Semester Long Internship Report

On

Designing of IP Blocks

& Verification using Mixed Signal Simulation in eSim

Submitted by

P. B. S. V. VAMSI

B.Tech (Electronics and Communication)

Rajiv Gandhi University of Knowledge Technologies, Nuzvid

Under the Guidance of

Prof. Prabhu Ramachandran

Principal Investigator,
Department of Aerospace Engineering,
Indian Institute of Technology Bombay

June 5, 2026

Acknowledgment

I express my sincere gratitude to Prof. Prabhu Ramachandran for providing me with the opportunity to be part of the FOSSEE internship program and for his continued efforts in promoting open-source engineering tool development. His leadership and vision have been instrumental in fostering meaningful student par-ticipation in the open-source ecosystem.

I also acknowledge Prof. Kannan M. Moudgalya for his foundational role in establishing and strengthening the FOSSEE initiative. His contributions to open-source education and the development of the FOSSEE fellowship framework have been pivotal in creating the academic and organisational platform through which this internship was undertaken.

My sincere appreciation extends to my mentor, Sumanto Kar, for his continual support, technical guidance, and encouragement throughout the duration of this project. His insights and feedback played a key role in refining ideas, overcoming challenges, and ensuring timely completion of the tasks assigned to me.

I would also like to thank my internal mentors, Mr. Varad Patil and Ms. Shanthi Priya K for their valuable guidance, coordination, and technical inputs during the internship. Their mentorship contributed significantly to the clarity, progress, and successful execution of the work.

Contents

Chapter 1

Introduction

The rapid growth of modern electronic systems has significantly increased the complexity of digital and analog circuit design. Designing reliable hardware systems requires extensive verification and testing before physical implementation to minimize design errors, reduce development cost, and improve overall system performance. Therefore, **Electronic Design Automation (EDA)** tools play an essential role in enabling efficient circuit modeling, simulation, verification, and analysis.

Although several commercial EDA tools are available for industrial applications, their high licensing cost often limits accessibility for students, researchers, and academic institutions. To address this challenge, open-source EDA platforms have emerged as cost-effective and flexible alternatives for electronic system development.

One such platform is **eSim**, developed under the **FOSSEE (Free/Libre and Open Source Software for Education)** project at **IIT Bombay**. eSim is an open-source EDA tool designed for circuit design, simulation, mixed-signal verification, and PCB development. It integrates multiple open-source simulation and verification tools into a single environment, enabling users to design and analyze **analog, digital, and mixed-signal systems** efficiently.

During this internship, eSim was utilized for the **design, implementation, and verification of multiple digital Intellectual Property (IP) blocks** using **mixed-signal simulation techniques**. Various reusable and industry-relevant IP blocks were modeled using **Verilog HDL**, integrated as subcircuits, and verified through waveform analysis in both **eSim** and **Vivado**. The internship provided practical exposure to **digital system design, mixed-signal simulation workflows, IP integration, and hardware verification methodologies**.

1.1 Overview of eSim

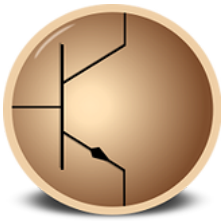


Figure 1.1: eSim Logo

eSim is an open-source **Electronic Design Automation (EDA)** software developed under the **FOSSEE** project for circuit design, simulation, and PCB development. It provides an integrated environment for creating, simulating, and verifying **analog, digital, and mixed-signal circuits**. By combining several open-source tools into a unified platform, eSim simplifies the overall electronic design process.

The platform supports the complete electronic design workflow, beginning with **schematic creation**, followed by **simulation and verification**, and extending to **PCB design implementation**. This integrated approach helps users analyze circuit functionality before hardware realization, thereby reducing implementation errors and improving reliability.

1.2 Tools Integrated in eSim

eSim integrates multiple open-source tools within a single environment to support the complete electronic system design workflow, including schematic capture, simulation, digital verification, device modeling, and PCB implementation. These integrated tools collectively enable efficient analysis and verification of analog, digital, and mixed-signal circuits.

