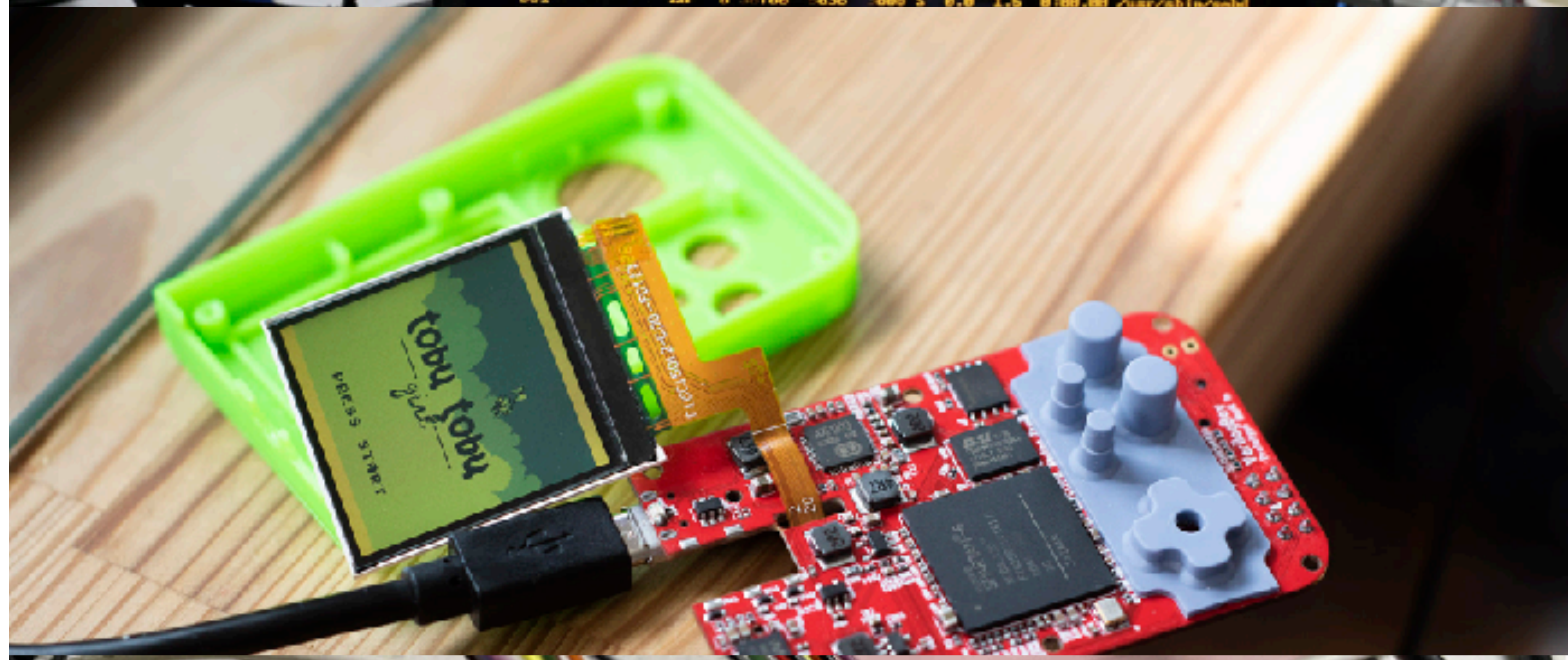
RISu064

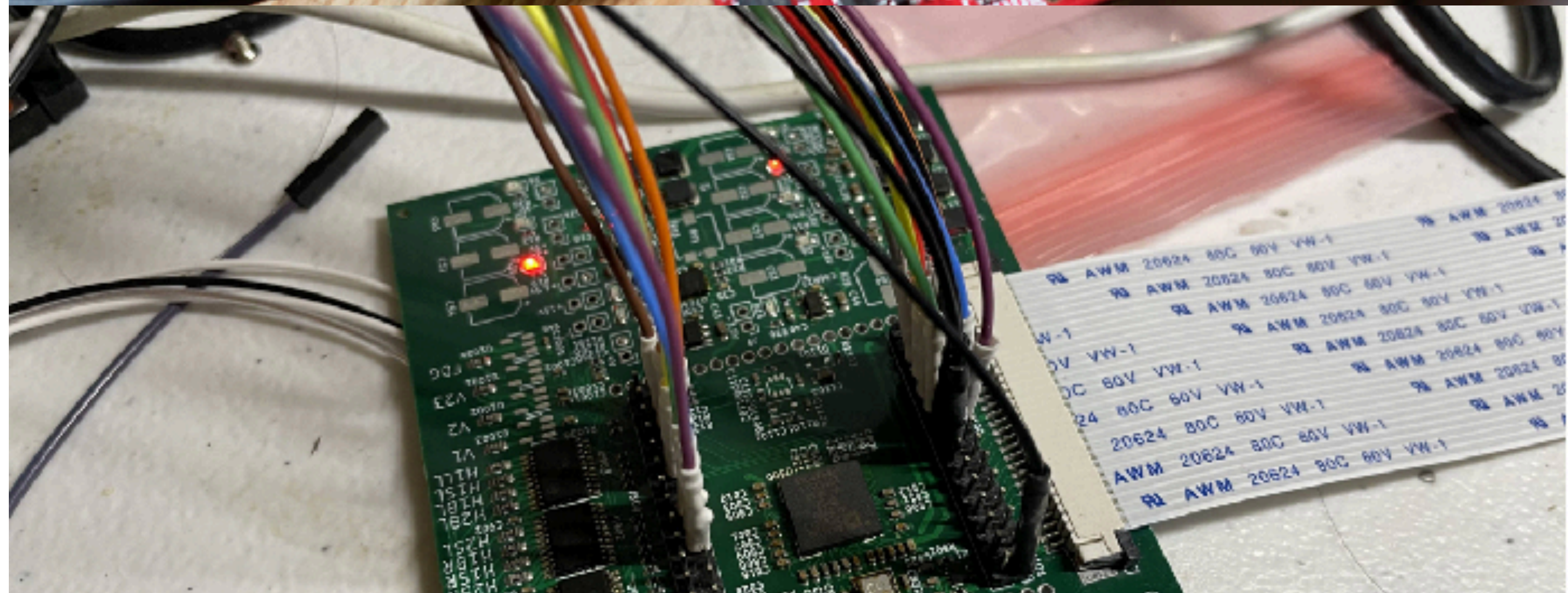
A non-blocking dual-issue RISC-V processor

