XDF XILINX DEVELOPER FORUM

Vivado Synthesis Tips & Tricks

Presented By

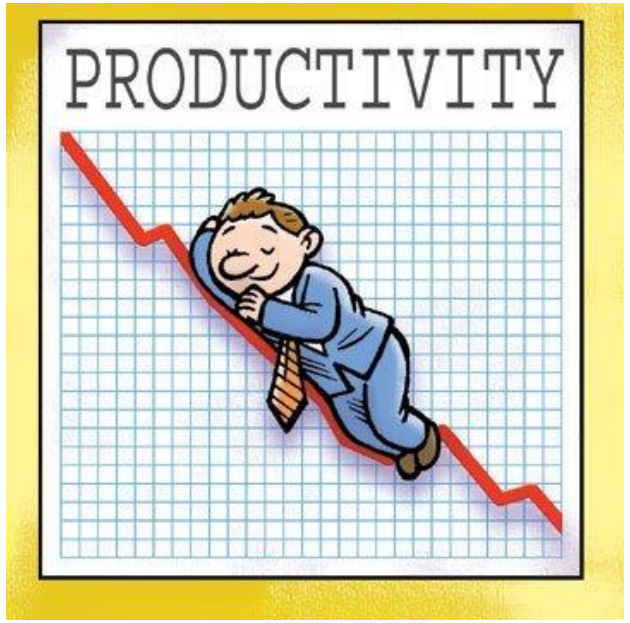
Balachander Krishnamurthy
Sr. Product Marketing Manager
October 2nd, 2018

XILINX

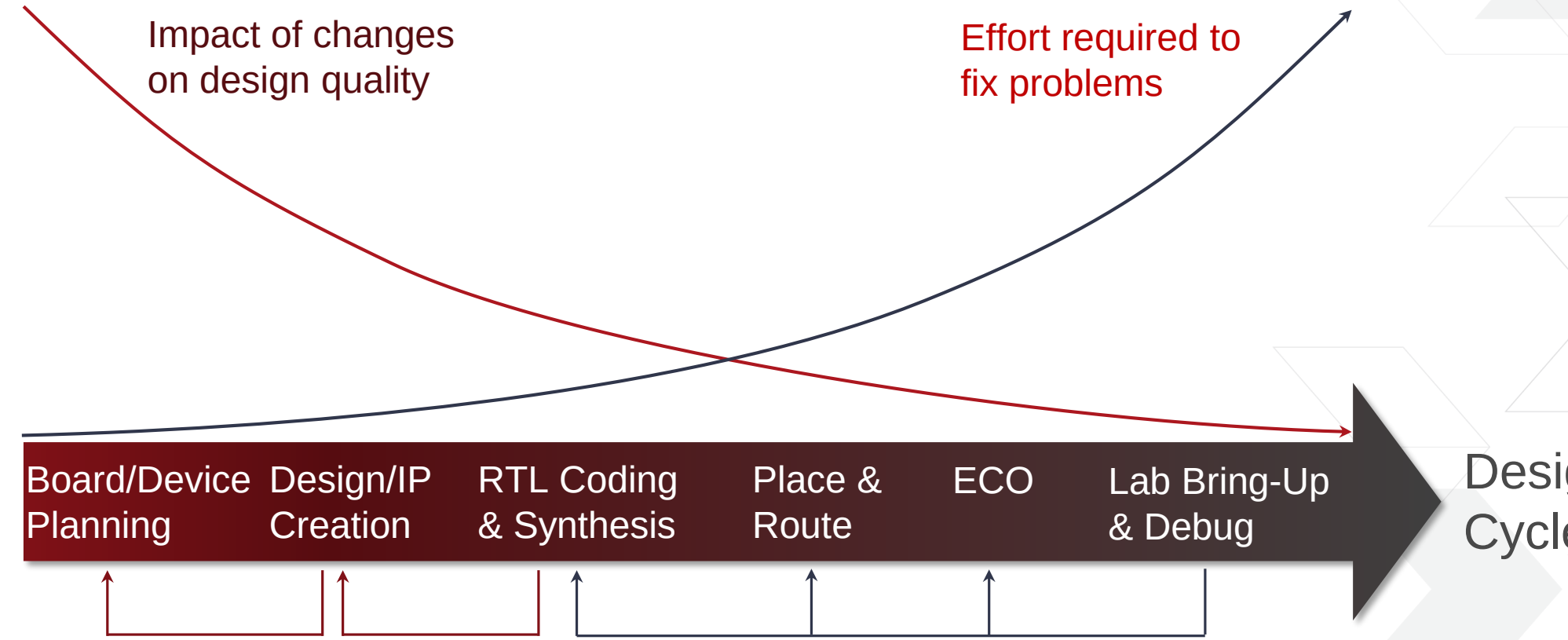
Topics for Today

- The UltraFast Design Methodology Philosophy
- UFDM: Customer Case Study
- Waiver Mechanism
- Vivado Incremental Synthesis
- QoR: Tips & Tricks

UltraFast Design Methodology Philosophy



UltraFast Design Methodology Philosophy



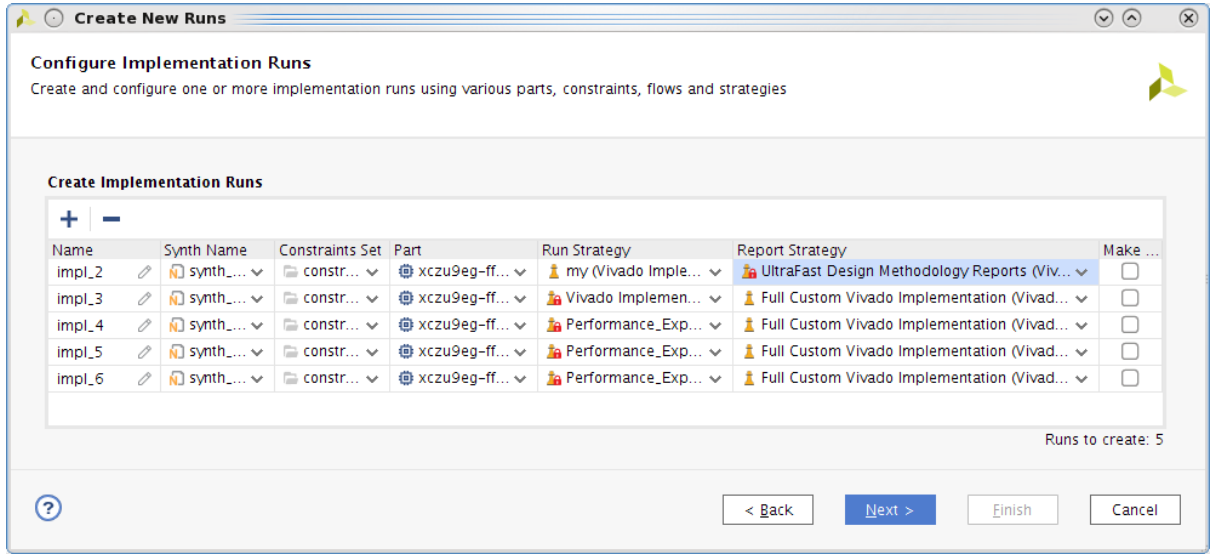
Earlier iterations:
Faster and more effective

