EC440-FINAL REPORT

32-BIT SIGNED PIPELINED MULTIPLIER

Submitted by

Adithi S Upadhya

191EC101

Poorvi S.H.M

191EC229

OBJECTIVE

To implement a 32-bit Pipelined Multiplier in ASIC configuration using OpenLANE tool chain which is open source with Skywater 130nm technology and observing various features of the layout in MAGIC.

INTRODUCTION

Pipelining is a popular technique that has been used in the design sector several years. This is an architectural option used by designers to reduce power.

To implement the pipeline into a multiplier architecture, we need to introduce registers to break the combinational path of multiplication and addition and compensate the delay added by these registers.

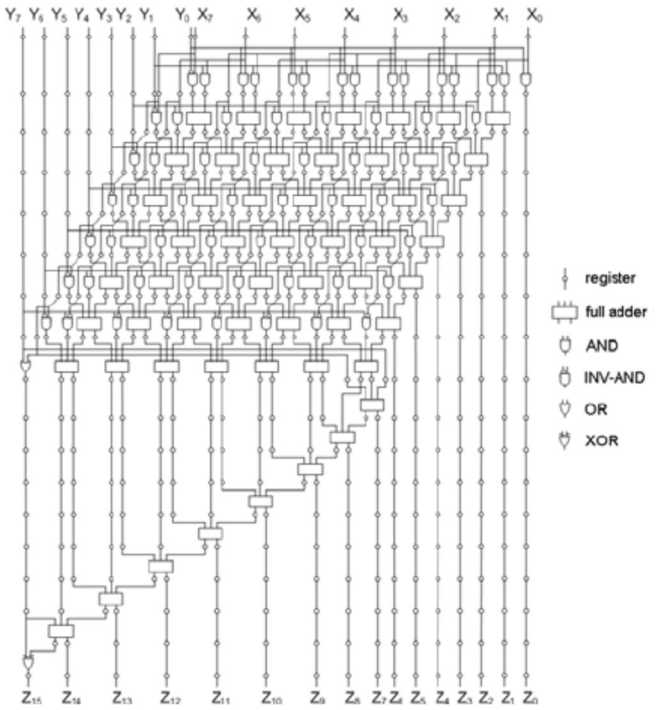


Fig 1. 16-bit Pipelined Multiplier Architecture

OpenLANE:

• OpenLane is an automated RTL to GDSII flow based on several components including OpenROAD, Yosys, Magic, Netgen, Fault, CVC, SPEF-Extractor, CU-GR, Klayout and a number of custom scripts for design exploration and optimization.

• The flow performs full ASIC implementation steps from RTL all the way down to GDSII.

• THE OPENLANE ARCHITECTURE:

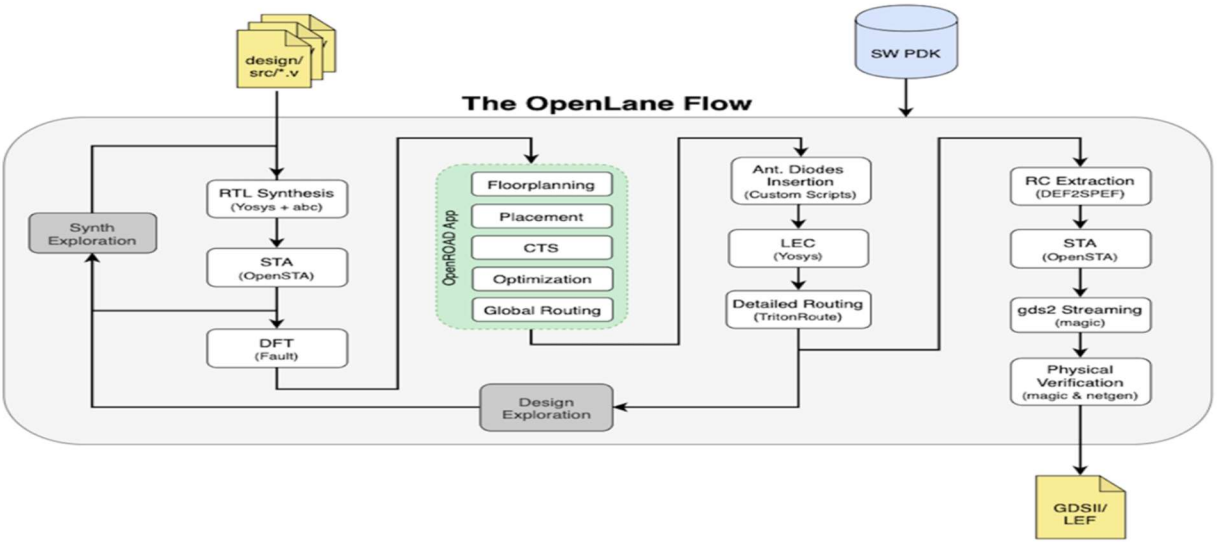


Fig 2: OpenLane Basic Flow

ASIC DESIGN STAGES:

1. *Synthesis:* Basic synthesis operations are carried out. RTL synthesis and technology mapping performed in this stage. Static timing analysis is also performed on the netlist.
2. *Floor plan and PDN:* Decides the approximate co-ordinates of the blocks in the design.
3. *Placement:* Place the blocks and fix the co-ordinates of the blocks.
4. *CTS (Clock Tree Synthesis):* Plans the clock distribution.
5. *Routing:* Interconnects the blocks using wires to achieve the functionality with least wire delay.
    - Global Routing- Input is floor plan. Decides loose plan for interconnects estimate the routes and routing congestion.
    - Detailed Routing- Actual routes of each interconnect with geometry, sizing and delay computation.
6. *Physical verification:* Verifies whether the actual layout represents the desired RTL.
7. *STA (Static Timing Analysis):* Method of validating the timing performance of a design by checking all possible paths for timing violations. STA breaks a

design down into timing paths, calculates the signal propagation delay along each path, and checks for violations of timing constraints inside the design and at the input/output interface.

