The Chip Designer's Guide to the Galaxy

Chronicles of RTL Engineering: From University Theory to Professional SystemVerilog Workflow

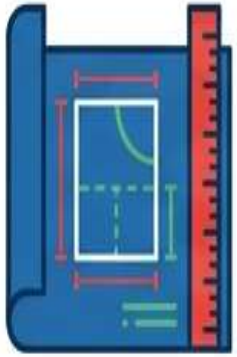
Tomer Habany, BSc Electrical Engineering

Linkedin: https://www.linkedin.com/in/tomer-habany/

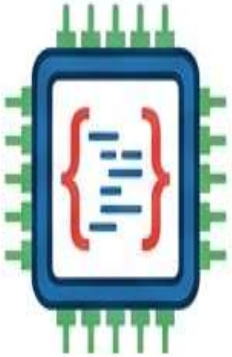
Email: Tntomer96@gmail.com

The Mission

To demystify the transition to professional RTL design by building a functional library of digital modules, emphasizing architecture, hardware-centric coding, and rigorous verification.

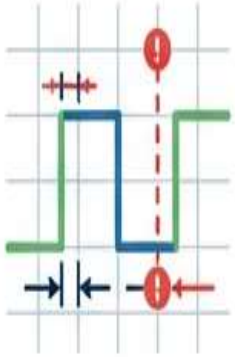


Architecture Before RTL



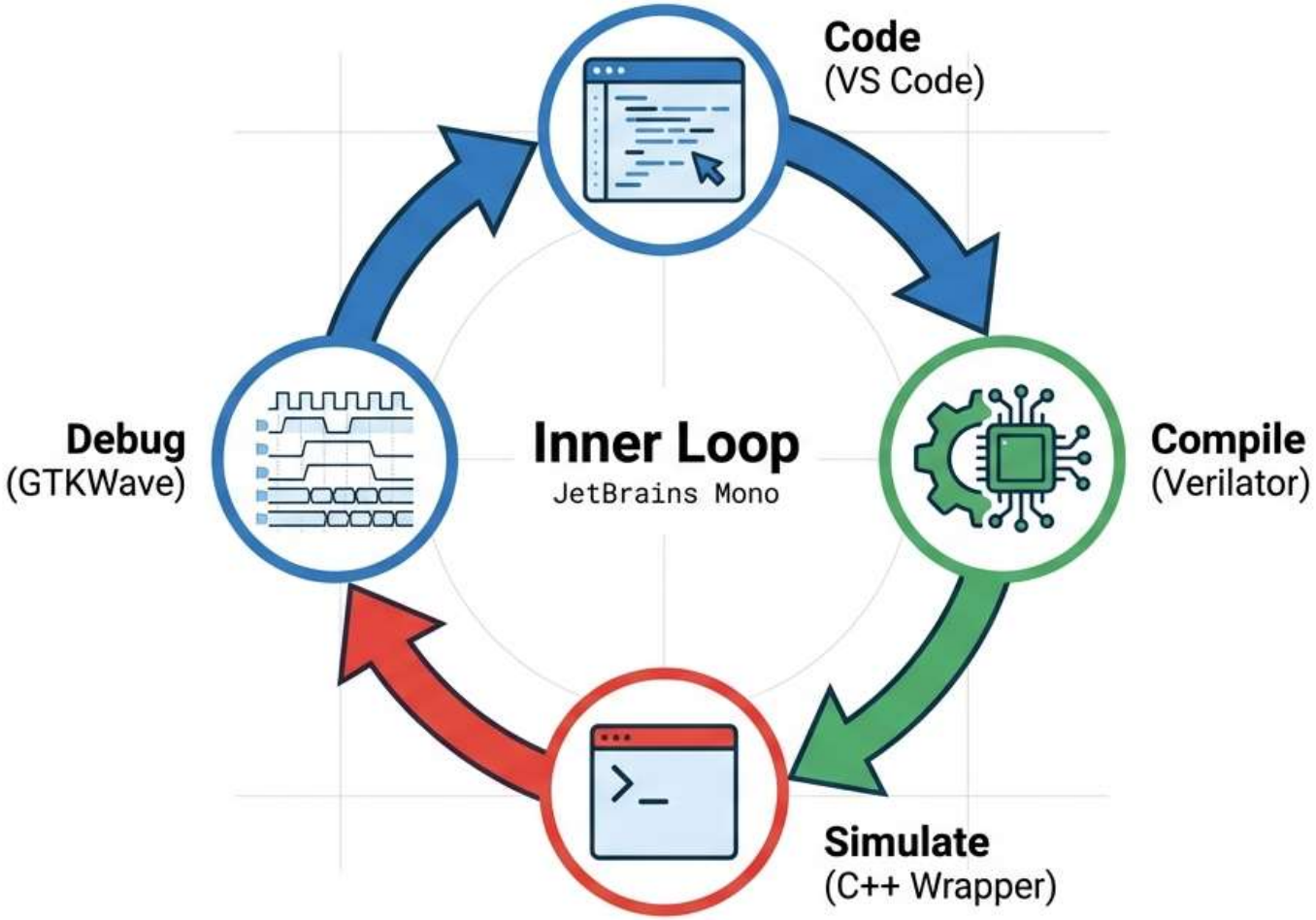
Hardware-Centric Coding

High-Level Verification



Waveform Debugging

Constructing the Laboratory: The Toolchain



The Stack

OS
Ubuntu via WSL2 (Windows Subsystem for Linux)
Industry standard environment.

The Compiler
