GamingCPU — Coding & Documentation Style Guide

Version 1.0

Prepared by: Rafeed Khan

Purpose — To establish a consistent, professional and easy to read standard for RTL, firmware, device-tree, register specifications, verification, and documentation across the GamingCPU codebase. This guide governs the naming, layout, formatting, and review practices to ensure readability, maintainability, and tool interoperability.

Scope — Applies to all sources under `rtl/`, `sw/`, `specs/`, `tools/`, `dts/`, `sim/`, `verification/`, `docs/`, build scripts, and CI.

Contents

1 Governance

1.1 Document Ownership

- Maintainers: Rafeed Khan, Sebastian Candelaria, Anh Pham.

1.2 Revision History

Date	Version	Notes
2025-10-19	1.0	First rendition of the Style Guide.

2 Repository Layout and Sources of Truth

2.1 Canonical Structure

Key top-level directories and responsibilities:

- `rtl/`: SystemVerilog RTL by functional domain (CPU, MMU, caches, bus/AXI, memory, IO, video, audio, SD, subsystems, top).
- `sw/`: BootROM, FSBL, drivers, libs, tests, and DOOM app integration.
- `specs/`: `memory_map.yaml` and per-IP `registers/*.yaml`. These generate headers and register files; they are the authoritative source.
- `dts/`: Device-tree sources describing the SoC for firmware/OS.
- `tools/`: Generators (e.g., `reggen/`), image packers, and helper scripts.
- `sim/` & `verification/`: Verilator/Cocotb harnesses, behavioral models, and ISA/MMU/-cache tests.
- `docs/`: Bring-up, memory map, boot flow, video/audio/storage pipelines, and test plan.

2.2 Generated Artifacts

- Do not hand-edit generated files (e.g., `regs_auto.h`, `regs_auto.sv`). Update YAML specs and re-generate via `tools/reggen`.
- Generated files must be clearly bannered with "AUTO-GENERATED; DO NOT EDIT."

3 General Naming & Conventions

3.1 File Naming

Domain	Convention
RTL modules	`snake_case.sv` (e.g., `axi_crossbar.sv`)
RTL packages	`snake_case_pkg.sv` (e.g., `rv32_pkg.sv`)
C headers	`snake_case.h` (e.g., `regs_auto.h`)
C sources	`snake_case.c` (e.g., `sd_spi.c`)
Device tree	`board_name.dts` (e.g., `GamingCPU_arty_a7.dts`)
YAML specs	`ipname.yaml` (e.g., `uart.yaml`)
Python tools	`snake_case.py` (e.g., `mkimage.py`)
Docs (md)	`topic.md` (e.g., `bringup.md`)

3.2 Identifiers

- **SystemVerilog**: `lower_snake_case` for signals/instances; `UPPER_SNAKE_CASE` for parameters/localparams/defines; `CamelCase` for type names (struct/enum) and interfaces.
- **C**: `lower_snake_case` for variables/functions; `g_` prefix for globals; `UPPER_SNAKE_CASE` for macros and constants; `typedef_t` suffix for typedefs.
- **DTS**: Node names `lowerdash`, unit-address as hex (e.g., `uart@10000000`). `compatible` strings use vendor,device (e.g., `"gamingcpu,uart16550"`).
- **Registers YAML**: Block names `lower_snake_case`; fields `UPPER_SNAKE_CASE`; bits `[msb:lsb]`.

3.3 Numbers, Units, and Constants

- Use explicit widths (e.g., `8'd255`) and SI units in comments (Hz, ms, MB).
- Time constants in RTL use parameters/localparams (e.g., `CLK_HZ`, `I2S_BCLK_HZ`). No magic numbers.

4 RTL (SystemVerilog) Standards

4.1 Module Template

Listing 1: Canonical module template

```
module foo_bar #(
  parameter int unsigned DATA_W = 32
) (
  input  logic              clk_i,
  input  logic              rst_ni,      // active-low, synchronous release
  // AXI-lite slave (example)
  input  logic [31:0]       s_awaddr_i,
  input  logic              s_awvalid_i,
  output logic              s_awready_o,
  // ...
  output logic [DATA_W-1:0] data_o
);
  // Parameters and localparams
  localparam int unsigned FIFO_DEPTH = 8;

  // Type definitions
  typedef enum logic [1:0] {
    IDLE, BUSY, ERR
  } state_e;

  // Registers and wires
  state_e state_q, state_d;
  logic [DATA_W-1:0] reg_q, reg_d;

  // Combinational
  always_comb begin
    state_d = state_q;
    reg_d   = reg_q;
    // ... unique/priority case for FSMs
  end

  // Sequential
  always_ff @(posedge clk_i) begin
```

```
    if (!rst_ni) begin
      state_q <= IDLE;
      reg_q   <= '0;
    end else begin
      state_q <= state_d;
      reg_q   <= reg_d;
    end
  end

  // Assertions (example)
  // assert property (@(posedge clk_i) disable iff (!rst_ni) s_awready_o |-> s_awvalid_i)
    ;
endmodule
```

