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
電子工業出版社

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

Synthesis

Lauren Gao

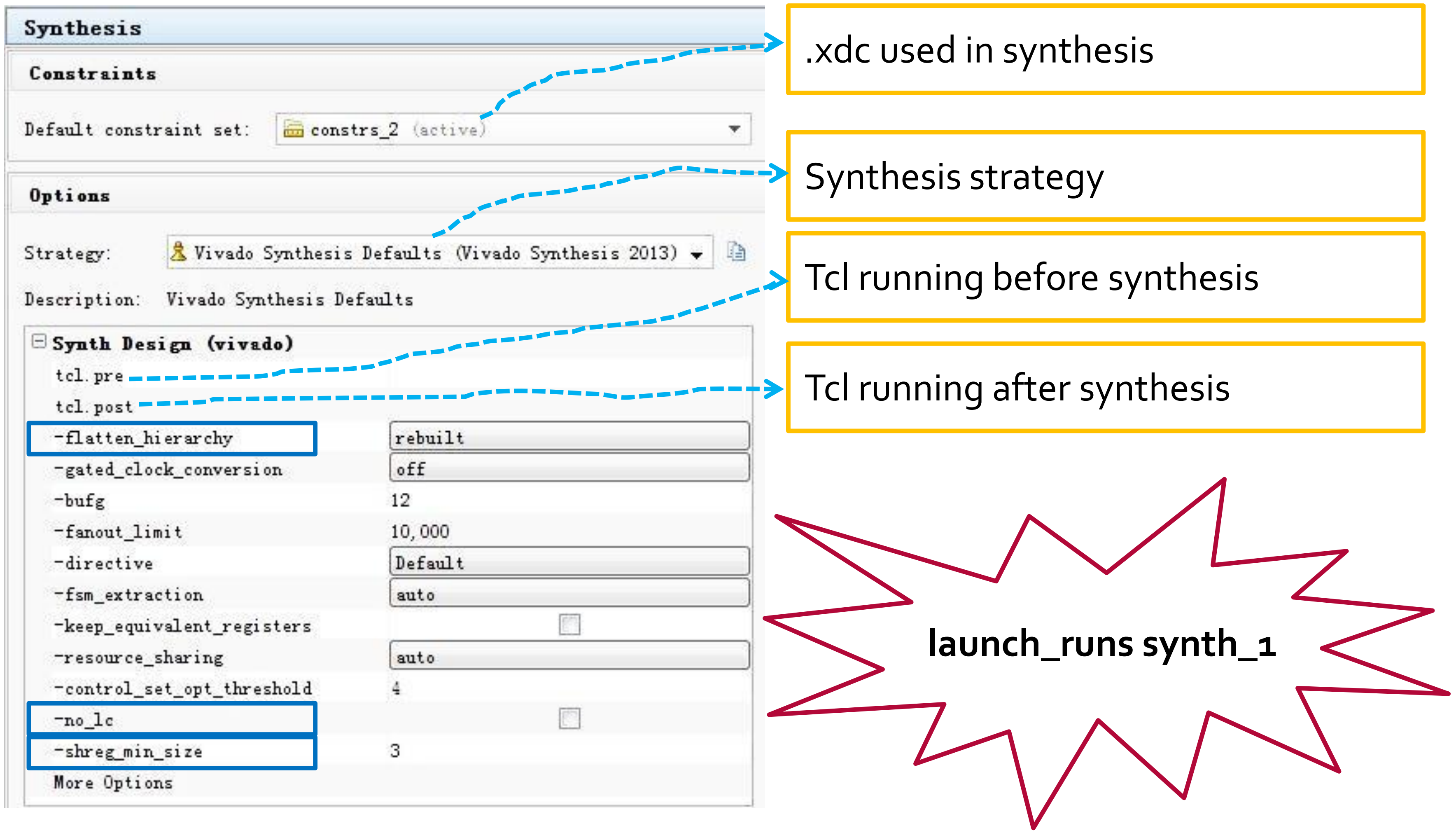
Agenda

- Basic Synthesis Settings
- Some Synthesis Attributes Commonly Used in the Design
- Create Multiple Runs

Agenda

- Basic Synthesis Settings
- Some Synthesis Attributes Commonly Used in the Design
- Create Multiple Runs

Synthesis Settings – Project Mode



Synthesis

Constraints

Default constraint set: constrs_2 (active)

Options

Strategy: Vivado Synthesis Defaults (Vivado Synthesis 2013)

Description: Vivado Synthesis Defaults

Synth Design (vivado)

tcl.pre	
tcl.post	
-flatten_hierarchy	rebuilt
-gated_clock_conversion	off
-bufg	12
-fanout_limit	10,000
-directive	Default
-fsm_extraction	auto
-keep_equivalent_registers	☐
-resource_sharing	auto
-control_set_opt_threshold	4
-no_lc	☐
-shreg_min_size	3