OPENLANE DESIGN STAGES:

OpenLane flow contains various stages. By default, all flow steps are run in a sequence. Each stage may have multiple sub-stages.

1. *Synthesis:*
   - yosys - Performs RTL synthesis
   - abc - Performs technology mapping
   - OpenSTA - Performs static timing analysis on the resulting netlist to generate timing reports

2. *Floor plan and PDN:*
   - init_fp - Defines the core area for the macro as well as the rows (used for placement) and the tracks (used for routing)
   - ioplacer - Places the macro input and output ports
   - pdn - Generates the power distribution network
   - tapcell - Inserts welltap and decap cells in the floorplan

3. *Placement:*
   - RePLace - Performs global placement
   - Resizer - Performs optional optimizations on the design
   - OpenDP - Perfroms detailed placement to legalize the globally placed Components

4. *CTS (Clock Tree Synthesis):*
   - TritonCTS - Synthesizes the clock distribution network (the clock tree)

5. *Routing:*
   - FastRoute - Performs global routing to generate a guide file for the detailed router
   - CU-GR - Another option for performing global routing

- TritonRoute - Performs detailed routing
- SPEF-Extractor - Performs SPEF extraction

6. *GDSII Generation:*
   - Magic - Streams out the final GDSII layout file from the routed def
   - Klayout - Streams out the final GDSII layout file from the routed def as a back-up

7. *Checks:*
   - Magic - Performs DRC Checks Antenna Checks
   - Klayout - Performs DRC Checks
   - Netgen - Performs LVS Checks
   - CVC - Performs Circuit Validity Checks

IMPLEMENTATION

I. Verilog Code

The Verilog code for a 32-bit Pipelined Multiplier is obtained it's working is verified using ModelSim.

```
module MULTI_32bit(clk, rst,A, B, F);
‘define BIAS 8’b01111111
input clk, rst;
input [31:0]A;
input [31:0]B;
output [31:0]F;
reg [31:0]F;
//////////////// PIPE-LINE REGISTERS /////////////////
reg [62:0] P1;
reg [66:0] P2;
reg [31:0] P3;
//////////////////////////////////////////////////////
initial
begin
P1 = 0;
P2 = 0;
P3 = 0;
end
```

```
wire [1:0]sign;
wire [49:0]mantissa;
assign sign = A[31]+B[31];
//solve for the sign bit part
always @ ( posedge clk )
begin
////////////////////////////////////////////////////
P1[0] <= (sign == 1'b1) ? 1'b1 : 1'b0;
P1[31:1] <= A[30:0];
P1[62:32] <= B[30:0];
////////////////////////////////////////////////////
P2[0] <= P1[0];
P2[50:1] <= P1[23:1] * P1[54:32];
P2[58:51] <= P1[31:24];
P2[66:59] <= P1[62:55];
////////////////////////////////////////////////////
P3[0] <= P2[0];
if(P2[50:24] == 0) begin
P3[23:1] = P2[23:1];
P3[31:24] = P2[58:51] + P2[66:59] - 'BIAS + 8'h00;
end
else if(P2[50] == 1) begin
P3[23:1] = P2[49:27];
P3[31:24] = P2[58:51] + P2[66:59] - 'BIAS + 8'h1a;
end
else if(P2[49] == 1) begin
P3[23:1] = P2[48:26];
P3[31:24] = P2[58:51] + P2[66:59] - 'BIAS + 8'h19;
end
else if(P2[48] == 1) begin
P3[23:1] = P2[47:25];
P3[31:24] = P2[58:51] + P2[66:59] - 'BIAS + 8'h18;
end
else if(P2[47] == 1) begin
P3[23:1] = P2[46:24];
P3[31:24] = P2[58:51] + P2[66:59] - 'BIAS + 8'h17;
end
else if(P2[46] == 1) begin
P3[23:1] = P2[45:23];
P3[31:24] = P2[58:51] + P2[66:59] - 'BIAS + 8'h16;
end
```

```
else if(P2[45] == 1) begin

P3[23:1] = P2[44:22];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h15;
end
else if(P2[44] == 1) begin
P3[23:1] = P2[43:21];

P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h14;
end
else if(P2[43] == 1) begin
P3[23:1] = P2[42:20];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h13;
end
else if(P2[42] == 1) begin
P3[23:1] = P2[41:19];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h12;
end
else if(P2[41] == 1) begin
P3[23:1] = P2[40:18];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h11;
end
else if(P2[40] == 1) begin
P3[23:1] = P2[39:17];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h10;
end
else if(P2[39] == 1) begin
P3[23:1] = P2[38:16];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h0f;
end
else if(P2[38] == 1) begin
P3[23:1] = P2[37:15];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h0e;
end
else if(P2[37] == 1) begin
P3[23:1] = P2[36:14];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h0d;
end
else if(P2[36] == 1) begin
P3[23:1] = P2[35:13];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h0c;
end
```

```
else if(P2[35] == 1) begin
P3[23:1] = P2[34:12];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h0b;
end
else if(P2[34] == 1) begin
P3[23:1] = P2[33:11];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h0a;
end
else if(P2[33] == 1) begin
P3[23:1] = P2[32:10];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h09;
end
else if(P2[32] == 1) begin
P3[23:1] = P2[31:09];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h08;
end
else if(P2[31] == 1) begin
P3[23:1] = P2[30:08];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h07;
end
else if(P2[30] == 1) begin
P3[23:1] = P2[29:07];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h06;
end
else if(P2[29] == 1) begin
P3[23:1] = P2[28:06];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h05;
end
else if(P2[28] == 1) begin
P3[23:1] = P2[27:05];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h04;
end
else if(P2[27] == 1) begin
P3[23:1] = P2[26:04];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h03;
end
else if(P2[26] == 1) begin
P3[23:1] = P2[25:03];
P3[31:24] = P2[58:51] + P2[66:59] - ‘BIAS + 8’h02;
end
else begin
P3[23:1] = P2[24:02];
```

```
P3[31:24] = P2[58:51] + P2[66:59] - 'BIAS + 8'h01;
end
////////////////////////////////////////////////////////
F[31] <= P3[0];
F[30:0] <= P3[31:1];
////////////////////////////////////////////////////////
end
endmodule
```

