Vivado从此开始（To Learn Vivado From Here）

Vivado
从此开始
To Learn Vivado From Here
高亚军 编著
· 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
中国工信出版集团
电子工业出版社

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

Understanding Implementation Strategies

Lauren Gao

Implementation

opt_design

power_opt_design

place_design

power_opt_design

phys_opt_design

route_design

phys_opt_design

Logic Optimization (opt_design)

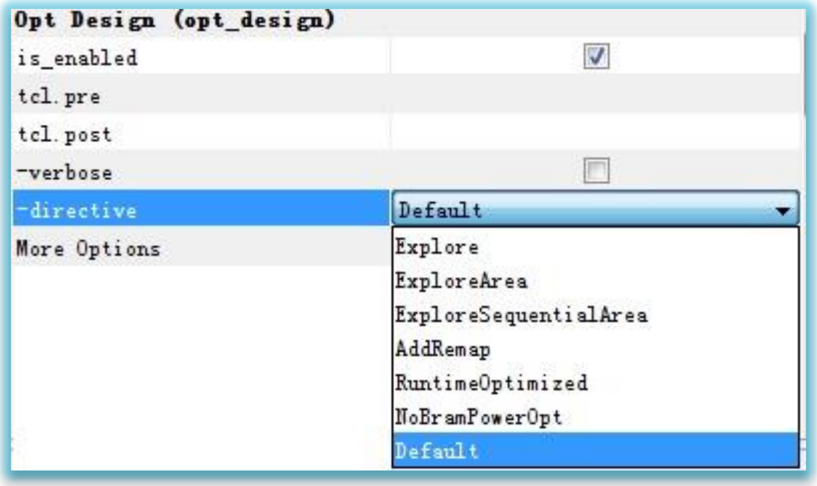
- **Retargeting**
    - Example: A MUXF7 replaced by a LUT3 can be combined with other LUTs
    - Example: Simple cells such as inverters are absorbed into downstream logic
- **Constant Propagation**
    - Eliminated logic
        - Example: an AND with a constant 0 input
    - Reduced logic
        - Example: A 3-input AND with a constant 1 input is reduced to a 2-input AND
    - Redundant logic
        - Example: A 2-input OR with a logic 0 input is reduced to a wire
- **Sweep**
    - Removes cells that have no loads
- **Block RAM Power Optimization**
    - Changing the WRITE_MODE on unread ports of true dual-port RAMs to NO_CHANGE
    - Applying intelligent clock gating to block RAM outputs

Logic Optimization (opt_design)

- **Remap**
  - – Remap combines multiple LUTs into a single LUT to reduce the depth of the logic
- **Resynth Area**
  - – Resynth Area performs re-synthesis in area mode to reduce the number of LUTs.
- **Resynth Sequential Area**
  - – Resynth Sequential Area performs re-synthesis to reduce both combinational and sequential logic

opt_design

opt_design [-retarget] [-propconst] [-sweep] [-bram_power_opt] [-remap] [-resynth_area] [-resynth_seq_area] [-directive <arg>] [-quiet] [-verbose]

