Digital IC Implementation of 'AXIS UART MATRIX VECTOR MULTIPLIER' using OSIC Tools and IHP SG13G2 PDK

Assignment 3.1 Report

Dulina Ranaweera - SKF2400190

May 21, 2025

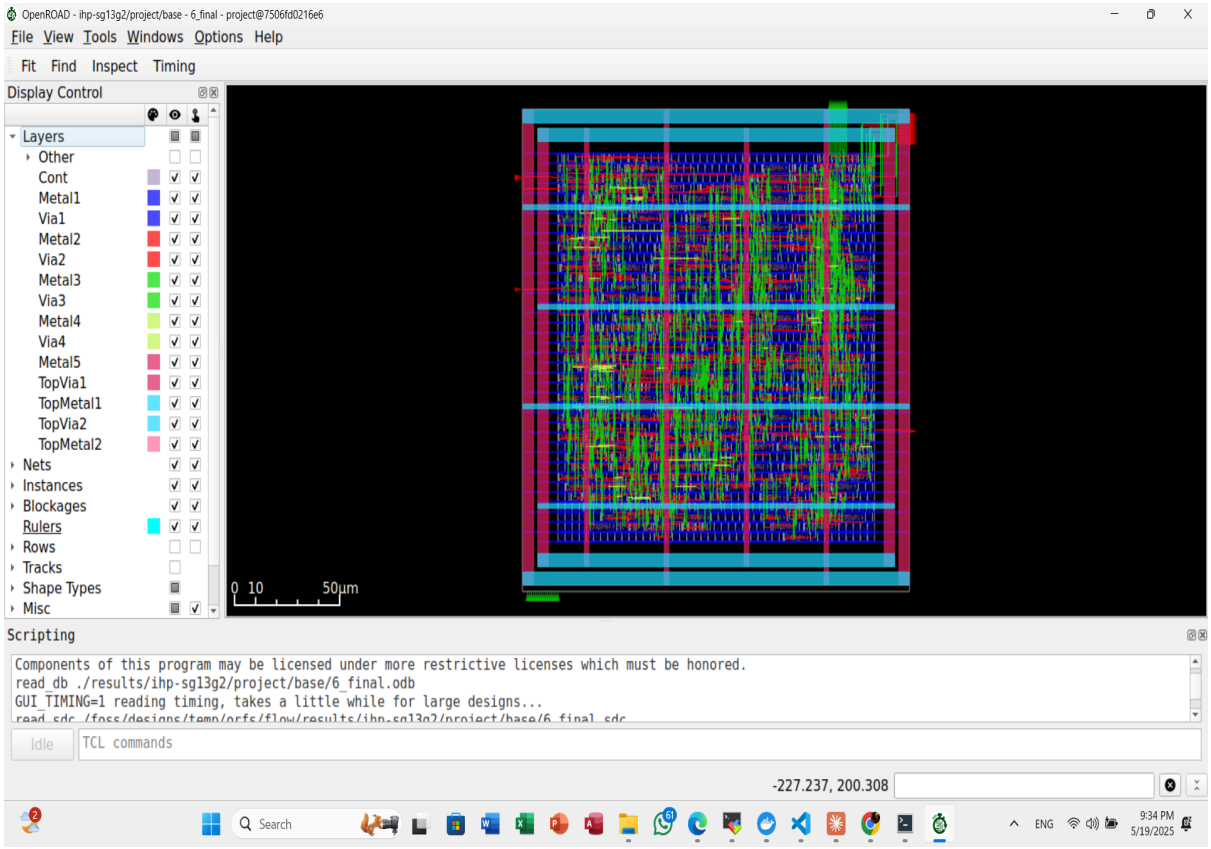

Figure 1: Final GDSII-equivalent layout of the 'Axis Uart Matrix Vector Multiplier' design as viewed in OpenROAD, showing the routed core, power distribution network, and metal layers.

Contents

1 Introduction

This report details the implementation of a digital design, referred to as "AXIS UART MATRIX VECTOR MULTIPLIER" from Register Transfer Level (RTL) to a GDSII layout. The primary objective was to successfully execute a complete Application-Specific Integrated Circuit (ASIC) design flow using open-source EDA tools provided within the OSIC ecosystem. Specifically, this project utilized the OpenROAD flow scripts (ORFS) targeting the IHP SG13G2 Process Design Kit (PDK). The design involves a Matrix Vector Multiplication (MVM) core and a UART interface.

2 Background

A typical digital ASIC design flow involves:

1. **RTL Design:** Describing hardware functionality in a Hardware Description Language (HDL) like Verilog or SystemVerilog.
2. **Synthesis:** Converting the HDL code into a gate-level netlist using a standard cell library.
3. **Floorplanning:** Defining the chip's overall area, aspect ratio, and placement of I/O pins and large macros.
4. **Placement:** Placing standard cells onto the chip floor.
5. **Clock Tree Synthesis (CTS):** Building a balanced clock distribution network.
6. **Routing:** Connecting the placed cells and pins according to the netlist.
7. **Post-Routing Optimization & Finishing:** Inserting filler cells, performing final timing checks, and metal fill.
8. **Verification:** Ensuring the design meets functional, timing, and physical requirements (DRC, LVS).
9. **GDSII Generation:** Creating the final layout file for fabrication.

3 Tools and Environment Setup

- **OSIC Tools Container:** Provides a pre-configured environment with necessary EDA tools.
- **Repository Cloning:** The project files and ORFS flow scripts were cloned into the working directory.
- **Key Tools Used:**
  - **Yosys (v0.51):** For RTL synthesis.
  - **OpenROAD (v2.0-19705-gbe6d1c688):** For the physical design (PnR) flow, including floorplanning, placement, CTS, routing, and final GDS generation steps. It internally uses OpenSTA for static timing analysis.
  - **KLayout:** For GDSII viewing and merging standard cell GDS files with the routed DEF.
- **PDK:** IHP SG13G2, utilizing the `sg13g2_stdcell_typ_1p20V_25C.lib` standard cell library and associated LEF/GDS files.

The flow was initiated using a shell script (`./run_all.sh`), which first cleaned previous runs (`./clean_all.sh`) and then executed the ORFS flow (`./run_orfs.sh`).