1.2.1 Ngspice

Ngspice is the primary simulation engine integrated within eSim for performing **analog, digital, and mixed-signal circuit simulations**. It is an open-source SPICE-based simulator widely used for analyzing circuit behavior under different operating conditions.

Ngspice supports several simulation techniques such as **DC analysis**, **AC analysis**, **transient analysis**, and **operating point analysis**. These analyses enable designers to study voltage, current, timing characteristics, frequency response, and overall circuit performance before physical hardware implementation. By allowing early-stage verification, Ngspice helps reduce design errors and improves circuit reliability.

1.2.2 NgVeri

NgVeri is a digital simulation and verification tool integrated with eSim for analyzing hardware designs written in **Verilog and SystemVerilog**. It provides an efficient environment for functional verification of digital circuits through waveform generation and behavioral analysis.

NgVeri enables users to validate the correctness of digital logic, observe signal transitions, and identify functional errors during the design phase. It plays a significant role in verifying reusable **digital Intellectual Property (IP) blocks** and ensuring proper operation before hardware synthesis.

1.2.3 NGHDL

NGHDL is an integrated simulation framework used for verifying digital systems modeled using the **VHDL hardware description language**. It allows designers to simulate VHDL-based digital circuits and analyze their functionality through waveform inspection.

In addition to digital verification, NGHDL supports interaction between analog and digital domains, making it suitable for **mixed-signal system simulation**. This capability helps designers validate complex hardware systems involving both analog and digital components.

1.2.4 Makerchip-App

Makerchip-App is a browser-based digital design and simulation platform integrated with eSim to simplify hardware design workflows. It provides an interactive environment for designing, simulating, debugging, and visualizing digital circuits.

The platform supports rapid prototyping of hardware logic and enables users to observe real-time simulation results, making it particularly useful for learning, experimentation, and functional verification of digital systems.

1.2.5 Model Builder

The **Model Builder** in eSim enables users to create, customize, and modify simulation models for semiconductor devices such as **diodes, BJTs, MOSFETs, JFETs, and IGBTs**. Accurate device modeling is essential for obtaining realistic simulation behavior and improving circuit analysis precision.

This feature allows designers to define custom electrical parameters and adapt simulation models according to specific design requirements, thereby enhancing overall simulation flexibility and accuracy.

1.2.6 Subcircuit Builder

The **Subcircuit Builder** is a useful feature in eSim that allows users to create **reusable hierarchical circuit modules**. Frequently used circuit components or functional blocks can be converted into subcircuits and reused across multiple projects.

This modular design approach improves circuit organization, reduces design complexity, minimizes repetitive work, and supports scalable hardware development. During this internship, multiple digital **IP blocks** were developed and integrated using the subcircuit framework in eSim.

1.2.7 PCB Design Tools

For Printed Circuit Board (PCB) implementation, eSim integrates **CvPcb** and **Pcbnew**. **CvPcb** is used to assign appropriate component footprints to schematic symbols, ensuring compatibility between circuit design and physical hardware implementation.

Pcbnew is the PCB layout editor used for board design, component placement, routing, and optimization of electrical connections. It enables designers to generate manufacturable PCB layouts after successful circuit verification.

Thus, the integration of these tools makes eSim a comprehensive and flexible **open-source Electronic Design Automation (EDA) platform** suitable for academic learning, research activities, digital system design, mixed-signal verification, and hardware prototyping.

Chapter 2

Features of eSim

eSim provides an integrated environment for electronic circuit design, simulation, verification, and PCB implementation. It combines multiple open-source tools to support analog, digital, and mixed-signal system design. The major features of eSim are listed below:

1. **Schematic Design:** eSim provides a schematic editor called Eeschema, which enables users to create and modify electronic circuit schematics efficiently. It supports component placement, wiring, annotation, and electrical rule checking (ERC).

2. **Circuit Simulation:** eSim integrates Ngspice for circuit simulation, allowing users to verify circuit functionality before hardware implementation. It supports DC analysis, AC analysis, transient analysis, and mixed-signal simulation.