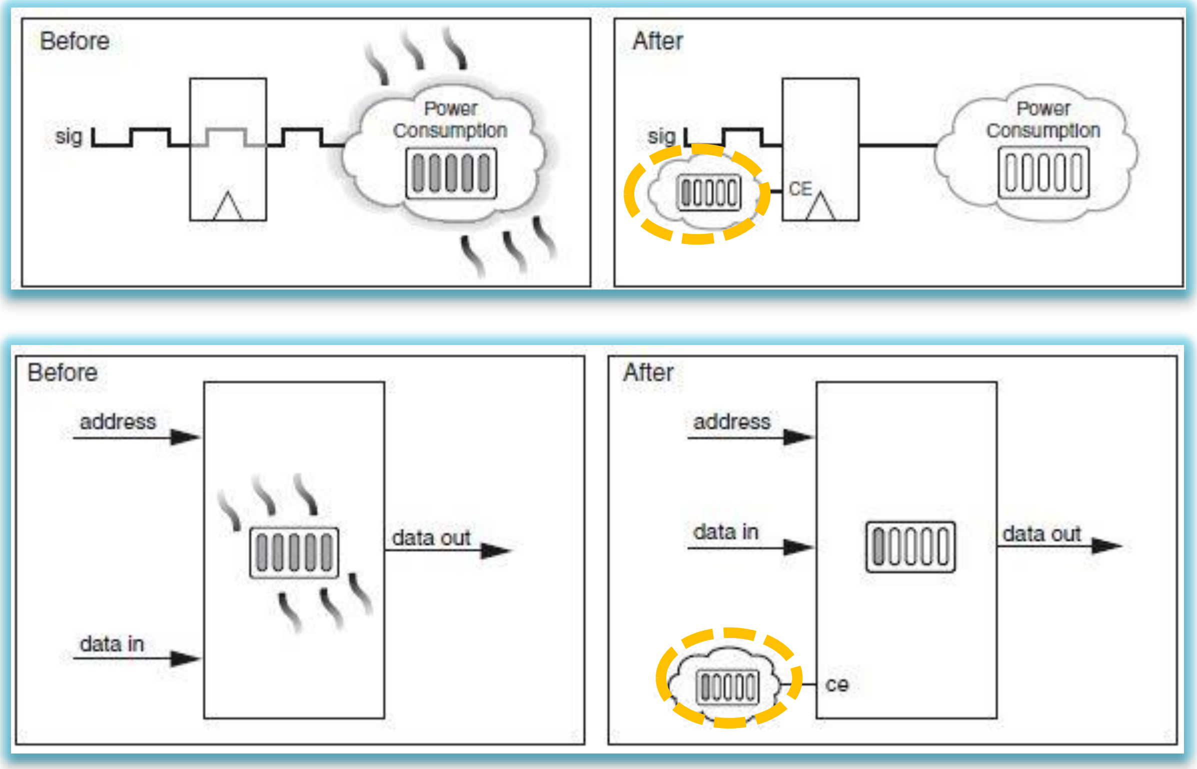


Opt Design (opt_design)	
is_enabled	✓
tcl.pre	
tcl.post	
-verbose	☐
-directive	Default
More Options	

```
opt_design -directive NoBramPowerOpt
opt_design -retarget -propconst -sweep
```

Power Optimization

- Optimize dynamic power using clock gating but do not change the clocks or logic of the design
- It can be run AFTER LOGIC OPTIMIZATION or after placement

