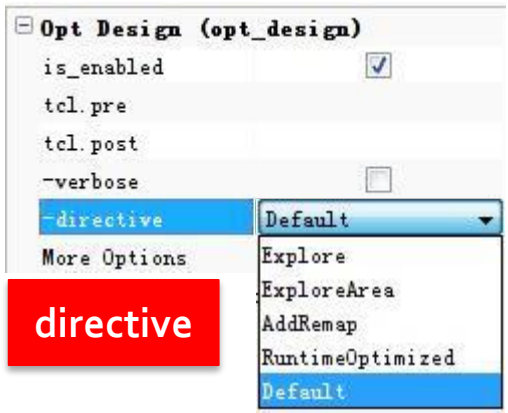
# Agenda

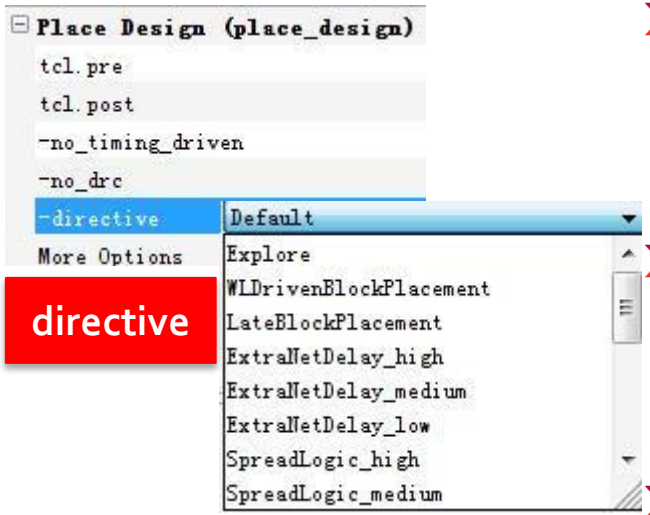
- New Feature: Directive
- Implementation Strategy
- Run Implementation in Project-Mode and Non-Project Mode

# Vivado Implementation Sub-processes



# New Feature: Directive




- Command option to "direct" behavior towrd alternate goal
  - Different algorithms
  - Different objectives
- Implementation commands with –directive
  - opt_design, place_design, phys_opt_design, route_design
- Alternative flows to use
  - When the default flow does not meet design goals
  - When tool runtime increase can be accommodated
- Uses different algorithms
  - Not random seeds like ISE cost tables
  - More consistent and predictable behavior
- Help -opt_design to find all the directives

# -directive Explore

- All commands have –directive Explore

**ISE MAP**


**ISE P&R**


| ISE –effort_level | | Vivado -directive |
|---|---|---|
| -effort_level high | ⇨ | -directive Explore |
| -effort_level high -area_mode | ⇨ | -directive ExploreArea (opt_design only) |