3. **PCB Layout Design:** eSim supports PCB implementation through Pcbnew, enabling users to convert schematics into PCB layouts with routing and track optimization.

4. **Component Footprint Association:** The CvPcb tool is used to assign footprints to schematic components, ensuring compatibility between circuit schematics and PCB layouts.

5. **KiCad to Ngspice Conversion:** eSim provides a KiCad-to-Ngspice converter for converting schematic netlists into a format compatible with Ngspice for simulation.

6. **Device Model Builder:** The Model Builder enables users to create and modify simulation models for devices such as diodes, BJTs, MOSFETs, JFETs, and IGBTs.

7. **Subcircuit Builder:** The Subcircuit Builder allows users to create reusable circuit modules and hierarchical circuit designs, reducing design complexity.

8. **Mixed Signal Simulation Support:** eSim supports digital and mixed-signal verification through NgVeri and NGHDL, enabling simulation using Verilog, SystemVerilog, and VHDL.

9. **Component Libraries:** eSim includes a wide range of component libraries consisting of analog, digital, hybrid, power, source, and user-defined components for flexible circuit development.

10. **Open Source and Cross Platform Support:** eSim is an open-source EDA tool accessible to students, educators, researchers, and hardware designers, supporting multiple operating systems.

Chapter 3

Design Methodology

3.1 Problem Identification

The problem identification phase focused on understanding the functionality, importance, and implementation requirements of various **digital Intellectual Property (IP) blocks** used in modern digital systems. Since IP blocks are fundamental building units in **embedded systems, FPGA, ASIC, and System-on-Chip (SoC)** architectures, proper analysis was essential before implementation.

The following methodology was adopted:

- Studied the role and industrial relevance of selected **digital IP blocks**.
- Identified the functional requirements, input-output behavior, and expected system responses of each design.
- Analyzed timing requirements, synchronization constraints, arbitration logic, signal conditioning, and control mechanisms.
- Examined the feasibility of implementation and verification using the **eSim mixed-signal simulation environment**.
- Defined design objectives for functional implementation and validation.

3.2 RTL Design and Verilog Implementation

After identifying the design requirements, the selected IP blocks were modeled using **Verilog Hardware Description Language (HDL)** at the **Register Transfer Level (RTL)**. The focus was on developing synthesizable and reusable hardware architectures.

The following design flow was followed:

- Studied the operational principle and architecture of each IP block.

- Designed RTL logic using **Verilog HDL** for implementing control, timing, synchronization, and sequential operations.
- Developed modular and reusable architectures suitable for hardware implementation.
- Used **Makerchip-App** integrated with eSim for understanding digital logic behavior and interactive verification wherever applicable.
- Ensured logical correctness and design consistency before schematic-level implementation.

3.3 Schematic Design and Subcircuit Development in eSim

The designed IP blocks were implemented at the schematic level using the **eSim platform**. To improve modularity and design reuse, the implemented circuits were converted into reusable subcircuits.

The following procedure was adopted:

- Created schematic representations using the **eSim schematic editor**.
- Used the **Subcircuit Builder** to develop reusable **IP modules**.
- Integrated **ADC** and **DAC** blocks for enabling mixed-signal interaction between analog and digital domains.
- Configured signal interconnections, clock inputs, reset logic, and control signals according to functional requirements.
- Verified schematic structure and module connectivity before simulation.

3.4 Mixed-Signal Simulation

The designed IP blocks were validated through **mixed-signal simulation** using the integrated toolchain available in eSim. Simulation was performed to observe functional correctness, timing behavior, and signal interaction under different operating conditions.

The following simulation methodology was followed:

- Used **NgVeri** for simulation and functional verification of digital circuits written in **Verilog HDL**.

