# miniRISC Processor

ASIC Implementation Report
SkyWater 130nm Technology

OpenROAD Flow Scripts (ORFS)

December 26, 2025

## Contents

# 1 Introduction

This report documents the ASIC implementation of the miniRISC processor using OpenROAD Flow Scripts (ORFS) targeting the SkyWater 130nm high-density (sky130hd) technology node. The design achieves a maximum frequency of 256 MHz with zero timing and DRC violations.

## 1.1 Design Overview

Table 1: Design Summary

| Parameter | Value |
|---|---|
| Design Name | miniRISC |
| Technology | SkyWater sky130hd (130nm) |
| Clock Frequency | 256.21 MHz |
| Die Area | 12,678.8 $\mu m^2$ |
| Core Utilization | 63.96% |
| Total Power | 1.31 mW |
| Standard Cells | 1,186 |

# 2 Processor Architecture

The miniRISC is a 32-bit lightweight RISC processor implemented in Verilog HDL. It employs a Harvard architecture with separate memories for instructions and data, controlled by a finite state machine (FSM) for sequential instruction execution.

## 2.1 Architectural Features

The processor's datapath centers around a 32-bit Instruction Register (IR) that holds the current instruction being executed. The IR is divided into fields for the opcode, destination register, source registers, mode selection, and immediate data. A register file provides 32 general-purpose registers (GPR), each 16 bits wide, allowing for flexible data manipulation. Additionally, a Special General Purpose Register (SGPR) captures the high-order 16 bits of multiplication results, enabling 32-bit multiplication output.

The arithmetic and logic unit (ALU) performs all computational operations including addition, subtraction, multiplication, and bitwise logical operations. The processor employs a Harvard architecture with physically separate instruction memory (32-bit wide) and data memory (16-bit wide), allowing simultaneous instruction fetch and data access in different clock cycles.

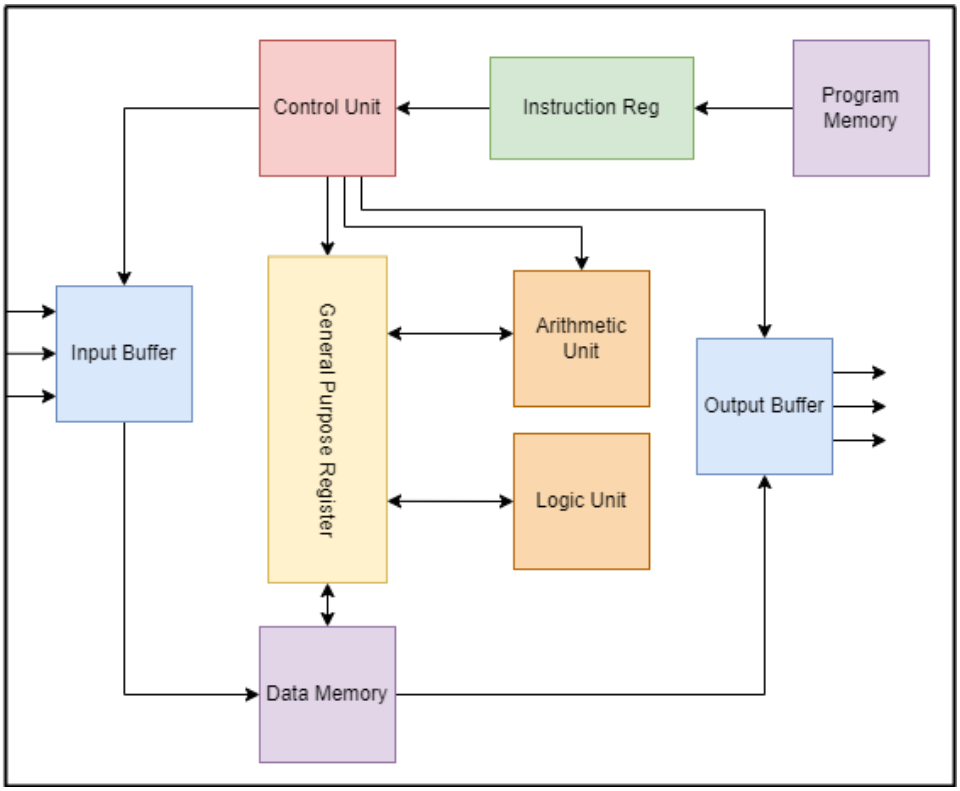



Figure 1: miniRISC Processor Block Diagram. The diagram illustrates the Harvard architecture with separate instruction and data memory paths, the central register file, ALU, and the FSM-based control unit that orchestrates instruction execution.

## 2.2 Instruction Format

The processor uses a 32-bit fixed-length instruction format, which simplifies instruction fetch and decode logic. All instructions share a common upper portion containing the opcode and register specifiers, while the lower portion varies based on the addressing mode. Table 2 shows the field allocation within the 32-bit instruction word.

Table 2: 32-bit Instruction Format

| **Bits** | 31:27 | 26:22 | 21:17 | 16 |
|---|---|---|---|---|
| **Field** | OPCODE | RDST | RSRC1 | MODE |
| **Width** | 5 bits | 5 bits | 5 bits | 1 bit |

The 5-bit opcode field (bits 31:27) supports up to 32 distinct operations. The destination register (RDST) and first source register (RSRC1) are each specified by 5-bit fields, allowing access to all 32 GPRs. The MODE bit (bit 16) determines whether the instruction uses register-to-register (R-Type) or register-immediate (I-Type) addressing.