# -directive: Replacement for –effort_level

- Automatic mapping for all implementation commands

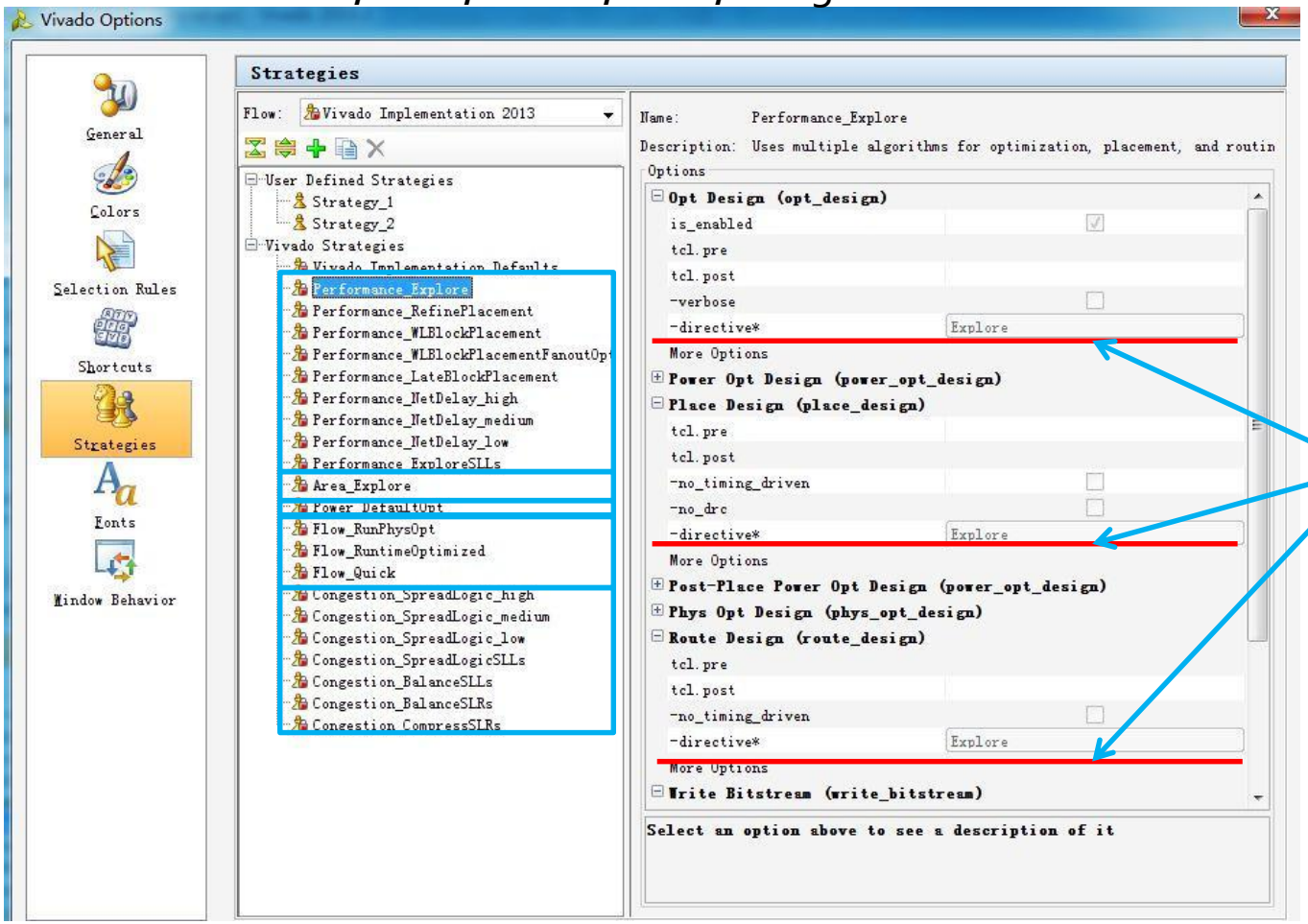
| ISE –effort_level | | Vivado -directive |
|---|---|---|
| -effort_level high | ⇨ | -directive Explore |
| -effort_level medium | ⇨ | -directive Default |
| -effort_level low | ⇨ | -directive RumtimeOptimized |
| -effort_level quick | ⇨ | -directive Quick |

# Agenda

- New Feature: Directive
- **Implementation Strategy**
- Run Implementation in Project-Mode and Non-Project Mode

# Implementation Strategy

- Tools → options → Strategies
- Strategies are grouped by category
  - Performance, Area, Power, Flow, Congestion



# Strategies Example

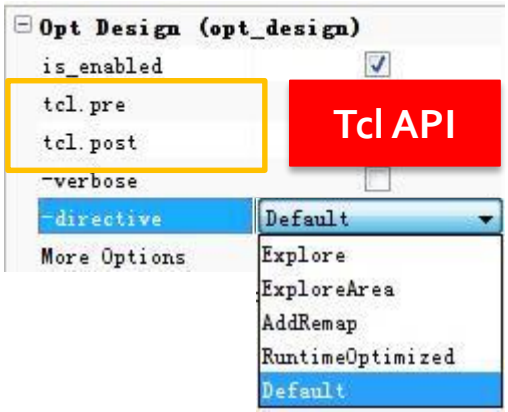
| Design flow | Performance_Explore | Area_Explore | Power_DefaultOpt | Flow_RunPhysOpt | Congestion_SpreadLogic_high |
|---|---|---|---|---|---|
| **opt_design**<br>-directive | √<br>Explore | √<br>ExploreArea | √<br>Default | √<br>Default | √<br>Default |
| **power_opt_design** | × | × | √ | × | × |
| **place_design**<br>-directive | √<br>Explore | √<br>Default | √<br>Default | √<br>Default | √<br>SpreadLogic_high |
| **power_opt_design (post-place)** | × | × | × | × | × |
| **phys_opt_design**<br>-directive | √<br>Explore | × | √<br>Default | √<br>Explore | √<br>AggressiveFanoutOpt |
| **route_design**<br>-directive | √<br>Explore | √<br>Default | √<br>Default | √<br>Default | MoreGlobalIterations |