Verilator
Converts SystemVerilog to optimized C++ models.
Speed + Strict Linting.

The Eyes
GTKWave
Visualization interface for debugging logic and timing.

The Engine
Make
Automates build dependencies with a single command.

From Behavioral Equations to Structural Architecture

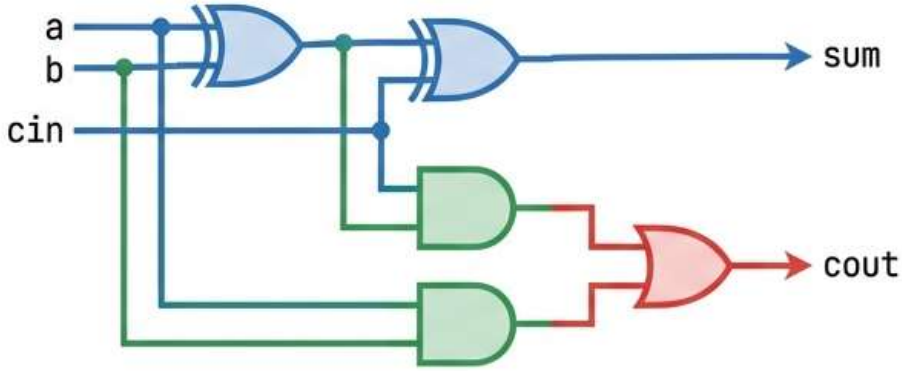
The 1-Bit Full Adder

The Theory (Behavioral)

```
sum  = a ^ b ^ cin
cout = (a & b) | (cin & (a ^ b))
```

⇉ Simulation Delay: `assign #1` represents gate propagation time (Ripple Effect).

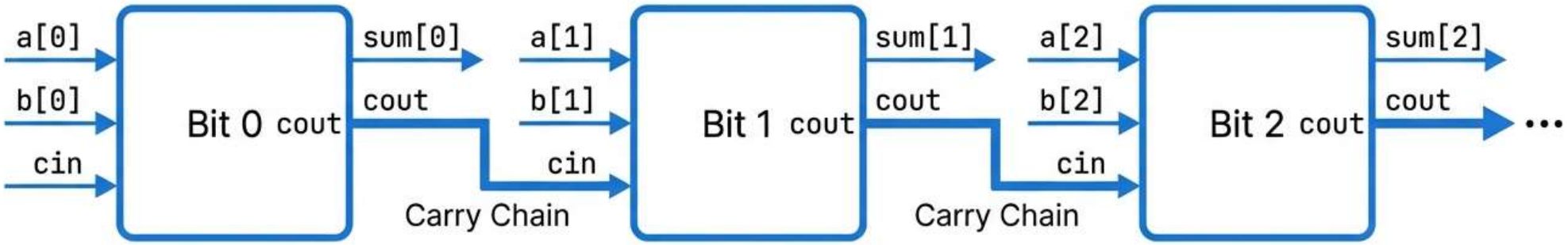
The Architecture (Structural)




```
module adder_1bit (
  input logic a, b, cin,
  output logic sum, cout
);
  assign #1 sum = a ^ b ^ cin;
  assign #1 cout = (a & b) | (cin & (a ^ b));
endmodule
```

Scaling Complexity: The Parameterizable Ripple Carry Adder

Using Generators to create scalable hardware IP.