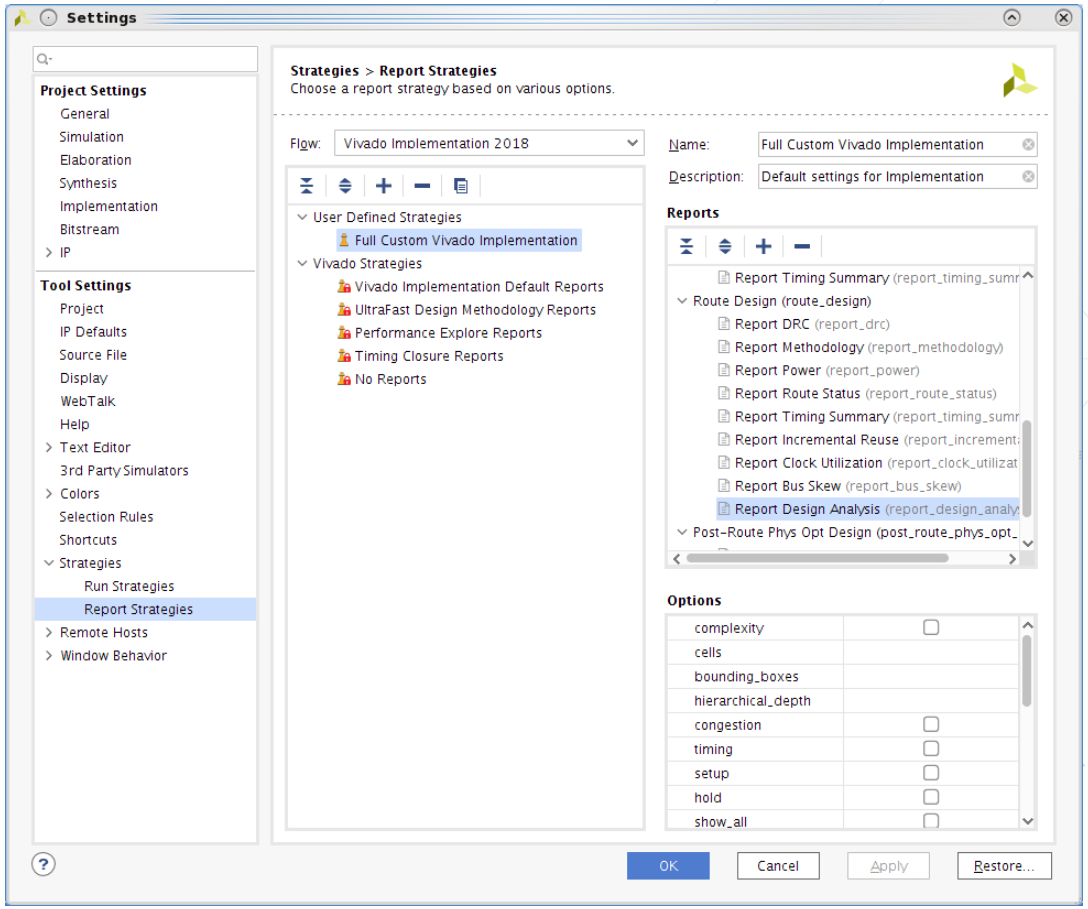
Later iterations:
Longer, difficult and less effective

Validate design at each stage, fix issues before proceeding to next stage

Report Run Strategies

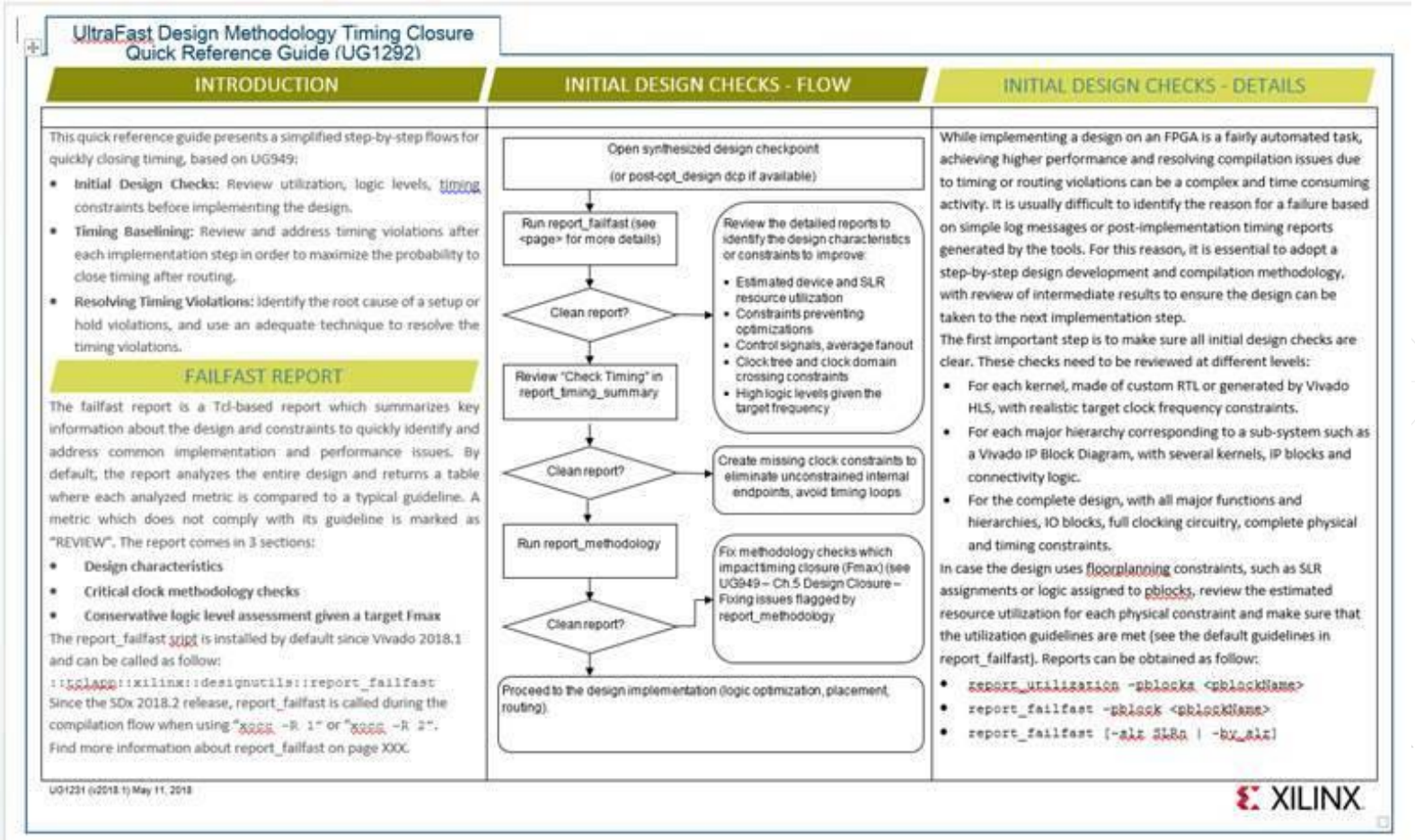
- **Create custom report strategies similar to custom run strategies**
- **Improve compile time**
  - Select which reports are generated for each run
  - Configure options for each report individually
- **Reuse report strategies across runs and projects**





NEW UG1292: UFDM Timing Closure Quick Reference Card

- **Step-by-step Analysis and Suggestions**
- **Address common timing closure challenges**
  - HLx and SDx
  - Project and Non-project



UltraFast Design Methodology Timing Closure Quick Reference Guide (UG1292)

INTRODUCTION

This quick reference guide presents a simplified step-by-step flows for quickly closing timing, based on UG949:

- **Initial Design Checks:** Review utilization, logic levels, timing constraints before implementing the design.
- **Timing Baselining:** Review and address timing violations after each implementation step in order to maximize the probability to close timing after routing.
- **Resolving Timing Violations:** Identify the root cause of a setup or hold violations, and use an adequate technique to resolve the timing violations.