- Directives -> command level behavior
- Strategies -> implementation run-level behavior, a combination of directives

# Implementation Strategy

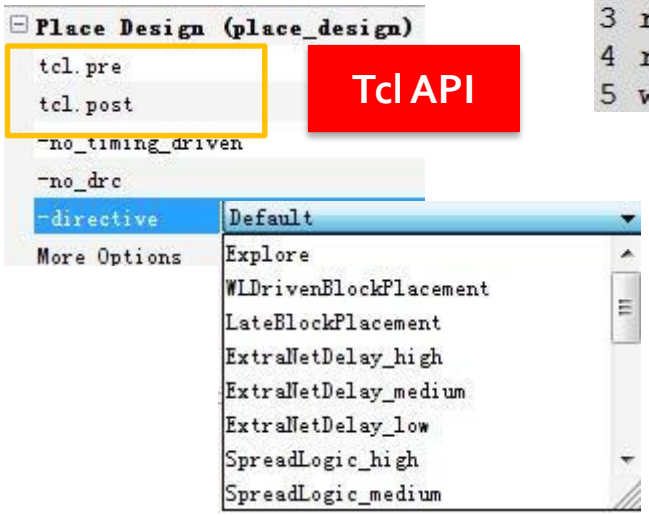
- **Different strategy may have**
  - Different directives
    - Vivado implementation default: -directive are all default
    - Performance_Explore: -directive are all Explore
  - Different design flows
    - Vivado implementation default:  opt_design, place_design , route_design
    - Performance_Explore: opt_design, place_design, phys_opt_design, route_design
- **Each strategy has different directives for the impl command steps**
- **You can customize your own strategy**
  - User defined strategies
- **The Performance_Explore strategy is a good first choice, because it covers all types of designs**
  - The Performance strategies aim to improve design performance at the expense of runtime
- **Strategies containing the terms SLL or SLR are for use with SSI devices only**

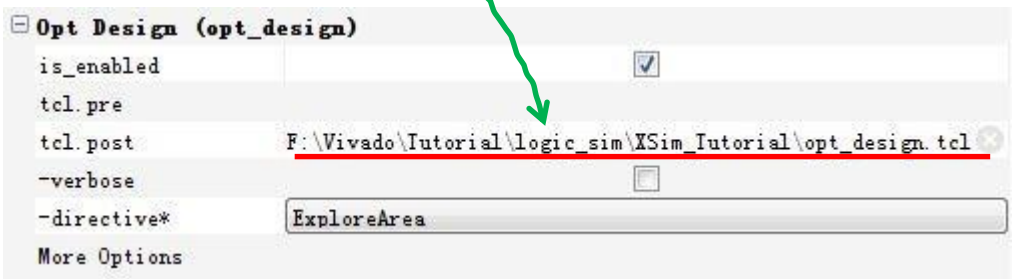
# Tcl API




- All implementation sub-processes have Tcl API
  - tcl.pre : commands should be executed before this process
  - tcl.post: commands should be executed after this process
- Example
  - opt_design.tcl

```
1 set mydir [get_property DIRECTORY [current_project]]
2 set prjname [get_property NAME [current_project]]
3 report_utilization -file $mydir/opt_design_util.txt
4 report_timing_summary -file $mydir/opt_design_timing.txt
5 write_checkpoint $mydir/${prjname}_opt_design -force
```



# Agenda

- New Feature: Directive
- Implementation Strategy
- Run Implementation in Project-Mode and Non-Project Mode

# Run Implementation in Project Mode with GUI

- **In Project Mode, the Vivado IDE allows you to**
  - Define multiple implementation runs
  - Run multiple strategies on a single design
  - Customize implementation strategies to meet specific design requirements
  - Save customized implementation strategies to use in other designs
- **Non-Project Mode does not support predefined implementation runs and strategies**



# Run Implementation in Project Mode with Tcl

- launch_runs [-jobs arg] [-scripts_only] [-all_placement] [-dir arg]
  [-to_step arg] [-next_step] [-host args] [-remote_cmd arg]
  [-email_to args] [-email_all] [-pre_launch_script arg]
  [-post_launch_script arg] [-force] [-quiet] [-verbose] runs...
- to_step
  - opt_design, power_opt_design, place_design, power_opt_design(post_place)
  - Phys_opt_design, route_design, write_bitstream
    - -to_step place_design
    - -to_step "power_opt_design (Post-Place)"

```
launch_runs impl_1
launch_runs impl_2 -to_step place_design
cd {F:\Vivado\Tutorial\logic_sim\XSim_Tutorial.runs\impl_2}
open_checkpoint sinegen_demo_placed.dcp
launch_runs impl_2 -next_step
```