In R-Type format (MODE = 0), the second source operand comes from a register specified in bits 15:11, with the remaining bits unused. This format is used for operations between two registers, such as `ADD R2, R0, R1`.

```
R-Type: [31:27] [26:22] [21:17] [16] [15:11] [10:0]
         OPCODE   RDST    RSRC1   0    RSRC2   unused
```

In I-Type format (MODE = 1), the entire lower 16 bits serve as an immediate value, allowing constants up to 65535 to be encoded directly in the instruction. This format supports operations like `MOV R0, #5` or `ADD R1, R1, #10`.

```
I-Type: [31:27] [26:22] [21:17] [16] [15:0]
         OPCODE   RDST    RSRC1   1   IMMEDIATE (16-bit)
```

### 2.2.1 Opcode Encoding

Table 3 presents the complete opcode encoding for all 26 instructions. The instruction set is organized into four categories: arithmetic operations for mathematical computations, logical operations for bitwise manipulation, memory operations for data transfer between registers and memory, and control flow operations for program sequencing.

Arithmetic operations (opcodes 00000-00100) include basic mathematical functions. The MOVSGPR instruction transfers the special register contents to a GPR, useful for retrieving the upper 16 bits after multiplication. MOV provides data movement between registers or from immediate values. ADD, SUB, and MUL perform addition, subtraction, and multiplication respectively, with the MUL instruction storing the upper 16 bits of the 32-bit product in SGPR.

Logical operations (opcodes 00101-01011) provide a complete set of bitwise functions including OR, AND, XOR, XNOR, NAND, NOR, and NOT. These operations work on 16-bit operands and support both register and immediate addressing modes.

Memory operations (opcodes 01101-10001) facilitate data transfer between the processor and external memory. STOREREG and STOREDIN write register contents or input bus data to memory, while SENDDOUT and SENDREG read from memory to the output bus or registers.

Control flow operations (opcodes 10010-11011) manage program execution. The unconditional JMP instruction transfers control to an absolute address. Conditional jumps (JC, JNC, JS, JNS, JZ, JNZ, JO, JNO) branch based on the state of condition flags, enabling implementation of loops and conditional logic. The HALT instruction terminates program execution.

Table 3: Complete Opcode Table

| Opcode | Binary | Mnemonic | Operation |
|---|---|---|---|
| *Arithmetic Operations* | | | |
| 00000 | 5'b00000 | MOVSGPR | RDST ← SGPR |
| 00001 | 5'b00001 | MOV | RDST ← RSRC1 or IMM |
| 00010 | 5'b00010 | ADD | RDST ← RSRC1 + RSRC2/IMM |
| 00011 | 5'b00011 | SUB | RDST ← RSRC1 - RSRC2/IMM |
| 00100 | 5'b00100 | MUL | RDST ← RSRC1 × RSRC2/IMM |
| *Logical Operations* | | | |
| 00101 | 5'b00101 | OR | RDST ← RSRC1 \| RSRC2/IMM |
| 00110 | 5'b00110 | AND | RDST ← RSRC1 & RSRC2/IMM |
| 00111 | 5'b00111 | XOR | RDST ← RSRC1 ⊕ RSRC2/IMM |
| 01000 | 5'b01000 | XNOR | RDST ← RSRC1 ⊙ RSRC2/IMM |
| 01001 | 5'b01001 | NAND | RDST ← ∼(RSRC1 & RSRC2/IMM) |
| 01010 | 5'b01010 | NOR | RDST ← ∼(RSRC1 \| RSRC2/IMM) |
| 01011 | 5'b01011 | NOT | RDST ← ∼RSRC1/IMM |
| *Memory Operations* | | | |
| 01101 | 5'b01101 | STOREREG | MEM[IMM] ← GPR[RSRC1] |
| 01110 | 5'b01110 | STOREDIN | MEM[IMM] ← DIN |
| 01111 | 5'b01111 | SENDDOUT | DOUT ← MEM[IMM] |
| 10001 | 5'b10001 | SENDREG | RDST ← MEM[IMM] |
| *Control Flow* | | | |
| 10010 | 5'b10010 | JMP | PC ← IMM (unconditional) |
| 10011 | 5'b10011 | JC | Jump if Carry = 1 |
| 10100 | 5'b10100 | JNC | Jump if Carry = 0 |
| 10101 | 5'b10101 | JS | Jump if Sign = 1 |
| 10110 | 5'b10110 | JNS | Jump if Sign = 0 |
| 10111 | 5'b10111 | JZ | Jump if Zero = 1 |
| 11000 | 5'b11000 | JNZ | Jump if Zero = 0 |
| 11001 | 5'b11001 | JO | Jump if Overflow = 1 |
| 11010 | 5'b11010 | JNO | Jump if Overflow = 0 |
| 11011 | 5'b11011 | HALT | Stop execution |

## 2.3 Condition Flags

The processor maintains four condition flags that are updated by arithmetic operations and used by conditional branch instructions. These flags enable the implementation of comparison operations, loop constructs, and conditional execution paths.

The Carry flag (C) is set when an arithmetic operation produces a result that exceeds 16 bits, indicating unsigned overflow. This occurs when the sum of two unsigned numbers cannot be represented in 16 bits. The Sign flag (S) reflects the most significant bit (bit 15) of the result, indicating whether the result is negative in two's complement representation. The Zero flag (Z) is set when an operation produces a zero result, commonly used for equality comparisons and loop termination conditions. The Overflow flag (V) indicates signed arithmetic overflow, which occurs when the result of an operation on two signed numbers has an incorrect sign bit.

Table 4: Condition Flags

| Flag | Condition | Set When |
|---|---|---|
| Carry (C) | Unsigned overflow | Result $>$ 16 bits |
| Sign (S) | Negative result | Bit 15 of result = 1 |
| Zero (Z) | Zero result | Result = 0 |
| Overflow (V) | Signed overflow | Sign bit incorrect |

### 2.4 Instruction Set Summary

The miniRISC instruction set comprises 26 instructions organized into four functional categories, as summarized in Table 5. This instruction set provides sufficient functionality for general-purpose computing while maintaining simplicity in the decode and execution logic.

