1-bit Full-Adder ASIC Project Starter Kit (OpenLane)

This document provides a **complete starter kit** for building a **1-bit full-adder** ASIC using the **OpenLane** digital design flow. It includes every required file along with its **purpose** and **role** in the flow.

Directory Layout

```
OpenLane/designs/full_adder/
├── config.json
├── run_config.json        ← optional
├── ag_config.json         ← optional
├── pin_order.cfg
└── src/
    ├── full_adder.v
    └── full_adder.sdc

simulation/                ← outside OpenLane
└── full_adder_tb.v

analysis/                  ← optional
└── export_gds_features.py
```

---

1. `src/full_adder.v`

```
`default_nettype none
module full_adder (
    input  wire A,
    input  wire B,
    input  wire Cin,
    output wire Sum,
    output wire Cout
);
    assign Sum  = A ^ B ^ Cin;
    assign Cout = (A & B) | (B & Cin) | (A & Cin);
endmodule
`default_nettype wire
```

- **Uses**: Synthesized by Yosys during RTL synthesis.

- **Reason**: Describes the actual digital logic of the full adder.

---

2. `src/full_adder.sdc`

```
# Intentionally empty - no clocks in a combinational circuit.
```

- **Uses**: Required by STA for timing analysis.
- **Reason**: OpenLane expects an SDC file; this avoids errors even if it's empty.

---

3. `pin_order.cfg`

```
#W
A
B
Cin

#E
Sum
Cout
```

- **Uses**: Read during floorplanning.
- **Reason**: Ensures predictable and controlled I/O pin placement.

---

4. `config.json`

```
{
    "DESIGN_NAME": "full_adder",
    "VERILOG_FILES": "dir::src/*.v",

    "CLOCK_TREE_SYNTH": false,
    "CLOCK_PORT": "",
    "PNR_SDC_FILE": "dir::src/full_adder.sdc",
    "SIGNOFF_SDC_FILE": "dir::src/full_adder.sdc",

    "FP_PDN_VOFFSET": 5,
    "FP_PDN_HOFFSET": 5,
    "FP_PDN_VWIDTH": 2,
    "FP_PDN_HWIDTH": 2,
    "FP_PDN_VPITCH": 30,
    "FP_PDN_HPITCH": 30,
    "FP_PDN_SKIPTRIM": true,

    "FP_PIN_ORDER_CFG": "dir::pin_order.cfg",

    "pdk::sky130*": {
        "FP_CORE_UTIL": 20
    },
    "pdk::gf180mcu*": {
        "FP_CORE_UTIL": 20,
        "PL_TARGET_DENSITY": 0.5
    }
}
```

- **Uses**: Master configuration for OpenLane.
- **Reason**: Governs synthesis, floorplanning, PDN grid, and PDK-specific options.

---

5. `run_config.json` (Optional)

```
{
    "DESIGN_NAME": "full_adder",
    "CLOCK_TREE_SYNTH": false,
    "FP_PDN_SKIPTRIM": true
}
```

- **Uses**: Passed via CLI as overrides.
- **Reason**: Allows experimenting without modifying the main config.

---

6. `ag_config.json` (Optional)

```
{
    "DESIGN_NAME": "full_adder",
    "VERILOG_FILES": "dir::src/*.v",
    "FP_PIN_ORDER_CFG": "dir::pin_order.cfg",
    "pdk::sky130*": { "FP_CORE_UTIL": 20 }
}
```

- **Uses**: Read by ML or analysis scripts.
- **Reason**: Keeps external tools independent of OpenLane internals.

---

7. `simulation/full_adder_tb.v`

```
`timescale 1ns/1ps
module full_adder_tb;

  reg A, B, Cin;
  wire Sum, Cout;

  full_adder dut (.A(A), .B(B), .Cin(Cin), .Sum(Sum), .Cout(Cout));

  integer i;
  initial begin
    $dumpfile("full_adder_tb.vcd");
    $dumpvars(0, full_adder_tb);
    $display(" A B Cin | Sum Cout");

    for (i = 0; i < 8; i = i + 1) begin
      {A, B, Cin} = i[2:0];
      #1;
      $display(" %b %b  %b  |  %b    %b", A, B, Cin, Sum, Cout);
    end
    $finish;
  end
endmodule
```

- **Uses**: Simulates RTL using tools like Icarus Verilog or Verilator.
- **Reason**: Ensures the logic is functionally correct before physical design.

---

8. `analysis/export_gds_features.py` (Optional)

```
import pya, csv, math

GDS_FILE = "full_adder.gds"
GRID_SIZE_UM = 10
LAYERS = [("metal1",(67,20)), ("metal2",(68,20)), ("metal3",(69,20)),
          ("via1",(67,44)), ("via2",(68,44))]

layout = pya.Layout(); layout.read(GDS_FILE)
top = layout.top_cell(); dbu = layout.dbu

bbox = top.bbox(); x0,y0 = bbox.left*dbu, bbox.bottom*dbu
x1,y1 = bbox.right*dbu, bbox.top*dbu
rows = math.ceil((y1-y0)/GRID_SIZE_UM); cols = math.ceil((x1-x0)/GRID_SIZE_UM)
density = {(r,c):0 for r in range(rows) for c in range(cols)}
```

```
with open("layout_features.csv","w",newline="") as f:
    w = csv.writer(f); w.writerow(
      ["layer","x1","y1","x2","y2","width","height","area","grid_r","grid_c"])
    for lname,(lnum,dt) in LAYERS:
        for s in top.shapes(layout.layer(lnum,dt)).each():
            if not s.is_box(): continue
            b=s.bbox();
xL,yB,xR,yT=[b.left*dbu,b.bottom*dbu,b.right*dbu,b.top*dbu]
            area=(xR-xL)*(yT-yB); r=int((yB-y0)/GRID_SIZE_UM); c=int((xL-x0)/
GRID_SIZE_UM)
            density[(r,c)] += area
            w.writerow([lname,xL,yB,xR,yT,xR-xL,yT-yB,area,r,c])

with open("layout_density_grid.csv","w",newline="") as f:
    w=csv.writer(f); w.writerow(["grid_r","grid_c","density_um2"])
    for (r,c),a in density.items(): w.writerow([r,c,a])

print("\u2705  layout_features.csv & layout_density_grid.csv written")
```

- **Uses**: Run with `klayout -b -r export_gds_features.py` post-GDS export.
- **Reason**: Extracts layout stats for machine learning or optimization.

Quick Reference Table

File	Uses	Reason
`full_adder.v`	RTL synthesis	Defines functionality
`full_adder.sdc`	Timing engine	Empty file to satisfy tool expectations
`pin_order.cfg`	Floorplanner	Fixes I/O order for repeatability
`config.json`	Flow config	Main driver of all OpenLane steps
`run_config.json`	Overrides	Tweaks without editing the main config
`ag_config.json`	Post-flow scripts	Keeps tools decoupled from OpenLane internals
`full_adder_tb.v`	Simulation	Validates logic pre-PnR
`export_gds_features.py`	Analysis	Extracts GDS layout metrics

How to Run

```
# Enter OpenLane container
docker run -it -v $PWD/OpenLane:/OpenLane efabless/openlane:latest

# Inside container:
cd /OpenLane
./flow.tcl -interactive
prep -design full_adder
run_flow

# View layout using KLayout:
klayout designs/full_adder/runs/*/results/signoff/full_adder.gds
```

With these files and flow steps, you can fully explore the RTL-to-GDSII lifecycle of a digital logic gate. ✅