XILINX

Vivado Adapt 2021 Design Closure

Methodology, tips, and tricks for achieving better Quality-of-Results (QoR)

Feb 11, 2021

Design Closure Sessions

- Session 1

  Methodology, tips, and tricks for achieving better Quality-of-Results

- Session 2

  Using Timing Closure Assistance tools to address tough timing issues

- Session 3

  Power Constraints, best practices for an accurate Report Power estimation

Agenda - Methodology, Tips, and Tricks

- Vivado Tool and Methodology Updates

- Synthesis
  - Key Synthesis Features (2020.X)
  - Tips and Tricks

- Implementation Updates
  - Key Implementation Features (2020.X)
  - Versal Implementation Guidelines

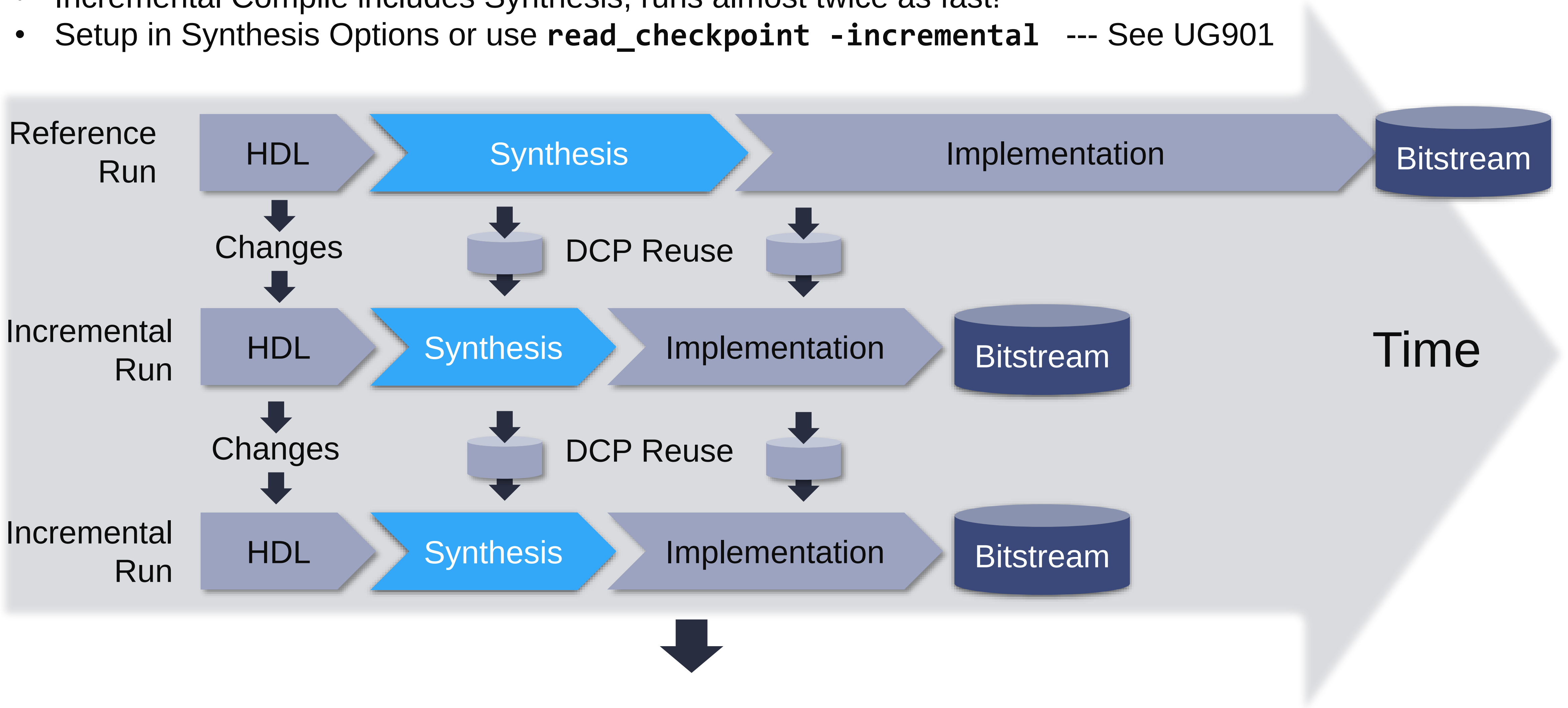
Tool and Methodology Updates

Vivado Compile Time Improvements

- Synthesis speedup
  - 2020.1: Constant RTL *function* compile times reduced to tiny fraction of 2019.2 time
  - 2020.2: Average **20%** overall improvement compared to 2020.1 (Versal devices)
- Placer **20%** average speedup on SSI designs with UltraThreads
  - Initial support (2020.2): enabled for Default, RuntimeOptimized, and Quick directives
  - Planned improvements (2021.x): enabled for other directives using place_design -ultrathreads
  - Use with general.maxThreads >= number of SLRs
- Router **33%** average speedup in 2020.2 compared to 2019.2
  - Major initialization tasks are now completed offline
  - Improved SLR crossing routing algorithms

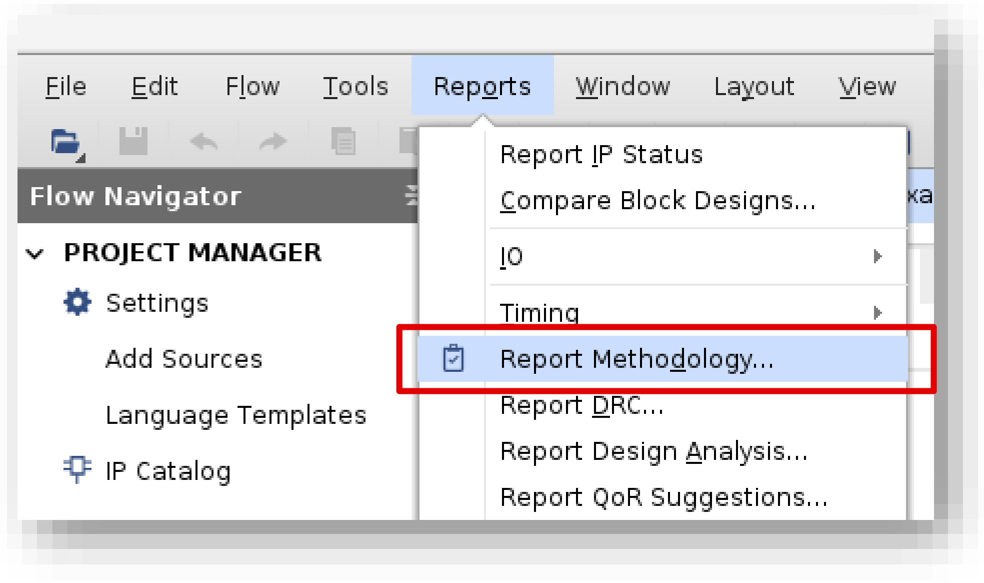
Incremental Synthesis

- Incremental Compile includes Synthesis, runs almost twice as fast!
- Setup in Synthesis Options or use **`read_checkpoint -incremental`** --- See UG901



New Methodology Checks

- UltraFast Methodology checks are built into Vivado reports
  - Access under Reports menu or Tcl command **`report_methodology`**
  - Automatically generated in Vivado projects
- Review and correct or waive warnings and critical violations!



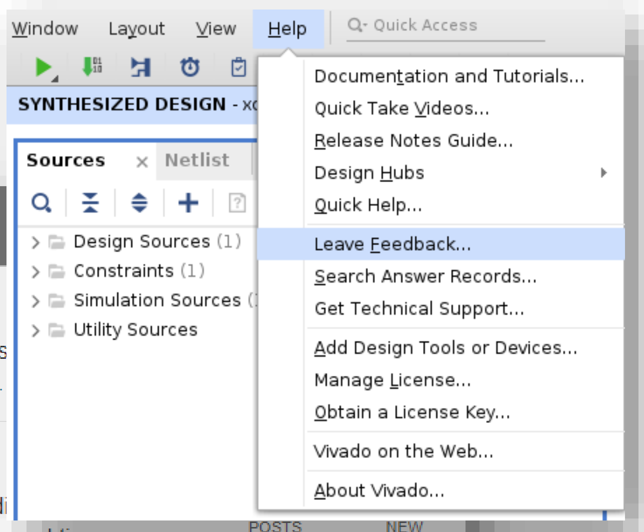
Recently added checks in 2020

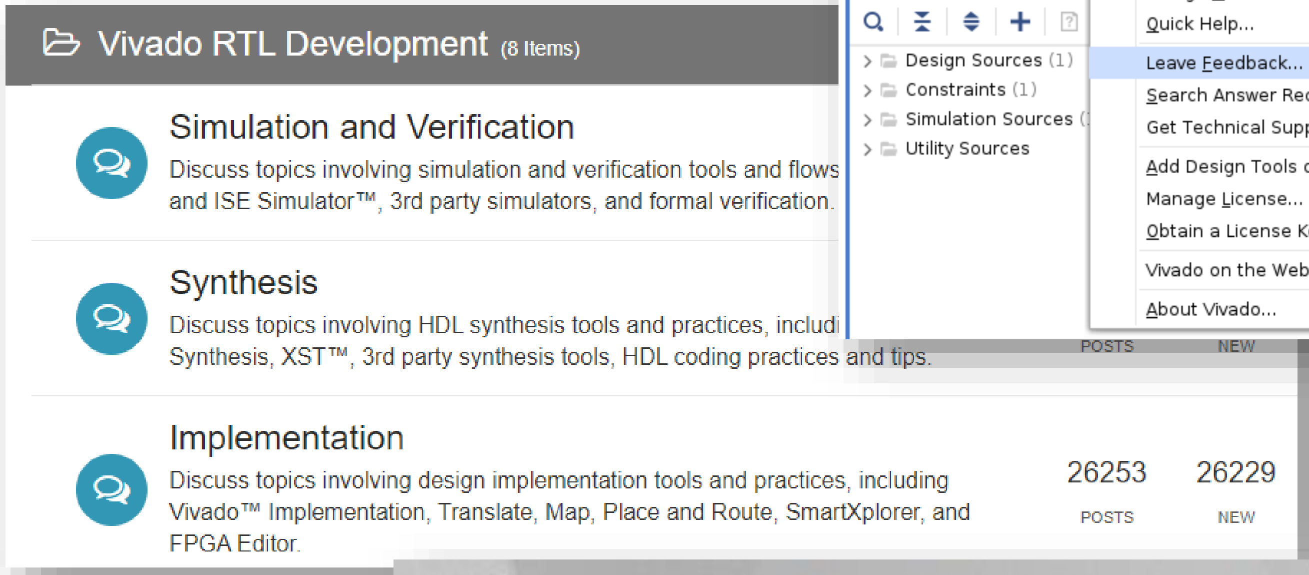
Rule ID	Severity	Description
XDCB-6	Advisory	Timing constraint pointing to hierarchical pins
TIMING-54	Critical Warning	Scoped false path or clock group constraint between clocks
TIMING-56	Warning	Missing logically or physically exclusive clock groups constraint

Vivado: Feedback, Discussions, and Blogs

- Help -> Leave Feedback

- Community Forums
  - Discussions
    - Categories for all tools
    - Experts and other users
  - Design Blogs
    - Written by Xilinx experts
    - Most new features are introduced in blogs
    - Leave comments at end!