4.2 Clocks, Resets, and CDC

- Clock names end with `_i`/`_o`; primary system clock is `clk_i`. Resets are active-low, synchronous release: `rst_ni`.
- CDC: use approved async FIFOs/synchronizers (e.g., AXI async FIFO wrappers). No ad-hoc CDC.

4.3 Coding Rules

1. **No latches.** Combinational blocks must assign defaults to all outputs/state-d.
2. **Use** `always_ff`/`always_comb`/`always_latch` appropriately; no legacy `always @(*)` unless justified.
3. **FSMs** use `typedef enum logic [`N`:0]` and `unique case`. Provide explicit reset state.
4. **Parameters** in UPPER_SNAKE; widths and array sizes parameterized; import shared widths from packages (e.g., `rv32_pkg`).
5. **AXI/APB/AHB** signal names follow official channel naming; ready/valid handshakes obey protocol timing.
6. **Assertions** for protocol handshakes, FIFO bounds, and illegal states; gate with synthesis defines if needed.
7. **Register maps** are generated from YAML via `tools/reggen`; do not duplicate logic.

4.4 Formatting

- Indent with 2 spaces, max line length 100. Align port lists/params sensibly; one signal per line.
- Module/file header banner includes brief, author, and references to spec section.

5 Register Specifications (YAML)

5.1 Principles

- One YAML per IP in `specs/registers/`. Include block description, address offset relative to IP base, reset values, and access types (rw/ro/wo/rc/w1c/etc.).
- All fields documented: function, side effects, and constraints. Reserve unused bits and mark them `RESERVED`.

5.2 Example

Listing 2: registers/uart.yaml (excerpt)

```
block: uart
base: 0x0000
registers:
  - name: TXDATA
    offset: 0x00
    desc: Transmit data
    fields:
      - { name: DATA,   bits: [7,0], access: rw, reset: 0 }
      - { name: FULL,   bits: [31,31], access: ro, reset: 0 }
  - name: RXDATA
    offset: 0x04
    desc: Receive data
    fields:
      - { name: DATA,   bits: [7,0], access: ro, reset: 0 }
      - { name: EMPTY,  bits: [31,31], access: ro, reset: 1 }
```

5.3 Generation

- Run `tools/reggen/reggen.py` to emit `regs_auto.h` (C) and IP register files (SV) using templates.
- CI validates YAML schema and re-generates on change; PRs failing regen are rejected.

6 Firmware (C) Standards

6.1 Layering

1. **HAL**: Thin MMIO accessors over `regs_auto.h`. No raw addresses in driver code.
2. **Drivers**: One driver per IP (UART, PLIC, CLINT, SD, DMA, VIDEO, I2S, GPIO). Interrupt-safe APIs where applicable.
3. **Apps/Tests**: Use drivers only; no direct MMIO in apps.

6.2 C Coding Rules

- Use `stdint.h` fixed-width types; avoid implicit promotion surprises.
- Functions: `lower_snake_case`; namespaced by IP (e.g., `uart_init()`, `video_set_palette()`).
- `const` correctness; no hidden state in drivers.
- ISRs: minimal work; set flags or push to ring buffers; no dynamic allocation.
- Logging: `printf` is allowed for bring-up, but production code should use lightweight debug macros that can be compiled out.

6.3 File Headers

Each C file starts with a banner noting purpose, references (spec sections), and ownership.

7 Device Tree (DTS) Standards

7.1 Basics

- Node names in lowercase with hyphens; include unit-address (e.g., `uart@10000000`).
- Properties: `reg` (base+size), `interrupts`, `clocks`, `compatible` (vendor,device).
- Use aliases for console and chosen nodes; keep board-specific overrides in board DTS.

