Vivado Synthesis Tips & Tricks

Presented By

Balachander Krishnamurthy
Sr. Product Marketing Manager
October 2\textsuperscript{nd}, 2018
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

Impact of changes on design quality

Effort required to fix problems

Board/Device Planning  Design/IP Creation  RTL Coding & Synthesis  Place & Route  ECO  Lab Bring-Up & Debug

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
Step-by-step Analysis and Suggestions

Address common timing closure challenges

- HLx and SDx
- Project and Non-project

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!

![Table of CDC violations]

💡 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

```bash
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}
    }
}
```

> Create from: manual specification of all arguments

>> Arguments are order dependent. They must match order inside the violation object

Notice: only description argument specified with this method.
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

**Top Module (M)**
- $M_1$, $M_3$, $G_1$, $G_2$, $G_3$, $G_4$

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

**Top Module (M')**
- $M_1$, $G_{1}'$, $G_{2}'$, $G_3$, $G_4$

- More cross boundary optimizations leads to more re-synthesis ($G_1' \rightarrow G_2'$)
- Changed / dissolved partitions also need to be re-synthesized
Log file and Non-Project Mode Flow

Report has 4 sections

1. Incremental synthesis was run or Not
2. Changed Modules and %Resynthesis
3. Check point details
4. RTL partitions (Reuse and Resynthesis)

> Reference run
  >> run.tcl
  - synth_design
  - write_checkpoint --incremental_synthesis --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_synthesis --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)
begin
  if (clk = '1' and clk'event) then
    AB xor:= std_logic_vector(63 downto 0);
    Box1, Box2, Box3, Box4, Box5, Box6, Box7, Box8, Box9, Box10 := std_logic_vector(3 downto 0);
    BoxAll, Box xor := std_logic_vector(39 downto 0);
    AB xor:=Box1 xor(63 downto 56));
    Box2:=Box2 xor(57 downto 52));
    Box3:=Box3 xor(51 downto 46));
    Box4:=Box4 xor(45 downto 40));
    Box5:=Box5 xor(39 downto 34));
    Box6:=Box6 xor(33 downto 28));
    Box7:=Box7 xor(27 downto 22));
    Box8:=Box8 xor(21 downto 16));
    Box9:=Box9 xor(15 downto 10));
    Box10:=Box10 xor(9 downto 4));
    BoxAll:=Box1 & Box2 & Box3 & Box4 & Box5 & Box6 & Box7 & Box8 & Box9 & Box10;
    Box xor:=BoxAll xor inA(1)(63 downto 24) xor inB(1)(39 downto 0);
    outA(i) <= Box xor & inA(1)(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

```vhDL
process(clk)
begin
  if (clk = '1' and clk'event) then
    AB xor:= std_logic_vector(63 downto 0);
    Box1, Box2, Box3, Box4, Box5, Box6, Box7, Box8, Box9, Box10 := std_logic_vector(3 downto 0);
    BoxAll, Box xor := std_logic_vector(39 downto 0);
    AB xor:=Box1 xor(63 downto 56));
    Box2:=Box2 xor(57 downto 52));
    Box3:=Box3 xor(51 downto 46));
    Box4:=Box4 xor(45 downto 40));
    Box5:=Box5 xor(39 downto 34));
    Box6:=Box6 xor(33 downto 28));
    Box7:=Box7 xor(27 downto 22));
    Box8:=Box8 xor(21 downto 16));
    Box9:=Box9 xor(15 downto 10));
    Box10:=Box10 xor(9 downto 4));
    BoxAll:=Box1 & Box2 & Box3 & Box4 & Box5 & Box6 & Box7 & Box8 & Box9 & Box10;
    Box xor:=BoxAll xor inA(1)(63 downto 24) xor inB(1)(39 downto 0);
    outA(i) <= Box xor & inA(1)(23 downto 0);
    outB(i) <= inB(i);
  end if;
end process;
```
Tips and Tricks: ROM Optimization

```vhdl
process(clk)
begin
    if (clk = '1' and clk'event) then
        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;
    end if;
end process;
```

Check the condition to access the data for address #63

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

This way, the ROM now can become 16-deep and 4-bit wide
### Tips and Tricks: ROM Optimization

