FV (Formal Verification) and UVM DV (Universal Verification Methodology Design Verification) engineer, here's how you can approach the RTL file (`mac_array_8x8.v`, presumably a 8x8 MAC array module):

🔍 1. Understand the Design (Review the RTL)

Before jumping into verification, understand what the RTL does:

- **Functionality:** It's likely an 8x8 Multiply-Accumulate (MAC) array — commonly used in neural networks and signal processing.
- **Ports & Parameters:** Identify all inputs, outputs, clock/reset signals, and any parameterizations.
- **Data Flow:** Trace how data moves through the design. Look for pipeline stages, accumulation logic, multipliers, etc.

✅ 2. Functional Verification with UVM

To verify this RTL using UVM, follow this roadmap:

📁 Testbench Structure

Create a modular UVM environment with:

- **Interface:** Bundle inputs/outputs (clock, reset, data, valid/ready, etc.)
- **Transaction Class:** Define stimulus packet (e.g., input operands, expected result).
- **Driver/Monitor:** Drive inputs, sample outputs.
- **Sequencer + Sequences:** Generate stimulus patterns (random, directed, edge-cases).
- **Scoreboard:** Compare DUT output with a golden model or reference behavior.
- **Coverage Collector:** Track functional and code coverage (e.g., corner cases like overflow, zero, negative inputs if signed).

🔒 3. Formal Verification (FV)

For FV, you can:

✅ Property Checks (SVA)

- **Assertions:** Use SystemVerilog Assertions (SVA) to check invariants like:
    - Output is updated correctly after N cycles.
    - No overflow occurs (unless designed to).
    - Result matches expected multiply-accumulate behavior.
    - Reset behavior is correct.

✅ Formal Tool Usage

Use tools like **JasperGold, VC Formal, QuestaPropCheck**, etc. Apply:

- **End-to-End assertions**
- **Assume-Guarantee modeling**
- **Cover properties** (to show stimulus reachability)

📊 4. Additional Considerations

- **Corner Cases:** Verify edge inputs (e.g., all 1s, all 0s, alternating bits).
- **Performance Testing:** Add latency, throughput checks.
- **Power/Timing Checks:** May be optional depending on the flow.

👉 Next Step:

step-by-step procedure tailored for you as an **FV and UVM DV engineer** to verify the `mac_array_8x8.v` RTL code using **industry-standard tools**. I'll split this into two parts:

🔷 Part 1: UVM-Based Simulation Verification (Functional DV)

📌 Tools Used:

- **Mentor QuestaSim/Modelsim**, **Synopsys VCS**, **Cadence Xcelium** (any SystemVerilog + UVM-supported simulator)

✅ Step 1: Analyze RTL

- Open `mac_array_8x8.v` and understand:
    - Inputs, outputs, clock/reset
    - Data format (bit-width, signed/unsigned)
    - Latency or pipelining stages
    - Any control signals (valid, ready, enable, etc.)

✅ Step 2: Create UVM Testbench

UVM testbench includes the following files:

1. **Interface (`mac_if.sv`)**
   Contains bundled I/O ports, clock, reset.

2. **Transaction class (`mac_trans.sv`)**
   Encodes inputs & expected outputs.

3. **Driver & Monitor (`mac_driver.sv`, `mac_monitor.sv`)**

4. **Sequencer + Sequences (`mac_seqr.sv`, `mac_seq.sv`)**

5. **Scoreboard (`mac_scoreboard.sv`)**
   Compares DUT output with golden reference.

6. **Test Class (`mac_test.sv`)**

7. **Top TB (`tb_top.sv`)**
   Instantiates DUT + UVM env.

✅ Step 3: Compile with Simulator

For example, with **QuestaSim**:

bash
```
vlib work
vlog +acc -sv mac_array_8x8.v mac_if.sv tb_top.sv
vsim -novopt tb_top
```

✅ Step 4: Run Simulation

tcl

```
run -all
```

Check logs, waveform (with `do wave.do`), and scoreboard results.

✅ Step 5: Check Coverage

bash

```
vcover merge tb_top.ucdb
vcover report -details tb_top.ucdb
```

🔶 Part 2: Formal Verification (Using JasperGold / QuestaPropCheck / VC Formal)

✅ Step 1: Set Up Design Environment

Organize your files:

arduino

```
fv/
├── rtl/
│   └── mac_array_8x8.v
├── properties/
│   └── mac_array_properties.sv
├── run/
    └── formal_run.tcl
```

✅ Step 2: Write Formal Properties

Use **SystemVerilog Assertions (SVA)** in `mac_array_properties.sv`, e.g.:

systemverilog

```
property reset_clears_output;
  @(posedge clk) disable iff (!rst_n)
    rst_n ##1 (out == 0);
endproperty
assert property(reset_clears_output);
```

✅ Step 3: Launch Tool (e.g., JasperGold)

bash

```
jaspergold formal_run.tcl
```

Sample `formal_run.tcl`:

tcl

```
read_verilog rtl/mac_array_8x8.v
read_sva properties/mac_array_properties.sv
set_top_module mac_array_8x8
run_prove
exit
```

✅ Step 4: Analyze Results

- **PASS:** Property proven ✅
- **FAIL:** Counterexample generated, check waveform

Use:

- **Cover checks** to ensure stimulus reachability.
- **Assume constraints** for valid input behavior.