Table 5: Instruction Categories

| Category | Instructions | Count |
|---|---|---|
| Arithmetic | MOVSGPR, MOV, ADD, SUB, MUL | 5 |
| Logical | OR, AND, XOR, XNOR, NAND, NOR, NOT | 7 |
| Memory | STOREREG, STOREDIN, SENDDOUT, SENDREG | 4 |
| Control | JMP, JC, JNC, JS, JNS, JZ, JNZ, JO, JNO, HALT | 10 |
| **Total** | | **26** |

### 2.5 Finite State Machine

The processor employs a finite state machine (FSM) to control the sequential execution of instructions. Unlike pipelined architectures, this FSM-based approach executes one instruction at a time, simplifying the control logic and eliminating hazards at the cost of throughput. The FSM cycles through six states for each instruction, as illustrated in Figure 2.

Upon system reset, the FSM enters the Idle state, initializing all registers and control signals to their default values. The processor then transitions to the Fetch state, where the instruction at the current program counter (PC) address is loaded into the instruction register (IR). In the Decode and Execute state, the opcode is decoded and the corresponding operation is performed; arithmetic operations update the condition flags, and memory operations initiate data transfers. A Delay state introduces a fixed wait period of several clock cycles, ensuring timing constraints are met for memory operations and providing a consistent instruction execution time. The Next Instruction state updates the program counter, either incrementing it sequentially or loading a jump target address if a branch was taken. Finally, the Sense Halt state checks whether a HALT instruction was executed; if so, the processor remains in this state until reset, otherwise it returns to Fetch for the next instruction.

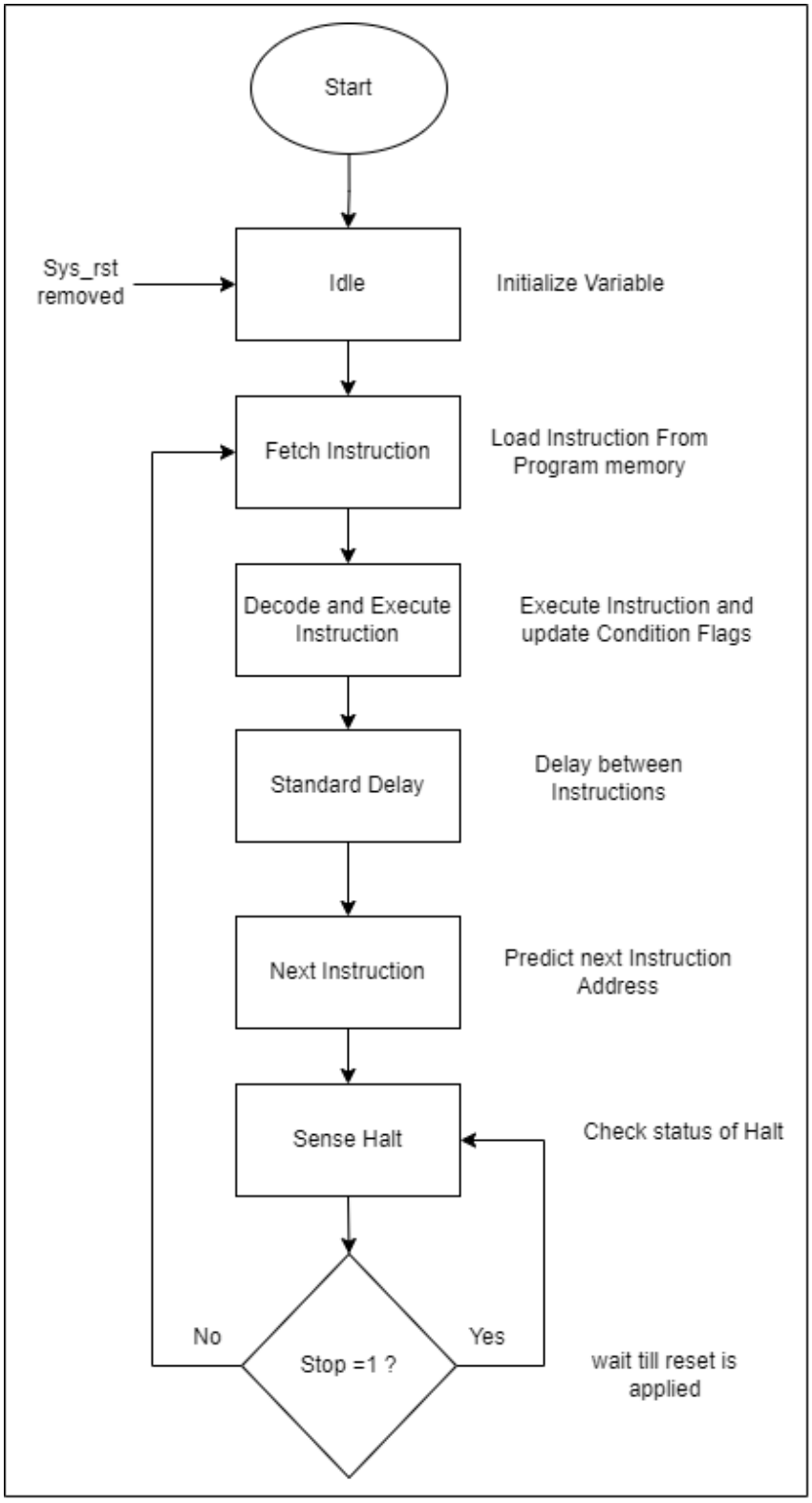



Figure 2: FSM State Diagram

### 2.6 I/O Ports

Table 6: I/O Port Specification

| Port | Width | Direction | Description |
|---|---|---|---|
| clk | 1 | Input | System clock |
| sys_rst | 1 | Input | Synchronous reset (active high) |
| din[15:0] | 16 | Input | Data input bus |
| dout[15:0] | 16 | Output | Data output bus |

## 3 Functional Verification

The processor functionality was verified using Verilog testbenches with automated checking.