4 Design Flow and Execution

4.1 RTL Preparation & Synthesis (Yosys)

- The Verilog RTL file (`project.v`) was processed.
- Initial Yosys canonicalization step (`synth_canonicalize.tcl`) was performed. Warnings related to continuous assignments to `reg` types (`s_packets`, `tx`, `s_ready`) and replacement of a memory (`\tree`) with registers were noted.
- Full synthesis (`synth.tcl`) was executed, targeting the IHP SG13G2 standard cell library (`sg13g2_stdcell_typ_1p20V_25C.lib`).
- The clock period was set to 125000 ps (125 ns, or 8 MHz).
- The output was a gate-level netlist (`1_synth.v`) and an initial SDC file (`1_synth.sdc`).
- The synthesis process reported **784 cells** in the initial netlist.

4.2 Floorplanning (OpenROAD)

- LEF files (`sg13g2_tech.lef`, `sg13g2_stdcell.lef`) were loaded.
- A warning "port 'clock_i' not found" was issued by STA, suggesting a potential mismatch or unhandled clock input at this stage.
- STA reported 153 unclocked register/latch pins and 177 unconstrained endpoints.
- Core area was initialized. For the successful run, a design area of **13519 µm²** with **61% utilization** was established after snapping to a 39-row core. (The prior failed run had snapped to 20 rows, leading to the high utilization error).
- I/O pins were placed randomly (`place_pins -random`).
- No macros were present, so macro placement was skipped.
- Tap cells were inserted.
- Power Distribution Network (PDN) was generated.
- **Timing (Floorplan):** TNS: 0.00, WNS: 0.00, Worst Slack: INF.
- **Power (Floorplan):** Total: 5.15e-06 W (Sequential: 90.7%, Combinational: 9.3%).

4.3 Placement (Global and Detailed - OpenROAD)

- **Global Placement (Skip IO):** Initial global placement was run with a target density of 0.65.
- **IO Placement:** I/O pins were placed again, this time considering netlist connectivity. HPWL for I/O nets was 1638.53 µm.
- **Global Placement (Full):**
  - Run with routability and timing-driven options.
  - The timing-driven mode noted "no slacks found" and was disabled, likely due to the largely unconstrained nature of the design.
  - Resizing steps (buffer insertion, gate sizing) were performed.

  - **Timing (Global Place):** TNS: 0.00, WNS: 0.00, Worst Slack: INF.
  - **Power (Global Place):** Total: 5.15e-06 W (Sequential: 90.6%, Combinational: 9.4%).

- **Resizing:**
  - Port buffering inserted 3 input and 1 output buffer.
  - Design repair inserted 83 buffers and resized 90 instances to fix fanout violations.
  - Tie cells (23 Lo, 38 Hi) were inserted.
  - Instance count increased from 784 to **930**. Design area became **14776 µm² (67% utilization)**.

- **Detailed Placement:**
  - Legalized cell positions, improving HPWL by 10.4% (from 19512.8 µm to 17474.6 µm after initial legalization, then further minor improvement).
  - Cell flipping and mirroring optimized HPWL.
  - **Timing (Detailed Place):** TNS: 0.00, WNS: 0.00, Worst Slack: INF.
  - **Physical Checks (Detailed Place):** No violations for max slew, max capacitance, or max fanout. Setup/Hold violation counts were 0.
  - **Power (Detailed Place):** Total: 6.36e-06 W (Sequential: 72.3%, Combinational: 27.7%).

4.4 Clock Tree Synthesis (CTS - OpenROAD)

- CTS was run with sink clustering and level balancing.
- A critical warning "[WARNING CTS-0082] No valid clock nets in the design" was issued. This indicates that the defined clock `core_clock` (likely virtual at this point) was not actually driving any synthesizable clock tree, or no flip-flops were properly associated with it.
- Design repair was run post-CTS.
- **Timing (CTS):** TNS: 0.00, WNS: 0.00, Worst Slack: INF. Clock skew for `core_clock` reported "No launch/capture paths found."
- **Physical Checks (CTS):** No violations for max slew, max capacitance, or max fanout. Setup/Hold violation counts were 0.
- **Power (CTS):** Total: 6.36e-06 W (Sequential: 72.3%, Combinational: 27.7%). (Identical to detailed place power, as no clock tree was effectively built for power calculation).

4.5 Routing (Global and Detailed - OpenROAD)

- **Global Routing:**
  - Performed with congestion iterations. Final congestion report showed 0 overflow.
  - Total wirelength estimated at 34444 µm.
  - No antenna violations were found at this stage.
  - Design repair (buffering/resizing) was performed again.
  - **Timing (Global Route):** TNS: 0.00, WNS: 0.00, Worst Slack: INF.

  - **Physical Checks (Global Route):** No violations for max slew, max capacitance, or max fanout. Setup/Hold violation counts were 0.
  - **Power (Global Route):** Total: 6.70e-06 W (Sequential: 68.8%, Combinational: 31.2%).

- **Detailed Routing (TritonRoute):**
  - Iteratively routed the design, fixing violations.
  - Final DRC check reported **0 violations**.
  - Final wirelength: **20420 µm**.
  - Total vias: **5419**.
  - No antenna violations reported after detailed routing.

4.6 Post-Routing and Finishing (OpenROAD)

- **Filler Cell Insertion:** 1014 filler instances (`sg13g2_fill_1`, `sg13g2_fill_2`, `sg13g2_decap_4`, `sg13g2_decap_8`) were placed.
- **Density Fill (Metal Fill):** Metal fill was added to layers Metal1 through TopMetal2 to meet density rules.
- **Parasitic Extraction (RCX):** Parasitics were extracted for final timing and power analysis. 5907 wires, 4757 RC segments, 4757 caps, 6267 coupling caps extracted.
- **IR Drop Analysis (PDNSim):**
  - VDD: Worst-case IR drop of 1.56e-06 V.
  - VSS: Worst-case IR drop of 3.36e-06 V.
  - Both negligible.
- **Final Design Area:** 20824 μm^2, **94% utilization** (due to filler cells).
- **Final Cell Count (including fillers):** 1944 instances. Detailed breakdown from final report:

sg13g2_a21o_1	9
sg13g2_a21oi_1	64
sg13g2_a221oi_1	4
sg13g2_a22oi_1	12
sg13g2_and2_1	8
sg13g2_and2_2	1
sg13g2_and3_1	5
sg13g2_and4_1	4
sg13g2_buf_1	67
sg13g2_buf_2	8
sg13g2_buf_4	2
sg13g2_buf_8	1
sg13g2_dfrbp_1	153
sg13g2_inv_1	39
sg13g2_mux2_1	53
sg13g2_nand2_1	45
sg13g2_nand2_2	1
sg13g2_nand2b_1	13

sg13g2_nand3_1	17
sg13g2_nand3b_1	12
sg13g2_nand4_1	16
sg13g2_nor2_1	72
sg13g2_nor2_2	2
sg13g2_nor2b_1	22
sg13g2_nor3_1	10
sg13g2_nor4_1	1
sg13g2_nor4_2	1
sg13g2_o21ai_1	70
sg13g2_or2_1	7
sg13g2_or3_1	2
sg13g2_or4_1	2
sg13g2_tiehi	1
sg13g2_tielo	1
sg13g2_xnor2_1	44
sg13g2_xor2_1	15

(Plus buffer cells inserted during PnR and filler cells)

- **Timing (Finish):** TNS: 0.00, WNS: 0.00, Worst Slack: INF.
- **Physical Checks (Finish):** No violations for max slew, max capacitance, or max fanout. Setup/Hold violation counts were 0.
- **Power (Finish):** Total: **7.08e-06 W** (Sequential: 65.4%, Combinational: 29.4%, Clock: 5.2%, Leakage: 7.7% of total). The clock power is now non-zero, likely due to RC extraction attributing some leakage to clock nets/buffers even if not fully switching.

4.7 GDSII Generation (KLayout)

- The final DEF file (`6_final.def`) was merged with the standard cell GDS (`sg13g2_stdcell.gds`) using KLayout's `def2stream.py` script.
- The output was `6_final.gds`.

5 Results

The project successfully completed the RTL-to-GDSII flow.

5.1 Layout Generation

A GDSII file (`6_final.gds`) representing the physical layout of the "project" design was successfully generated.

5.2 Timing Analysis

- **TNS (Total Negative Slack):** 0.00 ps across all stages.
- **WNS (Worst Negative Slack):** 0.00 ps across all stages.
- **Worst Slack:** INF (Infinity) across all stages. This indicates that while there are no violations on *constrained* paths, many paths in the design remain unconstrained.

- **Unconstrained Paths:** Examples were reported throughout the flow.
  - Floorplan Stage: `ui_in[0]` to `MVM_UART_SYSTEM.UART_RX.m_data[15]$_DFFE_PNOP_` (data arrival 25.09 ns).
  - Finish Stage: `rst_n` to `MVM_UART_SYSTEM.AXIS_MVM.SKID.m_data[11]$_DFFE_PNOP_` (data arrival 25.91 ns).
  - The consistent "No paths found" for `report_checks -path_delay min/max` and `reg to reg` checks further confirms the lack of comprehensive constraints.
- **Clock Skew:** The `core_clock` reported "No launch/capture paths found" or "No valid clock nets" throughout CTS and later stages, implying it was not properly synthesized as a propagating clock tree for timing analysis of synchronous paths.

5.3 Physical Verification (DRC, LVS-implied)

- **DRC (Design Rule Checks):**
  - Max Slew Violation Count: 0
  - Max Fanout Violation Count: 0
  - Max Capacitance Violation Count: 0
  - Setup Violation Count: 0
  - Hold Violation Count: 0
  - Detailed router reported 0 DRC violations in the final layout.
- **LVS (Layout vs. Schematic):** While not explicitly run as a separate step in the provided logs, the OpenROAD flow's correctness and KLayout's GDS merge imply a structurally correct netlist was physically implemented. A formal LVS check would be a recommended next step.

5.4 Power Analysis (Finish Stage)

- **Total Power:** 7.08e-06 Watts
  - Internal Power: 5.16e-06 W (72.9%)
  - Switching Power: 1.37e-06 W (19.4%)
  - Leakage Power: 5.45e-07 W (7.7%)
- **Breakdown by Component Type (Finish Stage):**
  - Sequential: 4.63e-06 W (65.4%)
  - Combinational: 2.08e-06 W (29.4%)
  - Clock: 3.69e-07 W (5.2%)
  - Macro/Pad: 0.00 W

5.5 Resource Utilization

- **Total Instances (Final, including fillers):** 1944
  - Standard Cells (excluding fillers, post-PnR): 930
  - Sequential Cells (`sg13g2_dfrbp_1`): 153
- **Design Area:** 20824 μm^2

- **Core Utilization (Final):** 94% (Initial floorplan was 61% for standard cells before fillers).
- **Total Wirelength (Detailed Route):** 20420 µm
- **Total Vias (Detailed Route):** 5419

5.6 IR Drop Analysis

- **VDD Net:**
  - Average IR Drop: 1.32e-06 V
  - Worst-case IR Drop: 1.56e-06 V (Negligible)
- **VSS Net:**
  - Average IR Drop: 3.05e-06 V
  - Worst-case IR Drop: 3.36e-06 V (Negligible)

5.7 Visual Layouts

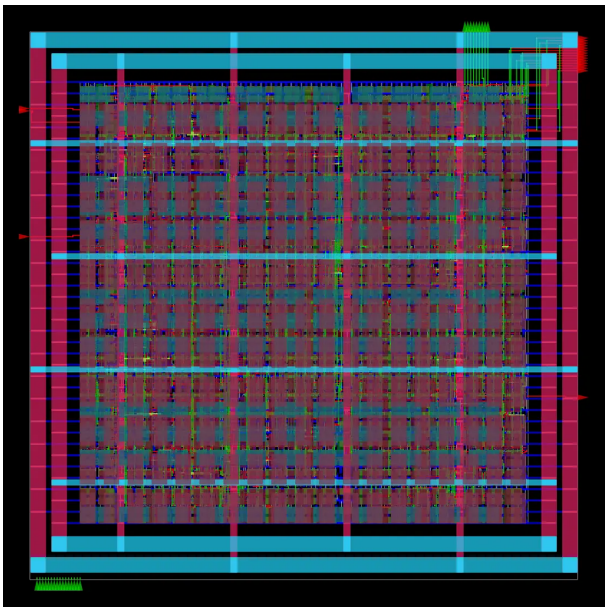

Figure 2: Complete GDSII layout view (Placeholder: final_all.png)