7.2 Example

Listing 3: GamingCPU board DTS (excerpt)

```
uart0: uart@10000000 {
  compatible = "gamingcpu,uart16550";
  reg = <0x10000000 0x1000>;
  interrupts = <5>;
  clock-frequency = <50000000>;
};
```

8 Verification & Simulation

8.1 Test Strategy

- **Unit**: IP-level directed tests (reset, basic transactions, error paths).
- **Protocol**: AXI-ready/valid, burst rules, error codes validated with assertions.
- **System**: Top-level bring-up (BootROM banner, DDR calibration, FSBL load), video/audio scanout sanity, SD read timing, DOOM smoke.

8.2 Cocotb/Verilator Conventions

- Test filenames: `test_feature.py`; mark seeds; avoid hidden nondeterminism.
- Waveforms: gate behind env var (e.g., `DUMP_VCD=1`). Store example `.vcd` only if small.
- Assertions: prefer self-describing messages; avoid arbitrary sleeps; await protocol events.

8.3 Coverage and Sign-off

- Checklists per IP: reset coverage, register R/W, IRQ routing, error injection, throughput/latency targets.
- System sign-off requires ISA compliance (rv32ui/um/ua), MMU paging tests, cache coherency policy checks, and IO regressions green.

9 Documentation Standards

9.1 Docs/ Markdown

- Audience-focused: bring-up guides, pipeline overviews, boot flow, and performance notes live under `docs/`.
- Each doc begins with: Purpose, Prerequisites, Procedure, Validation, Troubleshooting.
- Diagrams: use SVG/PNG checked into `docs/`; source (draw.io/mermaid) must be referenced.

9.2 API Reference

- Driver headers should be self-documenting; optionally use Doxygen comments. Do not rely on external wikis.

10 RTL Module Documentation (One-Pager Policy)

Policy (Normative)

Every RTL module **MUST** have a one-page brief stored at `docs/ip-briefs/<module_name>.tex`.

The filename **must** match the RTL file (e.g., `axi_crossbar.sv` → `docs/ip-briefs/axi_crossbar.tex`). Any PR that adds or modifies files under `rtl/` **must** add/update the corresponding brief. Briefs **must** compile to a single page with this document class.

10.1 What the one-pager must contain

- **Purpose & Role**: one paragraph on function and SoC placement.
- **Parameters**: names, defaults, effects.
- **Interfaces (Ports)**: signal, dir, width, description; protocol refs (AXI/APB/etc.).
- **Reset/Init**: reset values, enable sequences, configuration order.
- **Behavior & Timing**: FSM/pipeline overview, latency/throughput, ordering, CDC/FIFOs.
- **Programming Model**: cross-refs to `specs/registers/<ip>.yaml`.
- **Errors/IRQs**: status/clear mechanisms, recovery paths.
- **Performance Targets**: quantitative goals (bandwidth, fps, cycles/op).
- **Dependencies**: clocks/resets, other IPs, BootROM/driver assumptions.
- **Verification Links**: tests/coverage locations; known limitations/errata.

11 Build, CI, and Tooling

11.1 Make Targets

- `make sim`: Build and run Verilator simulation.
- `make bit`: Synthesize/implement FPGA bitstream.
- `make doom`: Build DOOM app and required assets.
- `make image`: Create SD/QSPI images; see `packaging/sd` and `packaging/flash`.
- `make lint/test`: Run linters and regression suites.

11.2 Linters/Formatters

- RTL: verible-format (or equivalent) with 2-space indent; Verilator `-Wall` clean.
- C: clang-format with project style; `-Werror -Wall -Wextra` clean.
- YAML: schema checked; no tabs; 2-space indent.
- DTS: `dtc` compile clean; keep base addresses aligned to memory map.

11.3 Docs CI Checks

- **IP Brief Presence**: any change under `rtl/` must include a brief at `docs/ip-briefs/<module>.tex` with the matching filename.
- **Registry Integrity**: `docs/ip-briefs/INDEX.md` lists each module and links to its brief, register YAML, driver, and tests.
- **Reggen Sync**: YAML diffs must include regenerated headers/SV; CI rejects stale artifacts.