### 3.1 Test Program

The verification program computes $5 \times 6 = 30$ using an iterative loop:

Table 7: Test Program Assembly

| Addr | Instruction | Encoding (32-bit) | Description |
|---|---|---|---|
| 0 | MOV R0, #5 | 00001_00000_00000_1_0000000000000101 | Load multiplier |
| 1 | MOV R1, #6 | 00001_00001_00000_1_0000000000000110 | Load multiplicand |
| 2 | MOV R2, #0 | 00001_00010_00000_1_0000000000000000 | Clear accumulator |
| 3 | MOV R3, #6 | 00001_00011_00000_1_0000000000000110 | Loop counter |
| 4 | ADD R2, R2, R0 | 00010_00010_00010_0_00000_00000000000 | Accumulate |
| 5 | SUB R3, R3, #1 | 00011_00011_00011_1_0000000000000001 | Decrement counter |
| 6 | JNZ 4 | 11000_00000_00000_0_0000000000000100 | Loop if R3 $\neq$ 0 |
| 7 | MOV R4, R2 | 00001_00100_00010_0_00000_00000000000 | Copy result |
| 8 | HALT | 11011_00000_00000_0_0000000000000000 | Stop execution |

### 3.2 Expected Results

Table 8: Expected Register Values After Execution

| Register | Expected Value | Description |
|---|---|---|
| GPR[0] | 5 | Multiplier (unchanged) |
| GPR[1] | 6 | Multiplicand (unchanged) |
| GPR[2] | 30 | Result: $5 \times 6 = 30$ |
| GPR[3] | 0 | Loop counter (decremented to 0) |
| GPR[4] | 30 | Copy of result |

## 3.3 Simulation Waveform

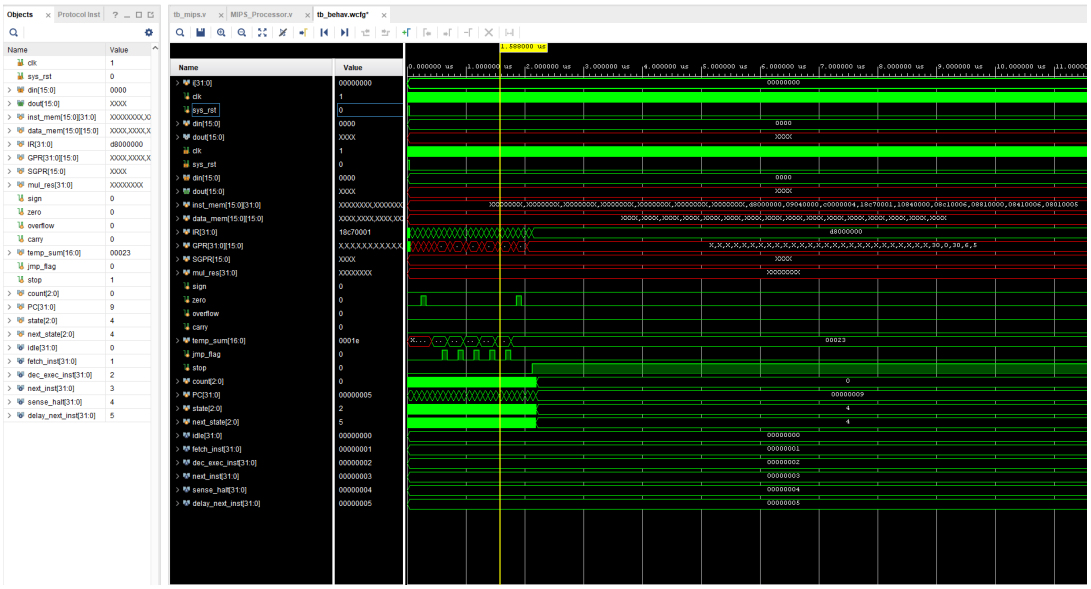


Figure 3: Simulation Waveform showing instruction execution

## 3.4 Verification Results

```
==========================================
miniRISC Synthesizable Design - Test Start
==========================================
Reset released at cycle 0
[Cycle 7] FETCH: PC=0 IR will be=08010005
[Cycle 14] EXEC: IR=08010005 oper=00001 GPR[0]=5 GPR[2]=0
...
HALT detected at cycle 287
------------------------------------------
Final Register Values:
  GPR[0] = 5 (expected: 5)
  GPR[1] = 6 (expected: 6)
  GPR[2] = 30 (expected: 30)
  GPR[3] = 0 (expected: 0)
  GPR[4] = 30 (expected: 30)
------------------------------------------
==========================================
TEST PASSED - All registers correct!
==========================================
```

Listing 1: Testbench Output

### 3.5 Testbench Coverage

Table 9: Verification Testbenches

| Testbench | File | Coverage |
|---|---|---|
| Functional Verify | tb_functional_verify.v | Full program execution |
| Arithmetic Unit | tb_arithmetic_unit.v | ADD, SUB, MUL operations |
| Logical Unit | tb_logical_unit.v | AND, OR, XOR, NOT operations |
| Conditional Flags | tb_conditional_flags.v | C, S, Z, V flag testing |
| Memory Interface | tb_data_&_inst_mem.v | Load/Store operations |

## 4 ASIC Implementation Flow

The ASIC implementation uses OpenROAD Flow Scripts (ORFS) with the following stages:

**RTL → Synthesis → Floorplan → Placement → CTS → Routing → Finishing → GDSII**

Figure 4: ASIC Design Flow

### 4.1 Tools Used

Table 10: EDA Tool Chain

| Tool | Function |
|---|---|
| Yosys | RTL Synthesis |
| OpenROAD | Floorplanning, Placement, CTS, Routing |
| OpenSTA | Static Timing Analysis |
| KLayout | GDSII Stream-out, DRC |
| Magic | DRC, LVS |

### 4.2 Flow Stages

#### 4.2.1 1. Synthesis (Yosys)

Converts Verilog RTL to gate-level netlist using sky130hd standard cells.