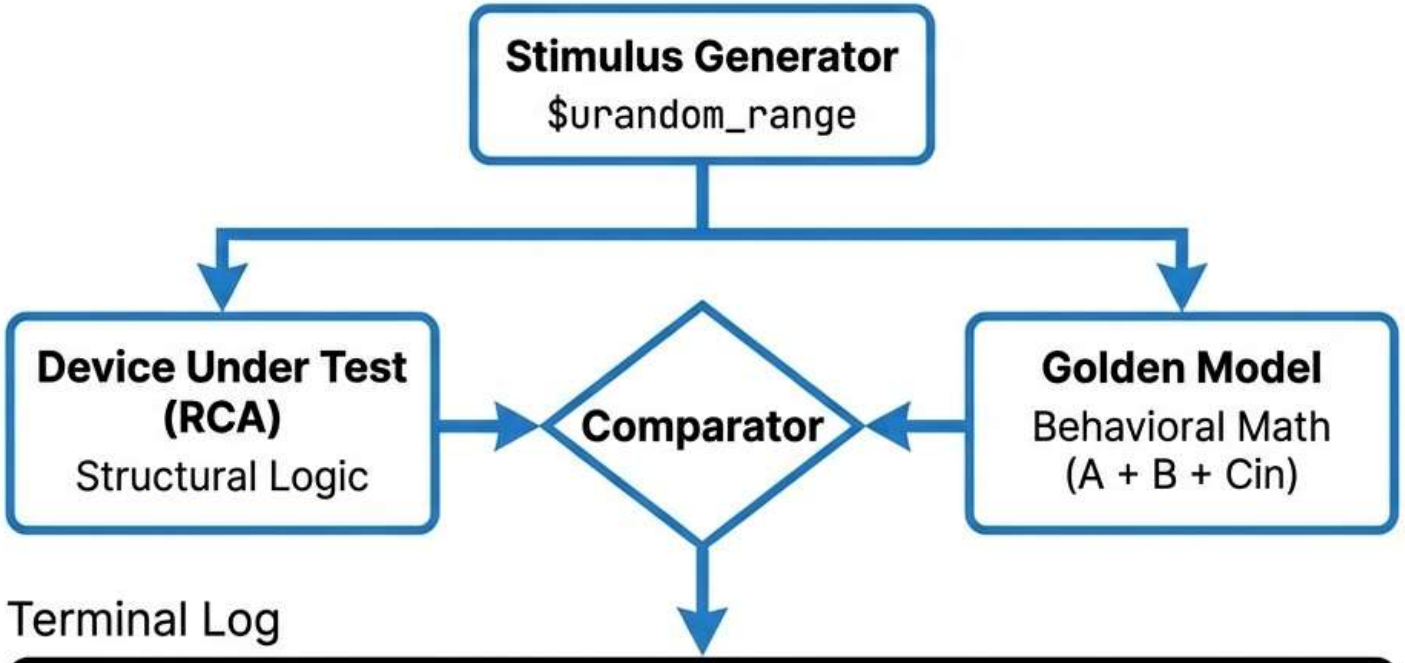

```
module rca #(parameter WIDTH = 8) (...);
    genvar i;
    generate
        for (i = 0; i < WIDTH; i++) begin : adder_loop
            adder_1bit u_adder (
                .a(a[i]), .b(b[i]), .cin(carry_chain[i]),
                .sum(sum[i]), .cout(carry_chain[i+1])
            );
        end
    endgenerate
endmodule
```