More Options

Synthesis Settings – Non-Project Mode

synth_design [**-name** *arg*] [**-part** *arg*] [**-constrset** *arg*] [**-top** *arg*] [**-include_dirs** *args*] [**-generic** *args*] [**-verilog_define** *args*] [**-flatten_hierarchy** *arg*] [**-gated_clock_conversion** *arg*] [**-directive** *arg*] [**-rtl**] [**-bufg** *arg*] [**-no_lc**] [**-fanout_limit** *arg*] [**-shreg_min_size** *arg*] [**-mode** *arg*] [**-fsm_extraction** *arg*] [**-keep_equivalent_registers**] [**-resource_sharing** *arg*] [**-control_set_opt_threshold** *arg*] [**-quiet**] [**-verbose**]

```
synth_design -top top -part xc7k70tfbg676-2 -flatten_hierarchy none
```

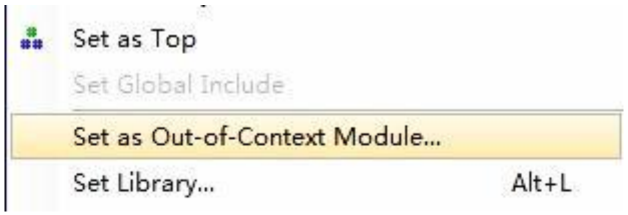
-mode: default, out_of_context (OOC)

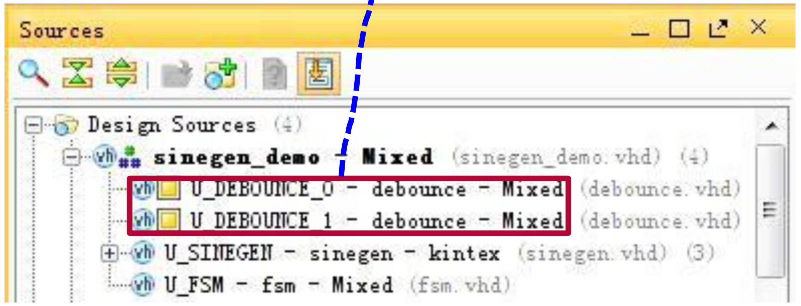
```
synth_design -top top -part xc7k70tfbg676-2 –mode out_of_context
```

Setting a Bottom-Up Flow Using the Out-of-Context Flow

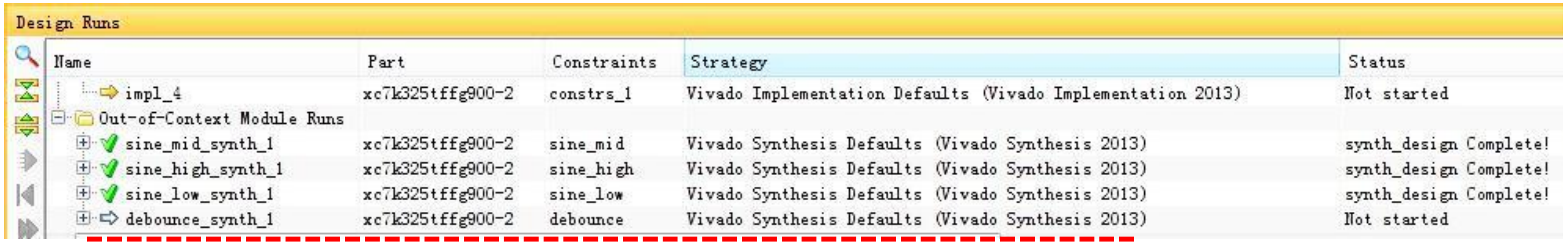
- In ISE
  - Uncheck “Add I/O Buffers” (-iobuf)
- In Vivado
  - Set the module as out of context module

Then have their own .dcp files





```
Design Sources (4)
  sinegen_demo - Mixed (sinegen_demo.vhd) (4)
    U_DEBOUNCE_0 - debounce - Mixed (debounce.vhd)
    U_DEBOUNCE_1 - debounce - Mixed (debounce.vhd)
    U_SINEGEN - sinegen - kintex (sinegen.vhd) (3)
    U_FSM - fsm - Mixed (fsm.vhd)
```



Name	Part	Constraints	Strategy	Status
impl_4	xc7k325tffg900-2	constrs_1	Vivado Implementation Defaults (Vivado Implementation 2013)	Not started
Out-of-Context Module Runs				
sine_mid_synth_1	xc7k325tffg900-2	sine_mid	Vivado Synthesis Defaults (Vivado Synthesis 2013)	synth_design Complete!
sine_high_synth_1	xc7k325tffg900-2	sine_high	Vivado Synthesis Defaults (Vivado Synthesis 2013)	synth_design Complete!
sine_low_synth_1	xc7k325tffg900-2	sine_low	Vivado Synthesis Defaults (Vivado Synthesis 2013)	synth_design Complete!
debounce_synth_1	xc7k325tffg900-2	debounce	Vivado Synthesis Defaults (Vivado Synthesis 2013)	Not started