#### 4.2.2 2. Floorplanning

Defines die/core dimensions, places I/O pins, and creates placement rows.

#### 4.2.3 3. Placement

- Global placement: Initial cell positions
- Detailed placement: Legalization to placement sites
- Timing-driven optimization

#### 4.2.4 4. Clock Tree Synthesis (CTS)

Builds balanced clock distribution network with buffers to minimize skew.

#### 4.2.5 5. Routing

- Global routing: Coarse routing paths
- Detailed routing: Exact metal layer connections

#### 4.2.6 6. Finishing

- Fill cell insertion
- Metal fill for density
- Final verification

## 5 How to Interpret Layout Images

This section explains how to read and understand the generated layout visualizations.

### 5.1 Final Layout View

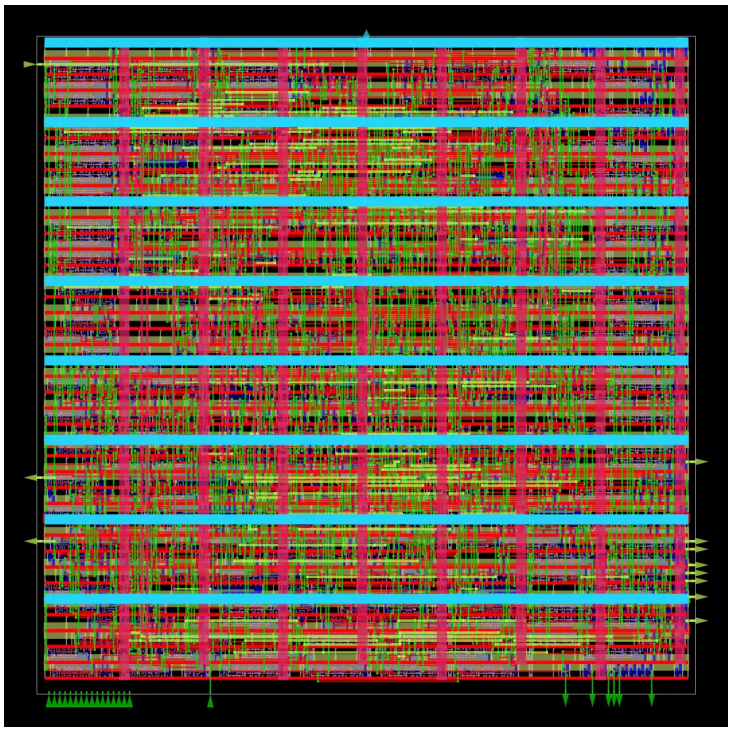


Figure 5: Final Layout - All Layers

The final layout view presents all physical design elements in a single image. The colored rectangles distributed across the core area represent standard cells, which include logic gates, flip-flops, and buffers that implement the processor's functionality. The horizontal and vertical lines traversing the design are metal routing layers that connect these cells together. Around the periphery, larger rectangular structures represent I/O pads and pins that interface with external signals. The regular grid pattern visible throughout the design is the power distribution network, consisting of VDD and VSS rails that supply power to all cells.

## 5.2 Routing View

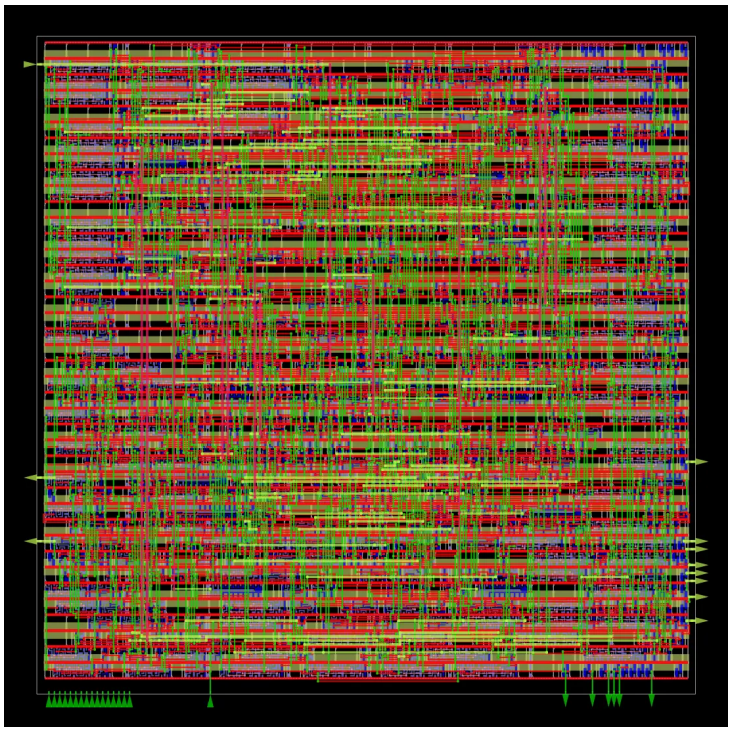


Figure 6: Detailed Routing View

The routing view highlights the interconnections between cells using different colors for each metal layer. The SkyWater 130nm process provides six routing layers. The local interconnect layer (li1) handles connections within individual cells and short-distance routing. The first metal layer (met1), typically shown in blue, runs primarily in the horizontal direction and handles most local signal routing. The second metal layer (met2), shown in pink or red, runs vertically and provides connections between horizontal met1 segments. Higher metal layers (met3 through met5) alternate between horizontal and vertical orientations, with the upper layers primarily used for power distribution and long-distance signal routing due to their lower resistance.