FAILFAST REPORT

The failfast report is a Tcl-based report which summarizes key information about the design and constraints to quickly identify and address common implementation and performance issues. By default, the report analyzes the entire design and returns a table where each analyzed metric is compared to a typical guideline. A metric which does not comply with its guideline is marked as "REVIEW". The report comes in 3 sections:

- **Design characteristics**
- **Critical clock methodology checks**
- **Conservative logic level assessment given a target Fmax**

The report_failfast script is installed by default since Vivado 2018.1 and can be called as follow:

```
::tclapp::xilinx::designutils::report_failfast
```

Since the SDx 2018.2 release, report_failfast is called during the compilation flow when using "xocc -R 1" or "xocc -R 2".
Find more information about report_failfast on page XXX.

INITIAL DESIGN CHECKS - FLOW

Open synthesized design checkpoint
(or post-opt_design dcp if available)

Run report_failfast (see <page> for more details)

Clean report?

Review the detailed reports to identify the design characteristics or constraints to improve:
- Estimated device and SLR resource utilization
- Constraints preventing optimizations
- Control signals, average fanout
- Clock tree and clock domain crossing constraints
- High logic levels given the target frequency

Review "Check Timing" in report_timing_summary

Clean report?

Create missing clock constraints to eliminate unconstrained internal endpoints, avoid timing loops

Run report_methodology

Clean report?

Fix methodology checks which impact timing closure (Fmax) (see UG949 – Ch.5 Design Closure – Fixing issues flagged by report_methodology

Proceed to the design implementation (logic optimization, placement, routing)

INITIAL DESIGN CHECKS - DETAILS

While implementing a design on an FPGA is a fairly automated task, achieving higher performance and resolving compilation issues due to timing or routing violations can be a complex and time consuming activity. It is usually difficult to identify the reason for a failure based on simple log messages or post-implementation timing reports generated by the tools. For this reason, it is essential to adopt a step-by-step design development and compilation methodology, with review of intermediate results to ensure the design can be taken to the next implementation step.
The first important step is to make sure all initial design checks are clear. These checks need to be reviewed at different levels:

- For each kernel, made of custom RTL or generated by Vivado HLS, with realistic target clock frequency constraints.
- For each major hierarchy corresponding to a sub-system such as a Vivado IP Block Diagram, with several kernels, IP blocks and connectivity logic.
- For the complete design, with all major functions and hierarchies, IO blocks, full clocking circuitry, complete physical and timing constraints.

In case the design uses floorplanning constraints, such as SLR assignments or logic assigned to pblocks, review the estimated resource utilization for each physical constraint and make sure that the utilization guidelines are met (see the default guidelines in report_failfast). Reports can be obtained as follow:

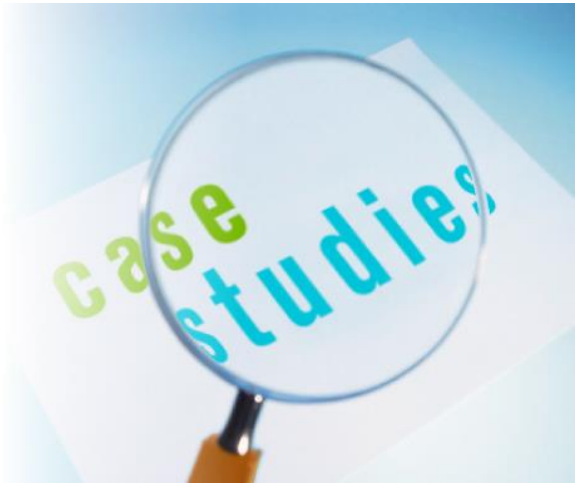
- `report_utilization -pblocks <pblockName>`
- `report_failfast -pblock <pblockName>`
- `report_failfast [-slr SLRn | -by_slr]`

UG1231 (v2018.1) May 11, 2018

XILINX

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1292-ultrafast-timing-closure-quick-reference.pdf

UFDM: Customer Case Study



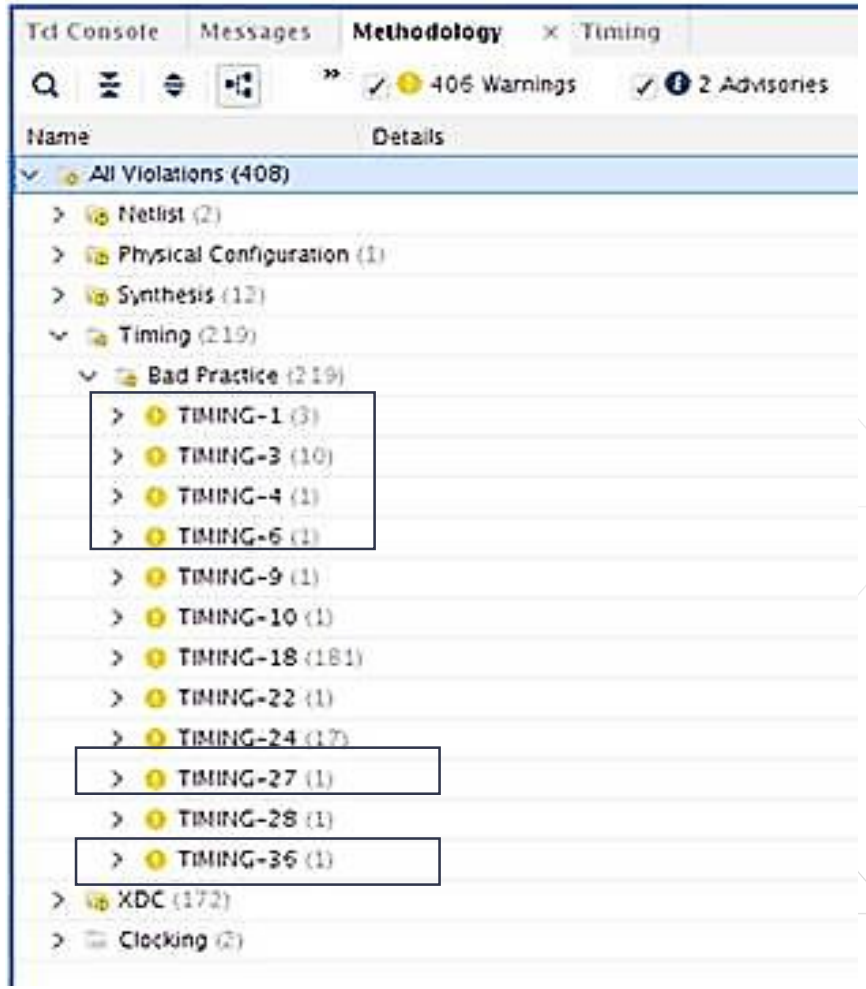
Customer Case Study: Design not functional

- **Major Xilinx customer with tight production deadline**
- **Customer claimed**
  - Running ‘place_design -fanout_opt’ caused functional issue
  - Adding ILA to DCP, design issue is gone
  - Not a CDC issue
- **OneSpin equivalency checking is clean**
  - opt_design DCP compared with place_design DCP
- **SR filed and escalated to factory**

Customer Case Study: Analysis by factory

> Many UltraFast Methodology Violations

- >> Timing -1 → Incorrect clock waveform
- >> Timing-3 → Breaking clock propagation delay and potentially skew accuracy
- >> Timing-6, 27 → Primary clock defined on hierarchical pin
- >> Timing-36 → Inaccurate skew due to missing insertion delay on a generated clock