11.4 EditorConfig (recommended)

```
root = true

[*]
charset = utf-8
end_of_line = lf
insert_final_newline = true
trim_trailing_whitespace = true

[*.{sv,svh,v,vh}]
indent_style = space
indent_size = 2
max_line_length = 100
```

```
[*.{c,h}]
indent_style = space
indent_size = 2
max_line_length = 100

[*.{yaml,yml,dts,dtsi,md}]
indent_style = space
indent_size = 2
```

12 Coding Examples

12.1 MMIO Access (C)

Listing 4: Driver reads/writes via regs_auto.h

```c
#include "regs_auto.h"
#include "hal.h"

void uart_putc(char c) {
  while (READ32(UART_BASE + UART_TXDATA) & UART_TXDATA_FULL_MASK) {}
  WRITE32(UART_BASE + UART_TXDATA, (uint32_t)c);
}
```

12.2 AXI Handshake (SV)

```
// Write address channel ready follows skid buffer fullness
assign s_awready_o = !aw_skid_full;

// Data must not be accepted without address
assert property (@(posedge clk_i) disable iff (!rst_ni)
  s_wvalid_i |-> $past(s_awvalid_i && s_awready_o));
```

13 Commit, Review, and Branch Policy

13.1 Commit Messages

- Use Conventional Commits: `feat:`, `fix:`, `docs:`, `refactor:`, `test:`, `build:`, `ci:`.
- Reference components (e.g., `rtl/axi: fix ready timing`); include brief rationale and validation notes.

13.2 Code Review Checklist (excerpt)

- RTL compiles and lints clean; no latches; assertions added for new protocols.
- Register YAML updated; regen checked in; docs updated.
- Drivers follow HAL; no magic addresses; ISR-safe.
- DTS matches memory map; interrupts wired; chosen/aliases correct.
- Tests added/updated; coverage impact noted; CI passes.

14 Security, Safety, and Reliability

- Input validation at IP boundaries; assert on illegal transactions.
- Error status/clear mechanisms documented for each IP; never reuse `RESERVED` bits.
- Deterministic resets: all state reset to known values; FIFO pointers cleared; IRQ masks disabled.

15 Appendices

15.1 Standard File Headers

```
// ----------------------------------------------------------------------
// <file>: <one-line purpose>
// Author: <owner@domain>
// References: <spec section(s)>
// Notes: <timing/protocols/errata>
// ----------------------------------------------------------------------
```

Table 2: GamingCPU IP Brief Registry

Module	RTL Path	Brief (Section)	Reg Spec	Driver
cache_l1	rtl/mem/cache/icache.sv, rtl/mem/cache/dcache.sv	Section 16.1	—	—
axi_crossbar	rtl/bus/axi/axi_crossbar.sv	Section 16.2	—	—
rv32core	rtl/cpu/core/rv32_core.sv	Section 16.3	—	—
fetch	rtl/cpu/core/fetch.sv	Section 16.4	—	—
decode	rtl/cpu/core/decode.sv	Section 16.5	—	—
execute	rtl/cpu/core/execute.sv	Section 16.6	—	—
mem_stage	rtl/cpu/core/mem_stage.sv	Section 16.7	—	—
writeback	rtl/cpu/core/writeback.sv	Section 16.8	—	—
sv32_mmu	rtl/cpu/mmu/sv32_mmu.sv	Section 16.9	—	—
tlb	rtl/cpu/mmu/tlb.sv	Section 16.10	—	—
uart16550	rtl/io/uart16550.sv	Section 16.11	specs/registers/uart.yaml	sw/drivers/uart.c
fb_ctrl	rtl/video/fb_ctrl.sv	Section 16.12	specs/registers/video.yaml	sw/drivers/video.c
sd_spi_ctrl	rtl/sd/sd_spi_ctrl.sv	Section 16.13	specs/registers/sdctrl.yaml	sw/drivers/sd_spi.c
i2s_tx	rtl/audio/i2s_tx.sv	Section 16.14	specs/registers/i2s.yaml	sw/drivers/audio_i2s.c
clint	rtl/irq/clint.sv	Section 16.15	specs/registers/clint.yaml	sw/drivers/clint.c
plic	rtl/irq/plic.sv	Section 16.16	specs/registers/plic.yaml	sw/drivers/plic.c

16 RTL Module One-Pagers

16.1 $\textbf{cache}_l1^-ModuleBrief(v0.1)$

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/mem/cache/icache.sv`, `rtl/mem/cache/dcache.sv`

Purpose & Role

GamingCPU implements a split Level-1 cache: a read-only **instruction cache (I$)** and a write-back **data cache (D$)**. I$ serves the fetch front-end; D$ serves the MEM stage. Both connect to the AXI fabric through dedicated ports and hide DDR latency from the core.