### 5.3 Placement View

Figure 7: Cell Placement View

The placement view shows the physical location of all standard cells within the core area. A high-quality placement exhibits several characteristics: uniform distribution of cells across the available area prevents localized congestion, while related logic blocks are clustered together to minimize wire lengths. Dense regions indicate areas of high logic complexity, typically around critical paths or complex functional units. Empty spaces between cell clusters serve as routing channels, providing room for interconnect wires. The placement algorithm optimizes for timing, routability, and power by iteratively adjusting cell positions to meet design constraints.

## 5.4 Routing Congestion Heatmap

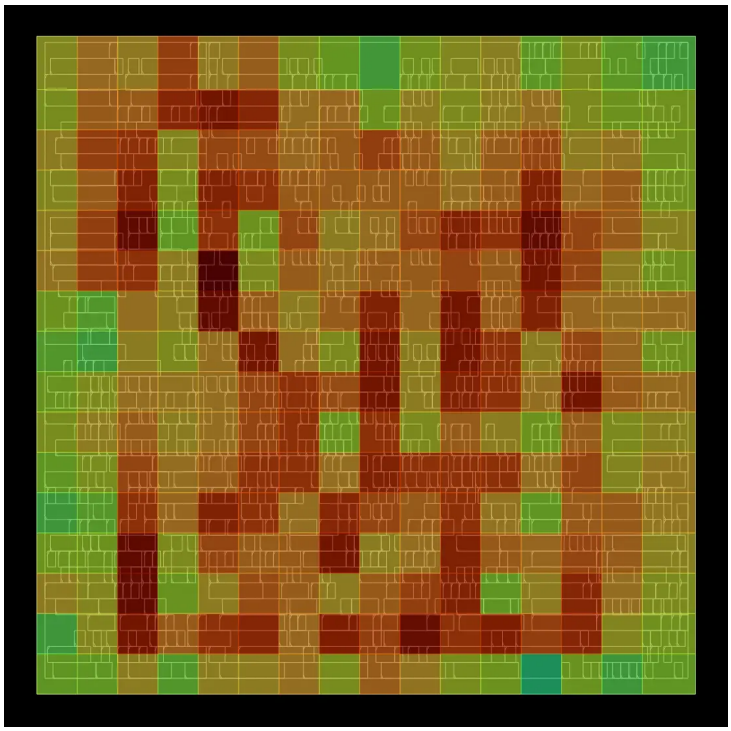


Figure 8: Routing Congestion Heatmap

The routing congestion heatmap uses color coding to indicate the demand for routing resources across different regions of the chip. Blue and green colors represent low congestion areas where routing resources are abundant and signal routing is straightforward. Yellow indicates moderate congestion, suggesting that routing in these areas requires careful management but remains feasible. Orange and red colors signal high congestion, where the demand for routing tracks approaches or exceeds the available capacity, potentially causing routing failures or requiring detours that increase wire length and delay.

Congestion typically arises from several sources: high-fanout nets that must connect to many destinations create routing hotspots; dense logic regions naturally require more interconnections; the clock tree, which must reach every sequential element, consumes significant routing resources; and the power distribution network reserves tracks on upper metal layers. The miniRISC implementation shows predominantly blue and green coloring, indicating excellent routability with no congestion issues.