Customer Case Study: Analysis by factory …2

- **Report CDC flagged ~10K Critical violations!**

Severity	ID	Count	Description
Critical	CDC-1	8823	1-bit unknown CDC circuitry
Critical	CDC-4	28	Multi-bit unknown CDC circuitry
Critical	CDC-7	335	Asynchronous reset unknown CDC circuitry
Critical	CDC-10	1089	Combinational logic detected before a synchronizer
Critical	CDC-11	247	Fan-out from launch flop to destination clock
Critical	CDC-13	681	1-bit CDC path on a non-FD primitive
Critical	CDC-14	8	Multi-bit CDC path on a non-FD primitive
Warning	CDC-2	478	1-bit synchronized with missing ASYNC_REG property

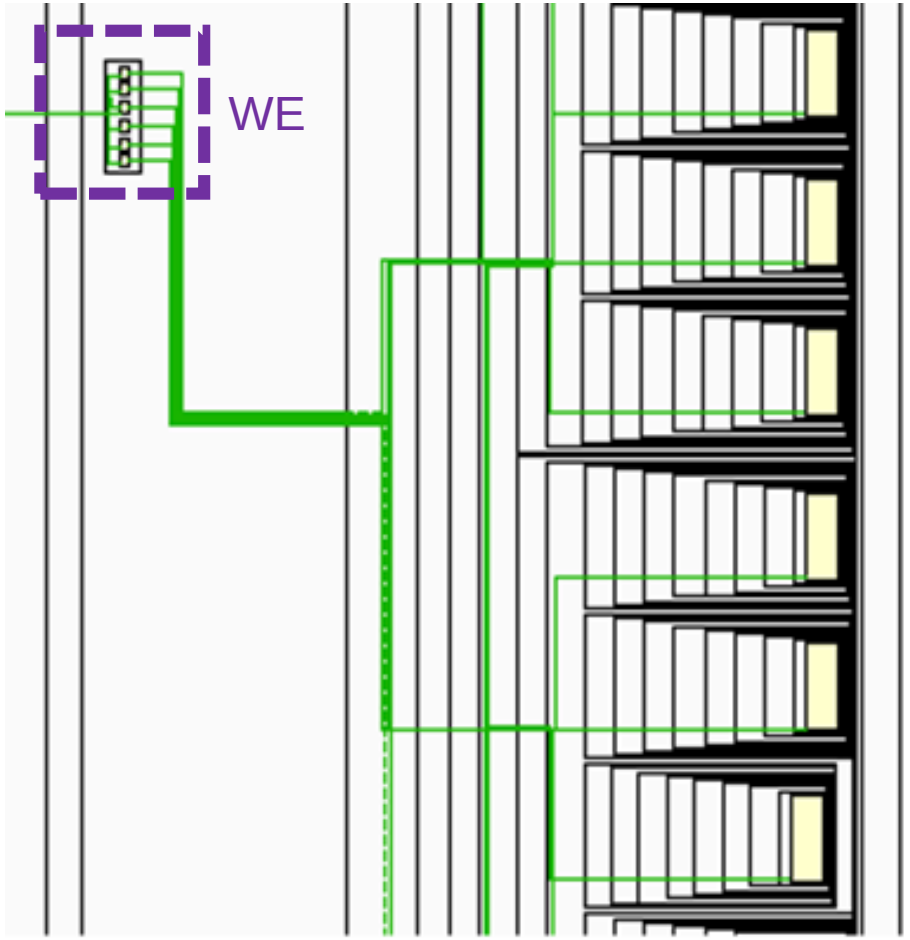
Waivers can help focus on new or un-reviewed issues

- **CDC-11 violations introduced by placer fanout opt**
  - User **allowed** **replication** of CDC endpoint (RAMB/WE control signal)

  => RAMBs written in different cycles

  Safe CDC topology would have prevented replication

- **Outcome**
  - Design working after addressing methodology and CDC violations



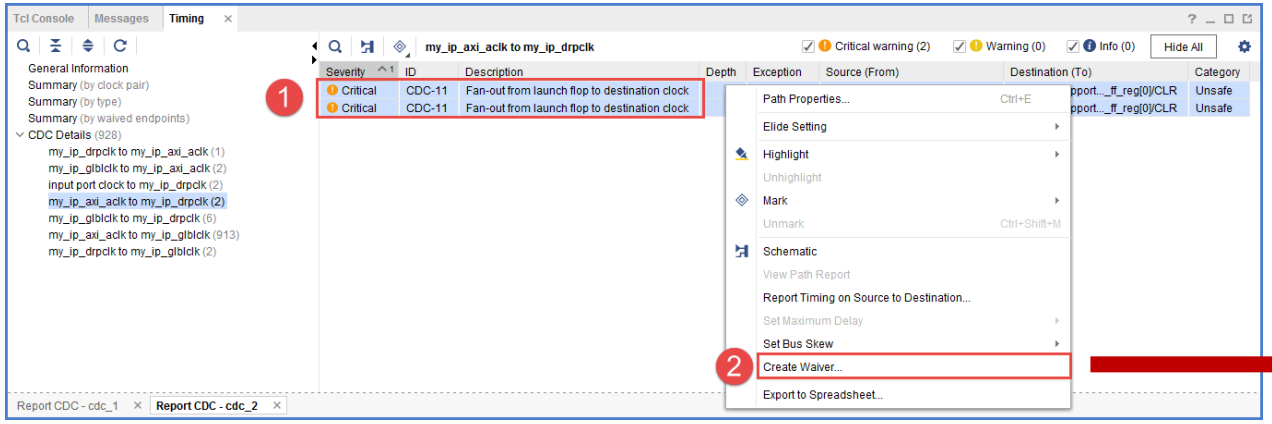
Waiver Mechanism

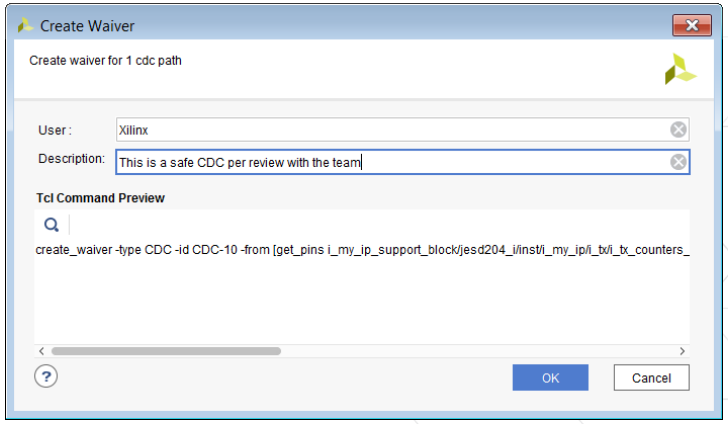
Waiver Mechanism

- **Hide violations in CDC/DRC/Methodology checks in the design**
  - Focus only on what is relevant
- **Waivers can be created, queried, reported against and deleted**
  - Track user, timestamp and description
  - Waivers should be reviewed by the design team
  - XDC Compatible, allows read/write and scoping
  - Duplicate waivers ignored
- **Recommend**
  - Don’t waive Critical violations
  - Waive Warning (after reviewing them) and Info types
- **Xilinx IPs have adopted waiver mechanism**
- **Documentation**
  - UG906: Design Analysis and Closure Techniques
  - UG938: Tutorial Design Analysis and Timing Closure (NEW)

Creating a Waiver

- **Create from: Report CDC / DRC / Methodology result window**





- **Create from: CDC / DRC / Methodology violation objects**

```
report_cdc -name cdc_1
foreach vio [get_cdc_violations -name cdc_1 -filter {CHECK == CDC-1}] {
  if {[regexp {^top/sync_1} [get_property STARTPOINT_PIN $vio]]} {
    create_waiver -of $vio -description {Safe by protocol}
  }
}
```

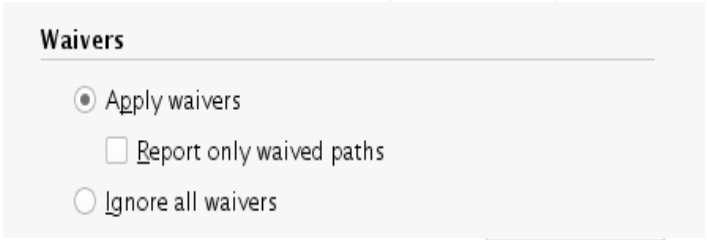
Notice: only **description** argument specified with this method.