Key Implementation Pieces

- `cache_tags.sv` — tag RAMs and valid/dirty bits (shared structure).
- `cache_miss_unit.sv` — miss/refill/evict controller with writeback buffer.
- `cache_coherency_none.sv` — single-core coherency stub (no inter-CPU snoop).
- AXI bridge: `rtl/bus/axi/axi_icache_port.sv`, `rtl/bus/axi/axi_dcache_port.sv`.

Interfaces (Ports)

Signal/Bundle	Dir	Width	Description
`clk_i`, `rst_ni`	In	1	Clock / active-low reset.
I$ CPU side (`fetch` ↔ I$)	I/O	—	PC/line request, hit/miss return, redirect flush on branch/exception.
D$ CPU side (`mem_stage` ↔ D$)	I/O	—	Load/store req with byte enables; data return; misalign/atomic handling.
AXI via `axi_i/dcache_port`	I/O	—	Read/write bursts for refills and evictions.

Parameters

Name	Default	Description
`LINE_BYTES`	project default	Cache line size in bytes (I$/D$).
`WAYS`	project default	Set associativity.
`SETS`	project default	Index count per way.

Behavior & Timing

- **I$:** read-only; serves aligned instruction fetches; pipeline redirect flushes pending hits; `FENCE.I` triggers invalidate path.
- **D$:** write-back, write-allocate; byte-enable stores; merges sub-line writes; evicts dirty lines through writeback buffer.
- Typical hit latency is single-cycle; misses stall the requesting stage until refill completes.
- Atomics (LR/SC) serialize as required by the reservation set; uncached/IO regions bypass via AXI port as needed.

Dependencies

- AXI fabric: `rtl/bus/axi/axi\_crossbar.sv`.
- CPU pipeline blocks: `rtl/cpu/core/fetch.sv`, `rtl/cpu/core/mem\_stage.sv`.
- MMU/TLB path: address translation precedes cache access in S-mode (`rtl/cpu/mmu/sv32\_mmu.sv`, `rtl/cpu/mmu/tlb.sv`).

Reset & Maintenance

- All valid/dirty bits clear on reset; writeback buffer emptied.
- Global invalidate hooks used at boot; `FENCE.I` integrated for I$ self-modifying code sequences.

Verification Links

`sim/cocotb/test_icache.py`, `sim/cocotb/test_dcache.py`;
`verification/cache/wb_wa_sequences.sv`, `verification/cache/selfmod_code_fencei.S`

16.2 axi_crossbar — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/bus/axi/axi_crossbar.sv`

Purpose & Role

Central AXI4 interconnect between masters and slaves. Performs address decode, per-slave channel arbitration, and ID-based routing so multiple transactions can be outstanding concurrently. Common widths/IDs are defined in `rtl/bus/interconnect_pkg.sv`.

System Context (Masters/Slaves)

- **Masters:** I$ port `axi_icache_port.sv`, D$ port `axi_dcache_port.sv`, DMA `axi_dma.sv`, and the MMU Page Table Walker `rtl/cpu/mmu/ptw.sv`.
- **Slaves:** DDR via `rtl/mem/ddr/mig_axi_wrap.sv`, peripheral MMIO via `rtl/bus/axi/axi_to_axi_lite.sv` into `rtl/subsys/periph_axi_shell.sv`.
- **CDC:** When a clock crossing is required, channels use `axi_async_fifo.sv`.

Interfaces (Ports)

Signal/Bundle	Dir	Width	Description
`s_axi_*[N_M]`	In	—	Slave-side inputs from masters (I$, D$, DMA, PTW).
`m_axi_*[N_S]`	Out	—	Master-side outputs to slaves (DDR, AXI-Lite bridge, etc.).
`clk_i, rst_ni`	In	1	Fabric clock and active-low reset.

Parameters & Configuration

- Number of masters/slaves, ID width, and data/address widths are set via `interconnect_pkg.sv`.
- Address windows correspond to the SoC memory map (DDR and MMIO ranges).

Behavior & Timing

- Per-slave arbitration and decode; maintains AXI ready/valid ordering on all channels (AW/W/B/AR/R).
- Supports multiple outstanding transactions using `AXI ID` tagging; responses are routed back by ID.
- Designed for the system clock domain; CDC handled externally via `axi_async_fifo.sv` where required.