- Used **Ngspice** to perform mixed-signal simulation involving interaction between digital circuits and analog interfaces.
- Used **NGHDL** wherever VHDL compatibility and digital verification support were required within the mixed-signal environment.
- Applied different clock conditions, control signals, and test cases to evaluate circuit behavior.
- Observed waveform responses, signal transitions, synchronization behavior, and timing characteristics for functional analysis.

3.5 Verification and Analysis

The final verification stage ensured that the implemented IP blocks performed according to the expected theoretical behavior. The outputs obtained through simulation were analyzed to confirm functional correctness and reliability.

The following analysis procedure was followed:

- Compared simulation outputs with expected design functionality.
- Analyzed waveform responses generated using **NgVeri** and **Ngspice**.
- Verified timing relationships, synchronization mechanisms, control operations, and signal integrity.
- Identified and corrected possible functional mismatches through iterative testing.
- Validated the overall performance of the designed IP blocks for reliable mixed-signal system integration.

Chapter 4

IP Blocks

4.1 Two Flip-Flop Synchronizer

4.1.1 Introduction

In digital system design, transferring data across different clock domains can lead to timing violations and metastability. When an asynchronous signal changes state near the setup or hold time window of a destination flip-flop, the output can settle into an unstable state between logic '0' and '1'. This phenomenon is known as metastability.

A 2-ff syncronizer is a fundamental circuit used to mitigate metastability. By cascading two edgetriggered flip-flops in series within the destination clock domain, the circuit provides an extra clock cycle for any metastable state to resolve into a stable logic level before the signal is sampled by the downstream logic.

4.1.2 Importance of Two Flip-Flop Synchronizer

- **Metastability Mitigation:** Prevents unstable logic states from propagating into destination clock domain logic.
- **Clock Domain Crossing (CDC):** Ensures safe, reliable communication of single-bit control signals between asynchronous clock boundaries.
- **System Reliability:** Dramatically increases the Mean Time Between Failures (MTBF) of complex SoC and ASIC designs.
- **Glitch Filtering:** Helps isolate downstream synchronous logic from raw, noisy external inputs.

4.1.3 Schematic Implementation

The testbench and top-level schematic implementation for verifying the synchronizer IP are designed within an eSim mixed-signal simulation framework. The structural setup is divided into an external verification environment and an inner bridging subcircuit.

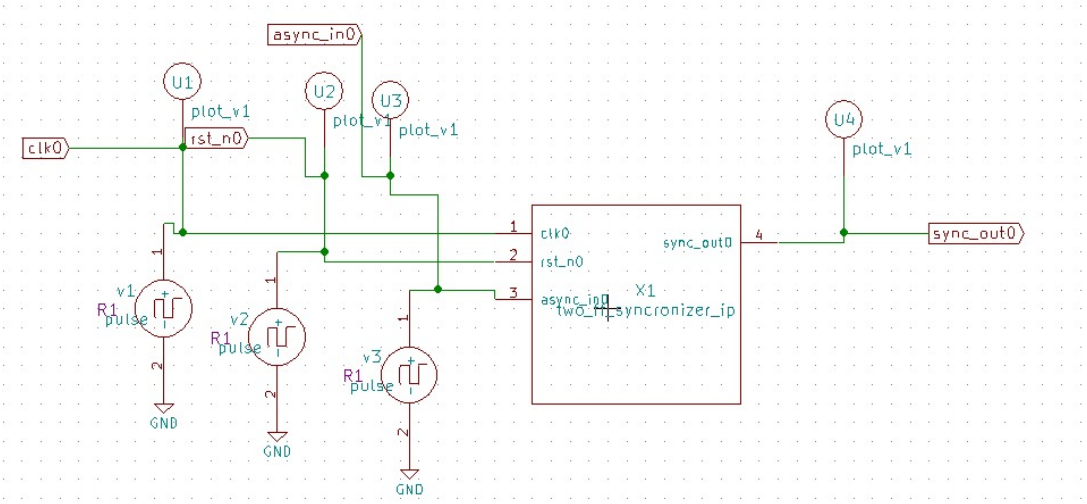


Figure 4.1: Top-level Testbench Circuit for the Synchronizer IP Block

As shown in Figure 4.1, the top-level testing schematic centers around the IP block component tagged as `X1` (`two_ff_synchronizer_ip`). The block features three primary inputs on the left and a synchronized output on the right:

- **Pin 1 (clk0):** Driven by a periodic pulse source generator (`v1`) representing the destination domain clock.
- **Pin 2 (rst_n0):** Driven by an independent pulse generator (`v2`) providing an active-low reset.
- **Pin 3 (async_in0):** Driven by pulse source `v3` to emulate an asynchronous signal changing completely out of phase with `clk0`.