- **Create from: manual specification of all arguments**
  - Arguments are order dependent. They must match order inside the violation object

Reporting Waivers

- **In Report CDC / DRC / Methodology GUI (and command line)**
  - Report can be generated with the waivers
  - Report can be generated by ignoring the waivers
  - Can report only waived violations

- **report_waivers**
  - Only Text Based
  - GUI Support coming soon
  - Report CDC/DRC/Methodology must be run prior to extract statistics



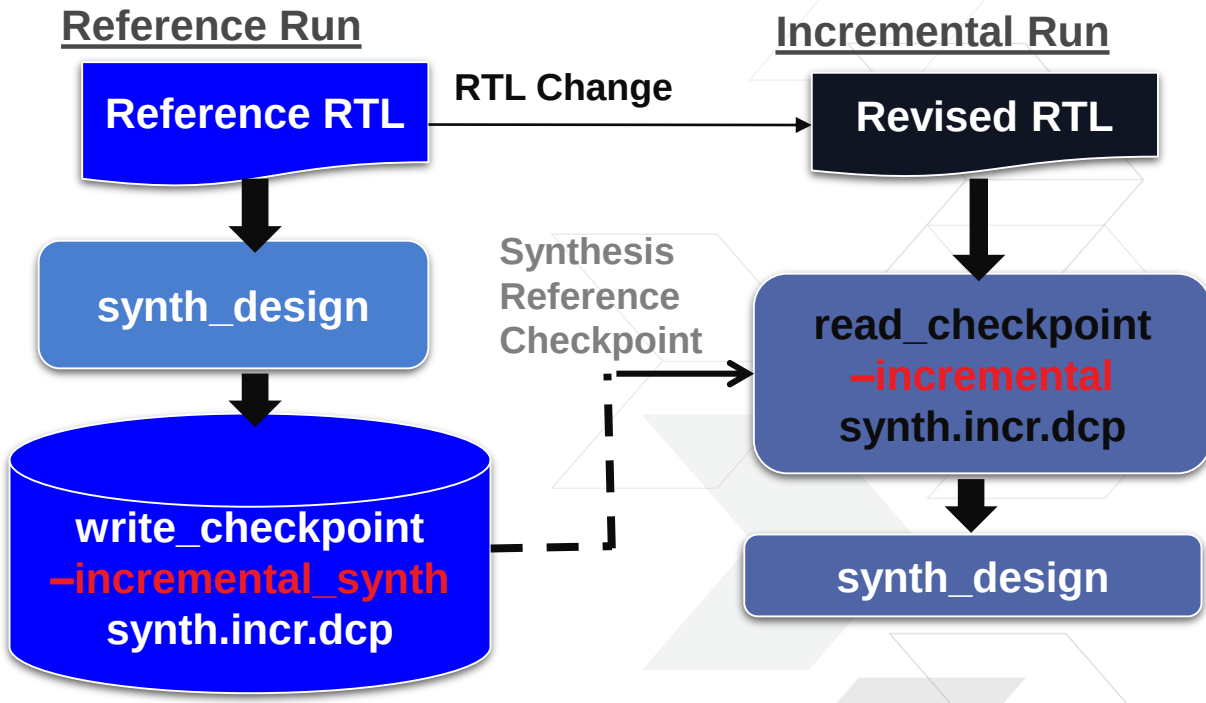
Useful Waiver Commands

create_waiver
get_waivers
delete_waivers
write_waivers
report_waivers

Vivado Incremental Synthesis

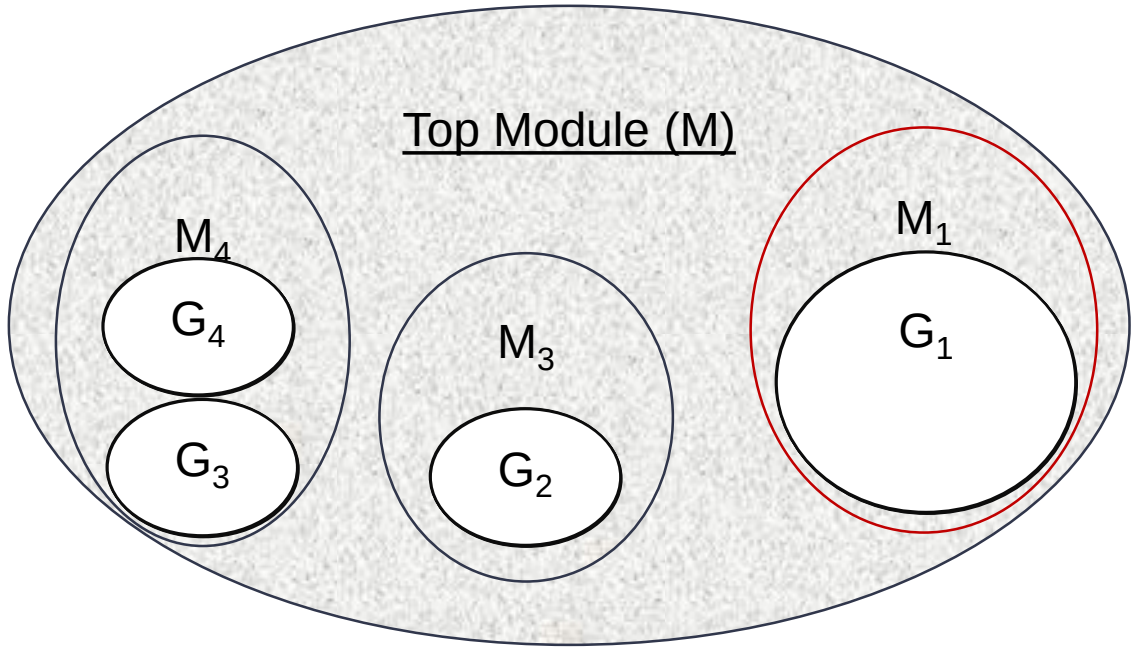
Incremental Synthesis

- **Flow similar to incremental P & R**
- **Benefits:**
  - 40% synthesis runtime reduction
    - Change is localized
  - Iterate quickly while working on a module
  - More design iterations in the front end
  - Improved predictability in results
  - Fewer changes in netlist structure when compared to previous flow
  - Improved results/QoR/runtime when used with Incremental P & R