Test Bench Code

```
timescale 1ns / 1ps
module MULTI_32bit_tb;
reg clk;
reg rst;
reg [31:0] A;
reg [31:0] B;
wire [31:0] F;
MULTI_32bit uut ( .clk(clk), .rst(rst), .A(A), .B(B), .F(F));
initial begin
A = 0;
B = 0;
clk <= 1'b0;
rst <= 1'b1;
#10 clk <= ~clk;

A = 32'h0000; B = 32'h0001;
#10 clk <= ~clk;
#10 clk <= ~clk;

A = 32'h0010; B = 32'h0001;
#10 clk <= ~clk;
#10 clk <= ~clk;

A = 32'h00F0; B = 32'h0040;
#10 clk <= ~clk;
#10 clk <= ~clk;

A = 32'hC000; B = 32'h1000;
#10 clk <= ~clk;
#10 clk <= ~clk;
```

```
A = 32'hAA00; B = 32'h0100;
#10 clk <= ~clk;
#10 clk <= ~clk;

A = 32'h0A01; B = 32'h0020;
#10 clk <= ~clk;
#10 clk <= ~clk;

A = 32'h0030; B = 32'h0009;
#10 clk <= ~clk;
#10 clk <= ~clk;

A = 32'h0020; B = 32'h00010;
#10 clk <= ~clk;
#10 clk <= ~clk;

A = 32'h9000; B = 32'h8000;
#10 clk <= ~clk;
#10 clk <= ~clk;

A = 32'h0003; B = 32'h0001;
#10 clk <= ~clk;
#10 clk <= ~clk;
#10 clk <= ~clk;
#10 clk <= ~clk;
#10 clk <= ~clk;
#10 clk <= ~clk;
end
endmodule
```

II. OpenLane

The necessary files for OpenLane and GDSII tools were installed. The procedure for implementation of a layout using these tools was practised and verified through a typical design.

The configuration file used is shown in Fig. 3

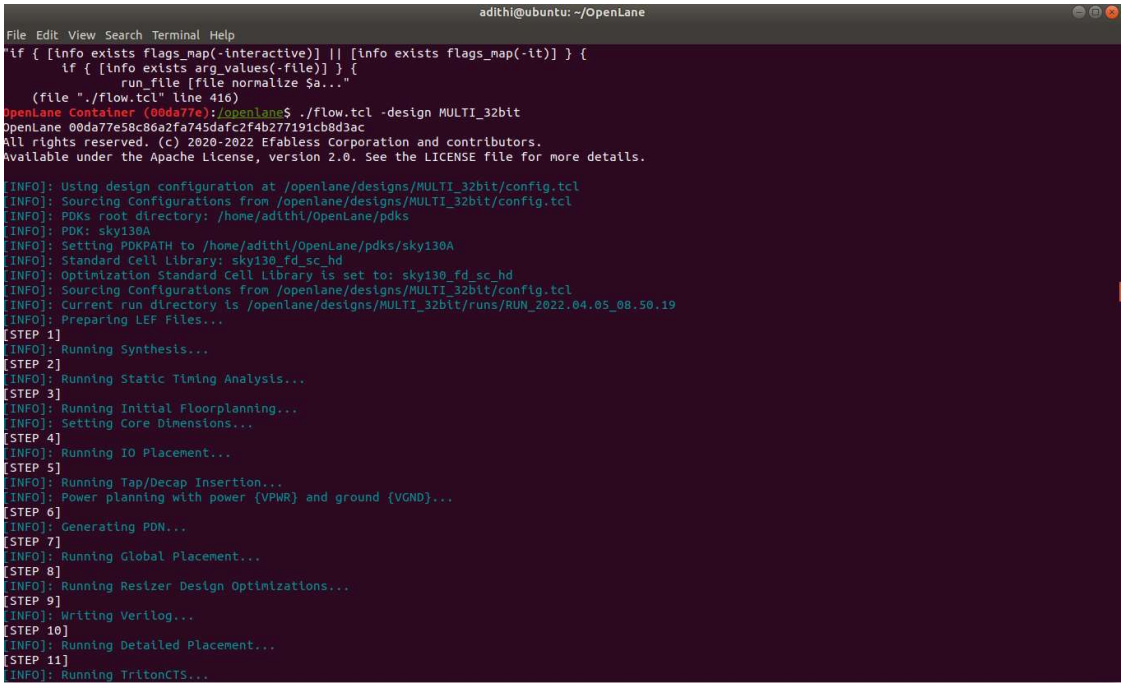
```
# User config
set ::env(DESIGN_NAME) MULTI_32bit

# Change if needed
set ::env(VERILOG_FILES) [glob $::env(DESIGN_DIR)/src/*.v]

# Design config
set ::env(CLOCK_PERIOD) "11.0"
set ::env(CLOCK_PORT) "clk"
set filename $::env(DESIGN_DIR)/$::env(PDK)_$::env(STD_CELL_LIBRARY)_config.tcl
if { [file exists $filename] == 1} {
        source $filename
}
```