-flatten_hierarchy

- -flatten_hierarchy options
  - none
    - This option instructs the synthesis tool to never flatten the hierarchy
    - The output of synthesis will have the exact same hierarchy as the original RTL
  - full
    - This option instructs the tool to fully flatten the hierarchy leaving only the top level
  - Rebuilt
    - This is the default flatten_hierarchy option
    - Rebuilt allows the synthesis tool to flatten the hierarchy, perform synthesis, and then rebuild the hierarchy based on the original RTL
    - This value allows the QoR benefit of cross-boundary optimizations, with a final hierarchy that is similar to the RTL for ease of analysis
- **Project mode: this option can be set using the Synthesis Settings button**
- **Non-project mode:**

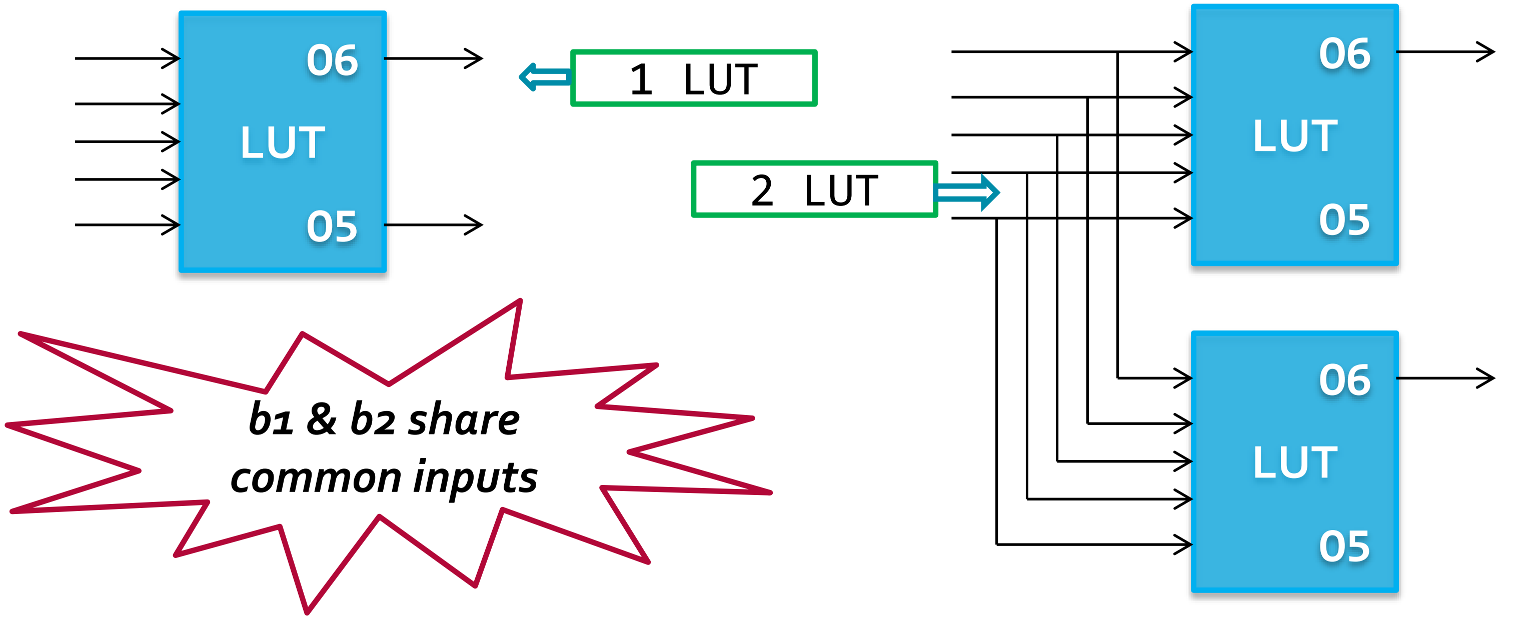
```
synth_design -top bft -part xc7k70tfbg484-2 -flatten_hierarchy none
```

-no_lc

- What is LUT Combining?