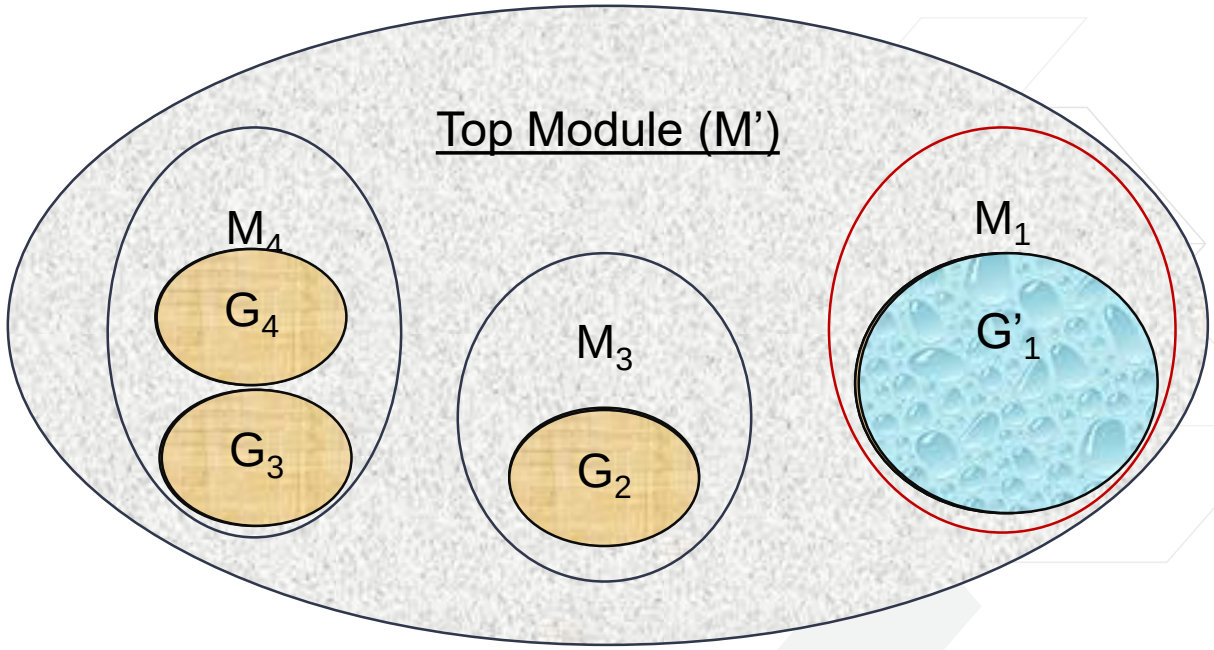


Incremental Synthesis - Internal Flow

Reference Run



Incremental Run

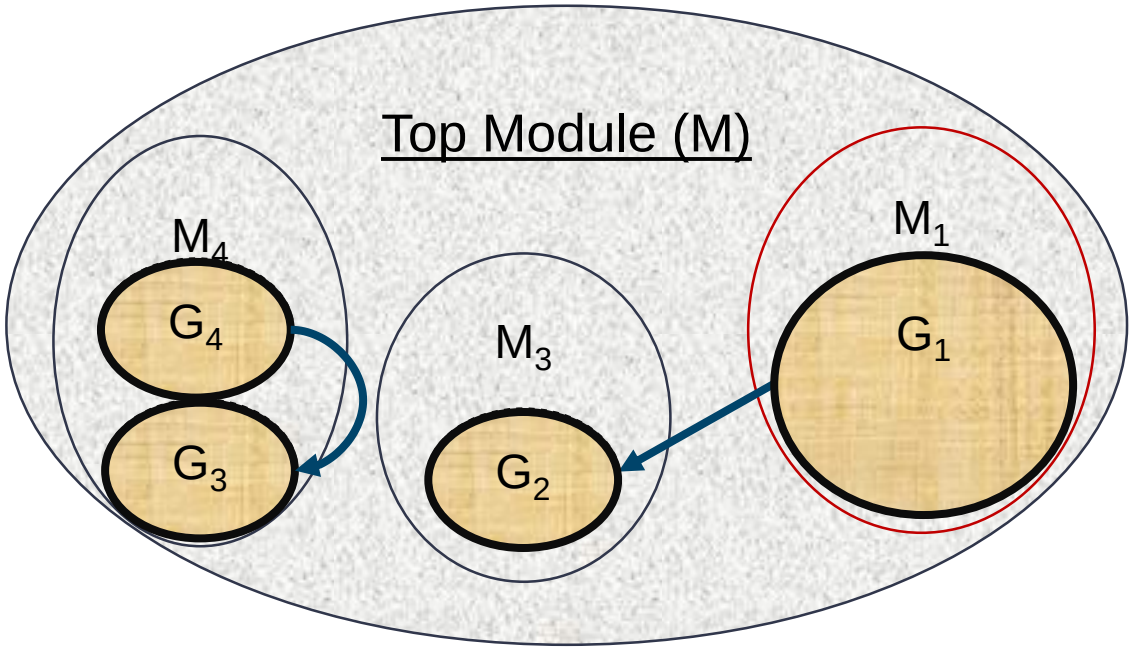


- G'1 must be re-synthesized
- G2, G3, G4 re-used

Incr. Synthesis - Cross-Boundary Optimizations

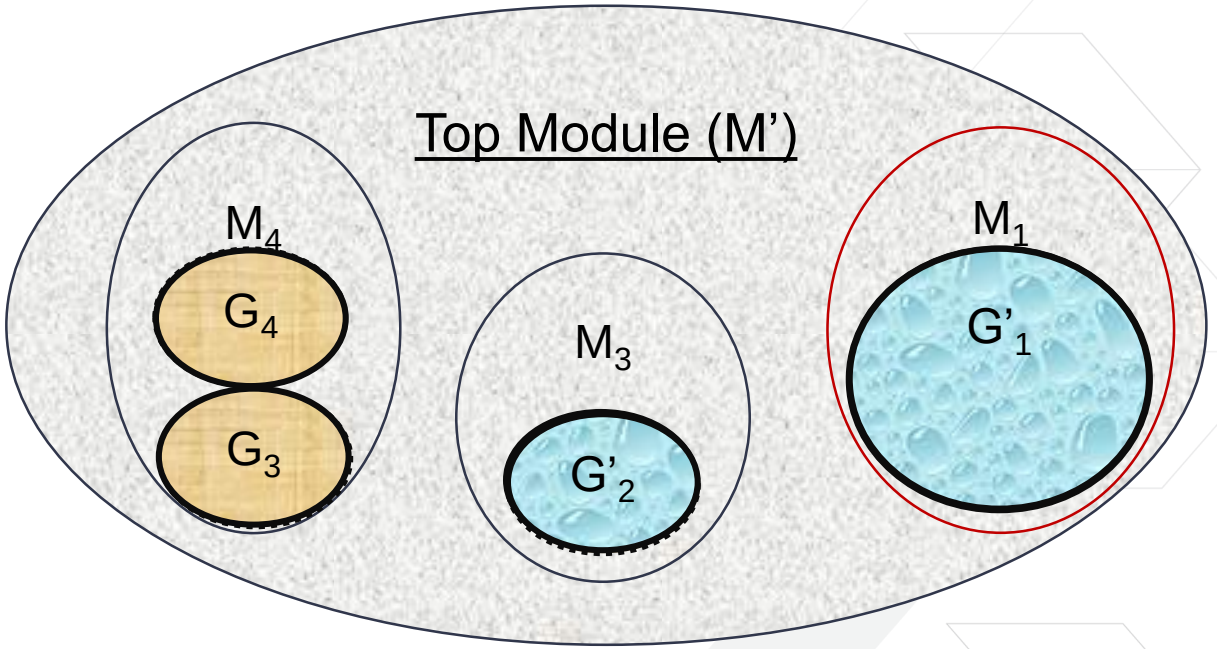
Reference Run

- **Track cross-boundary optimizations**



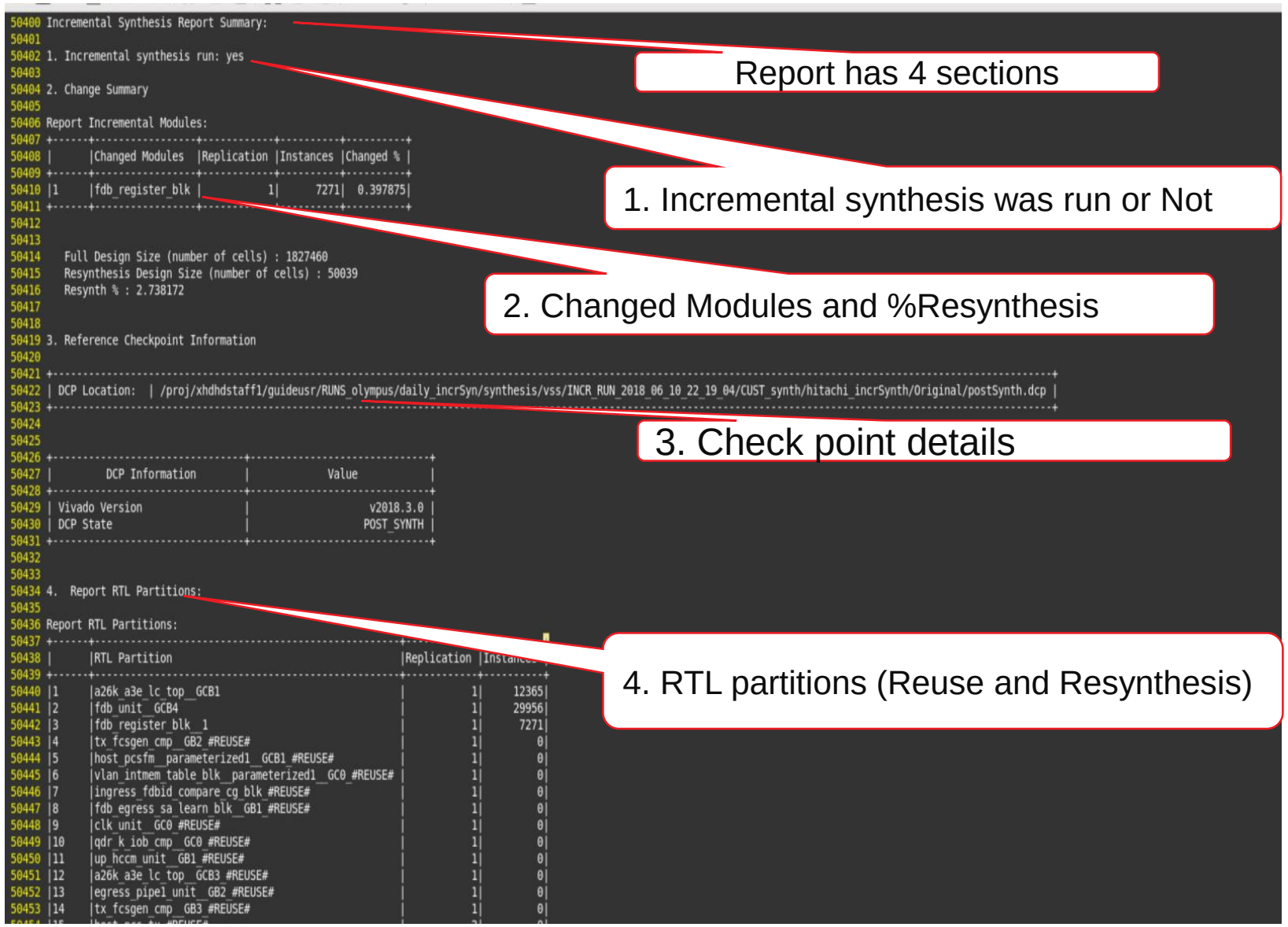
Incremental Run

- **Re-synthesize changed modules + its dependencies**



- More cross boundary optimizations leads to more re-synthesis (G'1 → G'2)
- Changed / dissolved partitions also need to be re-synthesized

Log file and Non-Project Mode Flow



- **Reference run**
  - run.tcl
    - synth_design
    - write_checkpoint –incremental_synth –force postSynth.dcp
    - opt_design
    - place_design
    - Phys-opt_design ← optimizations1
    - route_design
    - write_checkpoint routed.dcp
    - Phys-opt_design ←optimizations2
    - write_checkpoint ref_run_post-route_physopt.dcp
- **Incremental run**
  - run.tcl
    - read_checkpoint –incremental ../ReferenceRunDir/postSynth.dcp
    - synth_design
    - write_checkpoint –incremental_synth –force postSynth_incr.dcp
    - opt_design
    - read_checkpoint –incremental ../ReferenceRunDir/ref_run_post-route_physopt.dcp ← optimizations1 + optimizations2
    - place_design
    - route_design
    - write_checkpoint routed_incr.dcp

QoR: Tips & Tricks

Tips and Tricks: ROM Optimization

```
process(clk)