<table>
<thead>
<tr>
<th>Site Type</th>
<th>Used</th>
<th>Fixed</th>
<th>Available</th>
<th>Util%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice LUTs*</td>
<td>3087</td>
<td>0</td>
<td>303680</td>
<td>1.02</td>
</tr>
<tr>
<td>LUT as Logic</td>
<td>3083</td>
<td>0</td>
<td>303680</td>
<td>1.02</td>
</tr>
<tr>
<td>LUT as Memory</td>
<td>4</td>
<td>0</td>
<td>130800</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>LUT as Distributed RAM</td>
<td>0</td>
<td>0</td>
<td>130800</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>LUT as Shift Register</td>
<td>4</td>
<td>0</td>
<td>130800</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>3981</td>
<td>0</td>
<td>607200</td>
<td>0.66</td>
</tr>
<tr>
<td>Register as Flip Flop</td>
<td>3981</td>
<td>0</td>
<td>607200</td>
<td>0.66</td>
</tr>
<tr>
<td>Register as Latch</td>
<td>0</td>
<td>0</td>
<td>607200</td>
<td>0.66</td>
</tr>
<tr>
<td>F7 Muxes</td>
<td>0</td>
<td>0</td>
<td>151800</td>
<td>0.00</td>
</tr>
<tr>
<td>F8 Muxes</td>
<td>0</td>
<td>0</td>
<td>75900</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Site Type</th>
<th>Used</th>
<th>Fixed</th>
<th>Available</th>
<th>Util%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice LUTs*</td>
<td>1825</td>
<td>0</td>
<td>303600</td>
<td>0.60</td>
</tr>
<tr>
<td>LUT as Logic</td>
<td>1822</td>
<td>0</td>
<td>303600</td>
<td>0.60</td>
</tr>
<tr>
<td>LUT as Memory</td>
<td>4</td>
<td>0</td>
<td>130800</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>LUT as Distributed RAM</td>
<td>0</td>
<td>0</td>
<td>130800</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>LUT as Shift Register</td>
<td>4</td>
<td>0</td>
<td>130800</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>3981</td>
<td>0</td>
<td>607200</td>
<td>0.66</td>
</tr>
<tr>
<td>Register as Flip Flop</td>
<td>3981</td>
<td>0</td>
<td>607200</td>
<td>0.66</td>
</tr>
<tr>
<td>Register as Latch</td>
<td>0</td>
<td>0</td>
<td>607200</td>
<td>0.66</td>
</tr>
<tr>
<td>F7 Muxes</td>
<td>0</td>
<td>0</td>
<td>151800</td>
<td>0.00</td>
</tr>
<tr>
<td>F8 Muxes</td>
<td>0</td>
<td>0</td>
<td>75900</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**LUT difference = Original – Proposed (3087 - 1826) = 1261**
Tips and Tricks: 500 MHz Wide Multiplier

- Tip: Review log file
- 36 DSP’s for 100x100 multiplier
- Not meeting timing, needs pipeline registers
- 8 pipeline registers needed for timing closure

Design Timing Summary

<table>
<thead>
<tr>
<th>Setup</th>
<th>Hold</th>
<th>Pulse Width</th>
<th>Hold</th>
<th>Pulse Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Negative Slack (WNS)</td>
<td>0.162 ns</td>
<td>0.016 ns</td>
<td>Worst Hold Slack (WHS)</td>
<td>0.000 ns</td>
</tr>
<tr>
<td>Total Negative Slack (TNS)</td>
<td>0.000 ns</td>
<td>0.000 ns</td>
<td>Total Hold Slack (THS)</td>
<td>0.000 ns</td>
</tr>
<tr>
<td>Number of Failing Endpoints</td>
<td>0</td>
<td>0</td>
<td>Number of Failing Endpoints</td>
<td>0</td>
</tr>
<tr>
<td>Total Number of Endpoints</td>
<td>6626</td>
<td>6626</td>
<td>Total Number of Endpoints</td>
<td>1547</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Site Type</th>
<th>Used</th>
<th>Fixed</th>
<th>Available</th>
<th>Util%</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB LUTs*</td>
<td>13286</td>
<td>0</td>
<td>788160</td>
<td>1.68</td>
</tr>
<tr>
<td>LUT as Logic</td>
<td>13286</td>
<td>0</td>
<td>788160</td>
<td>1.68</td>
</tr>
<tr>
<td>LUT as Memory</td>
<td>0</td>
<td>0</td>
<td>394560</td>
<td>0.00</td>
</tr>
<tr>
<td>CLB Registers</td>
<td>979</td>
<td>0</td>
<td>1576320</td>
<td>0.06</td>
</tr>
<tr>
<td>Register as Flip Flop</td>
<td>979</td>
<td>0</td>
<td>1576320</td>
<td>0.06</td>
</tr>
<tr>
<td>Register as Latch</td>
<td>0</td>
<td>0</td>
<td>1576320</td>
<td>0.06</td>
</tr>
<tr>
<td>CARRY8</td>
<td>1474</td>
<td>0</td>
<td>98520</td>
<td>1.50</td>
</tr>
<tr>
<td>F7 Muxes</td>
<td>0</td>
<td>0</td>
<td>394560</td>
<td>0.00</td>
</tr>
<tr>
<td>F8 Muxes</td>
<td>0</td>
<td>0</td>
<td>197040</td>
<td>0.00</td>
</tr>
<tr>
<td>F9 Muxes</td>
<td>0</td>
<td>0</td>
<td>98520</td>
<td>0.00</td>
</tr>
</tbody>
</table>

### With use_dsp48 = “no” attribute

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