Fig 3: Config File Used

Running synthesis exploration and flow on OpenLane:



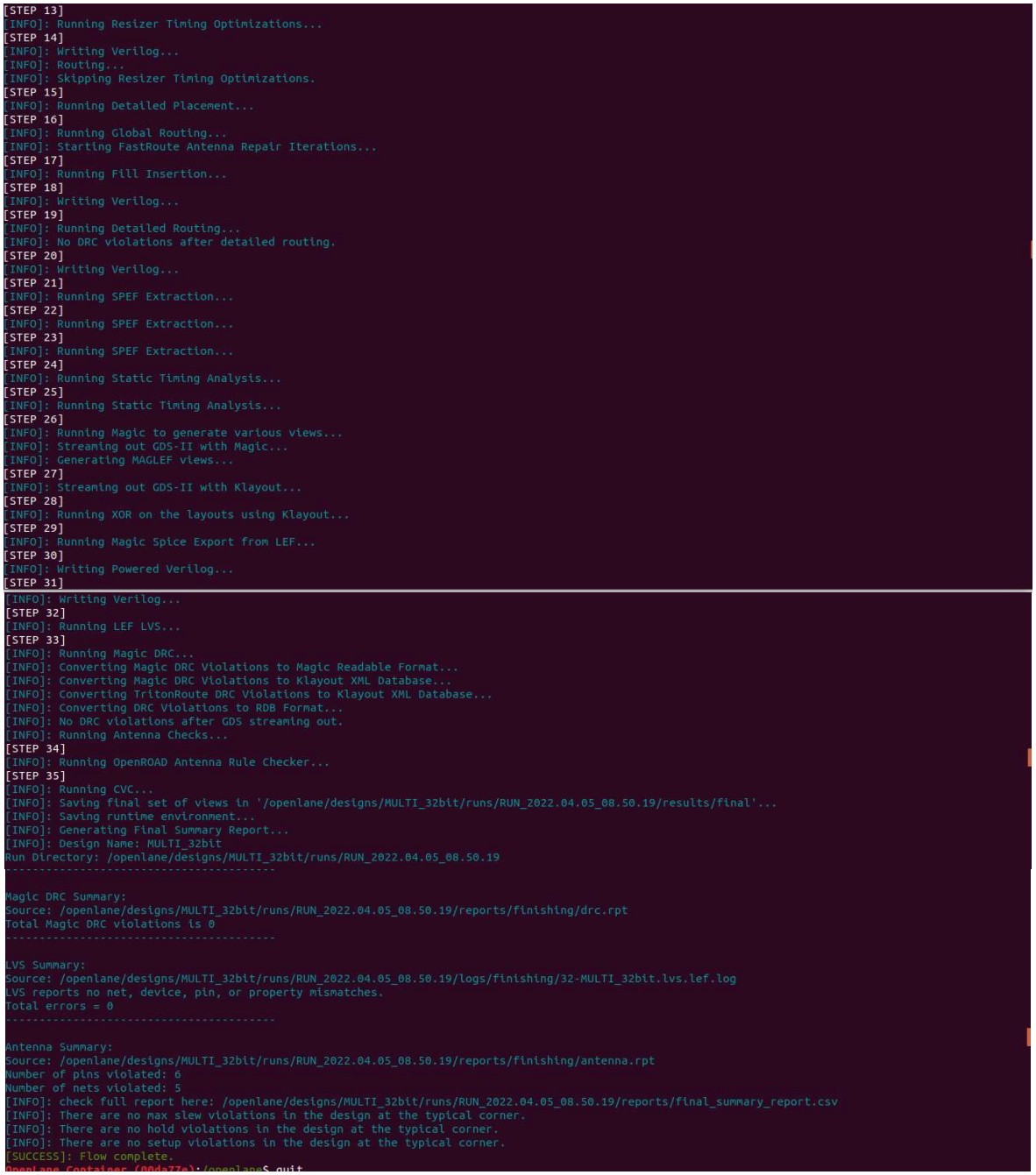


Fig 4: Running OpenLane

RESULTS:

- **From Synthesis exploration -**

Best Area	Best Gate Count	Best Delay
38740.91	4533.0	3736.05
AREA0	AREA0	AREA3

Strategy	Gate Count	Area (um^2)	Delay (ps)	Gates Ratio	Area Ratio	Delay Ratio
DELAY0	6365.0	54912.66	6411.51	1.404	1.417	1.716
DELAY1	6271.0	53074.65	6417.32	1.383	1.369	1.717
DELAY2	6267.0	52866.95	6362.21	1.382	1.364	1.702
DELAY3	6357.0	54958.96	6332.52	1.402	1.418	1.694
DELAY4	4947.0	45396.04	6474.41	1.091	1.171	1.732
AREA0	4533.0	38740.91	9264.61	1.0	1.0	2.479
AREA1	4556.0	38813.48	8109.81	1.005	1.001	2.17
AREA2	4577.0	39135.04	8571.86	1.009	1.01	2.294
AREA3	6896.0	49339.82	3736.05	1.521	1.273	1.0

