Full RTL-to-GDSII Flow Report for 1-bit Full Adder

OpenLane & Sky130 PDK

Contents

1 Introduction

The design and implementation of a 1-bit full adder forms the foundation for more complex arithmetic units in digital systems. This report details a structured five-day workflow, taking the design from Verilog RTL through to GDSII layout using the OpenLane physical design toolchain and the Skywater130 PDK. Each stage—RTL simulation, synthesis, floorplanning, placement, clock-tree synthesis, routing, and signoff—is accompanied by the exact commands, practical explanations, and theoretical insights needed to achieve a production-ready layout. Use this guide as a step-by-step reference to ensure correctness, timing closure, and manufacturability for custom standard-cell designs.

2 Folder Structure

```
openlane/designs/fulladder/
  config.tcl
  src/
    fulladder.v
    fulladder_tb.v
    base.sdc
  runs/<timestamp>/
    results/
    tmp/
```

3 Day 1 – RTL Design and Simulation

3.1 RTL: fulladder.v

```
module fulladder (
  input wire a, b, cin,
  output wire sum, cout
);
assign sum = a ^ b ^ cin;
assign cout = (a & b) | (b & cin) | (a & cin);
endmodule
```

Defines a combinational 1-bit full adder: sum = a ^b ^cin; cout = majority(a, b, cin).

3.2 Testbench: fulladder_tb.v

```
‘timescale 1ns/1ps
module fulladder_tb;
  reg a, b, cin;
  wire sum, cout;
  fulladder UUT(.a(a), .b(b), .cin(cin), .sum(sum), .cout(cout));
  initial begin
    $display("a b cin | sum cout");
    $monitor("%b %b %b | %b %b", a, b, cin, sum, cout);
    a=0; b=0; cin=0; #5;
    a=0; b=0; cin=1; #5;
    a=0; b=1; cin=0; #5;
    a=0; b=1; cin=1; #5;
    a=1; b=0; cin=0; #5;
    a=1; b=0; cin=1; #5;
    a=1; b=1; cin=0; #5;
    a=1; b=1; cin=1; #5;
    $finish;
  end
endmodule
```

Applies all 8 input combinations and prints the outputs to verify logic functionality before synthesis.

3.3 Simulation

```
cd openlane/designs/fulladder/src
iverilog -o fulladder_tb.vvp fulladder.v fulladder_tb.v
vvp fulladder_tb.vvp
```

iverilog: Compiles Verilog files into a simulation executable.

vvp: Runs the compiled simulation, printing the truth table and verifying correctness.

4 Day 2 – Synthesis and Floorplanning

4.1 config.tcl

```
set ::env(DESIGN_NAME) "fulladder"
set ::env(VERILOG_FILES) "dir::src/fulladder.v"
set ::env(CLOCK_PORT) "clk"
set ::env(CLOCK_PERIOD) 10.0
```

Defines design parameters: name, RTL path, clock port, and clock period.

4.2 OpenLane Commands

```
cd /openLANE_flow
./flow.tcl -interactive           # Launch OpenLane in interactive mode
# In the OpenLane TCL shell:
package require openlane 0.9       # Load OpenLane package
prep -design fulladder            # Prepare the design workspace
run_synthesis                     # Perform RTL synthesis (Yosys + ABC)
run_floorplan                     # Create floorplan and place IOs
```

* **prep**: Sets up run directories, merges LEFs.
* **run_synthesis**: Converts RTL to gate-level netlist, optimizes logic.
* **run_floorplan**: Defines die/core area, IO ring, tap/decap placement, outputs a ‘.def‘ file.

4.3 Magic Floorplan Viewer

```
magic -T $PDK_ROOT/sky130A/libs.tech/magic/sky130A.tech \
     lef read tmp/merged.lef \
     def read results/floorplan/fulladder.floorplan.def &
```

Loads the merged LEF and floorplan DEF into Magic for visual inspection of IO placement and core outline.