Compile-time iteration variable

User-definable bit width

Trust but Verify: Automation and the Golden Model

Self-Checking Testbench



Terminal Log

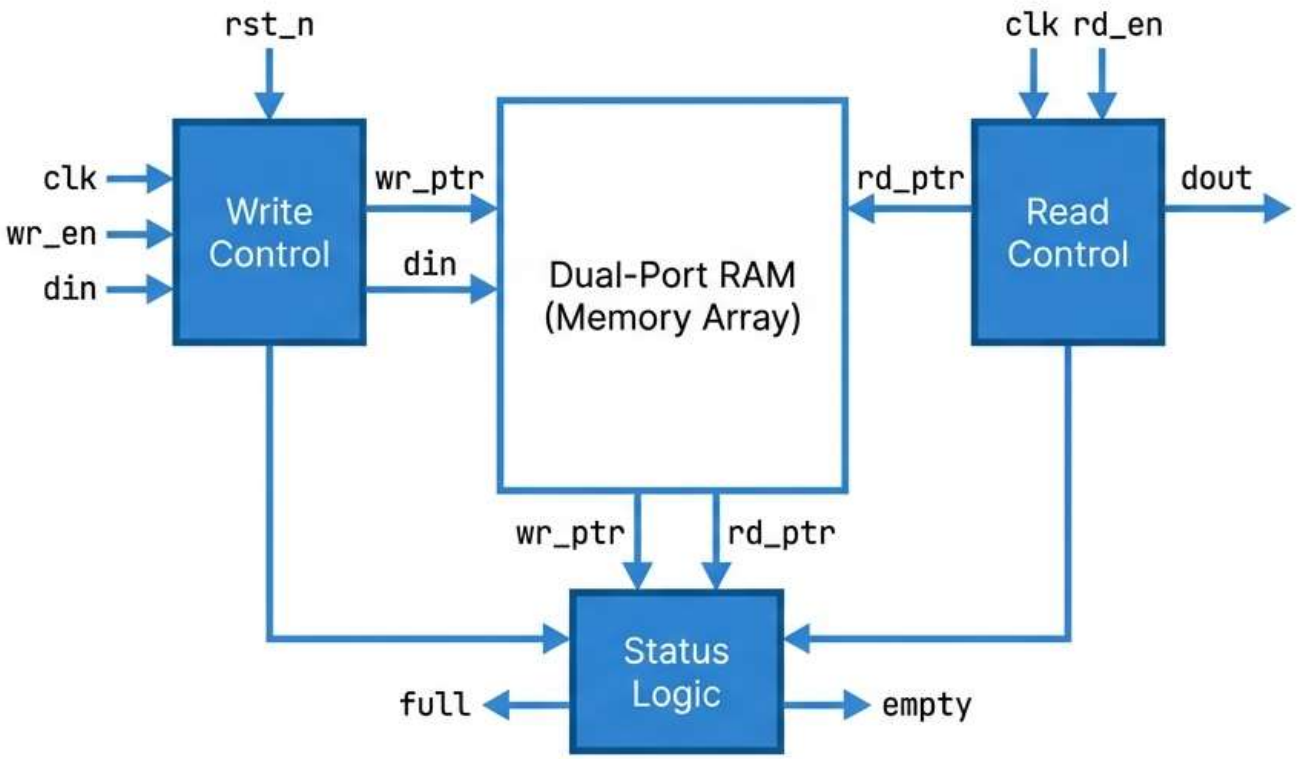
```
PASSED: a=104, b=22, cin=1 | Sum=127, Cout=0
PASSED: a=50, b=50, cin=0 | Sum=100, Cout=0
```

- **Stimulus:** Generates unpredictable inputs.
- **Golden Model:** The trusted mathematical truth (`expected = a + b + cin`).
- **The Check:**

```
if (sum !== expected)
  $error("Mismatch!");
```

The Waiting Room: Synchronous FIFO Architecture

Managing Data Flow with Sequential Logic and Memory



Interface:

- Inputs:
  - `clk`
  - `rst_n`
  - `wr_en`
  - `rd_en`
  - `din`
- Outputs:
  - `dout`
  - `full`
  - `empty`

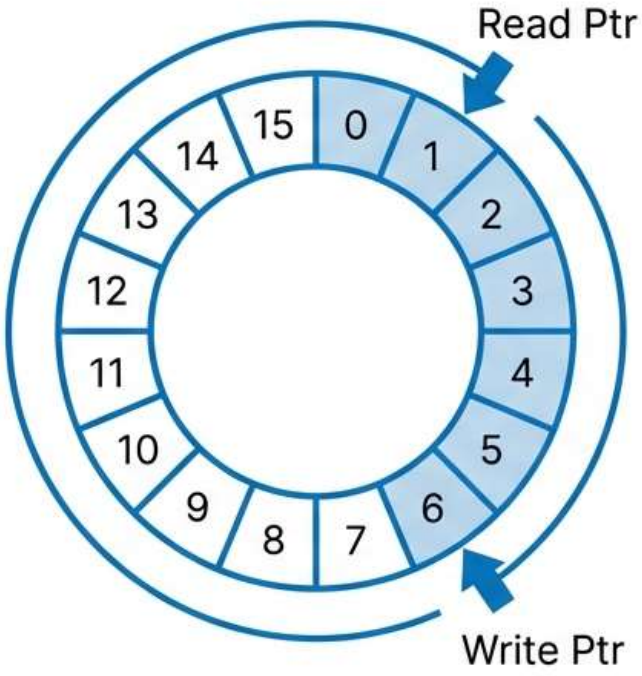
Constraints:

- `wr_valid = wr_en && !full` (Prevents Overflow)
- `rd_valid = rd_en && !empty` (Prevents Underflow)

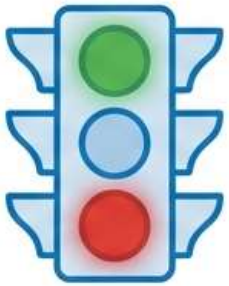
Technical Swiss Editorial

Solving the Empty vs. Full Dilemma

The N+1 Bit Trick for Circular Buffers



Empty
`count == 0`



Full
`count == DEPTH`
(Bit 4 is high)

The Problem: In a circular buffer, `wr_ptr == rd_ptr` could mean Empty OR Full.

The Solution: Use a counter with one extra bit.

Example: Depth 16 (4 bits). Counter is 5 bits.
- 00000 = Empty
- 10000 = Full (16 items)

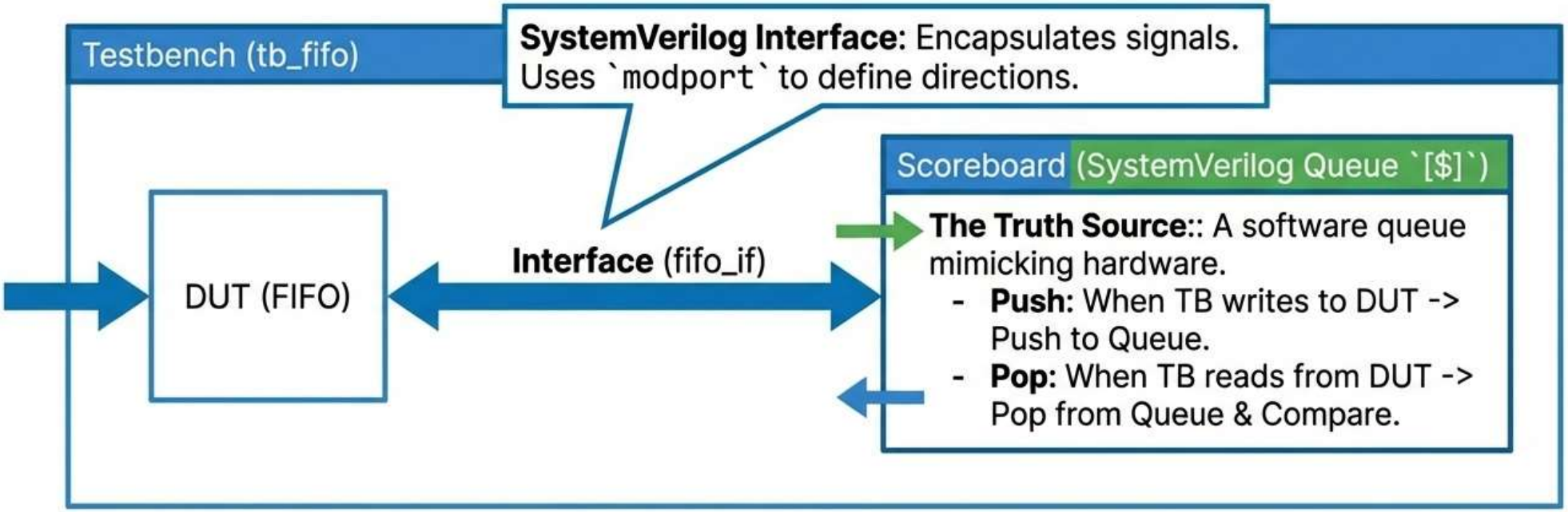
```
localparam PTR_WIDTH = $clog2(DEPTH);
logic [PTR_WIDTH:0] count; // N+1 bits

assign full = (count == DEPTH);
assign empty = (count == 0);
```