Dependencies

- **Clocks/Reset:** `clk_i`, `rst_ni` (system domain; sync release).
- **Upstream masters (initiate AXI):** I$ port, D$ port, DMA engine, Page Table Walker.
- **Downstream slaves (serve AXI):** DDR controller, AXI→AXI-Lite bridge into Peripheral Shell.
- **Configuration source:** address windows, ID/data/addr widths defined in the interconnect package.
- **CDC:** any clock crossings are handled *outside* the crossbar via AXI async FIFOs on the affected ports.

Verification Links

- **AXI memory model testbench:** `sim/common/tb_axi_mem.sv`
- **System integration (exercises crossbar paths):** `sim/cocotb/test_sd_spi.py`, `sim/cocotb/test_video_scanout.py`, `sim/cocotb/test_audio_i2s.py`, `sim/cocotb/test_mmu_sv32.py`

16.3 rv32core — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/cpu/core/rv32_core.sv`

Purpose & Role

5-stage RV32IMA core with M/S-mode, interrupts, debug hooks.

Name	Default	Description
HAS_M	1	Enable RV32M mul/div.
HAS_A	1	Enable LR/SC atomics.

Signal	Dir	Width	Description
`clk_i`, `rst_ni`	In	1	Clock/reset.
`i_axi_*`, `d_axi_*`	I/O	—	AXI via I/D caches.
`irq_ext_i`, `irq_timer_i`, `irq_soft_i`	In	1	PLIC/CLINT lines.
`dbg_*`	I/O	—	Debug module/JTAG.

Reset/Init

PC to BootROM; CSRs to reset; pipeline flushed.

Behavior & Timing

- In-order pipeline with forwarding, hazard/flush control.
- Optional BTB; exceptions to trap vector.

Dependencies

`icache`, `dcache`, Sv32 MMU, CLINT, PLIC.

Verification Links

`verification/riscv-isa/rv32ui,rv32um,rv32ua/`.

16.4 fetch — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/cpu/core/fetch.sv`

Purpose & Role

Instruction fetch front-end; issues I-cache requests; handles PC redirects.

Signal	Dir	Width	Description
`pc_q`	In	32	Program counter input.
`ic_req_*/ic_rsp_*`	I/O	—	I-cache request/response bundle.
`redir_i`	In	1	Branch/jump redirect from execute.

Behavior & Timing

Sequential fetch (optional BTB); bubble on miss/redirect; 1-cycle hit path.

Dependencies

- Clocks/Resets: $\texttt{clk}_i, \texttt{rst}_n i(syncrelease)$.
- `Upstream: optional BTB/static predictor; branch redirect from execute.`
- `Downstream: icache request/response; pipeline flush from trap/exception.`
- `Packages: rv32_pkg (opcodes/causes).`

Additional Notes

- PC is 32-bit; redirects take priority over hit returns.
- Bubbles on I$ miss; holds PC until `ic_rsp_valid`.
- `FENCE.I` triggers I$ invalidate hook (via writeback path).

16.5 decode — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/cpu/core/decode.sv`

Purpose & Role

Decodes RV32IMA; reads register file; emits control/imm bundles.

Signal	Dir	Width	Description
`instr_i`	In	32	Fetched instruction.
`rf_rdata_*`	In	32	Register operands from RF.
`ctrl_o/imm_o`	Out	—	Control + immediate to execute.

Behavior & Timing

One-cycle decode; hazard/flush inputs from execute/mem.

Dependencies

- Inputs: `instr_i`, RF read data, hazard/flush from `execute/mem_stage`.
- Outputs: control bundle + immediates to `execute`.
- Packages: `rv32_pkg` (formats, CSR map).

Additional Notes

- Generates I/S/B/U/J immediates; illegal instructions raise trap cause.
- Source operand hazards are flagged to the hazard unit for stall/forward.

16.6 execute — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/cpu/core/execute.sv`

Purpose & Role

ALU/branch stage; mul/div when enabled; provides redirect.

Signal	Dir	Width	Description
`op_a_i/op_b_i`	In	32	Operand A/B.
`alu_res_o`	Out	32	ALU result to MEM/WB.
`branch_taken_o/target_o`	Out	—	Redirect info to fetch.

Behavior & Timing

Single-cycle ALU; multi-cycle mul/div; unique-case FSM for ops.