Wenting Zhang, Apr. 2nd, 2023

About Me

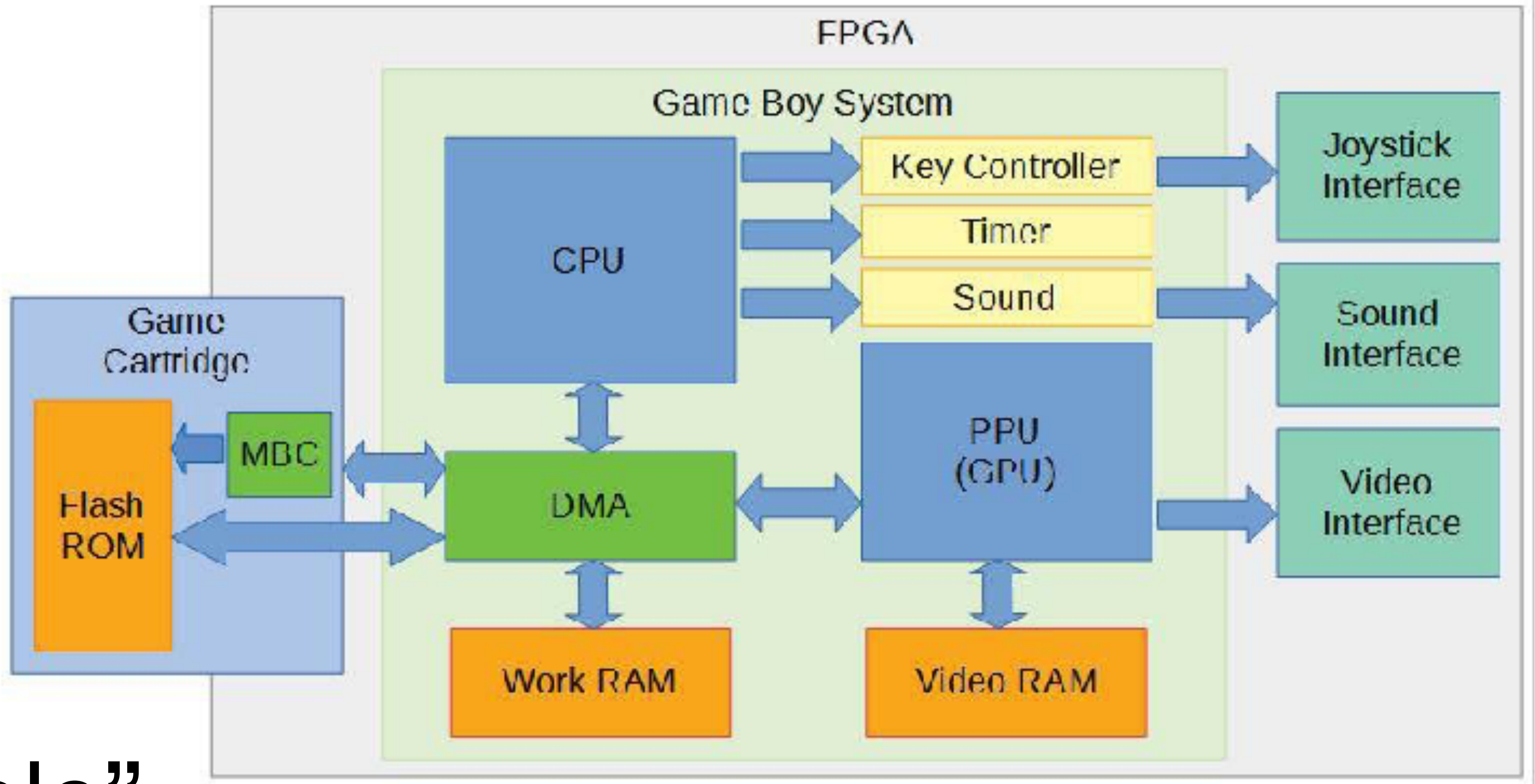
- Digital IC Designer at Zero ASIC
- Some of the personal projects I did:
    - E-ink controller + tablet device
    - RP2040 Serial Terminal
    - Handheld FPGA game console
    - Full-frame CCD camera