# Run Implementation in Non-Project Mode

```
# Step 1: Read in top-level EDIF netlist from synthesis tool
read_edif c:/top.edf
read_edif c:/core1.edf
read_edif c:/core2.edf
# Step 2: Specify target device and link the netlists
link_design -part xc7k325tfbg900-1 -top top
# Step 3: Read XDC constraints to specify timing requirements
read_xdc c:/top_timing.xdc
read_xdc c:/top_physical.xdc
# Step 4: Optimize the design with default settings
opt_design
# Step 5: Place the design
place_design
# Step 6: Route the design
route_design
# Step 7: Run Timing Summary Report and Resource Utilization
report_timing_summary -file post_route_timing.rpt
report_utilization -file post_route_utilization.rpt
# Step 8: Write checkpoint to capture the design database;
# The checkpoint can be used for design analysis
# in Vivado IDE or TCL API
write_checkpoint post_route.dcp
```

***Non-project based designs must be manually moved through each step of the implementation process using Tcl commands***

# place_design

- **place_design** [**-directive** *arg*] [**-no_timing_driven**] [**-unplace**]
  [**-cells** *args*] [**-post_place_opt**] [**-quiet**] [**-verbose**]
- **-post_place_opt**
  - Potentially improve critical path timing at the expense of additional runtime
  - This optimization can be run at any stage after placement, and can be particularly effective on a routed design
  - The optimization examines the worst case timing paths and tries to improve placement to reduce delay
  - The optimization is performed on a fully placed design with **timing violations**
- **The -directive option controls the overall placement strategy, and is not compatible with any specific place_design options.**

```
place_design
phys_opt_design
route_design
place_design -post_place_opt
route_design
```

# route_design

- **route_design** [**-unroute**] [**-re_entrant** *arg*] [**-nets** *args*] [**-physical_nets**] [**-pin** *arg*] [**-directive** *arg*] [**-no_timing_driven**] [**-preserve**] [**-delay**] [**-free_resource_mode**] **-max_delay** *arg* **-min_delay** *arg* [**-quiet**] [**-verbose**]
- **-preserve**
  - Preserve existing routing
- **-delay**
  - Use with -nets or -pin option to route in delay driven mode
- **-max_delay**
  - Use with -pin option to specify the max_delay constraint on the pin
- **-min_delay**
  - Use with -pin option to specify the max_delay constraint on the pin

```
route_design -delay -nets $myCriticalNets
route_design -preserve -directive RuntimeOptimized
route_design -unroute
```

# phys_opt_design

- **phys_opt_design** [**-fanout_opt**] [**-placement_opt**] [**-rewire**] [**-critical_cell_opt**] [**-dsp_register_opt**] [**-bram_register_opt**] [**-bram_enable_opt**] [**-shift_register_opt**] [**-hold_fix**] [**-retime**] [**-force_replication_on_nets** *args*] [**-directive** *arg*] [**-critical_pin_opt**] [**-quiet**] [**-verbose**]
- **Command options limit the optimization scope to only the specified options**
  - One option per optimization

# What Optimizations are Performed?

- **Timing-driven optimizations included by default**
  - Fanout: replicate drivers of high fanout nets
  - Placement: move cells
  - Rewiring: restructure logic cones
  - Critical cells: replicate critical path cells
  - DSP registers: move registers to/from DSP
  - BRAM registers: move registers to/from BRAM
  - Shift registers: move registers from SRLs
  - Critical pins: swap physical LUT input pins
  - Very high fanout: replicate drivers of very-high fanout nets
  - BRAM enable: improve power-optimized BRAM enables



XILINX ALL PROGRAMMABLE

# What Optimizations are Performed?

- **Optional optimizations**
  - Forced net replication: replicate regardless of timing slack
  - Register retiming: balance registers across combinational delays
  - Hold-fix: insert data path delay



XILINX ALL PROGRAMMABLE

# Summary

- Different strategy may have different design flows with different directives
- Directive is incompatible with other options in opt_design, place_design, phys_opt_design and route_design
- Directives and strategies provide alternate flows to try when the tool defaults don't meet timing on difficult designs
- Some Tcl command can help to meet timing
  - place_design –post_place_opt
  - phys_opt_design
  - route_design –delay –nets
- Performance_Explore is helpful for timing closure