Figure 3: Cell placement view, pre-routing (Placeholder: final_placement.png)

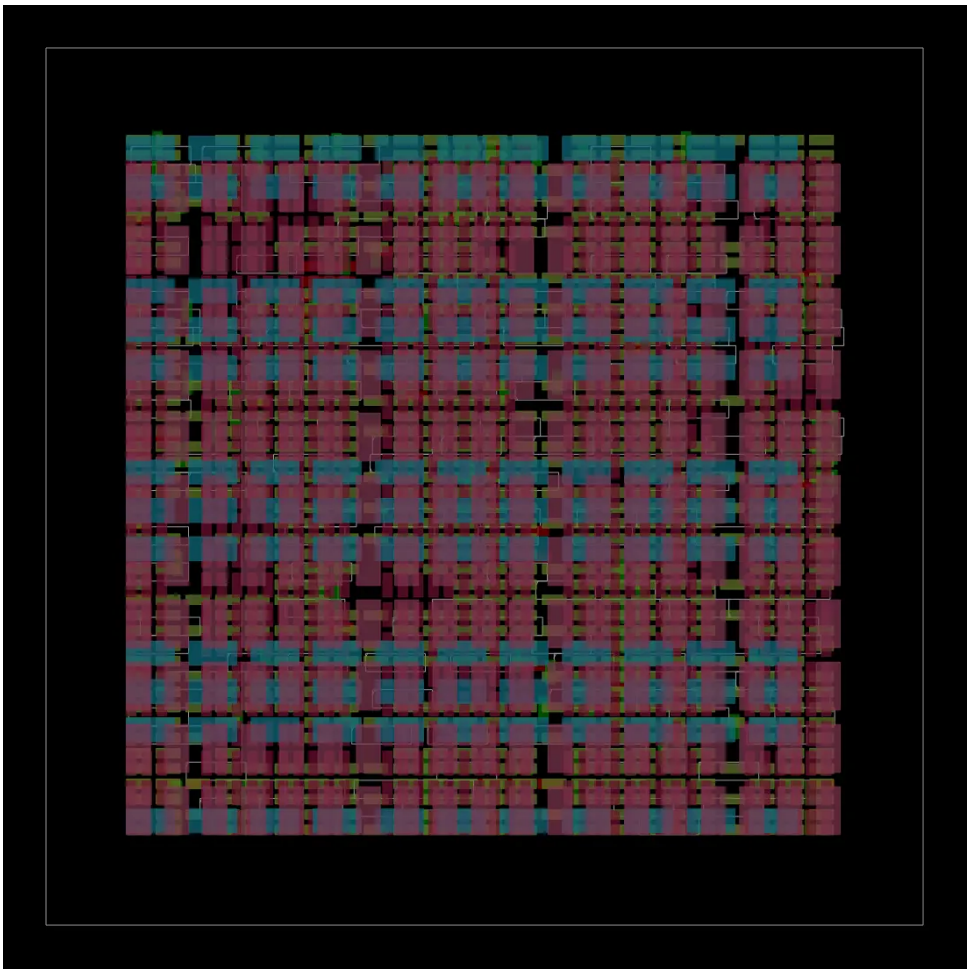

Figure 4: Clock net highlighting, if available (Placeholder: final_clocks.png)

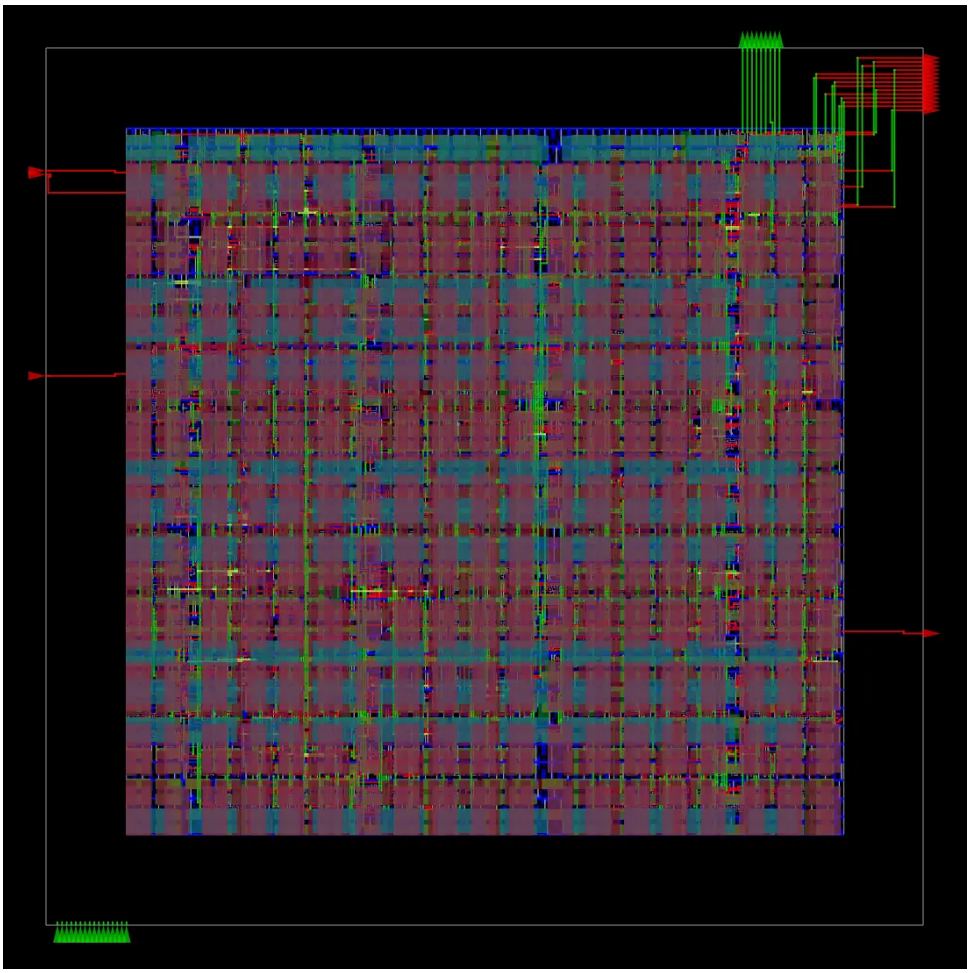

Figure 5: Zoomed-in view showing routing tracks (Placeholder: final_routing.png)