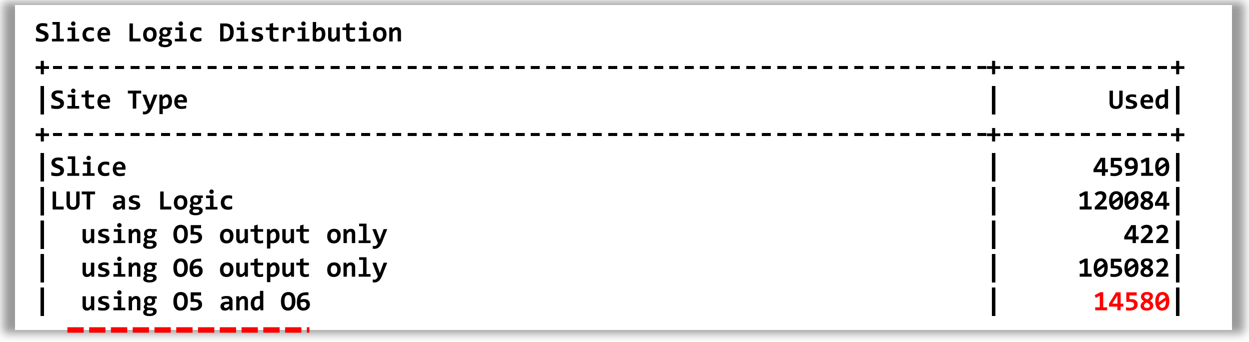
```
b1 <= (a1 and a2) or a3 or (a4 xor a5);
b2 <= (a1 and a2 and a3) or (not (a4 xor a5));
```

-no_lc unchecked (Enable LUT Combing)	Resource Utilization	-no_lc checked (DIsable LUT Combing)



-no_lc

- LUT combining leverages the dual-output LUT (O5/O6)
  - **Pro**: saves area
  - **Con**: could induce congestion
- Tools behavior
  - XST/Synplify combine by default
  - Vivado disable LUT combing by default in synthesis settings
- Use `report_utilization` and look for LUTs with O5 and O6 after implementation



```
Slice Logic Distribution
+----------------------------------------------------------+-----------+
|Site Type                                                 |       Used|
+----------------------------------------------------------+-----------+
|Slice                                                     |      45910|
|LUT as Logic                                              |     120084|
|  using O5 output only                                    |        422|
|  using O6 output only                                    |     105082|
|  using O5 and O6                                         |      14580|
```

- Guideline: If >15% of LUT use both O5 and O6, then
  - Consider turning off LUT combining in synthesis

Agenda

- Basic Synthesis Settings
- Some Synthesis Attributes Commonly Used in the Design
- Create Multiple Runs

Shift Register Coding Example with srl_style

```vhdl
architecture archi of shift_reg is
signal shreg: std_logic_vector(DEPTH-1 downto 0);

attribute srl_style : string;
attribute srl_style of shreg : signal is "reg_srl_reg";
begin

process (clk)
begin
  if rising_edge(clk) then
    if ce = '1' then
      shreg <= shreg(DEPTH-2 downto 0) & SI;
    end if;
  end if;
end process;
SO <= shreg(DEPTH-1);
end archi;
```

VHDL

```verilog
    (* srl_style = "register" *) reg [WIDTH-1:0] shreg;
    integer i;
    always @(posedge clk)
      begin
        if (clken)
        begin
          for (i = 0; i < WIDTH-1; i = i+1)
            shreg[i+1] <= shreg[i];
            shreg[0] <= SI;
        end
      end
      assign SO = shreg[WIDTH-1];
```