Dependencies

- Required units: `rv32m_muldiv` (M extension).
- Interfaces: redirect bus to `fetch`; ALU result to `mem_stage`/`writeback`.
- Packages: `rv32_pkg` (ALU ops, branch types).

Additional Notes

- Branch compare is single-cycle; redirect issued on the next cycle.
- Mul/Div are multi-cycle with busy/ready handshake to stall the pipe.
- `FENCE`/`FENCE.I` decode passes side-effects to memory/I$ paths.

16.7 mem_stage — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/cpu/core/mem_stage.sv`

Purpose & Role

D-side loads/stores; interfaces with D-cache/MMU.

Signal	Dir	Width	Description
`ls_ctrl_i`	In	—	Load/store control (size, sign, we).
`dc_req_*/dc_rsp_*`	I/O	—	D-cache request/response bundle.
`wb_data_o`	Out	32	Data to writeback stage.

Behavior & Timing

Align/merge as needed; issues AXI via D-cache on miss; atomics pass-through.

Dependencies

- Downstream: `dcache` + Sv32 `mmu`/`tlb`; atomics may bypass cache.
- Upstream: load/store control from `execute`; writeback sink.

Additional Notes

- Byte-enable generation for stores; sign/zero-extension on loads.
- Enforces LR/SC atomic semantics; aborts on reservation loss.
- D$ miss back-pressures pipeline until refill completes.

16.8 writeback — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/cpu/core/writeback.sv`

Purpose & Role

Final pipeline stage; writes RD/CSR; commits exceptions.

Signal	Dir	Width	Description
`rd_addr_i/rd_data_i`	In	5/32	Dest reg & value.
`rd_we_o`	Out	1	Register write enable.
`csr_*`	I/O	—	CSR read/write side effects.

Behavior & Timing

One-cycle commit; CSR/exception routing to trap logic.

Dependencies

- Sources: MEM/ALU results; CSR side-effects; trap/exception info.
- Sinks: register file write port; trap vector routing.

Additional Notes

- Commits in-order; exceptions take priority and squash the commit.
- CSR writes respect read/modify/write rules and cause checks.

16.9 sv32_mmu — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/cpu/mmu/sv32_mmu.sv`

Purpose & Role

Sv32 virtual memory translation unit. Handles page table walks, TLB management, and access permissions for S-mode operation.

Signal	Dir	Width	Description
`va_i`	In	32	Virtual address input.
`pa_o`	Out	34	Physical address output.
`ptw_req_*/ptw_rsp_*`	I/O	—	Page table walker interface.
`satp_i`	In	32	SATP register value.

Behavior & Timing

TLB hit: 1-cycle translation. TLB miss triggers page walk (multi-cycle).

Dependencies

- Cooperates with: `tlb` (I/D); `ptw` issues AXI reads for PTEs.
- Inputs: `satp`, privilege, SUM/UXN bits via CSR file.

Additional Notes

- Performs R/W/X and U/S permission checks; sets A/D bits via PTW.
- SFENCE.VMA (global or ASID-specific) invalidates affected entries.

16.10 tlb — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/cpu/mmu/tlb.sv`

Purpose & Role

Translation Lookaside Buffer for caching page table entries. Split I/D TLBs with fully-associative structure.

Name	Default	Description
ENTRIES	16	Number of TLB entries.

Behavior & Timing

CAM-based lookup; LRU replacement; SFENCE.VMA flush support.

Dependencies

- Connected to: `sv32_mmu` and `ptw` miss path.
- Control: SFENCE.VMA + `satp` write -> global shootdown.

Additional Notes

- Implements LRU (or pseudo-LRU) replacement; supports large-page entries if enabled.
- Distinct I/D TLBs; may share shootdown domain.

16.11 uart16550 — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/io/uart16550.sv`

Purpose & Role

16550-compatible UART with 16-byte FIFOs. Console interface for BootROM/FSBL/debug.

Signal	Dir	Width	Description
`axi_lite_*`	I/O	—	AXI-Lite slave interface.
`tx_o/rx_i`	Out/In	1	Serial TX/RX pins.
`irq_o`	Out	1	Interrupt output.

Behavior & Timing

Programmable baud rate; 8N1 format; TX/RX FIFOs with threshold IRQs.

Dependencies

- Bus: AXI-Lite via `periph_axi_shell` (or bridge).
- Interrupt: wired into `plic`.
- SW: BootROM/FSBL configures baud divisor before first use.