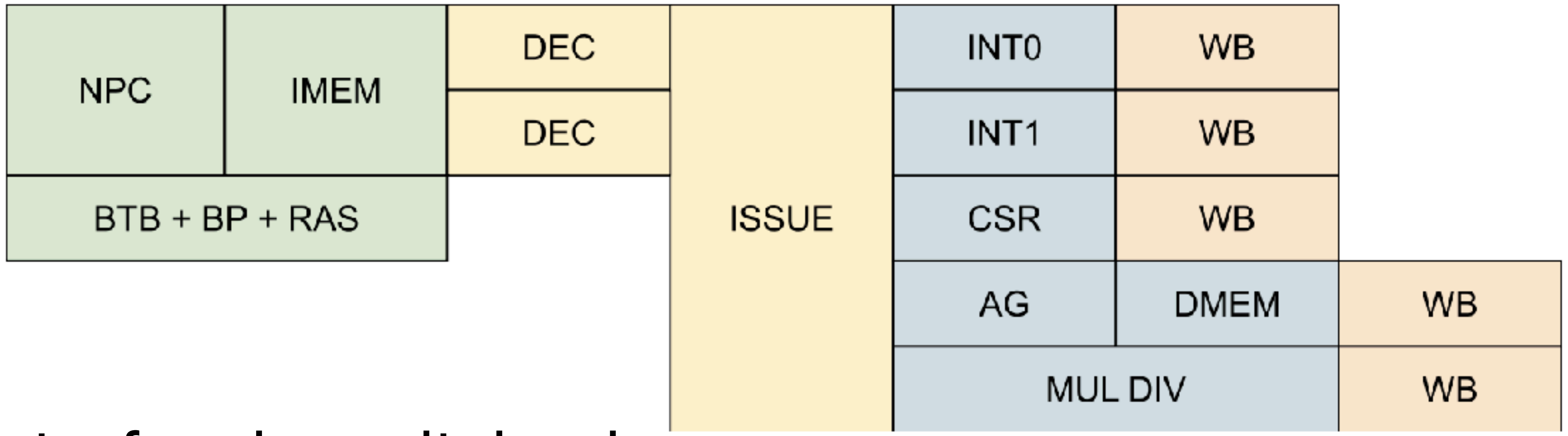


Project Background

- VerilogBoy
  - 8-bit CISC CPU + fixed 2D tiled graphics
  - ~4th generation handheld console
  - Compatible with real console
- Next Step:
  - 32/64-bit RISC CPU + 3D graphics
  - ~5th generation home console
  - More or less proof-of-concept “fantasy console”
- Reinventing things for fun and learning