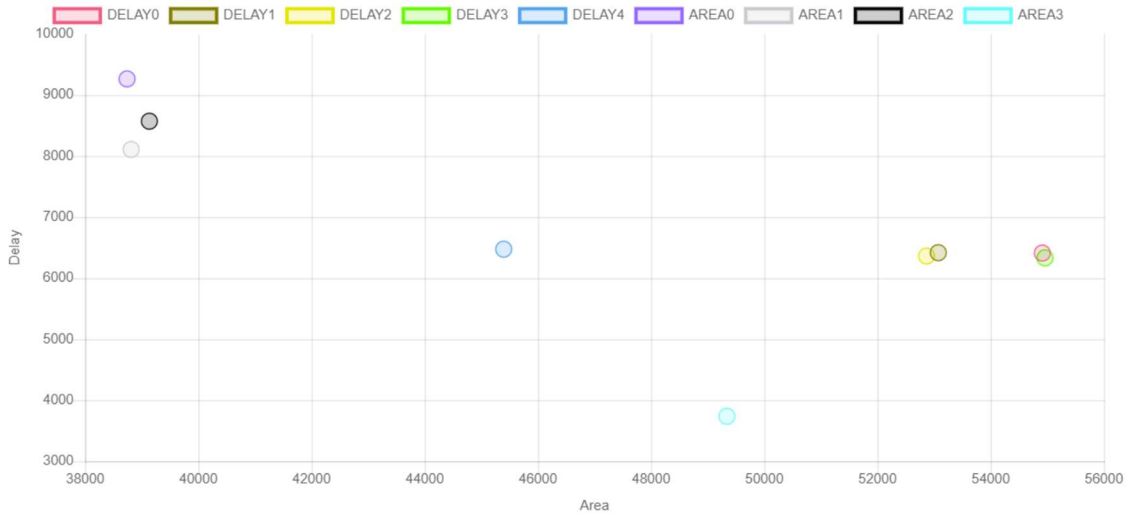


Fig 5: Synthesis Strategies Comparison

From the above comparison graph we see that AREA0 strategy gives both the best area and best count but maximum delay. Since the purpose of a pipelined multiplier is to increase the speed, minimum delay is desired. Therefore, we choose AREA3 strategy which gives the minimum delay.

Logs:

- Floor planning:

  initial_fp.log –

```
[INFO ODB-0223]     Created 13 technology layers
[INFO ODB-0224]     Created 25 technology vias
[INFO ODB-0225]     Created 441 library cells
```

```
set input_delay_value [expr $::env(CLOCK_PERIOD) * $::env(IO_PCT)]
set output_delay_value [expr $::env(CLOCK_PERIOD) * $::env(IO_PCT)]
puts "\[INFO\]: Setting output delay to: $output_delay_value"
[INFO]: Setting output delay to: 2.2
puts "\[INFO\]: Setting input delay to: $input_delay_value"
[INFO]: Setting input delay to: 2.2
set_max_fanout $::env(SYNTH_MAX_FANOUT) [current_design]
set clk_indx [lsearch [all_inputs] [get_port $::env(CLOCK_PORT)]]
#set rst_indx [lsearch [all_inputs] [get_port resetn]]
set all_inputs_wo_clk [lreplace [all_inputs] $clk_indx $clk_indx]
#set all_inputs_wo_clk_rst [lreplace $all_inputs_wo_clk $rst_indx $rst_indx]
set all_inputs_wo_clk_rst $all_inputs_wo_clk
# correct resetn
set_input_delay $input_delay_value  -clock [get_clocks $::env(CLOCK_PORT)] $all_inputs_wo_clk_rst
#set_input_delay 0.0 -clock [get_clocks $::env(CLOCK_PORT)] {resetn}
set_output_delay $output_delay_value  -clock [get_clocks $::env(CLOCK_PORT)] [all_outputs]
# TODO set this as parameter
set_driving_cell -lib_cell $::env(SYNTH_DRIVING_CELL) -pin $::env(SYNTH_DRIVING_CELL_PIN) [all_inputs]
set cap_load [expr $::env(SYNTH_CAP_LOAD) / 1000.0]
puts "\[INFO\]: Setting load to: $cap_load"
[INFO]: Setting load to: 0.033442
set_load  $cap_load [all_outputs]
puts "\[INFO\]: Setting clock uncertainity to: $::env(SYNTH_CLOCK_UNCERTAINITY)"
[INFO]: Setting clock uncertainity to: 0.25
set_clock_uncertainty $::env(SYNTH_CLOCK_UNCERTAINITY) [get_clocks $::env(CLOCK_PORT)]
puts "\[INFO\]: Setting clock transition to: $::env(SYNTH_CLOCK_TRANSITION)"
[INFO]: Setting clock transition to: 0.15
set_clock_transition $::env(SYNTH_CLOCK_TRANSITION) [get_clocks $::env(CLOCK_PORT)]
puts "\[INFO\]: Setting timing derate to: [expr {$::env(SYNTH_TIMING_DERATE) * 10}] %"
[INFO]: Setting timing derate to: 0.5 %
set_timing_derate -early [expr {1-$::env(SYNTH_TIMING_DERATE)}]S
set_timing_derate -late [expr {1+$::env(SYNTH_TIMING_DERATE)}]
[WARNING IFP-0028] Core area lower left (5.520, 10.880) snapped to (5.520, 10.880).
[INFO IFP-0001] Added 106 rows of 630 sites.
[INFO] Extracting DIE_AREA and CORE_AREA from the floorplan
[INFO] Floorplanned on a die area of 0.0 0.0 301.29 312.01 (microns). Saving to /openlane/designs/MULTI_32bit/ru
[INFO] Floorplanned on a core area of 5.52 10.88 295.32 299.2 (microns). Saving to /openlane/designs/MULTI_32bit
```