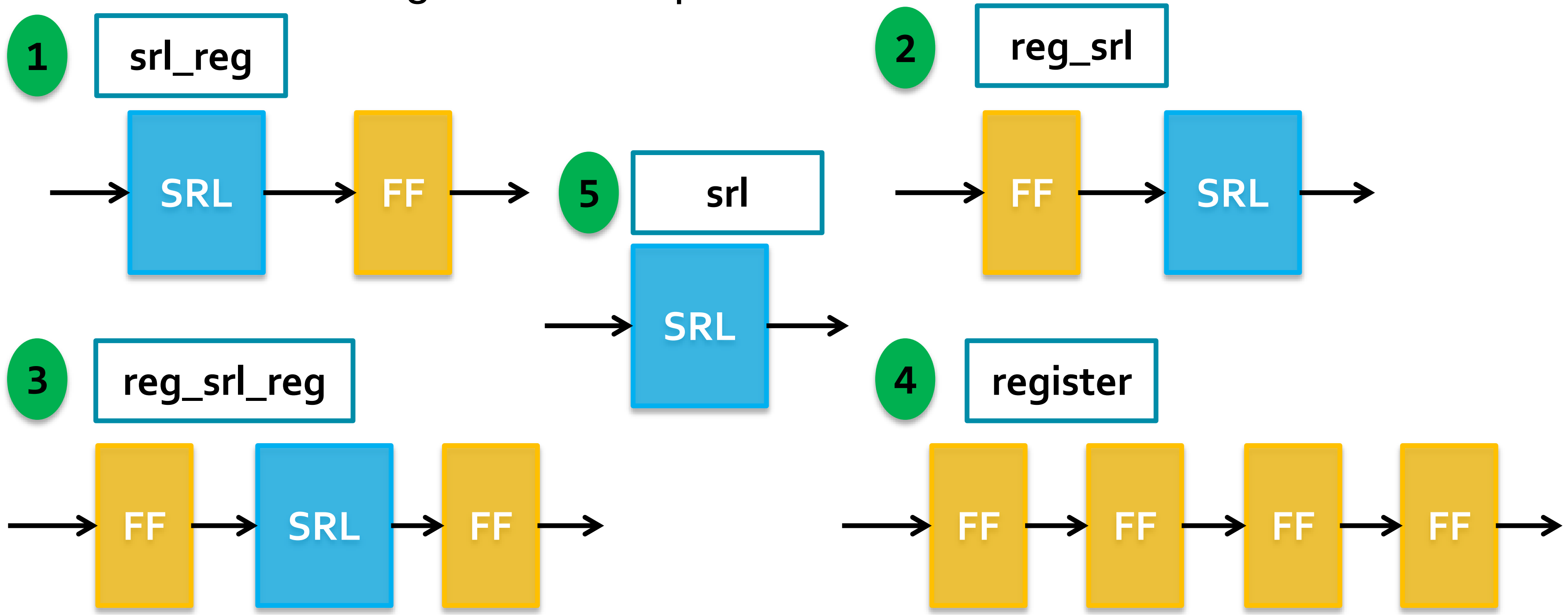
Verilog

- 7-series:
  - SRL16E
  - *SRLC32E*
- *Basic Ports:*
  - *Q,D,CLK,CE, A*
- *Do not support reset*

1 to 32 clock cycle shift register implemented within a single look-up table (LUT)

srl_style Values

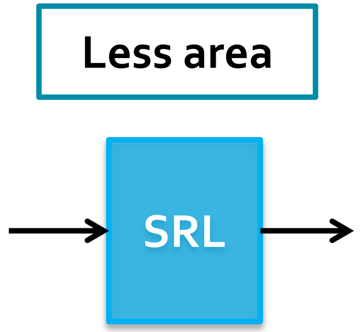
- controls shift register decomposition

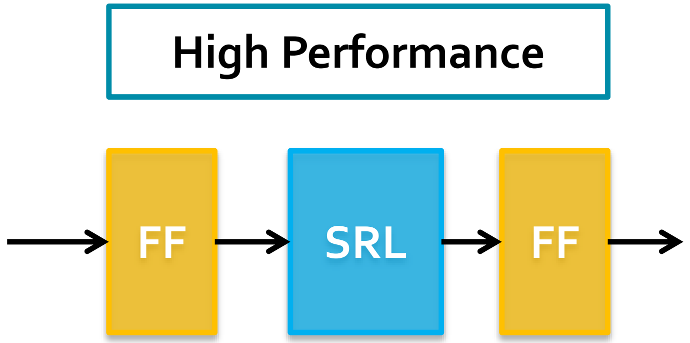


- **Do not support RESET with SRL**
- **Make sure –shreg_min_size is reasonable**
  - *If –shreg_min_size is 8, shift register depth is 4, srl_style is srl_reg, the synthesis result is not SRL + FF but cascading FFs*
- **Support for dynamic SRL inference**

When and How to Use srl_style?

- **Advantages of SRL**
  - less area
    - Using a single SRL you can implement up to 32-cycle shift register
  - Performance
    - Routing delays between cascaded ffs decreases the performance (Fmax) comparing to SRL
- **Tips**
  - The clock-to-output time of a flop is much lower than clock-to-output of an SRLThat's why it makes sense to add a flop on the output of an SRL





ram_style & rom_style

- Instructs the Vivado synthesis tool on how to infer memory
- Accepted values accepted are
  - **block**:
    - Instructs the tool to infer RAMB type components
  - **distributed**:
    - Instructs the tool to infer the LUT RAMs
- Place this attribute on the array that is declared for the RAM

Verilog:

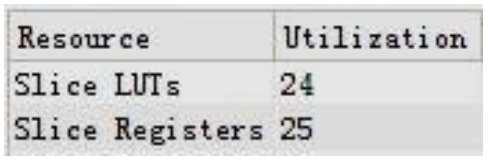
```
(* ram_style = "distributed" *) reg [data_size-1:0] myram[2**addr_size-1:0];
```

VHDL:

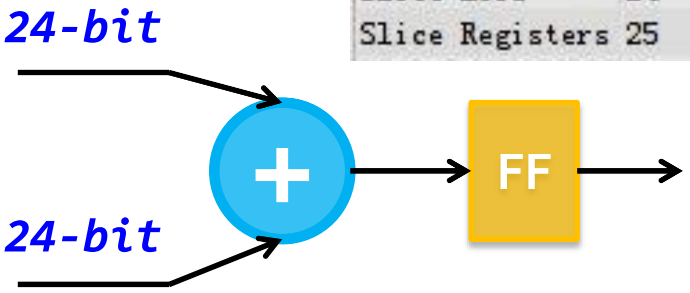
```
attribute ram_style : string;
attribute ram_style of myram : signal is "distributed ";
```

use_dsp48

- Instructs the synthesis tool how to deal with synthesis arithmetic structures
- Which arithmetic type structures go into DSP48 blocks by default
  - Mult
  - Mult-add & mult-sub
  - Mult-accumulate
- Adders, subtracters, and accumulators are implemented with the fabric instead of with DSP48 blocks by default
- Can they go into DSP48?
  - YES, apply use_dsp48 to them
- Accepted values
  - "yes" and "no"



Resource	Utilization
Slice LUTs	24
Slice Registers	25



Some Other Attributes

- black_box
  - It can turn a whole level of hierarchy off and enable synthesis to create a black box for that module or entity
  - The legal values are “yes” and “no”
  - This attribute can be placed on a module, entity, or component
- dont_touch
  - Use the DONT_TOUCH attribute in place of KEEP
  - Unlike KEEP, DONT_TOUCH is forward-annotated to place and route to prevent logic optimization
  - The legal values are “true” and “false”
  - This attribute can be placed on any signal, module, entity, or component
- fsm_enconding
  - It controls how a state machine is encoded
  - The legal values
    - "one_hot", "sequential", "johnson", "gray", and "auto“
  - It can be placed on the state machine registers

Examples 1

black_box

Verilog

```
(* black_box *) module test(in1, in2, clk, out1);
```

VHDL

```
attribute black_box : string;
attribute black_box of beh : architecture is "yes";
```

dont_touch

Verilog

```
(* dont_touch = "true" *) wire sig1;
assign sig1 = in1 & in2;
assign out1 = sig1 & in2;
```

VHDL

```
signal sig1 : std_logic
attribute dont_touch : string;
attribute dont_touch of sig1 : signal is "true";
```

Example 2

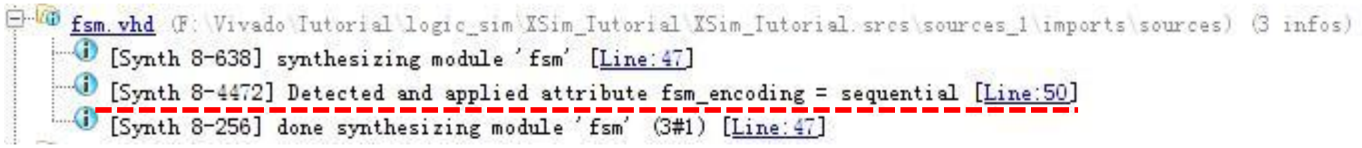
fsm_encoding

Verilog

```
(* fsm_encoding = "one_hot" *) reg [7:0] my_state;
```

VHDL

```
type count_state is (zero,one,two,three,four,five,six,seven);
signal my_state : count_state;
attribute fsm_encoding : string;
attribute fsm_encoding of my_state : signal is "sequential";
```

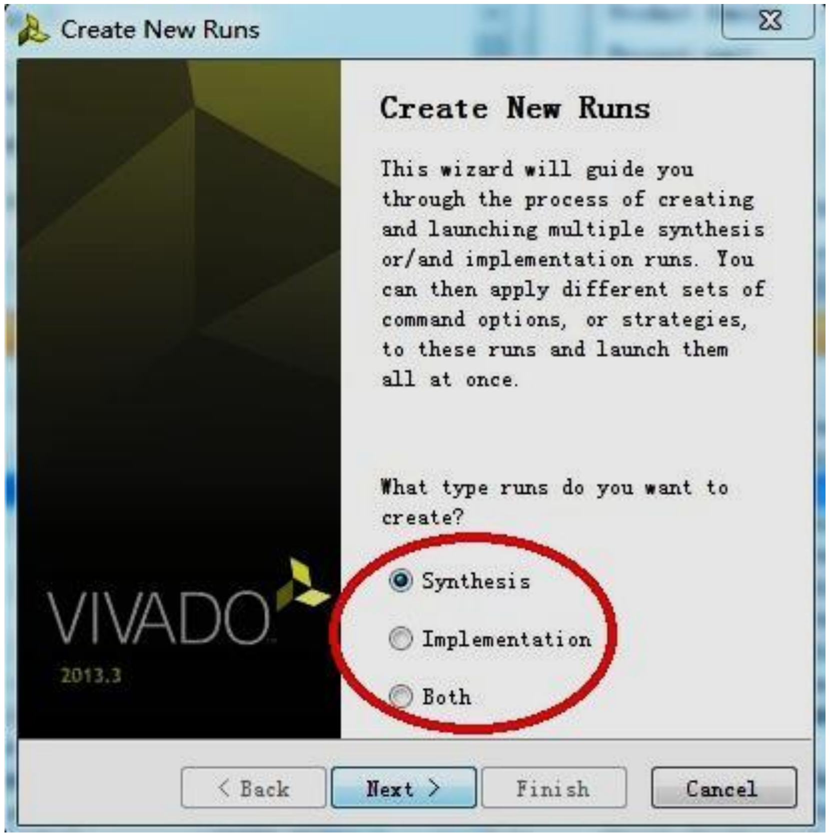


```
fsm.vhd (F:\Vivado Tutorial\logic_sim\XSim_Tutorial\XSim_Tutorial.srcs\sources_1\imports\sources) (3 infos)
    [Synth 8-638] synthesizing module 'fsm' [Line:47]
    [Synth 8-4472] Detected and applied attribute fsm_encoding = sequential [Line:50]
    [Synth 8-256] done synthesizing module 'fsm' (3#1) [Line:47]
```