✅ Summary: Tool-Specific Commands

Tool	Usage Command Example
QuestaSim	`vlog`, `vsim`, `run -all`, `vcover report`
Synopsys VCS	`vcs -sverilog`, `./simv`, `urg` for coverage
Cadence Xcelium	`xrun -uvm -sv`
JasperGold	`jaspergold run.tcl`
VC Formal	`vfc -f filelist.f`, `vfc_run`

8x8 MAC (Multiply-Accumulate) array, computing outputs like `mac_out0 = a0 * b0`, and so on, with 8-bit inputs and 16-bit outputs.

✅ PART 1: UVM Testbench Template for `mac_array_8x8`

Here's a working **UVM testbench template** (minimal, modular structure):

📁 `mac_if.sv` – Interface

Systemverilog

```
interface mac_if(input bit clk);
    logic rst;
    logic [7:0] a[8];
    logic [7:0] b[8];
    logic [15:0] mac_out[8];
endinterface
```

📄 `mac_trans.sv` – Transaction

systemverilog

```
class mac_trans extends uvm_sequence_item;
    rand bit [7:0] a[8];
    rand bit [7:0] b[8];
    bit [15:0] exp_mac_out[8];

    `uvm_object_utils(mac_trans)
    function new(string name = "mac_trans");
        super.new(name);
    endfunction
endclass
```

`mac_driver.sv` – Driver

systemverilog

```
class mac_driver extends uvm_driver#(mac_trans);
    virtual mac_if vif;

    function void build_phase(...);
        if (!uvm_config_db#(virtual mac_if)::get(..., vif))
            `uvm_fatal(...)
    endfunction

    task run_phase(...);
        forever begin
            seq_item_port.get_next_item(req);
            for (int i = 0; i < 8; i++) begin
                vif.a[i] = req.a[i];
                vif.b[i] = req.b[i];
            end
            seq_item_port.item_done();
        end
    endtask
endclass
```

`mac_monitor.sv` – Monitor

systemverilog

```
class mac_monitor extends uvm_monitor;
    virtual mac_if vif;
    uvm_analysis_port#(mac_trans) ap;

    task run_phase(...);
        mac_trans tx = new();
        wait (!vif.rst);
        forever begin
            for (int i = 0; i < 8; i++)
                tx.exp_mac_out[i] = vif.mac_out[i];
            ap.write(tx);
```

```
            @(posedge vif.clk);
        end
    endtask
endclass
```

`mac_scoreboard.sv` – Scoreboard

systemverilog

```
class mac_scoreboard extends uvm_component;
    uvm_analysis_imp#(mac_trans, mac_scoreboard) imp;

    function void write(mac_trans t);
        for (int i = 0; i < 8; i++) begin
            bit [15:0] expected = t.a[i] * t.b[i];
            if (expected !== t.exp_mac_out[i])
                `uvm_error("MAC_ERR", $sformatf("Mismatch at %0d",
i));
        end
    endfunction
endclass
```

`tb_top.sv` – DUT + Interface + Testbench

systemverilog

```
module tb_top;
    bit clk = 0;
    always #5 clk = ~clk;

    mac_if intf(clk);
    mac_array_8x8 dut (
        .clk(clk), .rst(intf.rst),
        .a0(intf.a[0]), .a1(intf.a[1]), ... .a7(intf.a[7]),
        .b0(intf.b[0]), .b1(intf.b[1]), ... .b7(intf.b[7]),
        .mac_out0(intf.mac_out[0]), ... .mac_out7(intf.mac_out[7])
    );
```

```
    initial begin
        run_test();
    end
endmodule
```

✅ PART 2: Formal Properties (SVA) for `mac_array_8x8`

Here's a solid starting set of **SystemVerilog Assertions** (SVA):

📄 `mac_array_properties.sv`

Systemverilog

```
module mac_array_properties(input clk, input rst, input [7:0] a[8],
b[8], input [15:0] mac_out[8]);

    // Property: Reset clears outputs
    genvar i;
    generate
        for (i = 0; i < 8; i++) begin : gen_rst_assert
            property reset_clears_output;
                @(posedge clk) rst |=> (mac_out[i] == 0);
            endproperty
            assert property(reset_clears_output);
        end
    endgenerate

    // Property: Output equals a[i]*b[i] on next clock (1-cycle
latency assumed)
    generate
        for (i = 0; i < 8; i++) begin : gen_mac_check
            property correct_mac;
                @(posedge clk) (!rst) |=> (mac_out[i] == a[i] * b[i]);
            endproperty
            assert property(correct_mac);
        end
```

```
    endgenerate

endmodule
```

🔧 Next Steps for You

1. **UVM Compilation (QuestaSim)**

bash

```
vlog +acc -sv tb_top.sv mac_driver.sv mac_monitor.sv mac_trans.sv ...
vsim tb_top
run -all
```

2. **Formal Verification (JasperGold)**

tcl

```
read_verilog mac_array_8x8.v
read_sva mac_array_properties.sv
set_top_module mac_array_8x8
run_prove
```