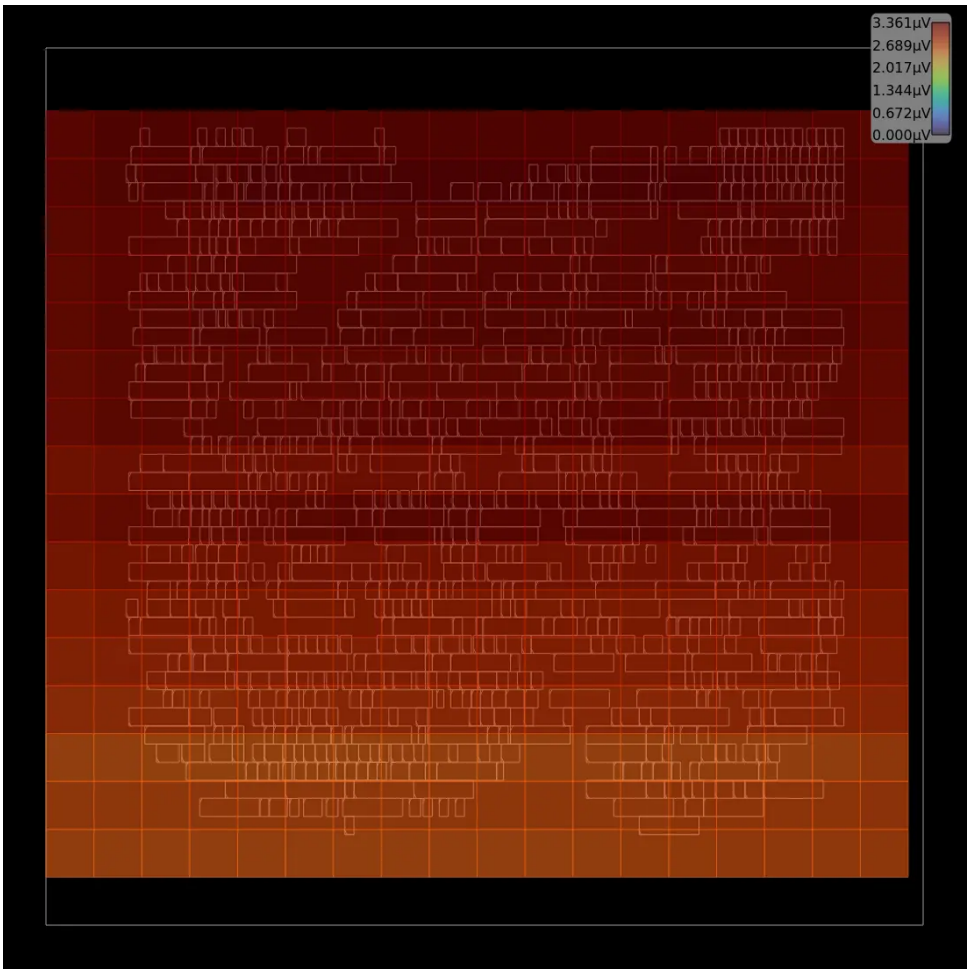

Figure 6: VDD/VSS IR drop heatmaps (Placeholder: final_ir_drop.png)

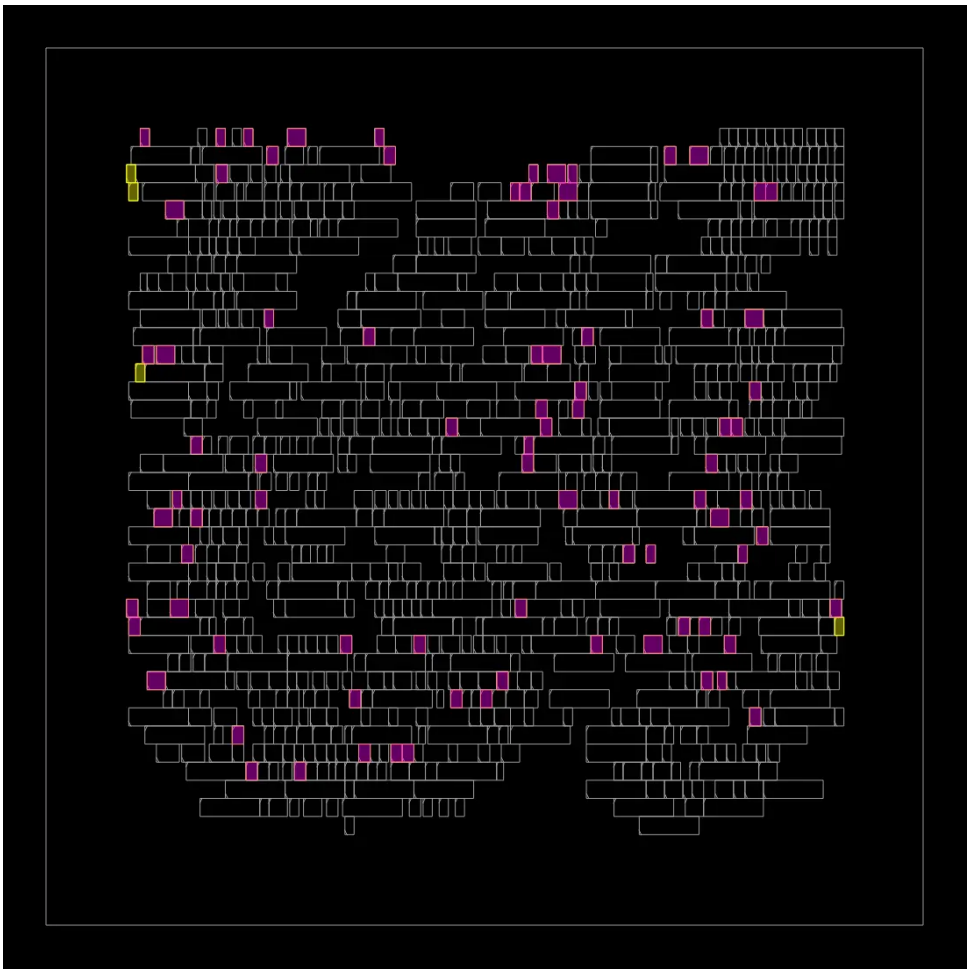

Figure 7: Conceptual view of resizer impact (Placeholder: final_resizer.png)