        variable AB_xor: std_logic_vector(63 downto 0);
        variable Box1, Box2, Box3, Box4, Box5, Box6, Box7, Box8, Box9, Box10: std_logic_vector(3 downto 0);
        variable BoxAll, Box_xor: std_logic_vector(39 downto 0);
begin
        if (clk = '1' and clk'event) then
                AB_xor:=inA(i) xor inB(i);
                Box1:=Box(conv_integer(AB_xor(63 downto 58)));
                Box2:=Box(conv_integer(AB_xor(57 downto 52)));
                Box3:=Box(conv_integer(AB_xor(51 downto 46)));
                Box4:=Box(conv_integer(AB_xor(45 downto 40)));
                Box5:=Box(conv_integer(AB_xor(39 downto 34)));
                Box6:=Box(conv_integer(AB_xor(33 downto 28)));
                Box7:=Box(conv_integer(AB_xor(27 downto 22)));
                Box8:=Box(conv_integer(AB_xor(21 downto 16)));
                Box9:=Box(conv_integer(AB_xor(15 downto 10)));
                Box10:=Box(conv_integer(AB_xor(9 downto 4)));
                BoxAll:=Box1 & Box2 & Box3 & Box4 & Box5 & Box6 & Box7 & Box8 & Box9 & Box10;
                Box_xor:=BoxAll xor inA(i)(63 downto 24) xor inB(i)(39 downto 0);
                outA(i) <= Box_xor & inA(i)(23 downto 0);
                outB(i) <= inB(i);
        end if;
end process;
```

- 64-deep ROM, 4-bit wide accessing different locations
- Loop with 30 iterations
- 10 ROM structures per iteration (300 ROMs in total)
- Data in 0-15 repeated in 16-31. 32-47 and 48-62
- Could this be 16 deep instead of 64 deep?