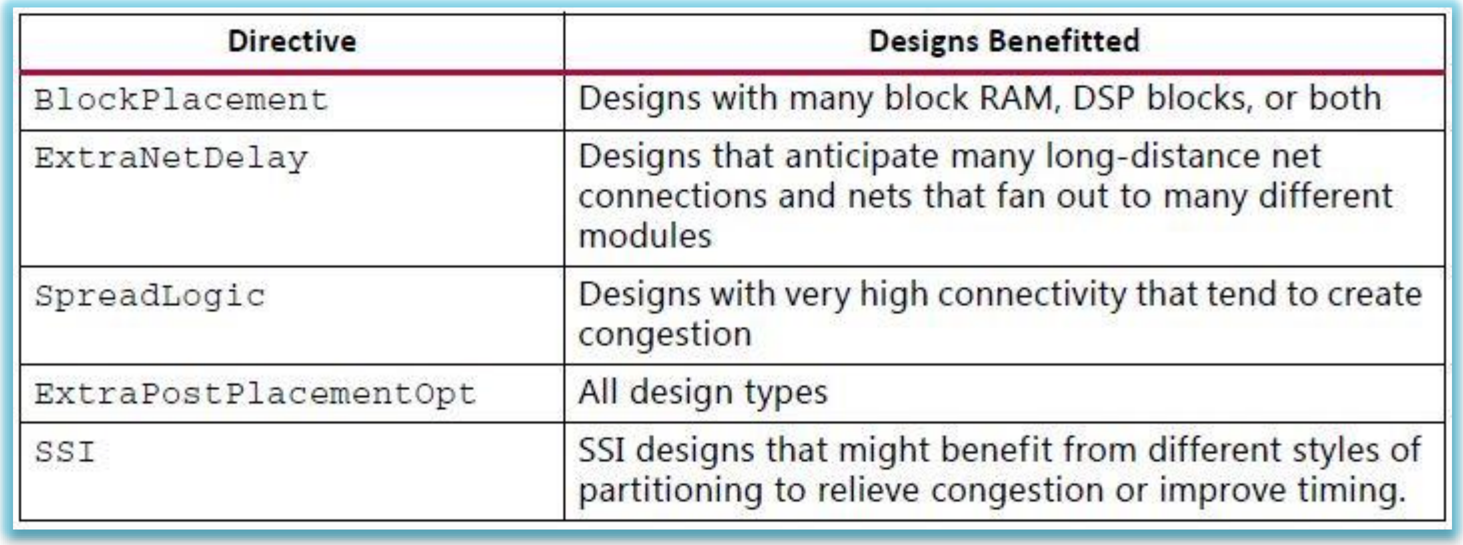


power_opt_design

- power_opt_design [-quiet] [-verbose]
- set_power_opt [-include_cells <args>] [-exclude_cells <args>] [-clocks <args>] [-cell_types <args>] [-quiet] [-verbose]

```
set_power_opt -cell_types {bram reg}
power_opt_design
```

Place Design (place_design)



Directive	Designs Benefitted
`BlockPlacement`	Designs with many block RAM, DSP blocks, or both
`ExtraNetDelay`	Designs that anticipate many long-distance net connections and nets that fan out to many different modules
`SpreadLogic`	Designs with very high connectivity that tend to create congestion
`ExtraPostPlacementOpt`	All design types
`SSI`	SSI designs that might benefit from different styles of partitioning to relieve congestion or improve timing.

Place Design Directives

Wire Length

WLDrivenBlockPlacement
AltWLDrivenPlacement
(UltraScale Only)
LateBlockPlacement

Extra Net Delay

ExtraNetDelay_high
ExtraNetDelay_medium
ExtraNetDelay_low

Spread Logic

SpreadLogic_high
SpreadLogic_medium
SpreadLogic_low

SSI

SSI_ExtraTimingOpt
SSI_SpreadSLLs
SSI_BalanceSLLs
SSI_BalanceSLRs
SSI_HighUtilSLRs

place_design

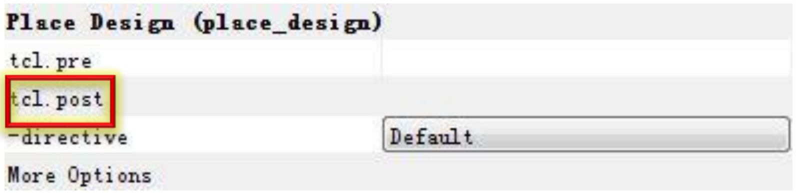
place_design [-directive <arg>] [-no_timing_driven] [-timing_summary] [-unplace] [-post_place_opt] [-quiet] [-verbose]

- **-unplace:**
  - Unplace all the instances which are not locked by constraints
- **-post_place_opt**
  - Run optimization after placement to improve critical path timing at the expense of additional placement and routing runtime
  - This optimization can be run at any stage after placement, and can be particularly effective on a routed design
  - Any placement changes will result in unrouted connections, so route_design will need to be run after -post_place_opt

place_design

```
proc runPPO { {numIters 1} {enablePhysOpt 1} } {
  for {set i 0} {$i < $numIters} {incr i} {
    place_design -post_place_opt
    if {$enablePhysOpt != 0} {
      phys_opt_design
    }
    route_design
    if {[get_property SLACK [get_timing_paths ]] >= 0} {break};
#stop if timing is met
  }
}
```

```
place_design
phys_opt_design
route_design
runPPO 4 1
```