6 Verilog Code

6.1 skid_buffer Module

```
/*
 * Basic building blocks first
 */

module skid_buffer #(parameter WIDTH = 8)(
  input  wire clk, rstn, s_valid, m_ready,
  input  wire [WIDTH-1:0] s_data,
  output reg  [WIDTH-1:0] m_data,
  output reg  m_valid, s_ready
);
  localparam EMPTY=0, PARTIAL=1, FULL=2;
  reg [1:0] state, state_next;

  always @* begin
    state_next = state;
    unique case (state)
      EMPTY  : if      ( s_valid)             state_next = PARTIAL;
      PARTIAL: if      (!m_ready &&  s_valid) state_next = FULL;
               else if ( m_ready && !s_valid) state_next = EMPTY;
      FULL   : if      ( m_ready)             state_next = PARTIAL;
    endcase
  end
  always @(posedge clk or negedge rstn)
    if (!rstn) state <= EMPTY;
    else       state <= state_next;

  reg [WIDTH-1:0] buffer;
  always @(posedge clk or negedge rstn)
    if (!rstn) {m_valid, s_ready, buffer, m_data} <= 0;
    else begin

      m_valid <= state_next != EMPTY;
      s_ready <= state_next != FULL;

      if (state == PARTIAL && state_next == FULL)
        buffer <= s_data;

      unique case (state)
        EMPTY  : if (state_next == PARTIAL) m_data <= s_data;
        PARTIAL: if (m_ready && s_valid)    m_data <= s_data;
        FULL   : if (m_ready)               m_data <= buffer;
      endcase
    end
endmodule
```

Listing 1: Skid Buffer Module

6.2 matvec_mul Module

```
/*
 * Matrix-Vector multiplication module
 */

module matvec_mul #(
    parameter R=8, C=8, W_X=8, W_K=8,
    localparam DEPTH = $clog2(C),
               W_M = W_X + W_K,
               W_Y = W_M + DEPTH
```

```
  )(
    input  wire  clk, cen,
    input  wire  [R*C*W_K-1:0] kf,
    input  wire  [C*W_X  -1:0] xf,
    output wire  [R*W_Y  -1:0] yf
  );

  // Padding
  localparam C_PAD = 2**$clog2(C);
  reg [W_Y-1:0] tree [R][DEPTH+1][C_PAD]; // adder tree

  wire signed [W_X-1:0] x_pad        [C_PAD-1:0];
  wire signed [W_K-1:0] k_pad [R-1:0][C_PAD-1:0];

  genvar r, c, d, a;
  generate

    for (c=0; c<C_PAD; c=c+1) begin
      assign x_pad[c] = (c < C) ? xf[(c+1)*W_X-1 : c*W_X] : 0;
      for (r=0; r<R; r=r+1)
        assign k_pad[r][c] = (c < C) ? kf[((r*C + c) + 1)*W_K-1 : (r*C + c)*W_K
    ] : 0;
    end

    for (r=0; r<R; r=r+1) begin
      for (c=0; c<C_PAD; c=c+1)
        always @(posedge clk)  // register after each mul
          if (cen) tree[r][0][c] <= $signed(k_pad[r][c]) * $signed(x_pad[c]);

      for (d=0; d<DEPTH; d=d+1)
        for (a=0; a<C_PAD/2**(d+1); a=a+1)
          always @(posedge clk)  // register after each add
            if (cen) tree [r][d+1][a] <= tree [r][d][2*a] + tree [r][d][2*a+1];

      assign yf[(r+1)*W_Y-1: r*W_Y] = tree [r][DEPTH][0];
    end
  endgenerate
endmodule
```

Listing 2: Matrix-Vector Multiplication Module

6.3 `uart_tx` Module

```
/*
 * UART modules
 */

module uart_tx #(
  parameter    CLOCKS_PER_PULSE = 4, //200_000_000/9600
               BITS_PER_WORD    = 8,
               PACKET_SIZE      = BITS_PER_WORD+5,
               W_OUT = 24, //R*C*W_K + C*W_X

  localparam   NUM_WORDS    = W_OUT/BITS_PER_WORD
)(
  input  wire clk, rstn, s_valid,
  input  wire [NUM_WORDS*BITS_PER_WORD-1:0] s_data_f,
  output reg  tx, s_ready
);

  genvar n;
  wire [BITS_PER_WORD-1:0] s_data [NUM_WORDS-1:0];
```

```
20
21  generate
22    for (n=0; n<NUM_WORDS; n=n+1)
23      assign s_data[n] = s_data_f[BITS_PER_WORD*(n+1)-1 : BITS_PER_WORD*n];
24  endgenerate
25
26  localparam END_BITS = PACKET_SIZE-BITS_PER_WORD-1;
27  reg [NUM_WORDS*PACKET_SIZE-1:0] s_packets;
28  reg [NUM_WORDS*PACKET_SIZE-1:0] m_packets;
29
30  generate
31    for (n=0; n<NUM_WORDS; n=n+1)
32      assign s_packets[PACKET_SIZE*(n+1)-1:PACKET_SIZE*n] = { ~(END_BITS'(0)),
    s_data[n], 1'b0};
33  endgenerate
34
35  assign tx = m_packets[0];
36
37  // Counters
38  localparam W_CPULSES = $clog2(NUM_WORDS*PACKET_SIZE),
39             W_CCLOCKS = $clog2(CLOCKS_PER_PULSE);
40
41  reg [W_CPULSES-1:0] c_pulses;
42  reg [W_CCLOCKS-1:0] c_clocks;
43
44  // State Machine
45
46  localparam IDLE=0, SEND=1;
47  reg state;
48
49  always @(posedge clk or negedge rstn) begin
50
51    if (!rstn) begin
52      state     <= IDLE;
53      m_packets <= -1;
54      {c_pulses, c_clocks} <= 0;
55    end else
56      case (state)
57        IDLE :  if (s_valid) begin
58                  state      <= SEND;
59                  m_packets  <= s_packets;
60                end
61
62        SEND :  if (c_clocks == W_CCLOCKS'(CLOCKS_PER_PULSE-1)) begin
63                  c_clocks <= 0;
64
65                  if (c_pulses == W_CPULSES'(NUM_WORDS*PACKET_SIZE-1)) begin
66                    c_pulses  <= 0;
67                    m_packets <= -1;
68                    state     <= IDLE;
69
70                  end else begin
71                    c_pulses <= c_pulses + 1;
72                    m_packets <= (m_packets >> 1);
73                  end
74
75                end else c_clocks <= W_CCLOCKS'(32'(c_clocks) + 1);
76      endcase
77  end
78
79  assign s_ready = (state == IDLE);
80
```

```
endmodule
```