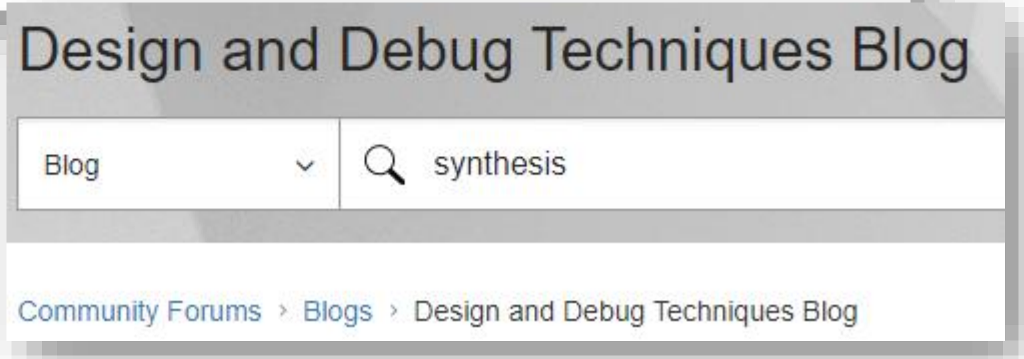

Vivado RTL Development (8 Items)

Simulation and Verification
Discuss topics involving simulation and verification tools and flows and ISE Simulator™, 3rd party simulators, and formal verification.

Synthesis
Discuss topics involving HDL synthesis tools and practices, includi Synthesis, XST™, 3rd party synthesis tools, HDL coding practices and tips.

Implementation
Discuss topics involving design implementation tools and practices, including Vivado™ Implementation, Translate, Map, Place and Route, SmartXplorer, and FPGA Editor.

26253	26229
POSTS	NEW


Design and Debug Techniques Blog
Blog
synthesis
Community Forums › Blogs › Design and Debug Techniques Blog

XILINX.

Tool and Methodology Takeaways

- Enable Incremental Synthesis with Incremental Compile to speed up iterations
- Review Methodology Reports and correct or waive warnings and critical violations
- Share feedback on Vivado, join online forum discussions, and review and comment on our blogs

Key Synthesis Features (2020.X)

Expanded Language Support

- VHDL-2008 IEEE fixed-point and floating-point packages
  - Now can target both packages using ieee statements instead of using an intermediate file
    - use ieee.float_pkg.all;
    - use ieee.fixed_pkg.all;
- SystemVerilog: constant strings
  - Strings can be used as parameters/localparams where the size of the string is fixed (Not to be used in logic)
  - Support for methods Len(), Getc(), Toupper(), Tolower(), Compare(), Atopi(), Atohex(), Atooct(), Atobin(), Atoreal()
- Mixed language support - passing generics and parameters in between VHDL and Verilog improved
  - Can handle multidimensional arrays/records/structs

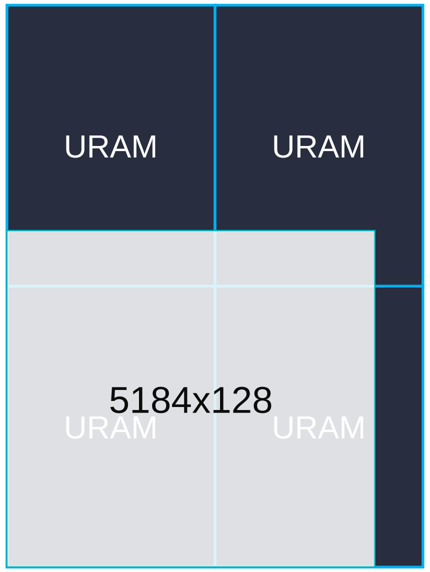
XILINX.

Heterogeneous RAM Mapping

- Maps to a mix of LUTRAM, Block RAM, and UltraRAM for highest efficiency
  - 2020.1: HDL attribute **ram_style=mixed** added
  - 2020.2: Pipeline register mapping
  - 2021.1: Planned support for XPMs

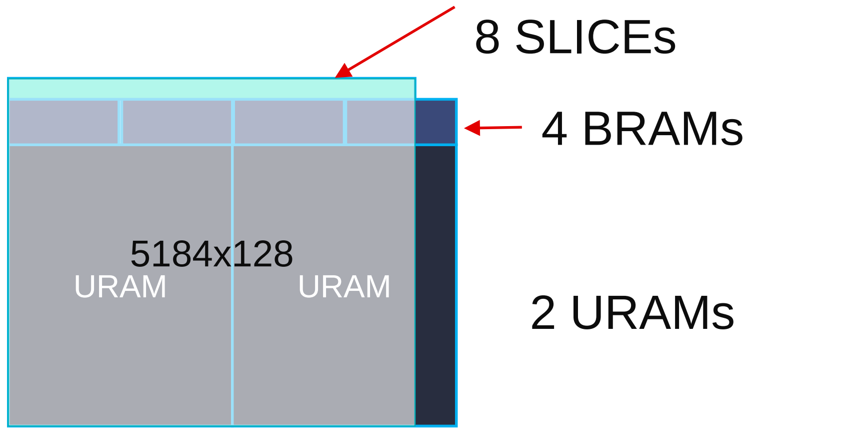
Note: report_ram_utilization reports bit utilization percentage (depth x width util)

Example:
5184x128 array



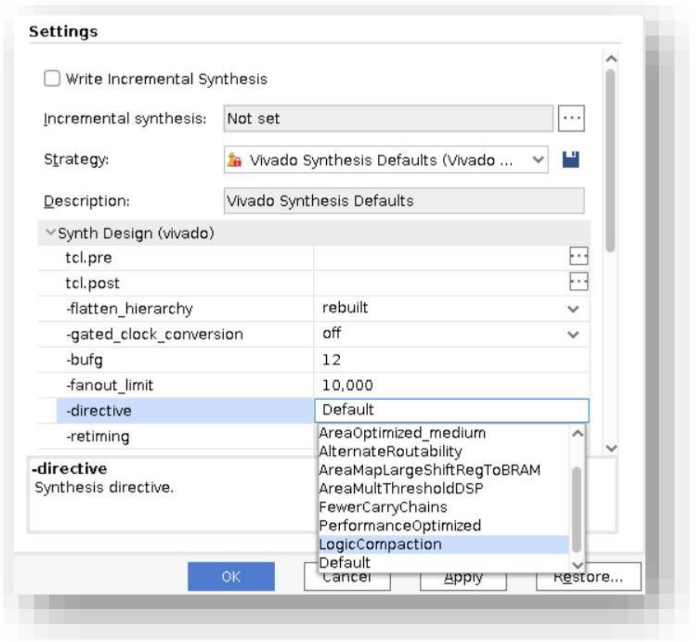
4 URAMs holds 8192x144 so portions are wasted

Heterogeneous Mapping:



Logic Compaction Optimization

- Reduce slice utilization of low-precision arithmetic
- Available globally as a directive or per-hierarchy using BLOCK_SYNTH cell property
- Supports both CARRY and LOOKAHEAD (Versal)


Settings

☐ Write Incremental Synthesis

Incremental synthesis:	Not set
Strategy:	Vivado Synthesis Defaults (Vivado ...
Description:	Vivado Synthesis Defaults

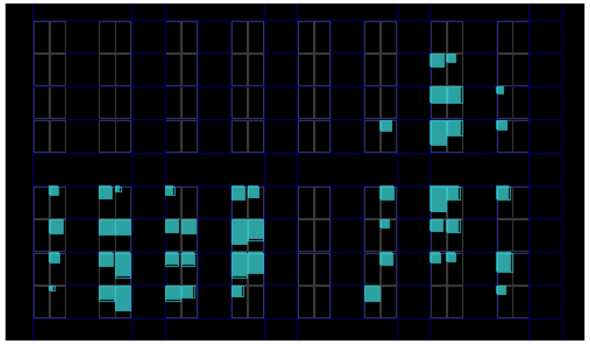
Synth Design (vivado)

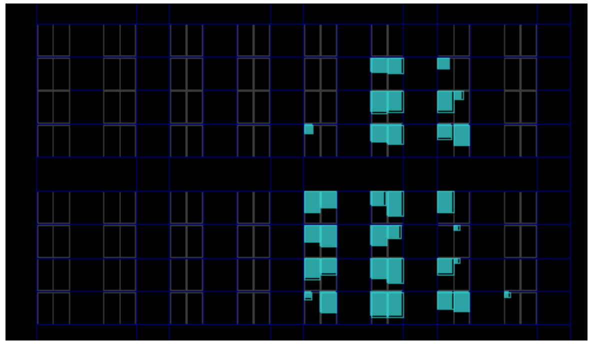
tcl.pre	
tcl.post	
-flatten_hierarchy	rebuilt
-gated_clock_conversion	off
-bufg	12
-fanout_limit	10,000
-directive	Default
-retiming	

-directive
Synthesis directive.

AreaOptimized_medium
AlternateRoutability
AreaMapLargeShiftRegToBRAM
AreaMultThresholdDSP
FewerCarryChains
PerformanceOptimized
LogicCompaction
Default

Versal example:
9x9 Multiply-Add, 3 stages


Default: timing-optimized
240 LUTs, 12 LOOKAHEADs
49 Slices


Smaller with Logic Compaction
186 LUTs, 27 LOOKAHEADs
40 Slices

Ease of Use Enhancements

- SRL_STYLE for static shift registers becomes a **global** option, additional usage now includes:
  - Hierarchical cells

```
set_property BLOCK_SYNTH.SRL_STYLE REG_SRL [get_cells mod_inst]
```

  - Tcl command

```
synth_design -top <top_name> -srl_style reg_srl_reg ...
```

Value	Style
register	no SRL, all FFs
srl	SRL only
srl_reg	SRL->FF
reg_srl	FF->SRL
reg_srl_reg	FF->SRL->FF
block	block RAM

- Override KEEP and DONT_TOUCH in RTL code
  - Set KEEP or DONT_TOUCH false in XDC to optimize away
  - Useful when RTL code cannot be modified

If used in XDC files, limit USED_IN to Synthesis - will generate critical warnings in Implementation due to constraints applied to optimized nets

HDL:

```
(* KEEP="true" *) reg [255:0] debug_signals;
```

Synthesis XDC:

```
set_property KEEP false [get_nets debug_signals*]
```

Synthesizing for Versal: RAM Mapping

- Fewer Block RAM aspect ratios: 8kx4, 16kx2, 32kx1 not supported
- UltraRAM initialization **is** supported, more aspect ratios: 8kx16, 16kx8, 32kx4