5 Day 3 – Placement

5.1 Place Standard Cells

```
run_placement
```

* **Purpose of Placement**: Assigns physical X–Y coordinates to each standard cell instance, respecting row structures, power rails, and routing congestion regions. Ensures minimal wirelength and balanced timing.
* **Global Placement**: The placer spreads cells across the core area to minimize estimated net lengths using analytical or partitioning algorithms to optimize a cost function combining wirelength and density.
* **Detailed Placement**: Legalizes the solution by aligning cells to discrete row tracks, resolving overlaps, and respecting site/blockage constraints. Uses cell-swapping and local refinement to fix density violations and overlaps.
* **Why ‘run**$_{p}$*lacement‘* : *InvokesOpenROAD′splacer*¯*performingbothglobalanddetailedphases.Generatesa‘.d*

5.2 View Placement in Magic

```
magic -T $PDK_ROOT/sky130A/libs.tech/magic/sky130A.tech \
     lef read tmp/merged.lef \
     def read results/placement/fulladder.placement.def &
```

*** Inspecting Placement: Loading the placement DEF into Magic verifies that:**

• Cells line up on standard rows.

• No cell overlaps or routing blockages exist.

• I/O pins remain correctly positioned on the periphery.

*** Theory Tie-in: Proper placement reduces interconnect capacitance and resistance, improves timing slack, and reduces dynamic power through shorter interconnects.**

6 Day 4 – CTS and STA

6.1 base.sdc

```
create_clock -name clk -period 10.0 [get_ports clk]
set_input_delay  -clock clk 1 [get_ports a b cin]
set_output_delay -clock clk 1 [get_ports sum cout]
```

Defines a dummy clock and I/O delays for static timing analysis.

6.2 CTS and STA Commands

```
# In OpenLane TCL shell:
run_cts                          # Insert clock buffers, build clock tree

# Enter OpenROAD for STA:
openroad
read_lef  tmp/merged.lef
read_def  results/cts/fulladder.cts.def
read_verilog results/synthesis/fulladder.synthesis_cts.v
read_liberty $::env(LIB_SYNTH_COMPLETE)
link_design fulladder
read_sdc src/base.sdc
set_propagated_clock [all_clocks]
report_checks -path_delay max -nworst 10   # Report worst timing paths
exit
```

*** run_cts: Balances clock skew; inserts buffers.**

*** STA: Checks setup and hold slack using OpenROAD's timing engine.**

7 Day 5 – Routing, DRC, LVS, GDS

7.1 Power Distribution Network

```
gen_pdn
```

*** Purpose of PDN Generation: Creates power rails and stripes for VDD/VSS, ensuring minimal IR drop and robust power delivery.**
*** Theory: A well-designed PDN balances resistance and inductance trade-offs to maintain voltage integrity under dynamic loads.**

7.2 Detailed Routing

```
run_routing
```

*** Purpose of Routing: Implements detailed pathfinding for nets across multiple metal layers, respecting DRC constraints.**
*** Global Routing Phase: Assigns preliminary track assignments to minimize congestion.**
*** Detail Routing Phase: Finalizes exact wire geometries and via placements, ensuring design rule compliance.**
*** Why ‘run_routing‘: Automates TritonRoute’s global and detailed engines, producing a routed DEF for signoff.**

7.3 GDSII Export and Signoff

```
write_gds
```

*** GDSII Generation: Exports the final layout to GDSII—the industry standard for mask data.**

7.4 DRC and LVS Checks

```
magic -T $PDK_ROOT/sky130A/libs.tech/magic/sky130A.tech fulladder.gds &
# In Magics TCL:
drc check
```

Verifies minimum widths, spacings, and enclosure rules.

```
lvs -interactive -gds fulladder.gds \
    -spi fulladder.synthesis.v \
    -t $PDK_ROOT/sky130A/libs.tech/netgen/sky130.lib
```

Compares layout-extracted netlist to the gate-level netlist for electrical equivalence.

8 Conclusion

This comprehensive report integrates precise commands, theoretical underpinnings, and practical workflows to guide a full adder from RTL to GDSII. It ensures functional correctness, timing closure, and manufacturability for standard-cell ASIC designs.