Agenda

- Basic Synthesis Settings
- Some Synthesis Attributes Commonly Used in the Design
- Create Multiple Runs

Create New Runs



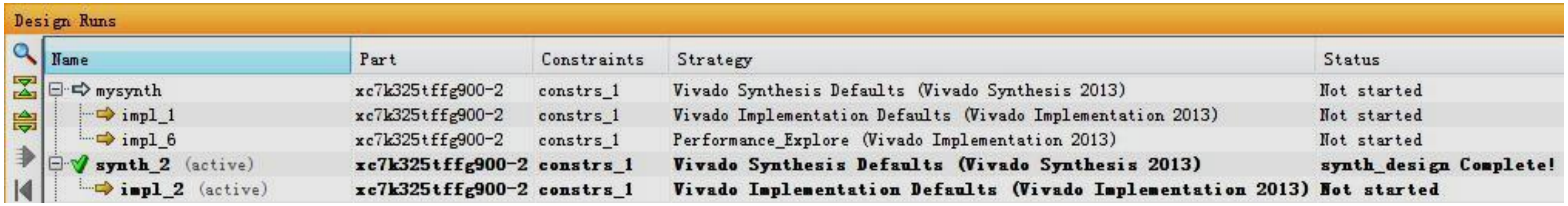
Create New Runs

This wizard will guide you through the process of creating and launching multiple synthesis or/and implementation runs. You can then apply different sets of command options, or strategies, to these runs and launch them all at once.

What type runs do you want to create?

[✓ Synthesis]
Implementation
Both

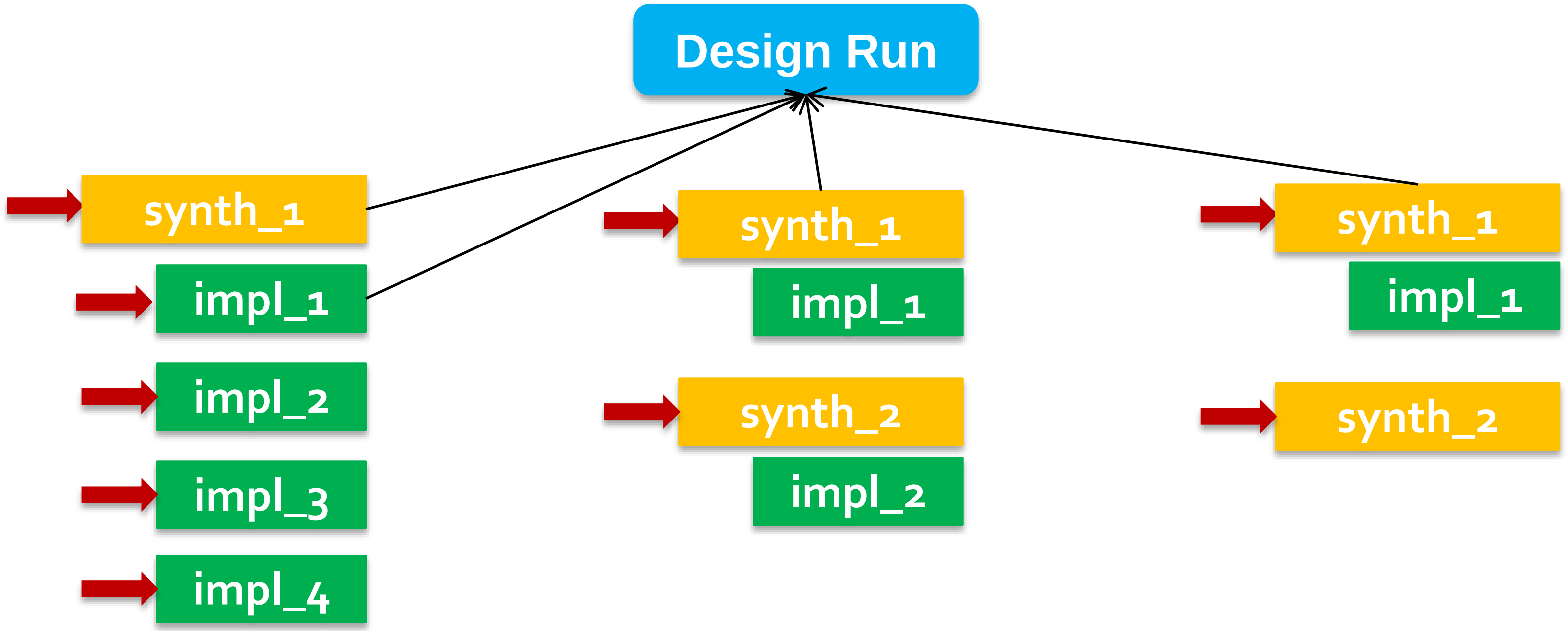
- Flow → Create Runs
- Design Runs → Create Runs
- Runs type:
  - Synthesis
  - Implementation
  - Both