Block RAMs vs array sizes

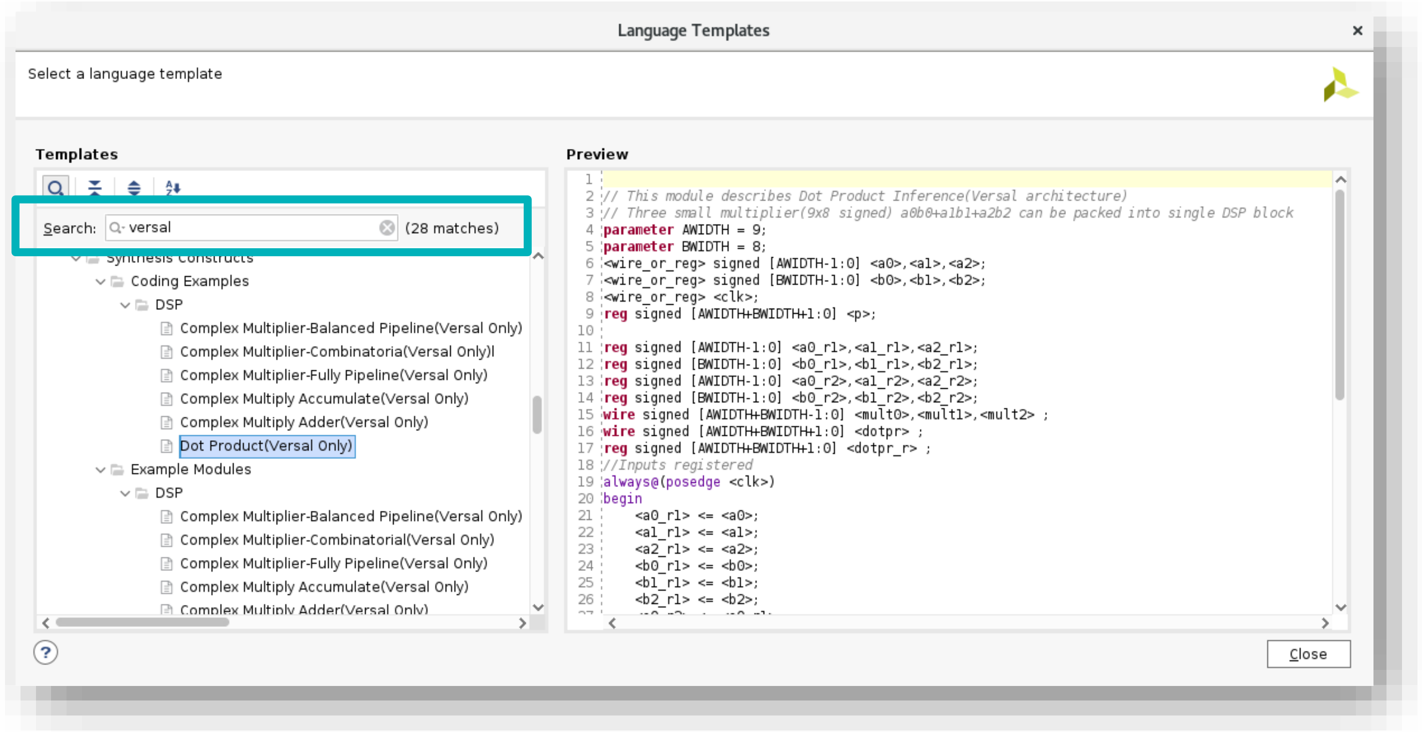
Depth	Width	UltraScale+	Versal
2^{10} (1k)	32	1	1
2^{11} (2k)	16	1	1
2^{12} (4k)	8	1	1
2^{13} (8k)	4	1	2
2^{14} (16k)	2	1	4
2^{15} (32k)	1	1	8

UltraRAMs vs array sizes

Depth	Width	UltraScale+	Versal
2^{12} (4k)	32	1	1
2^{13} (8k)	16	2	1
2^{14} (16k)	8	4	1
2^{15} (32k)	4	8	1

Synthesizing for Versal: DSP Block Mapping

- Complex Multiplier: 18x18 in 2 DSP blocks (3 required for UltraScale+)
- Dot Product: single DSP block holds 3 9x8 signed multiply-add
- See Language Templates


Language Templates
Select a language template
Templates
Search: versal
(28 matches)
Synthesis Constructs
Coding Examples
DSP
Complex Multiplier-Balanced Pipeline(Versal Only)
Complex Multiplier-Combinatoria(Versal Only)l
Complex Multiplier-Fully Pipeline(Versal Only)
Complex Multiply Accumulate(Versal Only)
Complex Multiply Adder(Versal Only)
Dot Product(Versal Only)
Example Modules
DSP
Complex Multiplier-Balanced Pipeline(Versal Only)
Complex Multiplier-Combinatorial(Versal Only)
Complex Multiplier-Fully Pipeline(Versal Only)
Complex Multiply Accumulate(Versal Only)
Complex Multiply Adder(Versal Only)
Preview
Close

```

// This module describes Dot Product Inference(Versal architecture)
// Three small multiplier(9x8 signed) a0b0+a1b1+a2b2 can be packed into single DSP block
parameter AWIDTH = 9;
parameter BWIDTH = 8;
<wire_or_reg> signed [AWIDTH-1:0] <a0>,<a1>,<a2>;
<wire_or_reg> signed [BWIDTH-1:0] <b0>,<b1>,<b2>;
<wire_or_reg> <clk>;
reg signed [AWIDTH+BWIDTH+1:0] <p>;

reg signed [AWIDTH-1:0] <a0_r1>,<a1_r1>,<a2_r1>;
reg signed [BWIDTH-1:0] <b0_r1>,<b1_r1>,<b2_r1>;
reg signed [AWIDTH-1:0] <a0_r2>,<a1_r2>,<a2_r2>;
reg signed [BWIDTH-1:0] <b0_r2>,<b1_r2>,<b2_r2>;
wire signed [AWIDTH+BWIDTH-1:0] <mult0>,<mult1>,<mult2> ;
wire signed [AWIDTH+BWIDTH+1:0] <dotpr> ;
reg signed [AWIDTH+BWIDTH+1:0] <dotpr_r> ;
//Inputs registered
always@(posedge <clk>)
begin
    <a0_r1> <= <a0>;
    <a1_r1> <= <a1>;
    <a2_r1> <= <a2>;
    <b0_r1> <= <b0>;
    <b1_r1> <= <b1>;
    <b2_r1> <= <b2>;
```

Floating point modes:
DSPFP32 instantiation required

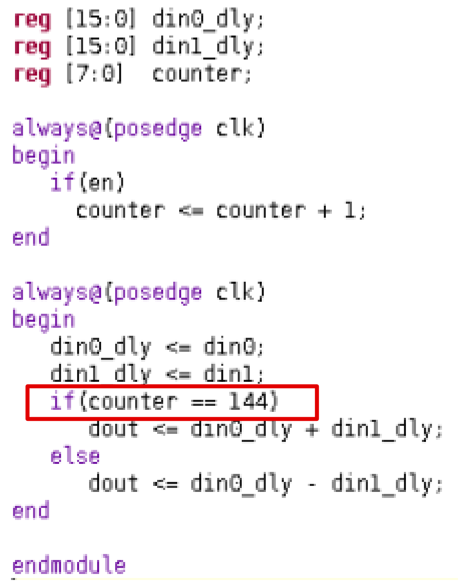
See AM004 for further details on Versal DSP

Synthesis Tips & Tricks

Device : xcvc1902-vsva2197-1LP-i-S
Frequency : 500 MHz

Optimizing a Critical Multiplexer 1/2

- Multiplexer in the critical path
  - Select path driven by counter->comparator
  - Mux inputs are driven by adder-subtractor
- Can the multiplexer be optimized further?

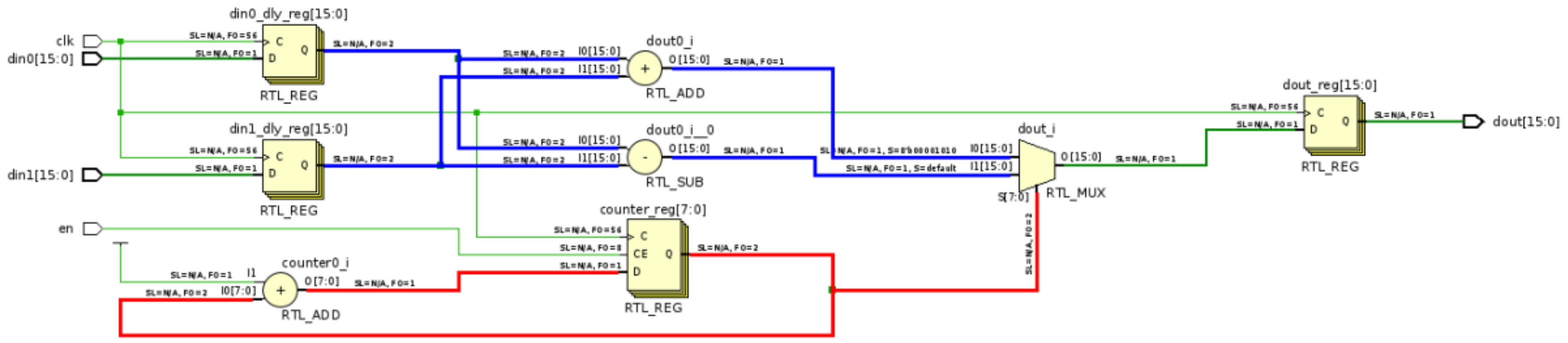


```
reg [15:0] din0_dly;
reg [15:0] din1_dly;
reg [7:0]  counter;

always@(posedge clk)
begin
   if(en)
     counter <= counter + 1;
end

always@(posedge clk)
begin
   din0_dly <= din0;
   din1_dly <= din1;
   if(counter == 144)
      dout <= din0_dly + din1_dly;
   else
      dout <= din0_dly - din1_dly;
end

endmodule
```

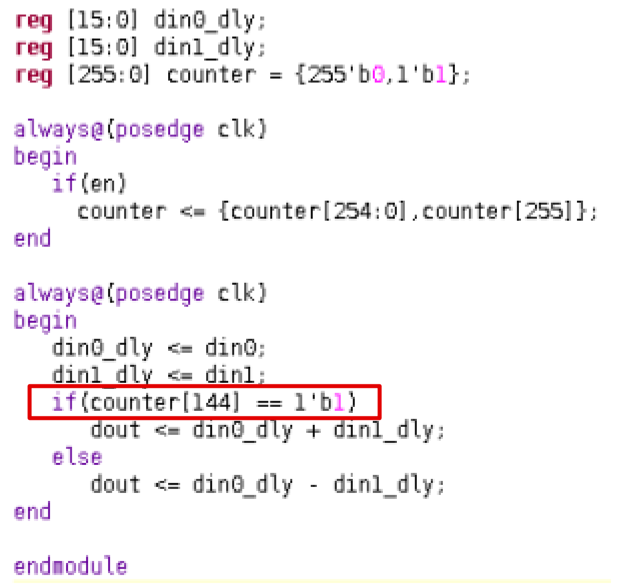


XILINX.