Fig 6: Initial Floor planning Log

tap.log –

```
[INFO TAP-0002] Original rows: 106
[INFO TAP-0003] Created 0 rows for a total of 106 rows.
[INFO TAP-0004] Inserted 212 endcaps.
[INFO TAP-0005] Inserted 1188 tapcells.
```

- Use of tapcells – Tap cells are used to limit resistance between power or ground connections to wells of the substrate. Taps are traditionally used so that VDD and GND are connected to substrate and n-wells respectively. This is to help tie them to VDD and GND levels so that they don't drift too

much (especially towards the middle of the chip) and cause latchup (condition where a low impedance path is created between a supply pin and ground).

- Use of endcells – The end cap cells are placed in the design because of the following reasons:
  - To protect the gate of a standard cell placed near the boundary from damage during manufacturing.
  - To avoid the base layer DRC (Nwell and Implant layer) at the boundary.
  - To make the proper alignment with the other block.

- Placement:

global.log –

```
================================================================================
 report_check_types -max_slew -max_cap -max_fanout -violators
================================================================================

================================================================================
max slew violation count 0
max fanout violation count 0
max cap violation count 0
================================================================================
check_slew_end
tns_report

================================================================================
 report_tns
================================================================================
tns -1.70
tns_report_end
wns_report

================================================================================
 report_wns
================================================================================
wns -0.57
wns_report_end
worst_slack

================================================================================
 report_worst_slack -max (Setup)
================================================================================
worst slack -0.57

================================================================================
 report_worst_slack -min (Hold)S
================================================================================
worst slack 0.08
```

Fig 7: Setup and Hold time violations

```
worst_slack_end
clock_skew

============================================================================
 report_clock_skew
============================================================================
Clock clk
Latency      CRPR       Skew
_9066_/CLK ^
   1.03
_9224_/CLK ^
   0.93      0.00       0.10

clock_skew_end
power_report

============================================================================
 report_power
============================================================================
Group                  Internal  Switching    Leakage      Total
                          Power      Power      Power      Power
----------------------------------------------------------------
Sequential             1.44e-03   1.36e-03   1.34e-09   2.80e-03  10.0%
Combinational          1.16e-02   1.35e-02   1.47e-08   2.51e-02  90.0%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
----------------------------------------------------------------
Total                  1.31e-02   1.49e-02   1.61e-08   2.79e-02 100.0%
                          46.8%      53.2%       0.0%
power_report_end
area_report
```

- tns - Total Negative Slack gives the sum of all the negative slacks in the design. From the value of TNS, we can know the severity of the slack in total design and whether to proceed or not with the current design model.
- wns - Worst Negative Path (WNS) points to the path having the maximum negative slack.
- Half Perimeter Wire Length (HPWL) of the circuit is used as an objective function to place blocks optimally within chip

```
============================================================================
 report_design_area
============================================================================
Design area 44405 u^2 53% utilization.
area_report_end
```

Fig 8: Area Design Report

For the AREA3 Strategy we see a greater area utilization (53%) than the default 50%.

- CTS:

```
[INFO ODB-0130]     Created 100 pins.
[INFO ODB-0131]     Created 6188 components and 38839 component-terminals.
[INFO ODB-0132]     Created 2 special nets and 22376 connections.
[INFO ODB-0133]     Created 4854 nets and 16463 connections.
[INFO ODB-0134] Finished DEF file: /openlane/designs/MULTI_32bit/runs/RUN_202
################################################################################
```

Fig 9: Clock Tree Synthesis

Max Fanout set is 5

```
report_design_area
============================================================================
Design area 40311 u^2 48% utilization.
area_report_end
```

Fig 10: Area Design Report for CTS

- Routing:

detailed.log –

```
[INFO ODB-0130]     Created 100 pins.
[INFO ODB-0131]     Created 13752 components and 69240 component-terminals.
[INFO ODB-0132]     Created 2 special nets and 52632 connections.
[INFO ODB-0133]     Created 4904 nets and 16608 connections.
[INFO ODB-0134] Finished DEF file: /openlane/designs/MULTI_32bit/runs/RUN_2022.
[INFO ORD-0030] Using 2 thread(s).
[INFO DRT-0149] Reading tech and libs.

Units:                1000
Number of layers:     13
Number of macros:     441
Number of vias:       25
Number of viarulegen: 25

[INFO DRT-0150] Reading design.

Design:                   MULTI_32bit
Die area:                 ( 0 0 ) ( 301290 312010 )
Number of track patterns: 12
Number of DEF vias:       4
Number of components:     13752
Number of terminals:      100
Number of snets:          2
Number of nets:           4904
```

Fig 11: Routing Log

```
[INFO DRT-0198] Complete detail routing.
Total wire length = 188746 um.
Total wire length on LAYER li1 = 0 um.
Total wire length on LAYER met1 = 74923 um.
Total wire length on LAYER met2 = 83653 um.
Total wire length on LAYER met3 = 22102 um.
Total wire length on LAYER met4 = 7936 um.
Total wire length on LAYER met5 = 128 um.
Total number of vias = 38903.
Up-via summary (total 38903):.

------------------------
 FR_MASTERSLICE        0
            li1    16577
           met1    18982
           met2     2674
           met3      666
           met4        4
------------------------
                  38903

[INFO DRT-0267] cpu time = 00:17:28, elapsed time = 00:10:41, memory = 744.74 (MB), peak = 928.34 (MB)
```

Fig 12: Wire Lengths in each layer

Detailed Routing step took the longest time to run.