## 5.5 IR Drop Heatmap

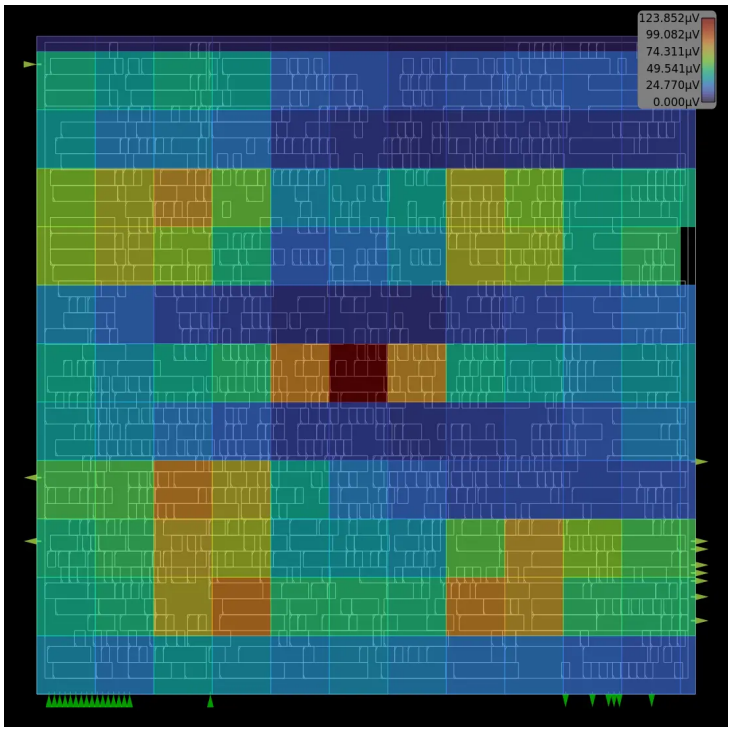


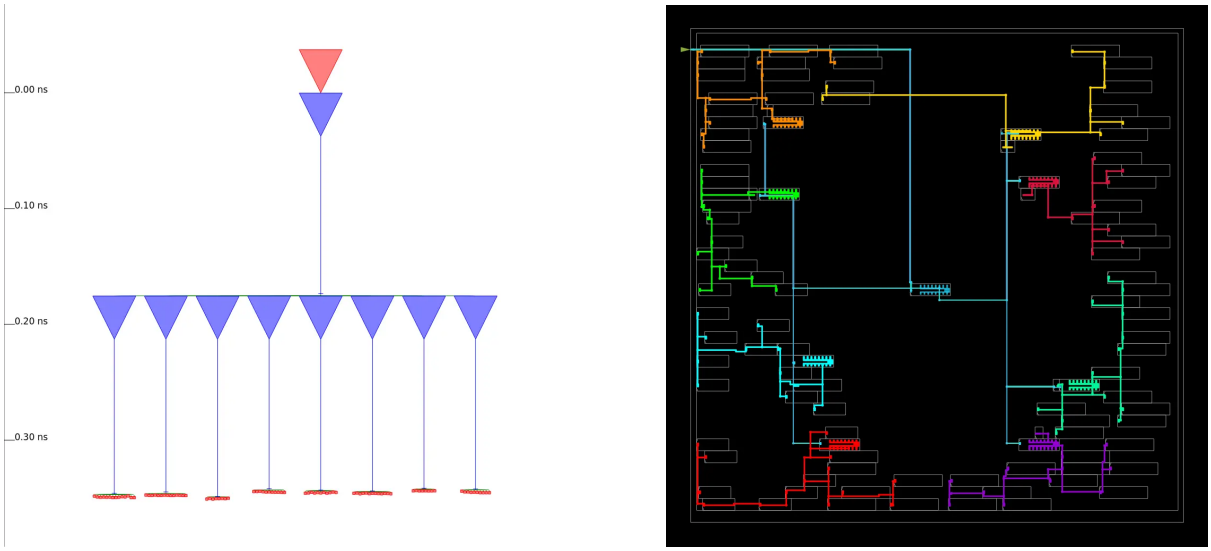

Figure 9: IR Drop Analysis

The IR drop heatmap visualizes the voltage drop across the power distribution network, which occurs due to the resistance of metal wires carrying supply current to the cells. This analysis is critical for ensuring reliable circuit operation, as excessive voltage drop can cause timing failures and functional errors.

The color scale represents the magnitude of voltage drop from the power supply rails to each location in the design. Blue and green regions experience minimal voltage drop, indicating robust power delivery where cells receive voltage close to the nominal supply level. Yellow and orange regions show moderate drop that remains within acceptable margins. Red areas, if present, indicate locations where the voltage drop is significant and may require design modifications such as additional power straps or decoupling capacitors.

For reliable operation, the voltage drop should remain below 5% of the supply voltage. With a 1.8V supply (VDD) in the SkyWater 130nm process, this translates to a maximum acceptable drop of approximately 90mV. The miniRISC implementation achieves excellent power integrity with only 0.01% IR drop, well within acceptable limits.

### 5.6 Clock Tree View




(a) Clock Tree Structure (b) Clock Tree Layout

Figure 10: Clock Tree Synthesis Results

The clock tree visualization shows how the clock signal is distributed from its source to all sequential elements (flip-flops) in the design. The left image displays the logical tree structure, showing the hierarchical arrangement of clock buffers that fan out the clock signal. A well-balanced tree exhibits symmetric branching, ensuring that the clock arrives at all endpoints with minimal variation in arrival time (skew).

The right image shows the physical layout of the clock tree, with clock buffers placed throughout the core and metal routing connecting them. The clock tree synthesis (CTS) tool automatically inserts buffers and determines their placement to achieve the target skew specification. Key metrics to evaluate include the number of buffer stages (affecting clock latency), the total buffer count (affecting power consumption), and the achieved skew (affecting timing margins). The miniRISC clock tree achieves a skew of only 7.4 picoseconds, indicating excellent clock distribution quality.

## 6 Implementation Results

### 6.1 Timing Analysis

Table 11: Timing Summary

| Metric | Value | Status |
|---|---|---|
| Clock Target | 6.50 ns | – |
| Achieved Period | 3.90 ns | MET |
| Fmax | 256.21 MHz | – |
| Setup Slack (WNS) | 2.60 ns | MET |
| Hold Slack | 0.42 ns | MET |
| TNS (Total Negative Slack) | 0.00 ns | CLEAN |
| Setup Violations | 0 | CLEAN |
| Hold Violations | 0 | CLEAN |

**Critical Path:** IR[17] → GPR[2][11] (3.97 ns data arrival time)

### 6.2 Clock Tree Results

Table 12: Clock Tree Metrics

| Metric | Value |
|---|---|
| Clock Buffers | 11 |
| Clock Inverters | 5 |
| Clock Skew (Setup) | 7.4 ps |
| Clock Skew (Hold) | 7.4 ps |
| Source Latency | 0.34 ns |
| Target Latency | 0.35 ns |

### 6.3 Design Rule Checks

Table 13: DRC Summary

| Check | Limit | Slack | Violations |
|---|---|---|---|
| Max Slew | 1.486 ns | 0.583 ns | 0 |
| Max Capacitance | 36.6 fF | 14.8 fF | 0 |
| Max Fanout | – | – | 0 |
| Routing DRC | – | – | 0 |
| Antenna | – | – | 0 |