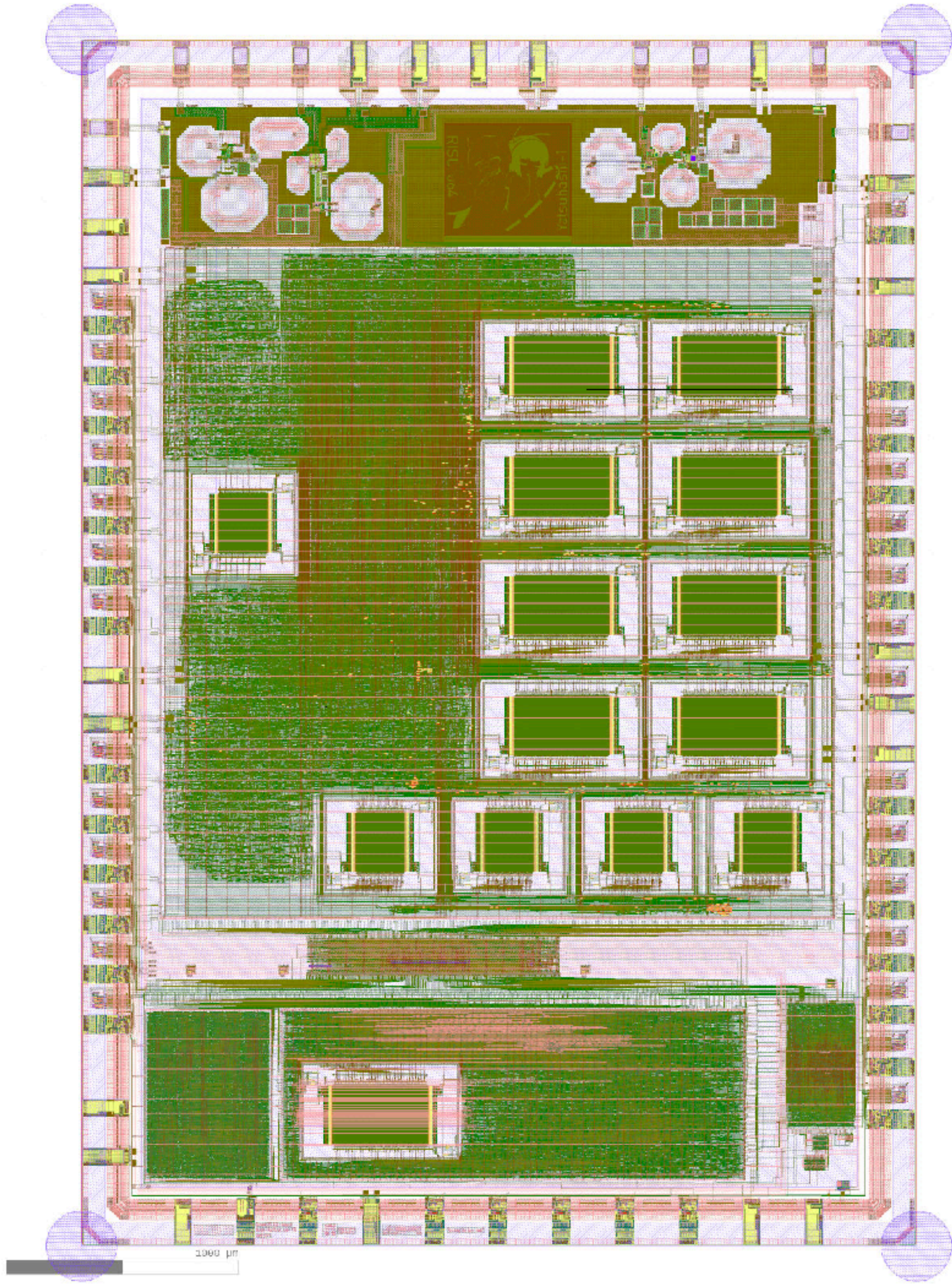
RISu064



- RV64IM Instruction Set
- 6/7-stage pipeline
- In-order dual-issue and out-of-order writeback
- Long latency instruction doesn't block non-dependent instructions
- BTB + Bimodal/Gselect/Gshare/Tournament + RAS branch predictors, up to 2 predictions per cycle
- Optional L1 I and D cache (2-way set associative blocking cache)
- Machine mode with precise exception and interrupt support
- Optional hardware refilled MMU + supervisor and user mode support
- Written in portable synthesizable Verilog

Results

- Coremark
  - Single-issue + TCM: 3.06 Coremark/MHz
  - Single-issue + 16KB L1$: 3.01 Coremark/MHz
  - Dual-issue + TCM: 4.31 Coremark/MHz
- ASIC (SiliconCompiler + yosys + OpenROAD)
  - SKY130HS w/o L1$: ~100 MHz
  - SKY130HD w/o L1$: ~80 MHz
  - SKY130HD w/ L1$: ~50 MHz
- FPGA
  - ~120 MHz fmax, 19.6K LUT, 6.9K FF (7A100T)



Next Steps

- Core optimizations
- Verilog is not ideal
  - Use a high level HDL like Chisel!
  - Use preprocessor to generate boilerplate code…

Manjuu

- Python-based Verilog preprocessor
- Python inside Verilog, not the other way around

```
req_t = [
    ["i", "data", 8],
    ["i", "cmd", 4]
]

module A(<% gen_port("in", handshake(req_t)) %>);
    <% gen_reg("cur", req_t) %>
    always @(posedge clk) begin
        if (in_valid) begin