Technical Swiss Editorial

Advanced Verification: Interfaces and Scoreboards

Systematic Approach to Ensuring FIFO Data Integrity and Ordering

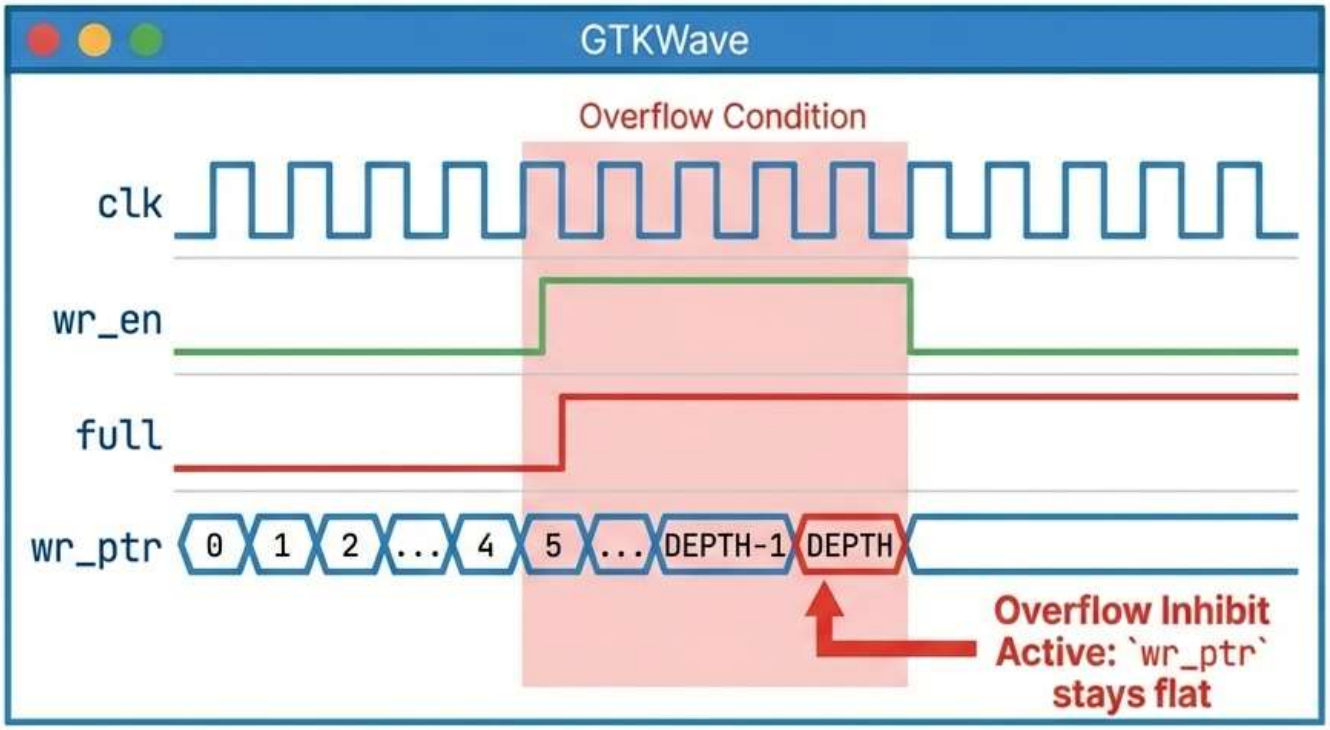


Benefit: "Ensures data integrity and strict First-In-First-Out ordering."

Technical Swiss Editorial

Trying to Break It: Corner Cases and Stress Testing

Systematic approach to identifying and validating boundary conditions and resilience



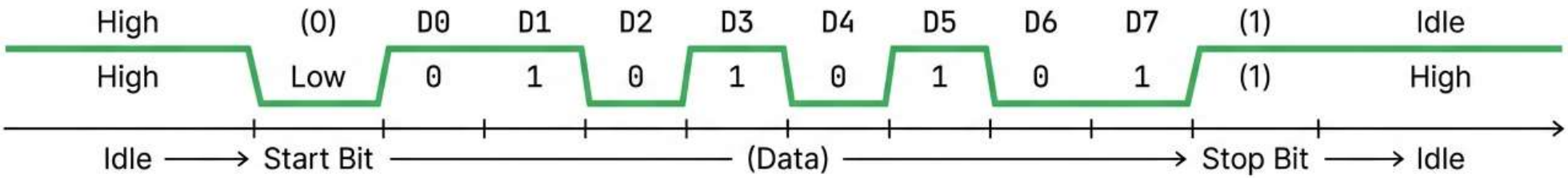
4 Verification Scenarios:

1. **Overflow (Fill Test):** Write `DEPTH + 5` items. Verify `full flag blocks excess.
2. **Underflow (Empty Test):** Read from empty. Verify `empty flag prevents garbage data.
3. **Sustained Throughput:** Simultaneous Read/Write (`fork-join`). Verify pointer independence.
4. **The "Surprise" Reset:** Trigger async reset mid-operation. Verify instant recovery.

Technical Swiss Editorial

The Bridge to the Outside World: UART Transmitter

Universal Asynchronous Receiver-Transmitter



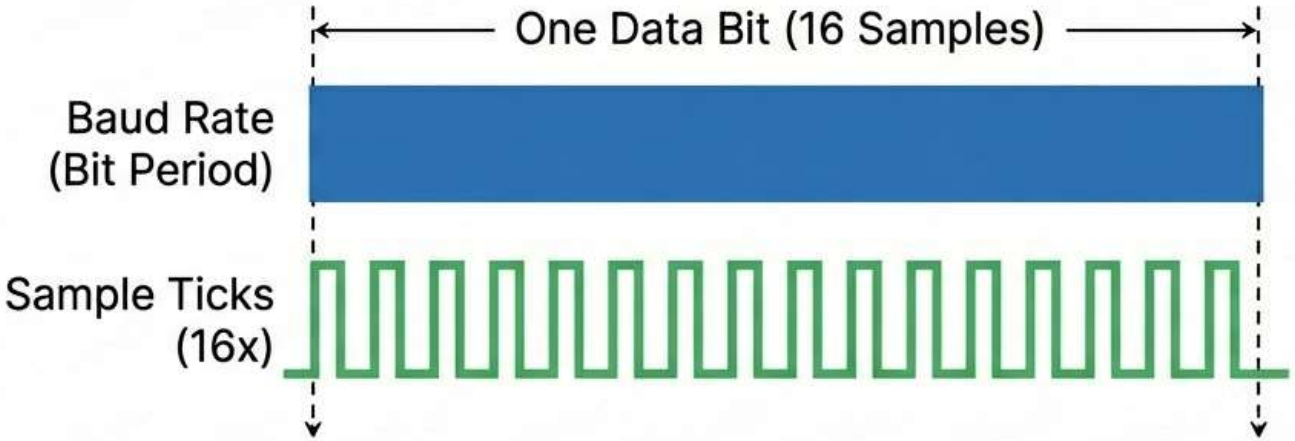
Key Concepts

- **Asynchronous:** No shared clock. Relies on agreed “Baud Rate”.
- **Serializer:** Converts 8-bit parallel to 1-bit serial.
- **Framing:** Wraps data in Start(0) and Stop(1) bits.

Technical Swiss Editorial

The Heartbeat: Precision Baud Rate Generation

Oversampling for Data Integrity



The Math:

Formula:

$$\texttt{TICK\_LIMIT} = \frac{\texttt{CLOCK\_FREQ}}{(\texttt{BAUD\_RATE} * 16)}$$

Example:

`100MHz / (115200 * 16)`

≈ 54 cycles per tick

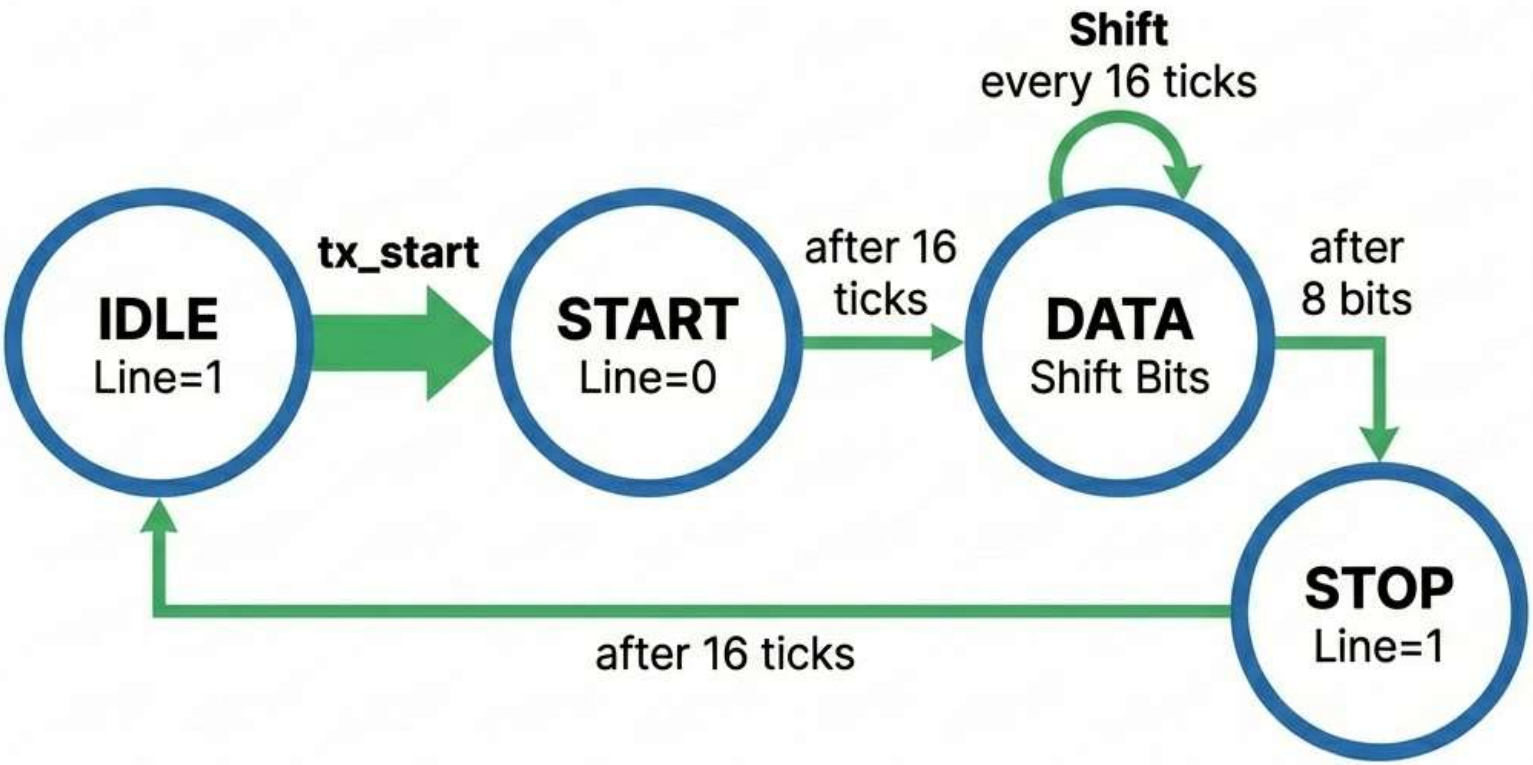
Logic Flow:

- Counter increments on `clk`.
- At `TICK_LIMIT`, pulse `sample_tick` and Reset.
- **Sync Feature:** Generator is enabled (en) only when transmission starts (aligns phase to Start Bit).

Technical Swiss Editorial

Orchestrating the Frame: The Finite State Machine

Finite State Machine (FSM) for UART Transmission Control



Internal Counters:

- `tick_count`: Tracks the 16 oversampling ticks.
- `bit_count`: Tracks the 8 data bits transferred.

Technical Swiss Editorial

Professional RTL Architecture: The Two-Block Style

Sequential Logic (`always_ff`)

```
always_ff @(posedge clk or negedge rst_n) begin
  if (!rst_n) state_reg <= IDLE;
  else        state_reg <= state_next;
end
```

Handles Reset & Clock updates.
Defines *what* the state is.

Combinational Logic (`always_comb`)

```
always_comb begin
  state_next = state_reg; // Default (Anti-Latch)
  case (state_reg)
    IDLE: if (start) state_next = START;
    // ... next state logic ...
  endcase
end
```

Calculates transitions.
Defines *how* the state changes.

Standard industry practice for timing stability and glitch prevention.

The Integrated System