Place Design (place_design)

tcl.pre	
tcl.post	
-directive	Default
More Options	

Physical Optimization (phys_opt_design)

- Physical optimization performs timing-driven optimization on the negative-slack paths of a design
- Two modes of operation
  - post-place
  - post-route

```
phys_opt_design [-fanout_opt] [-placement_opt] [-routing_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] [-clock_opt] [-quiet] [-verbose]
```

Routing

- Router can start with a placed design that is
  - Unrouted
  - Partially routed
  - Fully routed

route_design [-unroute] [-release_memory] [-nets <args>] [-physical_nets] [-pin <arg>] [-directive <arg>] [-tns_cleanup][-no_timing_driven] [-preserve] [-delay] [-auto_delay] -max_delay <arg> -min_delay <arg> [-timing_summary] [-finalize] [-quiet] [-verbose]

route_design

```
route_design
write_checkpoint -force $outputDir/post_route
report_timing_summary –file \
$outputDir/post_route_timing_summary.rpt
```

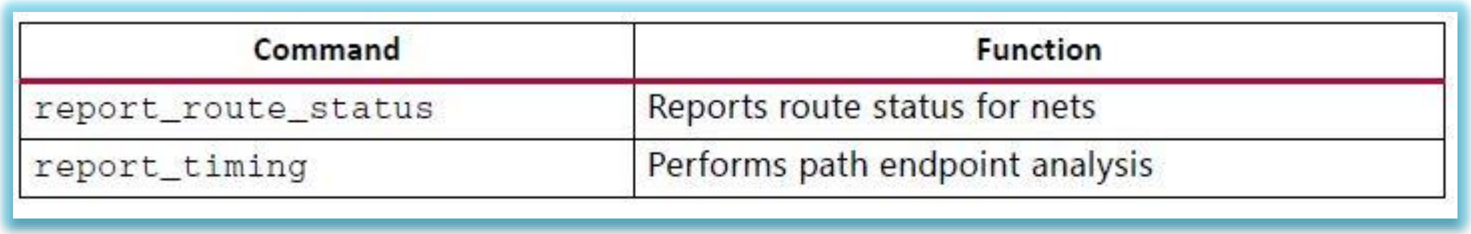
```
# Get the nets in the top 10 critical paths, assign to $preRoutes
set preRoutes [get_nets -of [get_timing_paths -max_paths 10]]
# route $preRoutes first with the smallest possible delay
route_design -nets [get_nets $preRoutes] -delay
# preserve the routing for $preRoutes and continue with the rest
# of the design
route_design -preserve
```

route_design

```
# get nets of the top 10 setup-critical paths
set preRoutes [get_nets -of [get_timing_paths -max_paths 10]]
# get nets of the top 10 hold-critical paths
lappend preRoutes [get_nets -of \
[get_timing_paths -hold -max_paths 10]]
# route $preRoutes based on timing constraints
route_design -nets [get_nets $preRoutes] -auto_delay
# preserve the routing for $preRoutes and continue with the rest
# of the design
route_design -preserve
```