            <% gen_capture("cur", req_t, "in") %>
        end
    end
    /* Module implementation */
endmodule

module B(<% gen_port("out", reverse(handshake(req_t))) %>);
    /* Module implementation */
endmodule

module Top();
    <% gen_wire("b_a", handshake(req_t)) %>
    A a(<% gen_connect("in", req_t, "b_a") %>);
    B b(<% gen_connect("out", req_t, "b_a") %>);
endmodule
```

➡

```
module A (
    input wire [7:0] in_data,
    input wire [3:0] in_cmd,
    input wire in_valid,
    output reg in_ready
);
    reg [7:0] cur_data;
    reg [3:0] cur_cmd;

    always @(posedge clk) begin
        if (in_valid) begin
            cur_data <= in_data;
            cur_cmd <= in_cmd;
        end
    end
    /* Module implementation */
endmodule

module B (
    output reg [7:0] out_data,
    output reg [3:0] out_cmd,
    output reg out_valid,
    input wire out_ready
);
    /* Module implementation */
endmodule

module Top ();
    wire [7:0] b_a_data;
    wire [3:0] b_a_cmd;
    wire b_a_valid;
    wire b_a_ready;

    A a (
        .in_data(b_a_data),
        .in_cmd(b_a_cmd),
    );
    B b (
        .out_data(b_a_data),
        .out_cmd(b_a_cmd),
    );
endmodule
```

Next Steps

- Core optimizations
- Verilog is not ideal
  - Use a high level HDL like Chisel!
  - Use preprocessor to generate boilerplate code…
- 3D GPU

Thank you