Device : xcvc1902-vsva2197-1LP-i-S
Frequency : 500 MHz

Optimizing a Critical Multiplexer 2/2

- Trick: replace counter - comparator with one-hot shift register
  - No complex decode logic, single bit for mux selection
  - Long shift register can be mapped to SRLs (LUTRAMs)

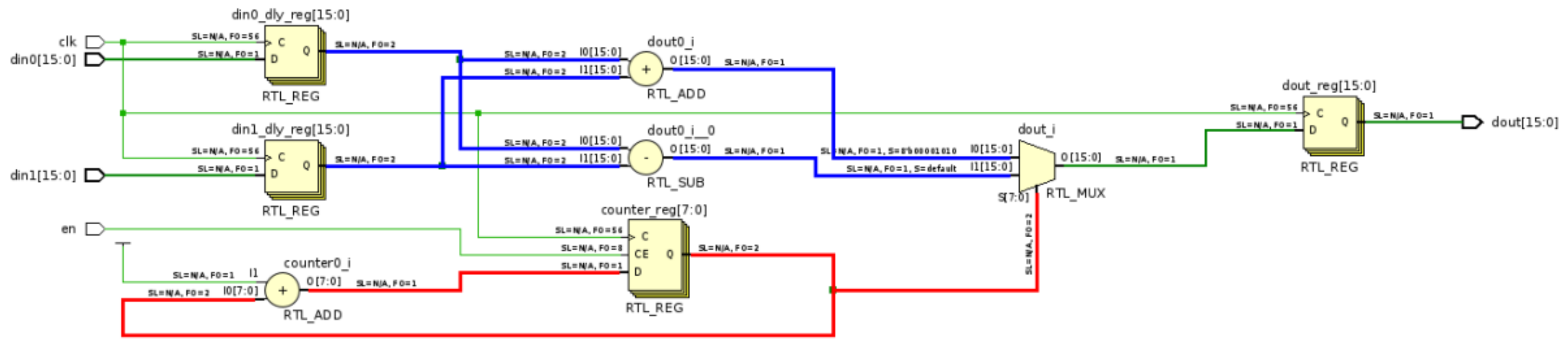


```
reg [15:0] din0_dly;
reg [15:0] din1_dly;
reg [255:0] counter = {255'b0,1'b1};

always@(posedge clk)
begin
   if(en)
     counter <= {counter[254:0],counter[255]};
end

always@(posedge clk)
begin
   din0_dly <= din0;
   din1_dly <= din1;
   if(counter[144] == 1'b1)
      dout <= din0_dly + din1_dly;
   else
      dout <= din0_dly - din1_dly;
end

endmodule
```



Version	LUTs	FFs	WNS	Critical Path
Original	24	56	-0.298 ns	FF -> LUT -> 2 LOOKAHEADs -> FF
Modified	24	49	0.232 ns	FF -> 2 LOOKAHEADs -> FF

Device : xcvc1902-vsva2197-1LP-i-S
Frequency : 500 MHz

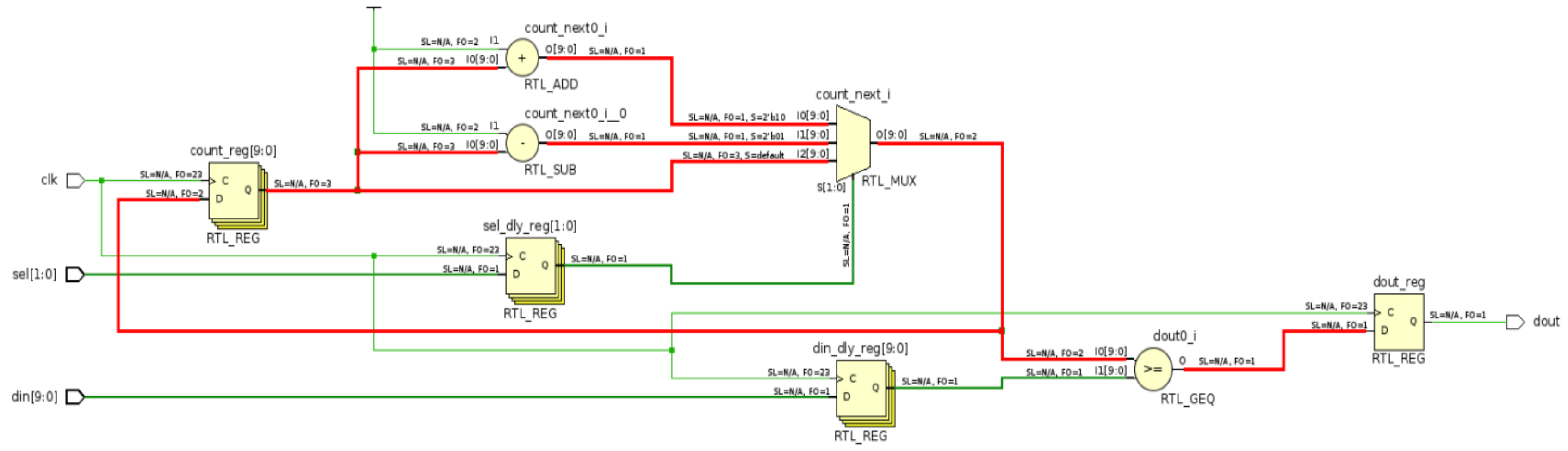
Optimizing Logical Comparisons 1/2

- Comparing two bit vectors: **count** and **din**
  - Is count >= din?
  - Critical path: add-sub -> 3-input mux -> comparator
- How can the critical path be improved?

```
reg [1:0] sel_dly;
reg [9:0] din_dly;
reg [9:0] count_next;
reg [9:0] count;

always@(*)
begin
   case(sel_dly)
     2'b10 : count_next = count + 1;
     2'b01 : count_next = count - 1;
     default : count_next = count;
   endcase
end

always@(posedge clk)
begin
   sel_dly <= sel;
   din_dly <= din;
   count <= count_next;
   dout <= count_next >= din_dly;
end
```

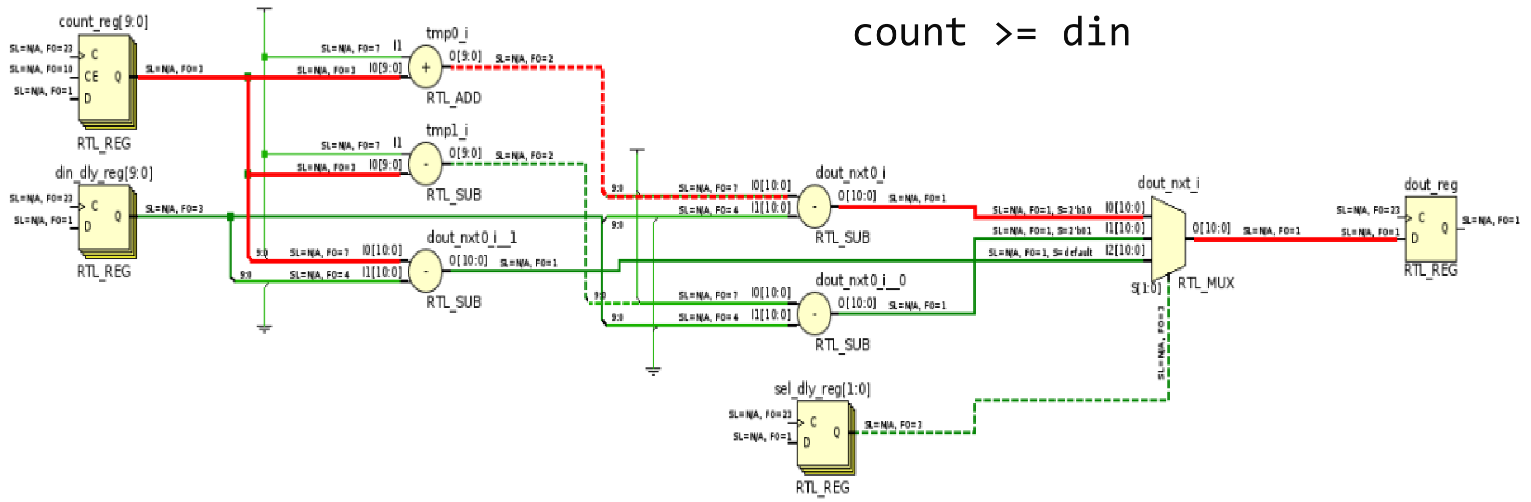


Device : xcvc1902-vsva2197-1LP-i-S
Frequency : 500 MHz

Optimizing Logical Comparisons 2/2

- Trick: move final comparison before 3-input mux
  - 3 comparators in parallel, area tradeoff

```
count_next >= din   ➡   count + 1 >= din
                        count - 1 >= din
                        count >= din
```



```verilog
reg [1:0] sel_dly;
reg [9:0] din_dly;
reg [9:0] count_next;
reg [9:0] count;
reg [10:0] dout_nxt;

always@(*)
begin
   case(sel_dly)
     2'b10 : count_next = count + 1;
     2'b01 : count_next = count - 1;
     default : count_next = count;
   endcase
end

wire [9:0] tmp0;
wire [9:0] tmp1;
assign tmp0 = count + 1;
assign tmp1 = count - 1;

always@(*)
begin
   case(sel_dly)
     2'b10 : dout_nxt = {1'b1,tmp0} - {1'b0,din_dly};
     2'b01 : dout_nxt = {1'b1,tmp1} - {1'b0,din_dly};
     default : dout_nxt = {1'b1,count} - {1'b0,din_dly};
   endcase
end

always@(posedge clk)
begin
   sel_dly <= sel;
   din_dly <= din;
   count <= count_next;
   dout <= dout_nxt[10];
end
```

Version	LUTs	FFs	WNS	Critical Path
Original	25	23	-0.501 ns	FF -> LOOKAHEAD -> LUT -> 2 LOOKAHEADs -> FF
Modified	42	23	0.311 ns	FF -> 2 LUTS -> LOOKAHEADs -> FF

Key Synthesis Takeaways

- New features help you use device resources more efficiently
  - Logic Compaction for low-precision arithmetic
  - RAM_STYLE = mixed for heterogeneous RAM mapping

- Remember key differences when migrating to Versal
  - Versal has more UltraRAM capabilities, fewer BRAM options
  - Versal DSP block natively supports complex multiplication and dot products

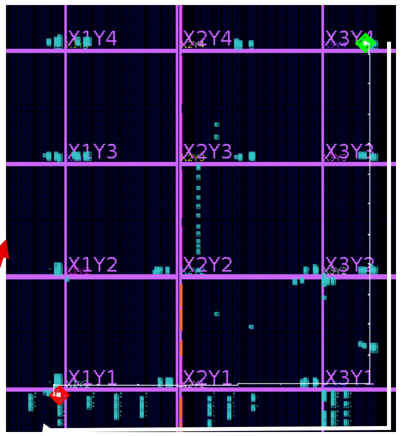
- To improve critical paths, look at the Elaborated Design and think of ways to improve the data flow