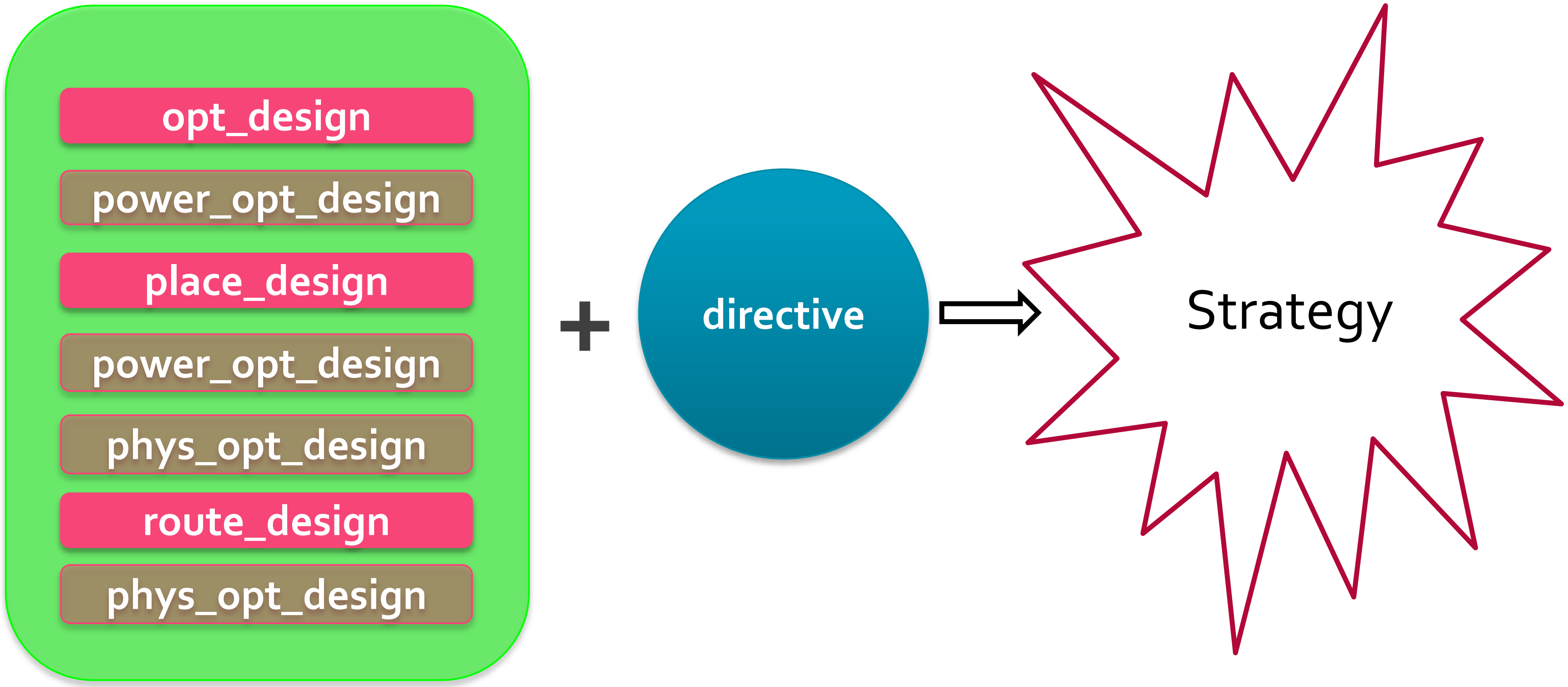
```
route_design
# Unroute all the nets in u0/u1, and route the critical nets
# first
route_design -unroute [get_nets u0/u1/*]
route_design -delay -nets [get_nets $myCritNets]
route_design -preserve
```

route_design



Command	Function
`report_route_status`	Reports route status for nets
`report_timing`	Performs path endpoint analysis

Implementation Strategies



Implementation Strategies

Performance_Explore
Performance_ExplorePostRoutePhysOpt
Performance_RefinePlacement
Performance_WLBlockPlacement
Performance_WLBlockPlacementFanoutOpt
Performance_LateBlockPlacement
Performance_NetDelay_high
Performance_NetDelay_medium
Performance_NetDelay_low
Performance_ExploreSLLs
Performance_Retiming

Congestion_SpreadLogic_high
Congestion_SpreadLogic_medium
Congestion_SpreadLogic_low
Congestion_SpreadLogicSLLs
Congestion_BalanceSLLs
Congestion_BalanceSLRs
Congestion_CompressSLRs

Flow_RunPhysOpt
Flow_RunPostRoutePhysOpt
Flow_RuntimeOptimized
Flow_Quick

Area_Explore

User Defined Strategy

Default

Power_DefaultOpt

Summary

- Implementation strategies are made of different design flows with its different directive
- Directive is an important option in its corresponding design flow
- Choose proper implementation strategy according to your design requirements instead of using default only