Listing 3: UART Transmitter Module

6.4 uart_rx Module

```
module uart_rx #(
  parameter CLOCKS_PER_PULSE = 4, //200_000_000/9600
            BITS_PER_WORD    = 8,
            W_OUT = 24 //R*C*W_K + C*W_X,
)(
  input  wire clk, rstn, rx,
  output reg  m_valid,
  output reg  [W_OUT-1:0] m_data
);
  localparam  NUM_WORDS = W_OUT/BITS_PER_WORD;

  // Counters

  localparam W_CCLOCKS = $clog2(CLOCKS_PER_PULSE),
             W_CBITS   = $clog2(BITS_PER_WORD),
             W_CWORDS  = $clog2(NUM_WORDS);

  reg [W_CCLOCKS-1:0] c_clocks;
  reg [W_CBITS  -1:0] c_bits  ;
  reg [W_CWORDS -1:0] c_words ;

  // State Machine


  localparam IDLE=0, START=1, DATA=2, END=3;
  reg [1:0] state;

  always @(posedge clk or negedge rstn) begin

    if (!rstn) begin
      {c_words, c_bits, c_clocks, m_valid, m_data} <= '0;
      state <= IDLE;
    end else begin
      m_valid <= 0;

      case (state)

        IDLE :  if (rx == 0)
                  state <= START;

        START:  if (c_clocks == W_CCLOCKS'(CLOCKS_PER_PULSE/2-1)) begin
                  state <= DATA;
                  c_clocks <= 0;
                end else
                  c_clocks <= c_clocks + 1;

        DATA :  if (c_clocks == W_CCLOCKS'(CLOCKS_PER_PULSE-1)) begin
                  c_clocks <= 0;
                  m_data   <= {rx, m_data[W_OUT-1:1]};

                  if (c_bits == W_CBITS'(BITS_PER_WORD-1)) begin
                    state  <= END;
                    c_bits <= 0;

                    if (c_words == W_CWORDS'(NUM_WORDS-1)) begin
                      m_valid <= 1;
```

```
                        c_words <= 0;

                      end else c_words <= c_words + 1;
                    end else c_bits <= c_bits + 1;
                  end else c_clocks <= c_clocks + 1;

          END   :  if (c_clocks == W_CCLOCKS'(CLOCKS_PER_PULSE-1)) begin
                     state <= IDLE;
                     c_clocks <= 0;
                   end else
                     c_clocks <= c_clocks + 1;
        endcase
        end
    end
endmodule
```

Listing 4: UART Receiver Module

6.5 `axis_matvec_mul` Module

```
/*
 * AXI Stream Matrix-Vector multiplication module
 */

module axis_matvec_mul #(
    parameter R=8, C=8, W_X=8, W_K=8,
              LATENCY = $clog2(C)+1,
              W_Y = W_X + W_K + $clog2(C)
  )(
    input  clk, rstn,
    output s_axis_kx_tready,
    input  s_axis_kx_tvalid,
    input  [R*C*W_K + C*W_X -1:0] s_axis_kx_tdata,
    input  m_axis_y_tready,
    output m_axis_y_tvalid,
    output [R*W_Y           -1:0] m_axis_y_tdata
  );
    wire [R*C*W_K-1:0] k;
    wire [C*W_X  -1:0] x;
    assign {k, x} = s_axis_kx_tdata;

    wire [R*W_Y-1:0] i_data;
    wire i_ready;

    matvec_mul #(
      .R(R), .C(C), .W_X(W_X), .W_K(W_K)
    ) MATVEC (
      .clk(clk    ),
      .cen(i_ready),
      .kf (k      ),
      .xf (x      ),
      .yf (i_data )
    );

    reg [LATENCY-2:0] shift;
    reg  i_valid;

    always @(posedge clk or negedge rstn)
      if     (!rstn)   {i_valid, shift} <= 0;
      else if(i_ready) {i_valid, shift} <= {shift, s_axis_kx_tvalid};

    skid_buffer #(.WIDTH(R*W_Y)) SKID (
```

```
      .clk     (clk    ),
      .rstn    (rstn   ),
      .s_ready (i_ready),
      .s_valid (i_valid),
      .s_data  (i_data ),
      .m_ready (m_axis_y_tready),
      .m_valid (m_axis_y_tvalid),
      .m_data  (m_axis_y_tdata )
    );

    assign s_axis_kx_tready = i_ready;

endmodule
```