- Final summary report:

Suggested Clock Frequency – 90.90
Clock Period Used – 11ns
Synthesis Strategy – AREA3
Max Fanout – 5 (default)
FP_CORE_UTIL – 50 (default)
PL_TARGET_DENSITY – 0.55 (default)

Errors:

Flow failed:

After going through the all the logs, we found that the error was due to set up violations. The worst slack (setup violation) is 0.73ns and the initial clock period used was 10ns (default value). In order to remove the violation, we increased the clock period to 11ns (based on the worst slack value)

Pin and net violations:

Could not find the cause of these violations and hence could not debug.

```
Antenna Summary:
Source: /openlane/designs/MULTI_32bit
Number of pins violated: 6
Number of nets violated: 5
```

Fig 13: Antenna Summary

MAGIC LAYOUT GENERATED:

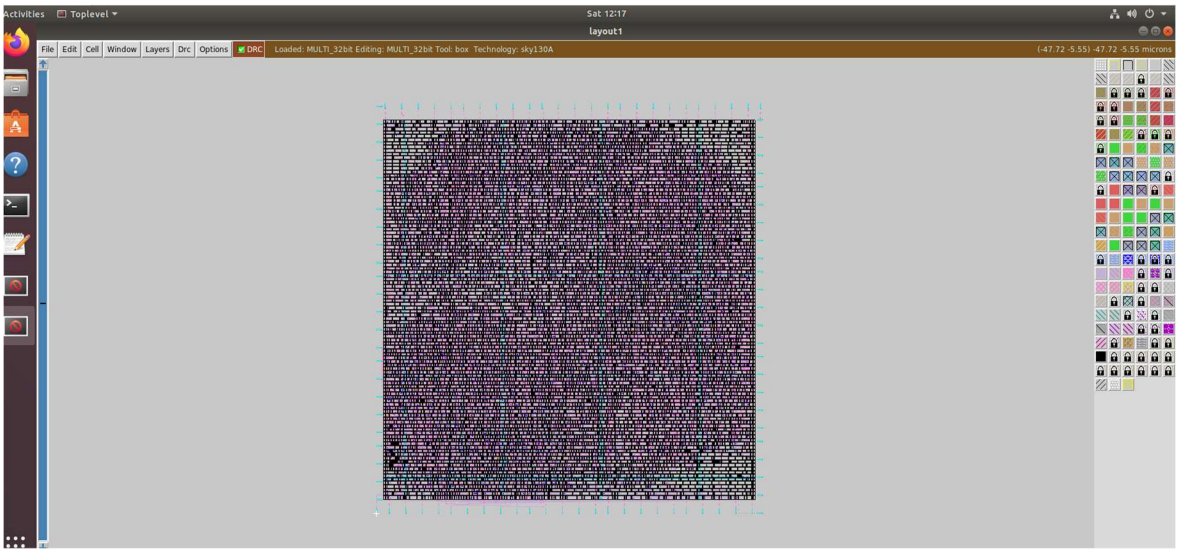


Fig 14: MAGIC Layout

GDS LAYOUT GENERATED:

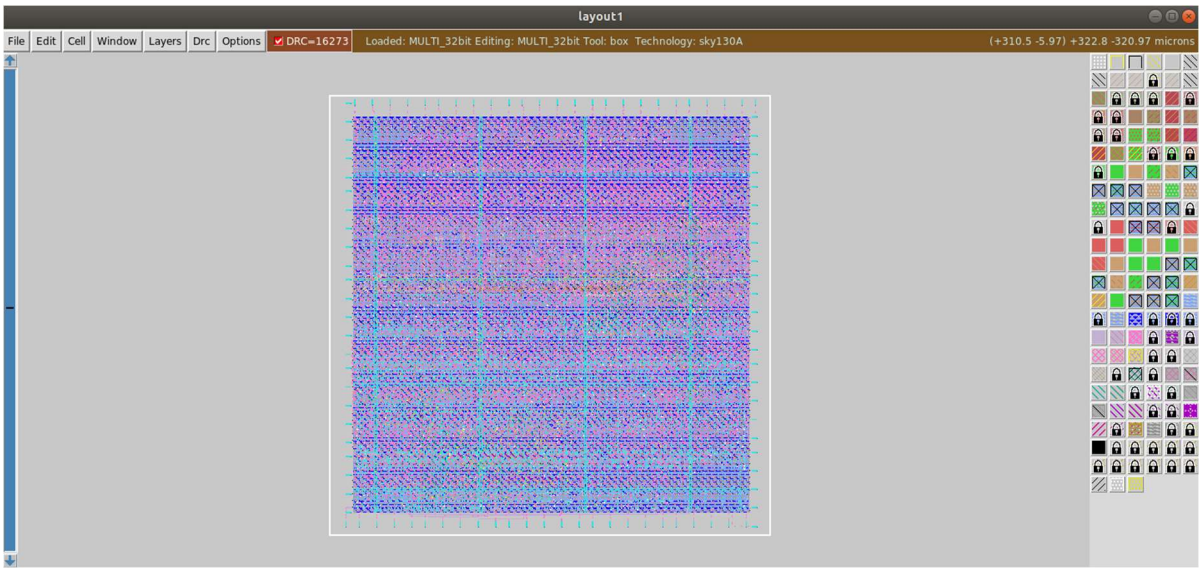


Fig 15: GDS Layout

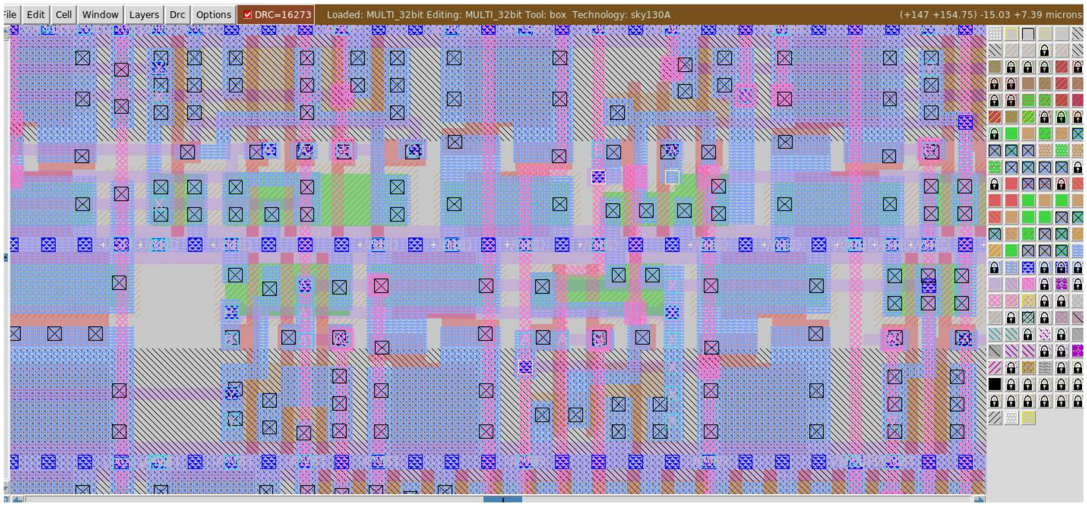


Fig 16: GDS Layout detailed (can see all the layers unlike MAGIC)

KLAYOUT:

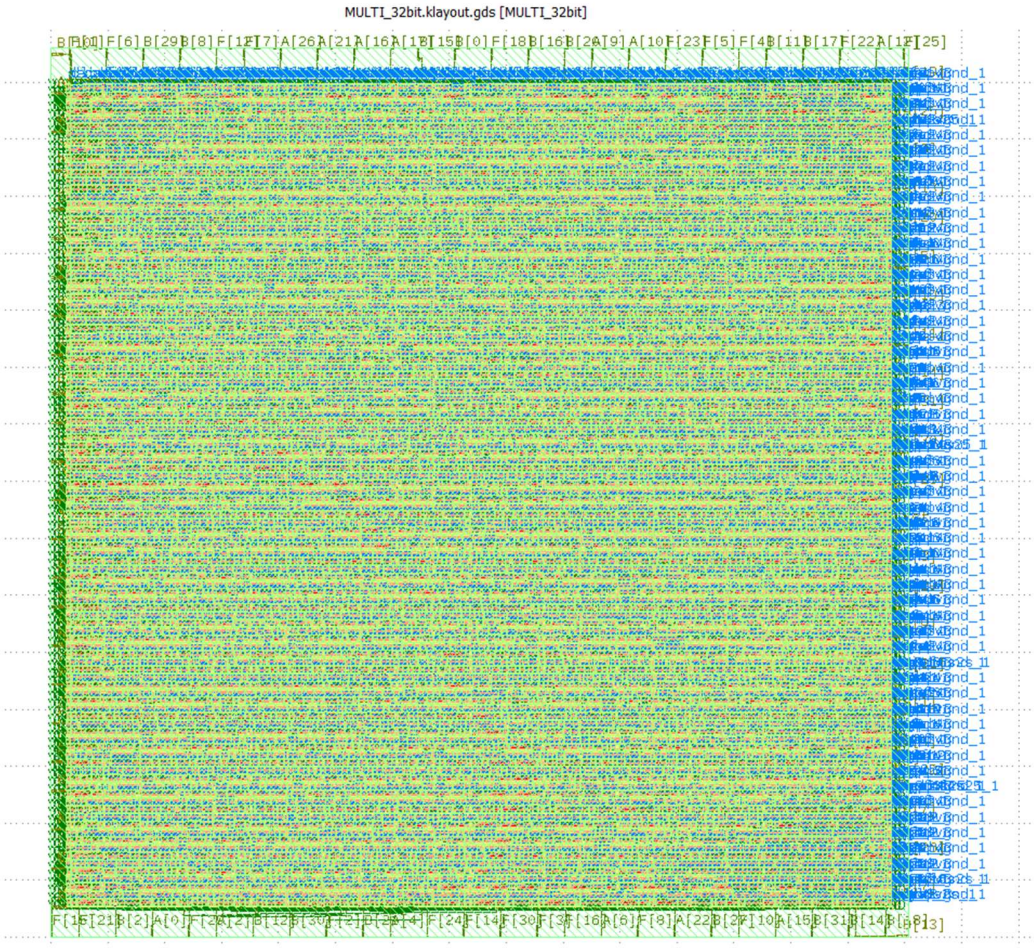


Fig 17: Klayout Generated

REFERENCES:

- Ahmed Ghazy and Mohamed Shalan, "OpenLANE: The Open-Source Digital ASIC Implementation Flow", Article No.21, Workshop on Open-Source EDA Technology (WOSET), 2020.
- S R Hitender, Dua Heena and S Sivanantham “ASIC Implementation of Two Stage Pipelined Multiplier”, International Journal of Engineering Research & Technology (IJERT), 2015
- Github for the verilog code
- Github for OpenLane