Missing uniformity in ROM data => 64th location

```
type ROM is array (integer range <>) of  std_logic_vector(3 downto 0);
constant Box: ROM(0 to 63):=(
        x"0", x"1",     x"2",   x"3",   x"4",   x"5",   x"6",   x"7",   x"8",   x"9",   x"a",   x"b",   x"c",   x"d",   x"e",   x"f",
        x"0", x"1",     x"2",   x"3",   x"4",   x"5",   x"6",   x"7",   x"8",   x"9",   x"a",   x"b",   x"c",   x"d",   x"e",   x"f",
        x"0", x"1",     x"2",   x"3",   x"4",   x"5",   x"6",   x"7",   x"8",   x"9",   x"a",   x"b",   x"c",   x"d",   x"e",   x"f",
        x"0", x"1",     x"2",   x"3",   x"4",   x"5",   x"6",   x"7",   x"8",   x"9",   x"a",   x"b",   x"c",   x"d",   x"e",   x"f");
```

Tips and Tricks: ROM Optimization

```
process(clk)
        variable AB_xor: std_logic_vector(63 downto 0);
        variable Box1, Box2, Box3, Box4, Box5, Box6, Box7, Box8, Box9, Box10: std_logic_vector(3 downto 0);
        variable BoxAll, Box_xor: std_logic_vector(39 downto 0);
begin
        if (clk = '1' and clk'event) then
                AB_xor:=inA(i) xor inB(i);
    if(AB_xor(63 downto 58) = "111111") then
      Box1:= "0000";
    else
      Box1:=Box(conv_integer(AB_xor(63 downto 58)));
    end if;
    if(AB_xor(57 downto 52) = "111111") then
      Box2:= "0000";
    else
      Box2:=Box(conv_integer(AB_xor(57 downto 52)));
    end if;
    if(AB_xor(51 downto 46) = "111111") then
      Box3:= "0000";
    else
      Box3:=Box(conv_integer(AB_xor(51 downto 46)));
    end if;
```

4-bit address as the two MSB bits doesn't play any role

Check the condition to access the data for address#63

This way, the ROM now can become 16-deep and 4-bit wide

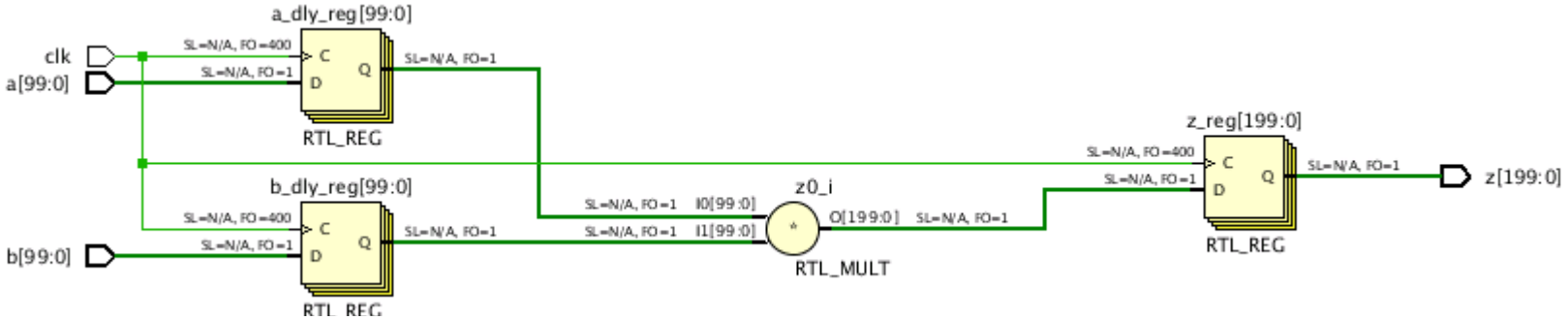
Tips and Tricks: ROM Optimization

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	3087	0	303600	1.02
LUT as Logic	3083	0	303600	1.02
LUT as Memory	4	0	130800	<0.01
LUT as Distributed RAM	0	0		
LUT as Shift Register	4	0		
Slice Registers	3981	0	607200	0.66
Register as Flip Flop	3981	0	607200	0.66
Register as Latch	0	0	607200	0.00
F7 Muxes	0	0	151800	0.00
F8 Muxes	0	0	75900	0.00

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	1826	0	303600	0.60
LUT as Logic	1822	0	303600	0.60
LUT as Memory	4	0	130800	<0.01
LUT as Distributed RAM	0	0		
LUT as Shift Register	4	0		
Slice Registers	3981	0	607200	0.66
Register as Flip Flop	3981	0	607200	0.66
Register as Latch	0	0	607200	0.00
F7 Muxes	0	0	151800	0.00
F8 Muxes	0	0	75900	0.00

LUT difference = Original – Proposed (3087 - 1826) = 1261

Tips and Tricks: 500 MHz Wide Multiplier



Site Type	Used	Fixed	Available	Util%
DSPs	36	0	4560	0.79
DSP48E2 only	36			

- **Tip: Review log file**
- **36 DSP's for 100x100 multiplier**

- **Not meeting timing, needs pipeline registers**
- **8 pipeline registers needed for timing closure**

```
INFO: [Synth 8-5845] Not enough pipeline registers after wide multiplier. Recommended levels of pipeline registers is 36 [/archive/marketing_designs/balacha/customer_d
```

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.162 ns	Worst Hold Slack (WHS):	0.016 ns	Worst Pulse Width Slack (WPWS):	0.532 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6626	Total Number of Endpoints:	6626	Total Number of Endpoints:	1547

```
//wide multiplier

module wide_mult(clk, h_0, h_1, Y);

  input clk;
  input [99:0] h_0;
  input [99:0] h_1;

  output [199:0] Y;
```

```
module wide_mult(clk, h_0, h_1, Y);

input clk;
input unsigned  [99:0] h_0;
input unsigned [99:0] h_1;
output [199:0] Y;

reg [199:0] y_1;

// add 5 pipeline reg
reg [199:0] ypip1, ypip2, ypip3, ypip4,

reg  [99:0] A, B;

(* keep *) reg [99:0]  Areg, Breg;

 always @(posedge clk) begin
    A <= h_0 ;
    B <= h_1;
    Areg <= A;
    Breg <= B;
    ypip1 <= Areg*Breg;
    ypip2 <= ypip1;
    ypip3 <=ypip2;
    ypip4 <= ypip3;
    ypip5 <= ypip4;
    y_1 <= ypip5;
nd

sign Y = y_1;

dmodule
```

Tips and Tricks: Multiplier => LUT mapping

- **Higher utilization v/s competition for multipliers**

- **Need to compare LUT based mapping**
  - Map to DSP (use_dsp48 = “no”)
  - Convert to LUT based (-max_dsp 0)

With –max_dsp 0

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	13206	0	788160	1.68
LUT as Logic	13206	0	788160	1.68
LUT as Memory	0	0	394560	0.00
CLB Registers	979	0	1576320	0.06
Register as Flip Flop	979	0	1576320	0.06
Register as Latch	0	0	1576320	0.00
CARRY8	1474	0	98520	1.50
F7 Muxes	0	0	394080	0.00
F8 Muxes	0	0	197040	0.00
F9 Muxes	0	0	98520	0.00

With use_dsp48 = “no” attribute

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	10512	0	788160	1.33
LUT as Logic	10512	0	788160	1.33
LUT as Memory	0	0	394560	0.00
CLB Registers	494	0	1576320	0.03
Register as Flip Flop	494	0	1576320	0.03
Register as Latch	0	0	1576320	0.00
CARRY8	687	0	98520	0.70
F7 Muxes	0	0	394080	0.00
F8 Muxes	0	0	197040	0.00
F9 Muxes	0	0	98520	0.00

Summary

- **Following the UltraFast Design Methodology reduces Time-to-Market**
- **Waiver Mechanism for CDC, Methodology and DRCs enables clean reports and design sign-off**
- **Ensure Clock Domain Crossing issues are reviewed and fixed**
  - Use the waiver mechanism to focus on real issues
- **Vivado Incremental synthesis reduces compile time**
  - Reach out to your FAE for details/issues