Listing 5: AXI Stream Matrix-Vector Multiplication Wrapper

6.6 `mvm_uart_system` Module

```
/*
 * Integration of UART and Matrix-Vector multiplication
 */

module mvm_uart_system #(
  parameter CLOCKS_PER_PULSE = 200_000_000/9600, //200_000_000/9600
            BITS_PER_WORD    = 8,
            PACKET_SIZE_TX   = BITS_PER_WORD + 5,
            W_Y_OUT          = 32,
            R=8, C=8, W_X=8, W_K=8
)(
  input  clk, rstn, rx,
  output tx
);

  localparam  W_BUS_KX = R*C*W_K + C*W_X,
              W_BUS_Y  = R*W_Y_OUT,
              W_Y      = W_X + W_K + $clog2(C);

  wire s_valid, m_valid, m_ready;
  wire [W_BUS_KX-1:0] s_data_kx;

  uart_rx #(
    .CLOCKS_PER_PULSE (CLOCKS_PER_PULSE),
    .BITS_PER_WORD (BITS_PER_WORD),
    .W_OUT (W_BUS_KX)
  ) UART_RX (
    .clk    (clk),
    .rstn   (rstn),
    .rx     (rx),
    .m_valid(s_valid),
    .m_data (s_data_kx)
  );

  wire [R*W_Y  -1:0] m_data_y;
  wire s_ready;
  wire _unused = s_ready; // Suppress unused warning if s_ready isn't read
    elsewhere
  axis_matvec_mul #(
    .R(R), .C(C), .W_X(W_X), .W_K(W_K)
  ) AXIS_MVM (
    .clk    (clk      ),
    .rstn   (rstn     ),
    .s_axis_kx_tready(s_ready), // assume always valid
```

```
    .s_axis_kx_tvalid(s_valid  ),
    .s_axis_kx_tdata (s_data_kx),
    .m_axis_y_tready (m_ready  ),
    .m_axis_y_tvalid (m_valid  ),
    .m_axis_y_tdata  (m_data_y )
  );

  // Padding to 32 bits to be read in computer
  wire [W_Y      -1:0] y_up     [R-1:0];
  wire [W_Y_OUT  -1:0] o_up   [R-1:0];
  wire [R*W_Y_OUT-1:0] o_flat;

  genvar r_gen; // Changed genvar name to avoid conflict with wire r if any
  generate
  for (r_gen=0; r_gen<R; r_gen=r_gen+1) begin : gen_padding
    assign y_up  [r_gen] = m_data_y[W_Y*(r_gen+1)-1 : W_Y*r_gen];
    assign o_up  [r_gen] = W_Y_OUT'($signed(y_up[r_gen])); // sign extend to 32
    b
    assign o_flat[W_Y_OUT*(r_gen+1)-1 : W_Y_OUT*r_gen] = o_up[r_gen];
    // assign o_flat[W_Y_OUT*(r_gen+1)-1 : W_Y_OUT*r_gen] = $signed(m_data_y[
    W_Y*(r_gen+1)-1 : W_Y*r_gen]);
  end
  endgenerate

  uart_tx #(
   .CLOCKS_PER_PULSE (CLOCKS_PER_PULSE),
   .BITS_PER_WORD    (BITS_PER_WORD),
   .PACKET_SIZE      (PACKET_SIZE_TX),
   .W_OUT            (W_BUS_Y)
  ) UART_TX (
   .clk     (clk      ),
   .rstn    (rstn     ),
   .s_ready (m_ready  ),
   .s_valid (m_valid  ),
   .s_data_f(o_flat   ),
   .tx      (tx       )
  );

endmodule
```

Listing 6: MVM UART System Integration Module

6.7 `project` (Top-Level Module)

```
/*
 * Top-level module
 */

`default_nettype none

module project (
    input  wire [7:0] ui_in,    // Dedicated inputs
    output wire [7:0] uo_out,   // Dedicated outputs
    input  wire [7:0] uio_in,   // IOs: Input path
    output wire [7:0] uio_out,  // IOs: Output path
    output wire [7:0] uio_oe,   // IOs: Enable path (active high: 0=input, 1=
    output)
    input  wire       ena,      // always 1 when the design is powered, so you
    can ignore it
    input  wire       clk,      // clock
    input  wire       rst_n     // reset_n - low to reset
);
```

```
17
18    localparam
19      CLOCKS_PER_PULSE = 50_000_000/19200,
20      BITS_PER_WORD    = 8                 ,
21      PACKET_SIZE_TX   = BITS_PER_WORD + 5,
22      W_Y_OUT          = 8                 ,
23      R                = 2                 ,
24      C                = 2                 ,
25      W_X              = 4                 ,
26      W_K              = 2                 ;
27
28    mvm_uart_system #(
29      .CLOCKS_PER_PULSE (CLOCKS_PER_PULSE),
30      .BITS_PER_WORD    (BITS_PER_WORD   ),
31      .PACKET_SIZE_TX   (PACKET_SIZE_TX  ),
32      .W_Y_OUT          (W_Y_OUT         ),
33      .R                (R               ),
34      .C                (C               ),
35      .W_X              (W_X             ),
36      .W_K              (W_K             )
37    ) MVM_UART_SYSTEM (
38      .clk (clk),
39      .rstn(rst_n),
40      .rx  (ui_in [0]),
41      .tx  (uo_out[0])
42    );
43
44    // All output pins must be assigned. If not used, assign to 0.
45    assign uo_out[7:1] = 0;
46    assign uio_out = 0;
47    assign uio_oe  = 0;
48
49    // List all unused inputs to prevent warnings
50    wire _unused_inputs; // Changed name for clarity
51    assign _unused_inputs = &{uio_in, ui_in[7:1], ena, clk, rst_n, 1'b0};
52
53  endmodule
```

Listing 7: Top-Level Project Module

7 Conclusion

This project successfully demonstrated the capability of the OSIC open-source tools, particularly the OpenROAD flow with the IHP SG13G2 PDK, to take a Verilog design ("project") from RTL to a GDSII layout. The flow completed without DRC violations, and the final design showed negligible IR drop.

Key Achievements:

- Successful execution of the entire RTL-to-GDSII flow using open-source tools.
- Generation of a DRC-clean layout.
- Achieved 0 TNS and 0 WNS for the constrained parts of the design.
- Detailed power and resource utilization metrics were obtained.

Challenges and Observations:

1. **Unconstrained Design:**
    - `Worst Slack:  INF` throughout the flow.

- Numerous unconstrained path reports.
- The `core_clock` not being properly propagated by CTS ("No valid clock nets").
- The initial STA warning about "port 'clock_i' not found."

This implies that the SDC file lacked proper clock definitions (e.g., `create_clock` on an actual input port, not just a virtual clock), input/output delay constraints, and potentially timing exceptions.

2. **Initial Utilization Error:** An early attempt failed due to placement utilization exceeding 100% (reported as 224%). This was rectified in the successful run by adjusting floorplan parameters, leading to an initial 61% utilization for standard cells. This highlights the sensitivity of placement to initial floorplan density.

3. **High Final Utilization:** The final utilization of 94% after filler cell insertion is quite high and could pose challenges for routability or future engineering change orders (ECOs) in more complex designs.

4. **Synthesis Warnings:** Yosys reported warnings about continuous assignments to `reg` types in the RTL, which, while handled, could be improved by using `wire` or proper procedural blocks. The replacement of memory `\tree` with registers should also be noted as a synthesis decision.