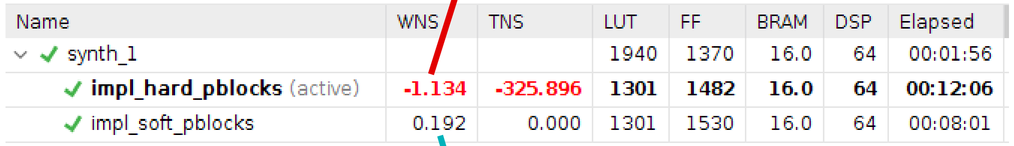
Key Implementation Features (2020.X)

Pblocks are Treated as Soft By Default

- Soft Pblocks have been supported since Vivado 2019.1
  - In 2020.2 Pblocks are treated as Soft by default and are honored until Physical-Synthesis-In-Placer in Global Placement
  - Reduces need to update pblocks when changing design
  - Reduces pblock restriction on congestion handling in rest of placer flow
  - Allows all physical optimizations (PSIP, phys_opt_design)
- DFX parent & child pblocks = hard by default
  - HD.ISOLATED
  - HD.RECONFIGURABLE
  - HD.TANDEM
  - HD.TANDEM_IP_PBLOCK
  - HD.RECONFIGURABLE_CONTAINER
- User constraint IS_SOFT=FALSE carried forward in DCPs from previous Vivado releases when loaded in Vivado 2020.2

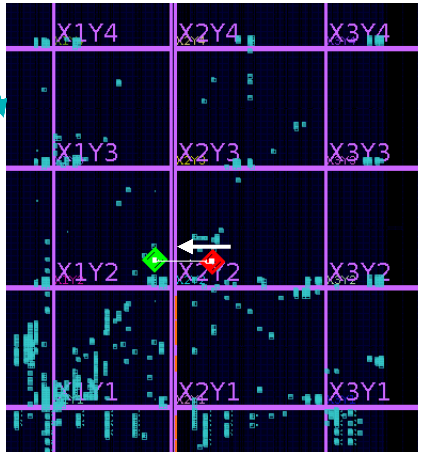
Hard Pblocks





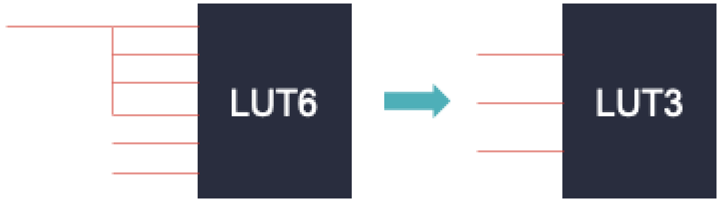
Name	WNS	TNS	LUT	FF	BRAM	DSP	Elapsed
✓ synth_1			1940	1370	16.0	64	00:01:56
✓ **impl_hard_pblocks** (active)	-1.134	-325.896	**1301**	**1482**	**16.0**	**64**	**00:12:06**
✓ impl_soft_pblocks	0.192	0.000	1301	1530	16.0	64	00:08:01

Soft Pblocks

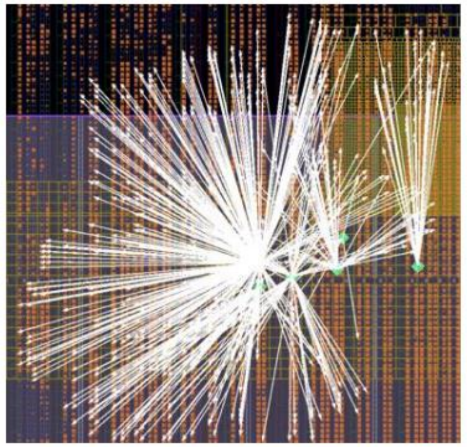


Physical-Synthesis-In-Placer (PSIP) Improvements

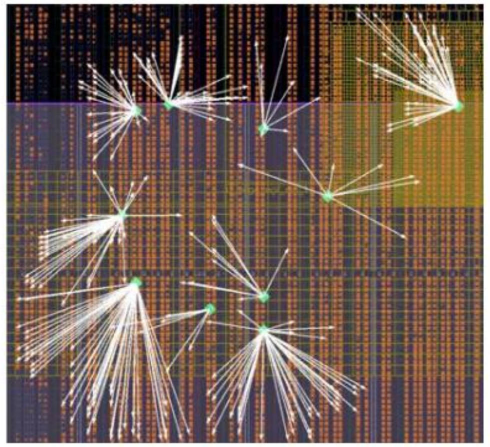
- Equivalent Driver Re-wire Optimization
  - Loads are redistributed between logically-equivalent drivers based on their placements
  - Helps reduce routing resource utilization and congestion
- After rewiring it is possible that some inputs of a LUT are connected to the same net and LUT reduction can result



Before



After

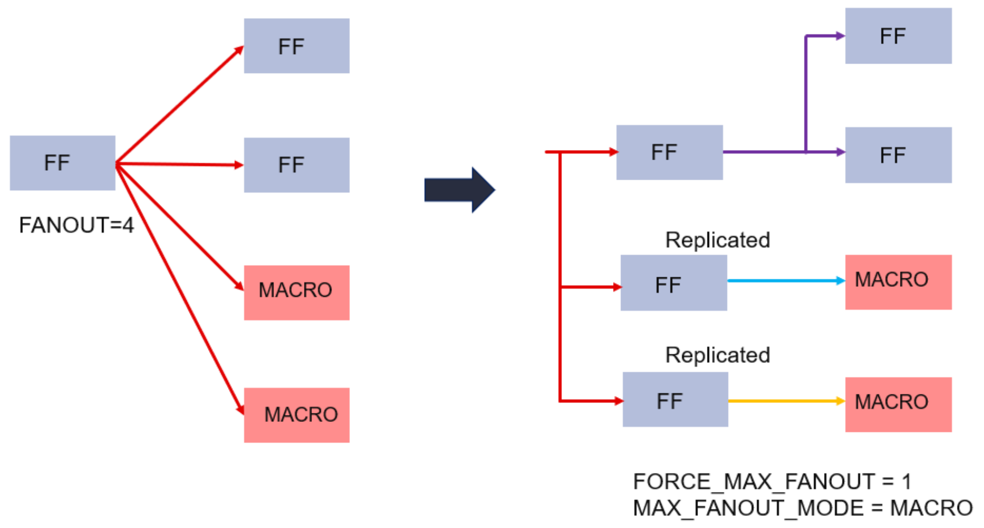


```
Summary of Physical Synthesis Optimizations
============================================

---------------------------------------------------------------------------------------------------------------------------------------------------
| Optimization                                   | WNS Gain (ns) | TNS Gain (ns) | Added Cells | Removed Cells | Optimized Cells/Nets | Dont
---------------------------------------------------------------------------------------------------------------------------------------------------
| LUT Combining                                  |         0.000 |         6.524 |           0 |          2255 |                 2255 |            0 |           1 | 00:00:07 |
| Equivalent Driver Rewiring                     |         0.000 |      1120.810 |        2185 |          4271 |                  121 |            3 |           1 | 00:00:55 |
```

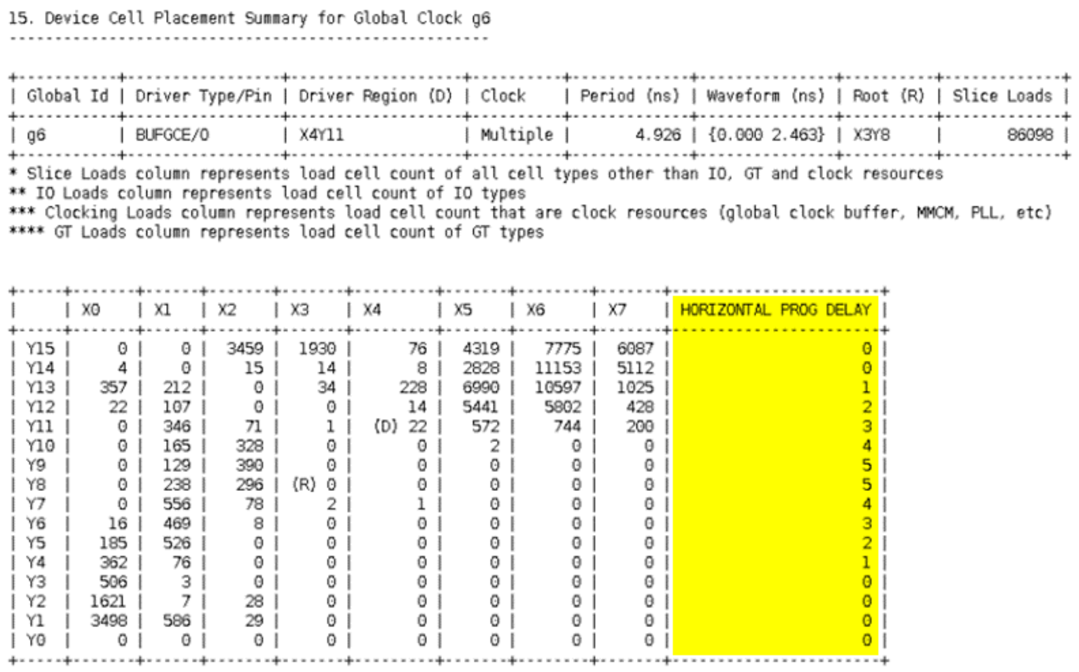
PSIP Replication Properties

- MAX_FANOUT_MODE and FORCE_MAX_FANOUT allow user to direct replication in PSIP
  - Works for FF and LUT
- For replication of drivers with far-apart loads
  - MAX_FANOUT_MODE values
    - MACRO (Block RAM, UltraRAM, DSP)
    - CLOCK_REGION
    - SLR