Design Runs

Name	Part	Constraints	Strategy	Status
mysynth	xc7k325tffg900-2	constrs_1	Vivado Synthesis Defaults (Vivado Synthesis 2013)	Not started
impl_1	xc7k325tffg900-2	constrs_1	Vivado Implementation Defaults (Vivado Implementation 2013)	Not started
impl_6	xc7k325tffg900-2	constrs_1	Performance_Explore (Vivado Implementation 2013)	Not started
synth_2 (active)	**xc7k325tffg900-2**	**constrs_1**	**Vivado Synthesis Defaults (Vivado Synthesis 2013)**	**synth_design Complete!**
impl_2 (active)	**xc7k325tffg900-2**	**constrs_1**	**Vivado Implementation Defaults (Vivado Implementation 2013)**	**Not started**

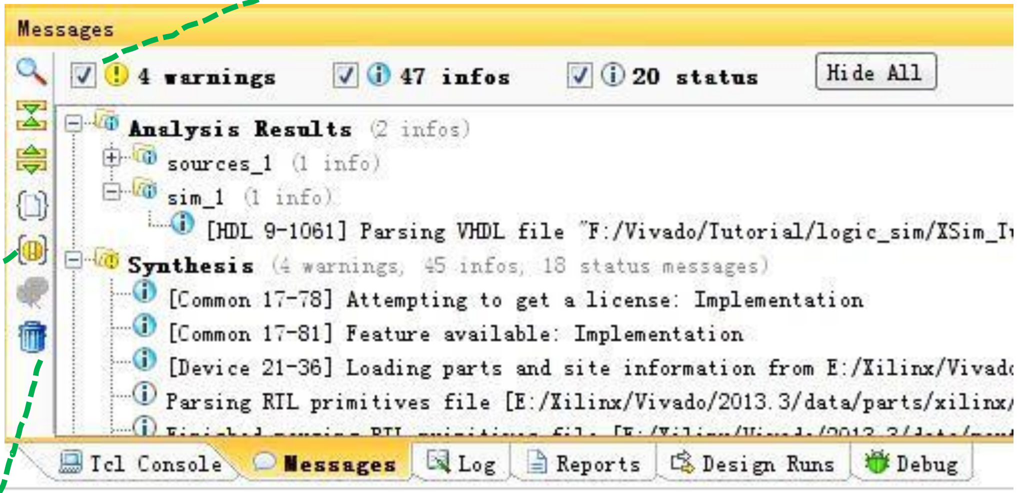
Create and Manage Multiple Runs



- Each run can have different constraint, different part and different strategy
- Impl run is based on synth run which is impl run's parent run
- Only one run is active. *current_run [get_runs synth_1]*
- Run result is called design. *current_design*
- Run can also be deleted. *delete_run synth_2*

XILINX ALL PROGRAMMABLE.

Message Management



Analysis Results (2 infos)
sources_1 (1 info)
sim_1 (1 info)
[HDL 9-1061] Parsing VHDL file "F:/Vivado/Tutorial/logic_sim/XSim_T
Synthesis (4 warnings, 45 infos, 18 status messages)
[Common 17-78] Attempting to get a license: Implementation
[Common 17-81] Feature available: Implementation
[Device 21-36] Loading parts and site information from E:/Xilinx/Vivad
Parsing RTL primitives file [E:/Xilinx/Vivado/2013.3/data/parts/xilinx/

Filter different types

Group info

Discard old message

Summary

- Support both top-down and bottom-up synthesis flow
- Support multiple runs for experiments
- Powerful messages management