### 6.4 Power Analysis

Table 14: Power Breakdown by Category

| Category | Internal | Switching | Leakage | Total | % |
|---|---|---|---|---|---|
| Sequential | 611 $\mu$W | 12.3 $\mu$W | 0.75 nW | 623 $\mu$W | 47.6% |
| Combinational | 56.4 $\mu$W | 122 $\mu$W | 2.57 nW | 178 $\mu$W | 13.6% |
| Clock | 298 $\mu$W | 209 $\mu$W | 0.14 nW | 507 $\mu$W | 38.7% |
| **Total** | **965 $\mu$W** | **343 $\mu$W** | **3.46 nW** | **1.31 mW** | 100% |

**Power Distribution:** 73.8% Internal, 26.2% Switching, ∼0% Leakage

### 6.5 IR Drop Analysis

Table 15: IR Drop Results

| Net | Worst Voltage | Avg Drop | Worst Drop | % Drop |
|---|---|---|---|---|
| VDD | 1.7998 V | 25.6 $\mu$V | 159.6 $\mu$V | 0.01% |
| VSS | 123.9 $\mu$V | 19.6 $\mu$V | 123.9 $\mu$V | 0.01% |

## 6.6 Congestion Analysis

Table 16: Global Routing Congestion by Layer

| Layer | Resource | Demand | Usage (%) | Overflow |
|---|---|---|---|---|
| li1 | 0 | 0 | 0.00% | 0 |
| met1 | 2,291 | 1,145 | 49.98% | 0 |
| met2 | 2,415 | 1,236 | 51.18% | 0 |
| met3 | 1,803 | 169 | 9.37% | 0 |
| met4 | 885 | 103 | 11.64% | 0 |
| met5 | 240 | 0 | 0.00% | 0 |
| **Total** | **7,634** | **2,653** | **34.75%** | **0** |

Table 17: Routing Statistics

| Metric | Value |
|---|---|
| Total Wirelength | 30,546 $\mu$m |
| Routed Nets | 1,055 |
| Total Overflow | 0 (no congestion) |
| Weighted Congestion | 1.025 |

## 6.7 Area Utilization

Table 18: Area Summary

| Metric | Value |
|---|---|
| Die Area | 12,678.8 $\mu m^2$ |
| Core Area | 11,961.5 $\mu m^2$ |
| Core Utilization | 63.96% |
| Placement Rows | 40 |
| Placement Sites | 9,560 |

## 6.8 Cell Statistics

Table 19: Cell Count by Type

| Cell Type | Count | Area ($\mu m^2$) |
|---|---|---|
| Standard Cells | 1,186 | 7,651.1 |
| Sequential Cells | 89 | 2,422.3 |
| Combinational Cells | 815 | 4,271.6 |
| Inverters | 110 | 412.9 |
| Clock Buffers | 11 | 232.7 |
| Clock Inverters | 5 | 57.6 |
| Timing Repair Buffers | 9 | 70.1 |
| Fill Cells | 1,079 | 4,310.4 |
| Tap Cells | 147 | 183.9 |

# 7 AutoTuner Optimization

AutoTuner was used to explore the design space using Bayesian optimization (Tree-structured Parzen Estimator).

## 7.1 Configuration

- **Algorithm:** HyperOpt (TPE)
- **Samples:** 20
- **Parallel Jobs:** 2
- **Objective:** Minimize effective clock period (maximize frequency)

## 7.2 Best Parameters Found

Table 20: AutoTuner Optimized Parameters

| Parameter | Default | Optimized | Description |
|---|---|---|---|
| SDC_CLK_PERIOD | 6.50 ns | 5.26 ns | Clock target |
| CORE_UTILIZATION | 60% | 52% | Core utilization |
| CORE_ASPECT_RATIO | 1.0 | 0.999 | Die aspect ratio |
| CORE_MARGIN | 2 | 2 | Core margin |
| PLACE_DENSITY_LB_ADDON | 0.0 | 0.10 | Placement density |
| CTS_CLUSTER_SIZE | 30 | 22 | CTS cluster size |
| CTS_CLUSTER_DIAMETER | 100 | 69 | CTS cluster diameter |

## 7.3 Results Comparison

Table 21: Default vs AutoTuner Results

| Metric | Default | AutoTuner | Change |
|---|---|---|---|
| Fmax | 256.21 MHz | 263 MHz | **+2.7%** |
| Clock Period | 3.90 ns | 3.80 ns | **-2.6%** |
| Die Area | 12,679 $\mu m^2$ | 15,081 $\mu m^2$ | +19% |
| Core Utilization | 63.96% | 52% | -12% |

**Conclusion:** AutoTuner achieved 2.7% higher frequency at the cost of 19% larger die area (speed-optimized mode).

# 8 Conclusion

The miniRISC processor was successfully implemented in SkyWater 130nm technology achieving:

- **256 MHz** maximum operating frequency
- **1.31 mW** total power consumption
- **12,679** $\mu m^2$ die area with 64% utilization

- **Zero** DRC, timing, and antenna violations
- **0.01%** IR drop (excellent power integrity)

The design is clean and ready for tape-out. AutoTuner optimization demonstrated potential for further frequency improvement with area trade-off.

# 9 Appendix: Running the Flow

## 9.1 Prerequisites

```
docker pull openroad/orfs:latest
docker build -t orfs-autotuner - < Dockerfile.autotuner-lite
```

## 9.2 Running the Flow

```
# Full RTL-to-GDS flow
./run_flow

# Individual stages
./run_flow synth
./run_flow floorplan
./run_flow place
./run_flow cts
./run_flow route
./run_flow finish

# AutoTuner optimization
./run_autotuner -s 20 -j 4 -a hyperopt
```

## 9.3 Output Files

| Output | Path |
|---|---|
| GDSII | results/sky130hd/miniRISC/base/6_final.gds |
| DEF | results/sky130hd/miniRISC/base/6_final.def |
| Netlist | results/sky130hd/miniRISC/base/6_final.v |
| Reports | logs/sky130hd/miniRISC/base/6_report.json |