MAX_PROG_DELAY Capped For SLR Crossing Performance

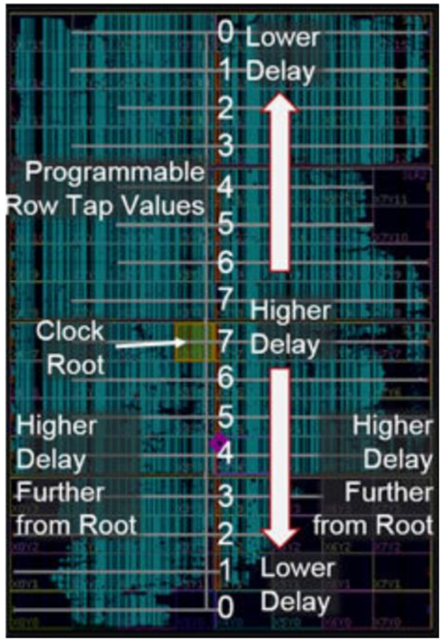
- Placer limits MAX_PROG_DELAY for UltraScale+ devices
  - Minimizes clock skew on SLR crossing when balancing clock network delays
  - USER_MAX_PROG_DELAY property allows user to cap delays further if required
- Clock Utilization Report shows programmable delays used for each clock



```
15. Device Cell Placement Summary for Global Clock g6
-----------------------------------------------------

+------------+-----------------+--------------------+----------+-------------+-----------------+----------+-------------+
| Global Id  | Driver Type/Pin | Driver Region (D)  | Clock    | Period (ns) | Waveform (ns)   | Root (R) | Slice Loads |
+------------+-----------------+--------------------+----------+-------------+-----------------+----------+-------------+
| g6         | BUFGCE/O        | X4Y11              | Multiple |       4.926 | {0.000 2.463}   | X3Y8     |       86098 |
+------------+-----------------+--------------------+----------+-------------+-----------------+----------+-------------+
* Slice Loads column represents load cell count of all cell types other than IO, GT and clock resources
** IO Loads column represents load cell count of IO types
*** Clocking Loads column represents load cell count that are clock resources (global clock buffer, MMCM, PLL, etc)
**** GT Loads column represents load cell count of GT types


+-----+-------+------+-------+-------+----------+-------+-------+-------+------------------------+
|     | X0    | X1   | X2    | X3    | X4       | X5    | X6    | X7    | HORIZONTAL PROG DELAY  |
+-----+-------+------+-------+-------+----------+-------+-------+-------+------------------------+
| Y15 |     0 |    0 |  3459 |  1930 |       76 |  4319 |  7775 |  6087 |                      0 |
| Y14 |     4 |    0 |    15 |    14 |        8 |  2828 | 11153 |  5112 |                      0 |
| Y13 |   357 |  212 |     0 |    34 |      228 |  6990 | 10597 |  1025 |                      1 |
| Y12 |    22 |  107 |     0 |     0 |       14 |  5441 |  5802 |   428 |                      2 |
| Y11 |     0 |  346 |    71 |     1 |   (D) 22 |   572 |   744 |   200 |                      3 |
| Y10 |     0 |  165 |   328 |     0 |        0 |     2 |     0 |     0 |                      4 |
| Y9  |     0 |  129 |   390 |     0 |        0 |     0 |     0 |     0 |                      5 |
| Y8  |     0 |  238 |   296 | (R) 0 |        0 |     0 |     0 |     0 |                      5 |
| Y7  |     0 |  556 |    78 |     2 |        1 |     0 |     0 |     0 |                      4 |
| Y6  |    16 |  469 |     8 |     0 |        0 |     0 |     0 |     0 |                      3 |
| Y5  |   185 |  526 |     0 |     0 |        0 |     0 |     0 |     0 |                      2 |
| Y4  |   362 |   76 |     0 |     0 |        0 |     0 |     0 |     0 |                      1 |
| Y3  |   506 |    3 |     0 |     0 |        0 |     0 |     0 |     0 |                      0 |
| Y2  |  1621 |    7 |    28 |     0 |        0 |     0 |     0 |     0 |                      0 |
| Y1  |  3498 |  586 |    29 |     0 |        0 |     0 |     0 |     0 |                      0 |
| Y0  |     0 |    0 |     0 |     0 |        0 |     0 |     0 |     0 |                      0 |
+-----+-------+------+-------+-------+----------+-------+-------+-------+------------------------+
```



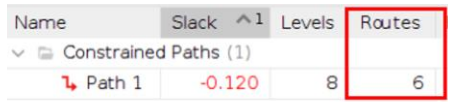
Versal Implementation Guidelines

Versal Changes to Fabric

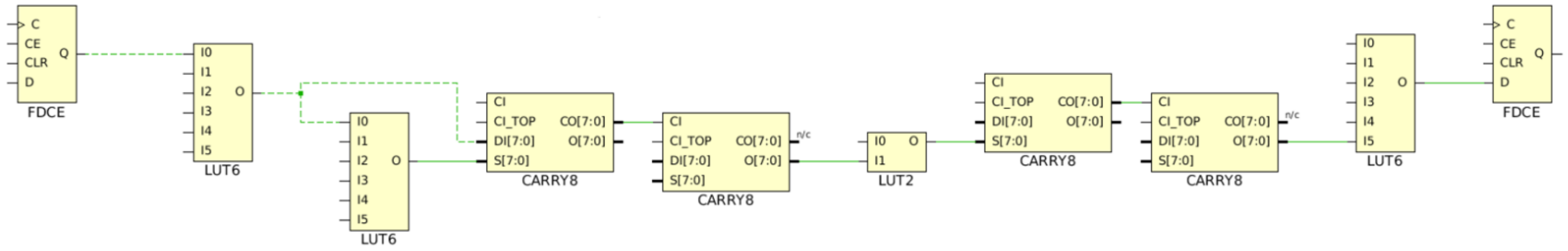
- Versal has a Uniform fabric
  - Half of LUTs in every CLB are LUTRAM/SRL capable
  - Even Block RAM & UltraRAM distribution
- Simplified CLB architecture with fast LUT cascade
  - No F7/F8/F9 muxes
  - CARRY8 replaced by LOOKAHEAD8 and LUTCY
  - Fast LUT cascade
  - More LUT combining options
- To take full advantage of the architectural changes need to re-synthesize
  - Remove instantiated legacy primitives and synthesis attributes
  - Re-targeting prior architecture netlist will result in sub-optimal implementation

Comparing CARRY8 vs. LOOKAHEAD8/LUTCY

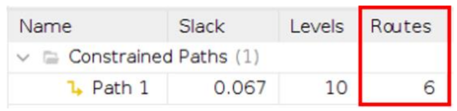
- UltraScale+ uses 8 logic levels, 6 routes with CARRY8s
  - Datapath delay = 3.822 ns



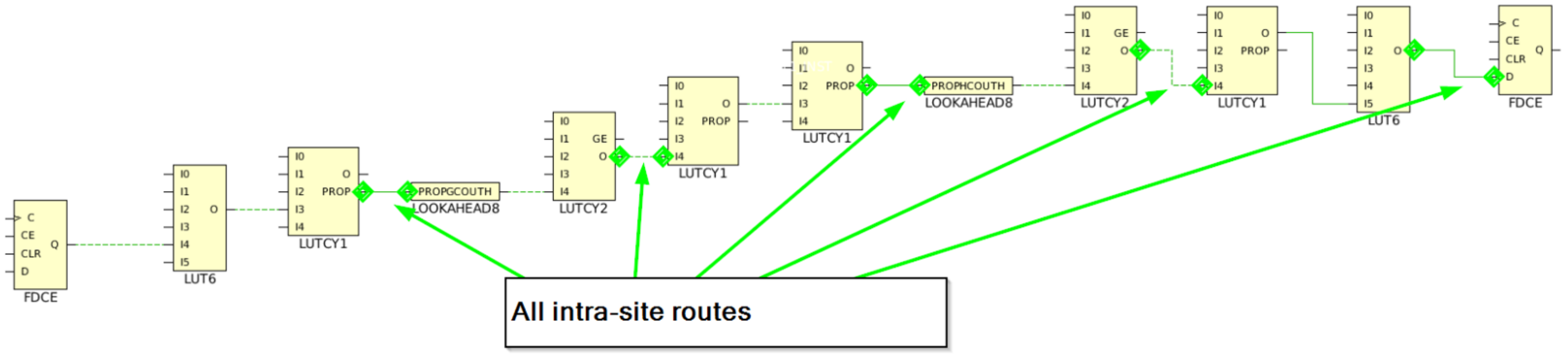
Name	Slack ^1	Levels	Routes
Constrained Paths (1)			
Path 1	-0.120	8	6



- Versal uses 10 logic levels but still only 6 routes with LOOKAHEAD8s
  - Datapath delay = 3.635 ns



Name	Slack	Levels	Routes
Constrained Paths (1)			
Path 1	0.067	10	6

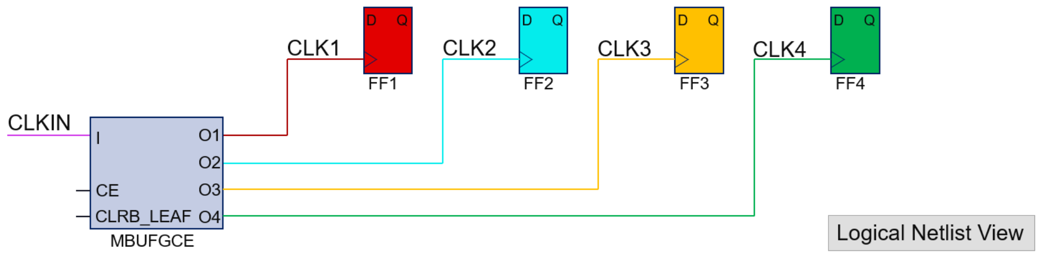


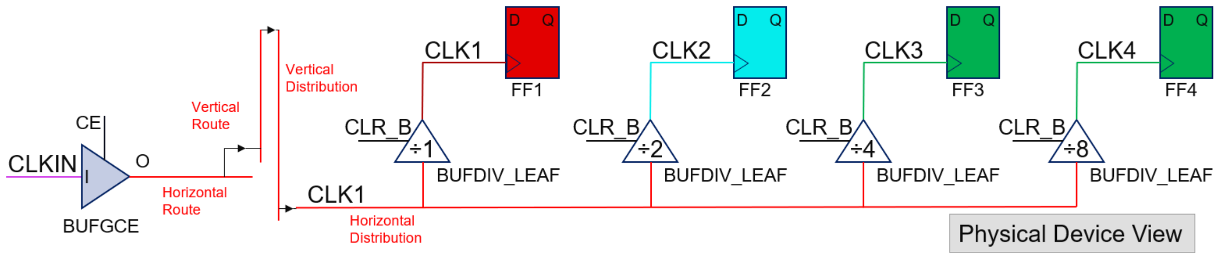
```
wr_ptr <= resize(dol(5 downto 3) * unsigned(img_size_x), log2(C_MAX_LINE_WIDTH*C_NUM_LINES)) +
resize(dol(15 downto 6) * 8, log2(C_MAX_LINE_WIDTH*C_NUM_LINES));
```