Additional Notes

- LSR/IER/ISR follow 16550 behavior; TX empty and RX threshold IRQs.
- Baud generator derives from `clk_i`; document divisor table in driver.

16.12 fb_ctrl — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/video/fb_ctrl.sv`

Purpose & Role

Framebuffer controller. DMA fetches pixels from DDR, applies palette/COLORMAP, drives VGA timing.

Name	Default	Description
H_RES	640	Horizontal resolution.
V_RES	480	Vertical resolution.

Behavior & Timing

320×200 8bpp upscaled to 640×480; double-buffering; vsync IRQ at 60Hz.

Dependencies

- DMA: reads via AXI from DDR; optional `dma_subsys` assist.
- Timing: `vga_timing` block; vsync IRQ to `plic`.
- Regs: `specs/registers/video.yaml` (FB base/stride, palette, colormap).

Additional Notes

- Double-buffer swap on vsync to avoid tearing.
- Palette/COLORMAP expand 8bpp to RGB; stride must be line-aligned.
- AXI bursts sized to cache lines for sustained bandwidth.

16.13 sd_spi_ctrl — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/sd/sd_spi_ctrl.sv`

Purpose & Role

SD card SPI-mode controller for boot media and WAD file access.

Signal	Dir	Width	Description
`spi_mosi_o`	Out	1	SPI master out.
`spi_miso_i`	In	1	SPI master in.
`spi_sclk_o`	Out	1	SPI clock.
`spi_cs_o`	Out	1	Chip select.

Behavior & Timing

CMD17/18 single/multi-block reads; CRC16 checking; 25MHz SPI clock.

Dependencies

- SPI pins to PMOD; optional DMA for block copy to DDR.
- Interrupt: data-ready/error IRQ -> `plic`.
- SW: FSBL driver handles init sequence (CMD0, CMD8, ACMD41).

Additional Notes

- 512-byte block length; single/multi-block reads with STOP_TRAN token.
- CRC enable optional; error/status latched until cleared by SW.

16.14 i2s_tx — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/audio/i2s_tx.sv`

Purpose & Role

I2S audio transmitter for PMOD I2S2 DAC. Streams PCM samples from ring buffer.

Signal	Dir	Width	Description
`i2s_sdata_o`	Out	1	Serial data.
`i2s_bclk_o`	Out	1	Bit clock.
`i2s_lrclk_o`	Out	1	L/R word select.

Behavior & Timing

16-bit stereo @ 11025Hz; DMA from audio ring buffer.

Dependencies

- Clocking: BCLK/LRCLK derived from system clock (dividers in regs).
- Source: audio DMA/ring buffer managed by driver.

- IRQ: underrun interrupt -> `plic`.

Additional Notes

- Left-then-right frame ordering; MSB-first per I2S.
- Underrun forces zeros until buffer refilled; sticky status bit.

16.15 clint — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/irq/clint.sv`

Purpose & Role

Core Local Interruptor. Provides mtime, mtimecmp timer interrupts, and software interrupts.

Signal	Dir	Width	Description
`mtime_o`	Out	64	Current timer value.
`timer_irq_o`	Out	1	Timer interrupt.
`soft_irq_o`	Out	1	Software interrupt.

Behavior & Timing

Free-running 64-bit timer @ system clock; compare triggers IRQ.

Dependencies

- Memory map base from `specs/memory`$_{map.yaml}$; $timer tick based on$ `CLK`$_{HZ}$.
- `Interrupts: timer/software lines into core.`

Additional Notes

- 64-bit `mtime` increments monotonically; `mtimecmp` match latches IRQ.
- SW uses MSIP for inter-processor events (reserved here for single-core).

16.16 plic — Module Brief (v0.1)

Owner: Rafeed Khan & Sebastian Candelaria **RTL:** `rtl/irq/plic.sv`

Purpose & Role

Platform-Level Interrupt Controller. Priority-based external interrupt routing.

Name	Default	Description
N_SOURCES	32	Number of interrupt sources.

Behavior & Timing

Priority arbitration; claim/complete protocol; threshold filtering.

Dependencies

- Sources: UART, SD, VIDEO, AUDIO, GPIO, etc.
- Sink: external interrupt line to core; context 0 (machine-mode).

Additional Notes

- Implements claim/complete with priority thresholding.
- Level-sensitive inputs; edge semantics (if any) handled at source.
- SW must clear device status before completing to avoid reassertion.

This document is versioned in the repository alongside code. Treat it as a normative standard.