Above results require re-synthesis of RTL

Versal Multi-Clock Buffer - MBUFG

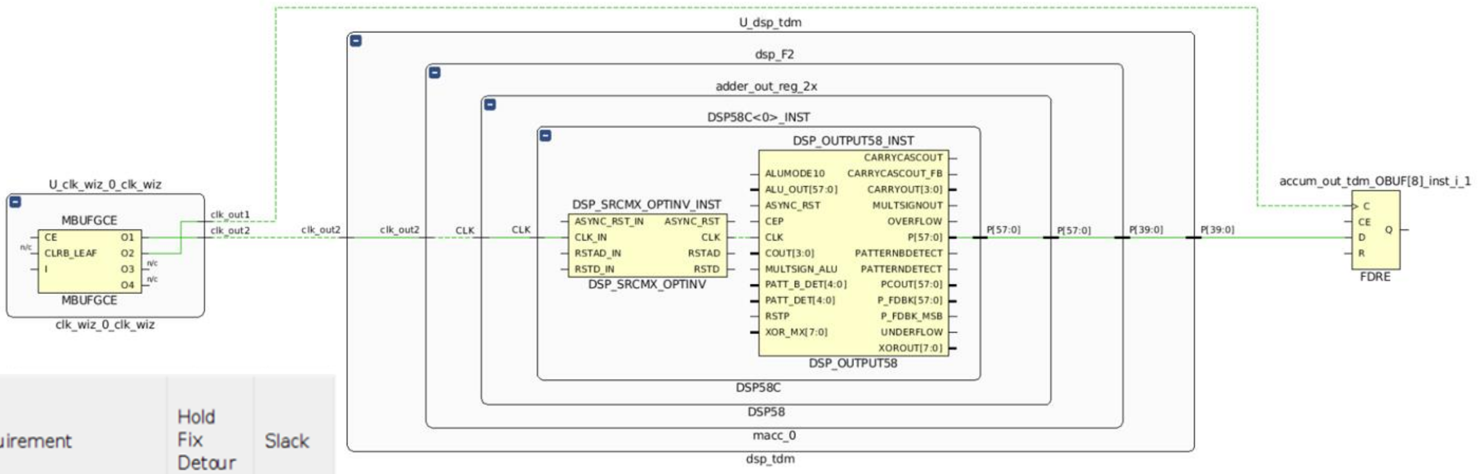
- Versal supports a new Multi-Clock Buffer (MBUFG) that generates up to 4 output clocks from a single input clock
  - Output clocks are /1, /2, /4, and /8 versions of input clock
- MBUFG versions exist for BUFGCE, BUFGCE_DIV, BUFG_PS, BUFG_GT and BUFGCTRL
- MBUFG is a logical buffer with 4 outputs (O1, O2, O3, O4)
  - Physical implementation uses BUFG* and BUFDIV_LEAF leaf clock dividers
    - BUFDIV_LEAF buffers are driven by horizontal clock distribution and are the final clock buffer for fabric loads (CLB, DSP, BRAM, URAM) and most hard-IP blocks (GTYP_QUAD, MRMAC, etc.)

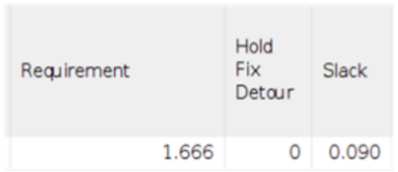




Versal MBUFG For Synchronous CDC

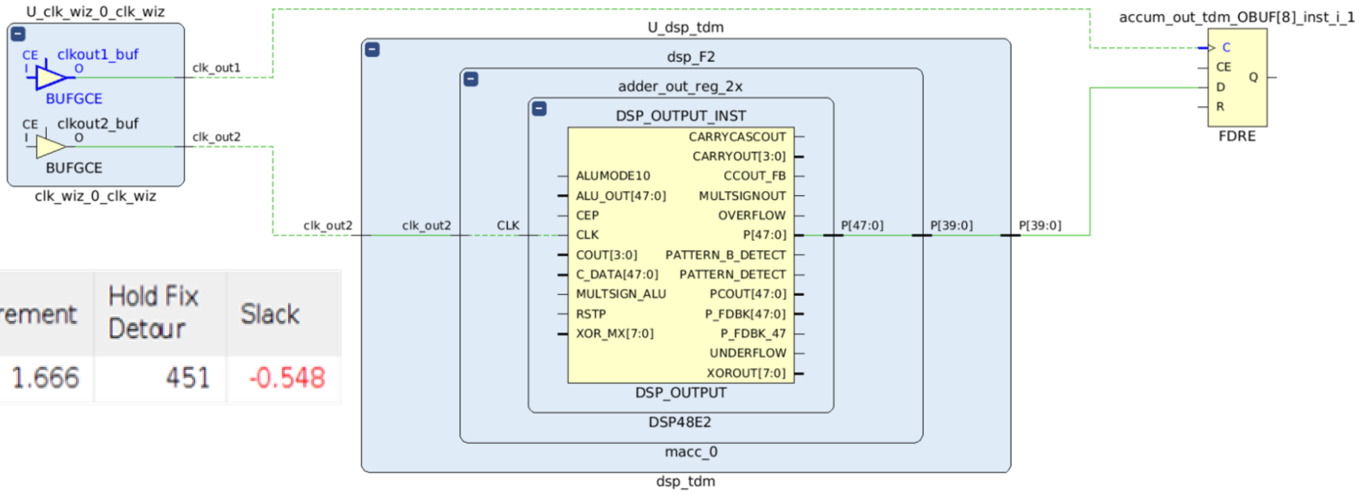
- MBUFGCE
  - Common node closer to path
    - Inter Clock Skew ~ 0.174ns
    - Inter Clock FMAX > 600 MHz

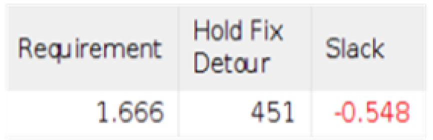




Requirement	Hold Fix Detour	Slack
1.666	0	0.090

- Parallel BUFGCE
  - Common node at driver
    - Can be far away if driver in XPIO clock region or GT Column
  - Inter Clock Skew > 0.500ns
  - Inter Clock FMAX < 500 MHz





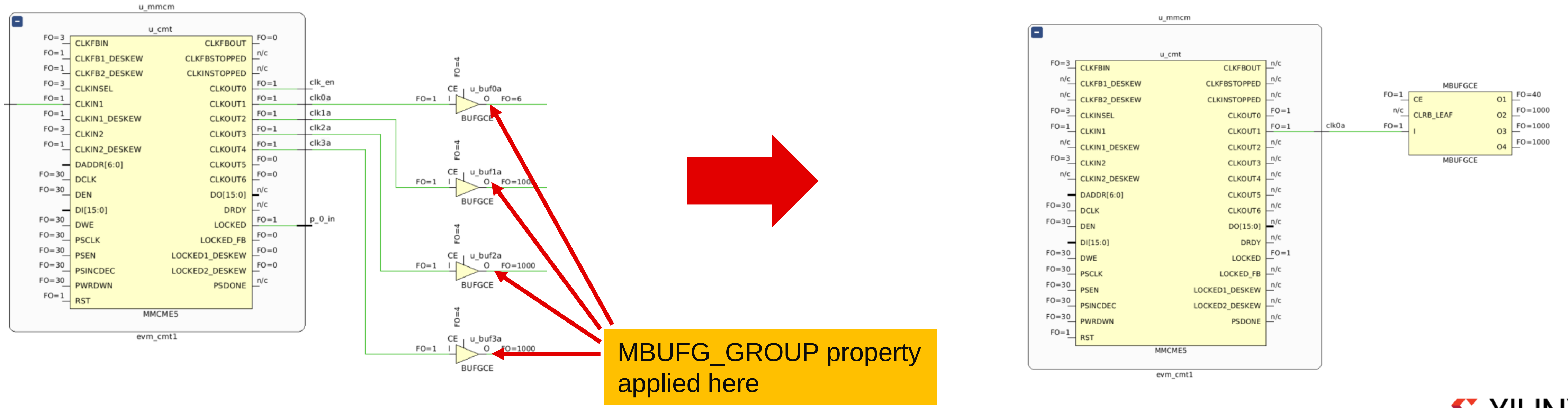
Requirement	Hold Fix Detour	Slack
1.666	451	-0.548

MBUFG Transform In Logical Optimization Phase

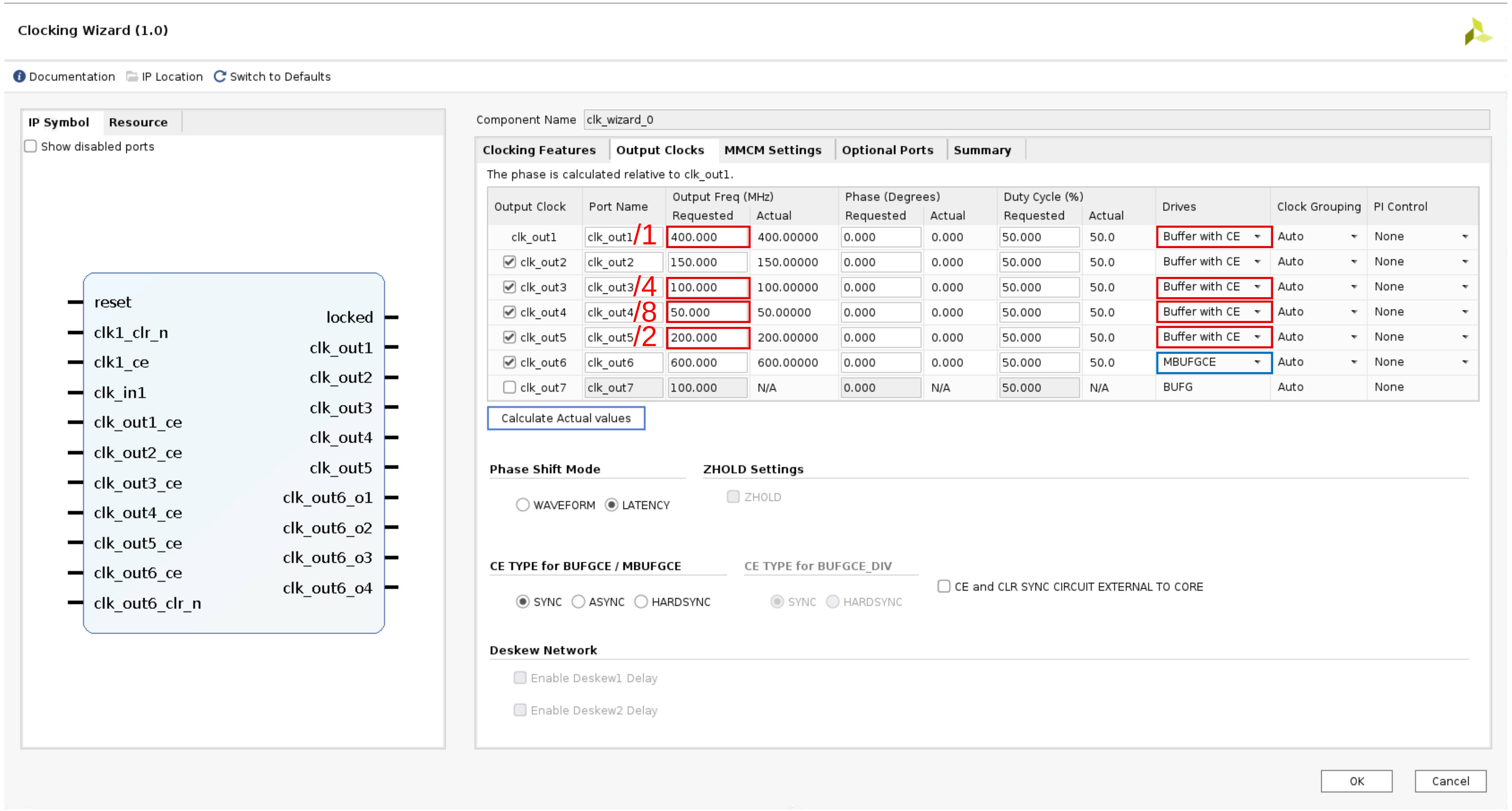
- opt_design –mbufg_opt for global transformation of parallel BUFG -> MBUFG
- MBUFG_GROUP property allows for targeted transformation
  - Set precedents over which BUFG* should get converted to MBUFG*

```
set_property MBUFG_GROUP group1 [get_nets -of [get_pins {u_buf0a/O u_buf1a/O u_buf2a/O u_buf3a/O }]]
```

- Transformations are prevented if timing constraints could result in mismatch



Clocking Wizard Support for MBUFG



Clocking Wizard (1.0)

Documentation | IP Location | Switch to Defaults

IP Symbol | Resource

☐ Show disabled ports

Component Name: clk_wizard_0

Clocking Features | Output Clocks | MMCM Settings | Optional Ports | Summary

The phase is calculated relative to clk_out1.

Output Clock	Port Name	Output Freq (MHz)		Phase (Degrees)		Duty Cycle (%)		Drives	Clock Grouping	PI Control
		Requested	Actual	Requested	Actual	Requested	Actual			
clk_out1	clk_out1 /1	400.000	400.00000	0.000	0.000	50.000	50.0	Buffer with CE	Auto	None
☑ clk_out2	clk_out2	150.000	150.00000	0.000	0.000	50.000	50.0	Buffer with CE	Auto	None
☑ clk_out3	clk_out3 /4	100.000	100.00000	0.000	0.000	50.000	50.0	Buffer with CE	Auto	None
☑ clk_out4	clk_out4 /8	50.000	50.00000	0.000	0.000	50.000	50.0	Buffer with CE	Auto	None
☑ clk_out5	clk_out5 /2	200.000	200.00000	0.000	0.000	50.000	50.0	Buffer with CE	Auto	None
☑ clk_out6	clk_out6	600.000	600.00000	0.000	0.000	50.000	50.0	MBUFGCE	Auto	None
☐ clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	Auto	None

Calculate Actual values

Phase Shift Mode

WAVEFORM [✓ LATENCY]

ZHOLD Settings

☐ ZHOLD

CE TYPE for BUFGCE / MBUFGCE

[✓ SYNC] ASYNC HARDSYNC

CE TYPE for BUFGCE_DIV

SYNC HARDSYNC

☐ CE and CLR SYNC CIRCUIT EXTERNAL TO CORE

Deskew Network

☐ Enable Deskew1 Delay

☐ Enable Deskew2 Delay

OK | Cancel

Inferred MBUFGCE

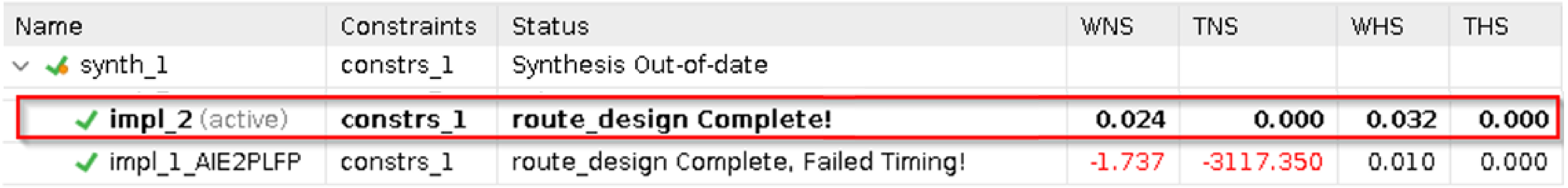
1. Select Output Frequencies that are /1, /2, /4, /8
2. Select "Buffer" or "Buffer with CE" as clock driver

Instantiated MBUFGCE

1. Select "MBUFGCE" as clock driver

Real World Example Of QoR Improvement with MBUFG

- WNS went from -1.737ns to timing closed!
  - impl_1_AIE2PLFP WNS=-1.737ns
    - Default strategy implementation
  - impl_2 WNS=0.024ns
    - Default strategy implementation
    - MBUFG transform using MBUFG_GROUP constraint



Name	Constraints	Status	WNS	TNS	WHS	THS
synth_1	constrs_1	Synthesis Out-of-date				
impl_2 (active)	**constrs_1**	**route_design Complete!**	**0.024**	**0.000**	**0.032**	**0.000**
impl_1_AIE2PLFP	constrs_1	route_design Complete, Failed Timing!	-1.737	-3117.350	0.010	0.000

Top Takeaways

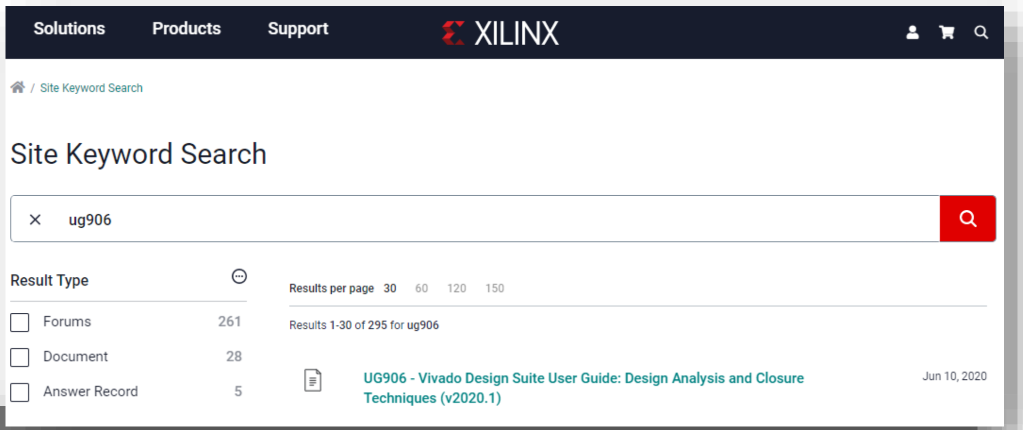
- Run methodology reports, review and fix critical violations and warnings
- Synthesis has many options to drive improved results. In addition, you can develop your own bag of tricks to fine tune critical logic
- Vivado placement has very comprehensive replication to improve QoR, both automatic and user-driven
- Versal architecture brings many improvements over prior architectures, be aware of key differences
  - Re-synthesize for optimal results, and recode if necessary
  - Take advantage of new capabilities like MBUFG, URAM initialization, DSP complex and dot-product modes

XILINX.

Where to Find More Information

- User Guides on xilinx.com
  - UG901 - Synthesis
  - UG904 - Implementation
  - UG906 - Design Analysis & Closure
  - UG949 - UltraFast Methodology

- Xilinx Community Forums
  - Vivado RTL Development
  - Blogs: Design and Debug Techniques



Site Keyword Search

ug906

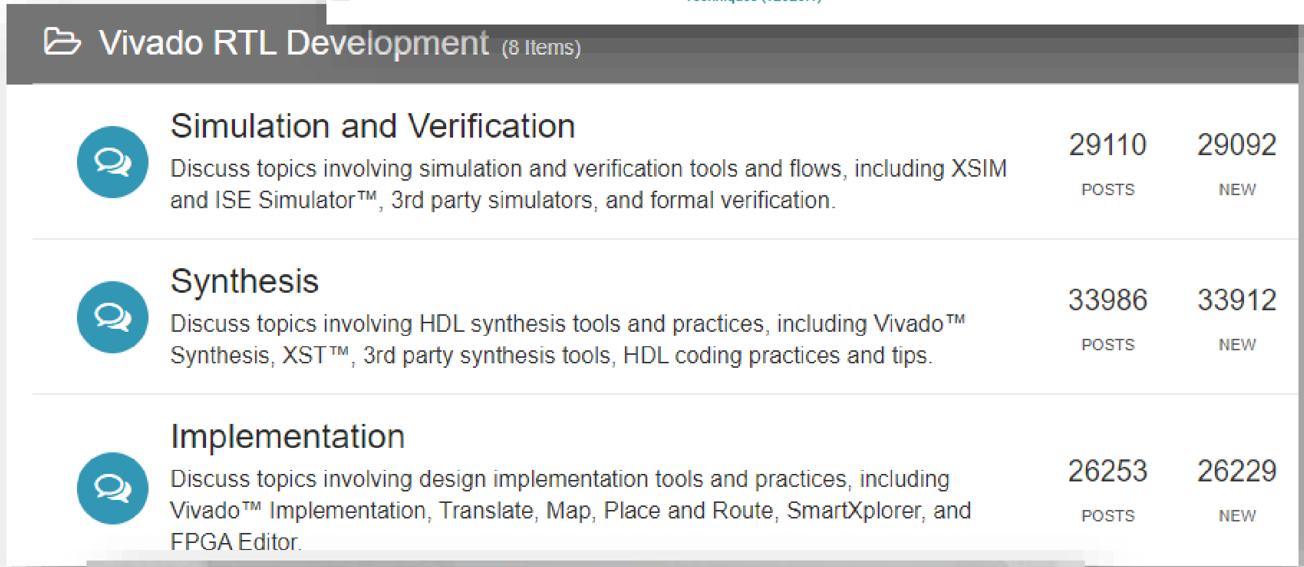
Result Type

Forums	261
Document	28
Answer Record	5

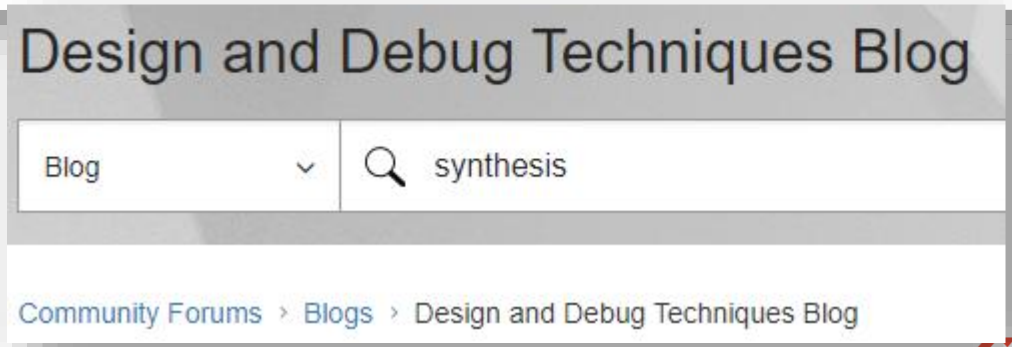
Results per page 30 60 120 150

Results 1-30 of 295 for ug906

UG906 - Vivado Design Suite User Guide: Design Analysis and Closure Techniques (v2020.1) Jun 10, 2020


Vivado RTL Development (8 Items)

Simulation and Verification Discuss topics involving simulation and verification tools and flows, including XSIM and ISE Simulator™, 3rd party simulators, and formal verification.	29110 POSTS	29092 NEW
Synthesis Discuss topics involving HDL synthesis tools and practices, including Vivado™ Synthesis, XST™, 3rd party synthesis tools, HDL coding practices and tips.	33986 POSTS	33912 NEW
Implementation Discuss topics involving design implementation tools and practices, including Vivado™ Implementation, Translate, Map, Place and Route, SmartXplorer, and FPGA Editor.	26253 POSTS	26229 NEW


Design and Debug Techniques Blog

Community Forums › Blogs › Design and Debug Techniques Blog

XILINX®

Thank You