Voltage plot nodes `U1`, `U2`, `U3`, and `U4` (`plot_v1`) are strategically placed on each signal trace to capture real-time waveforms during transient analysis.

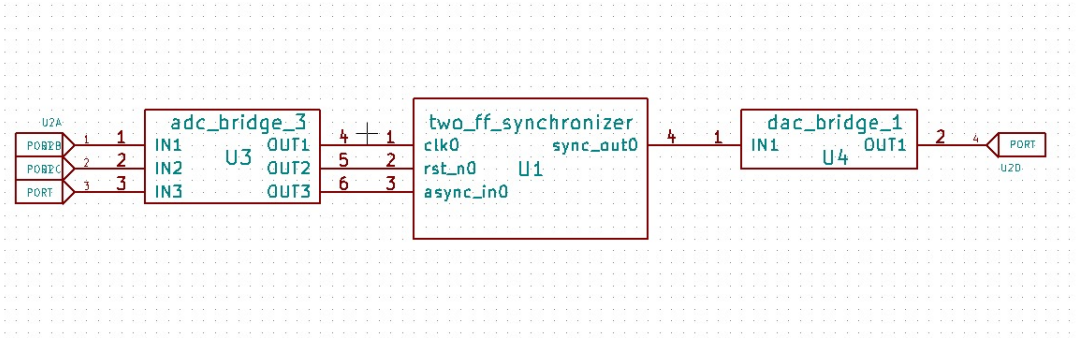


Figure 4.2: Internal Subcircuit Block Interface Configuration

Figure 4.2 details the mixed-signal bridge interface utilized by eSim to interface the digital synchronizer core with analog simulation elements. Since external sources provide analog voltage values, an Analog-to-Digital bridge component (`adc_bridge_3`) converts the three continuous-time signals into digital vectors (`IN1` to `IN3`). These vectors feed directly into the digital core instance `U1` (`two_ff_synchronizer`). The output is then processed by a Digital-to-Analog bridge (`dac_bridge_1`) to reconstruct a clean analog voltage wave out of port `U2D` (`sync_out0`).

4.1.4 RTL Design and Verilog Implementation

The digital core inside the subcircuit relies on two sequentially connected D-flip-flops sharing a single clock and reset line. The Verilog hardware description code is structured as follows:

```
1  `timescale 1ns / 1ps
2
3  module two_ff_synchronizer (
4      input  wire clk0,
5      input  wire rst_n0,
6      input  wire async_in0,
7
8      output wire sync_out0
9  );
10
11     reg q1;
12     reg q2;
13
14     always @(posedge clk0 or negedge rst_n0) begin
15         if (!rst_n0) begin
16             q1 <= 1'b0;
17             q2 <= 1'b0;
18         end else begin
```

```
19              q1 <= async_in0; // First staging flip-flop
20              q2 <= q1;        // Second stabilizing flip-flop
21          end
22      end
23
24      // Output assigned from the second stable stage
25      assign sync_out0 = q2;
26
27  endmodule
```

Listing 4.1: Two Flip-Flop Synchronizer Verilog Code

4.1.5 Functional Description

The architecture acts as a two-stage shift register running on the destination clock. It functions via distinct lifecycle phases:

- **Reset Phase:** When `rst_n0` drops to logic '0', both internal registers (`q1`, `q2`) instantly clear to '0'. The output `sync_out0` remains anchored low, disregarding incoming state changes.
- **First Catching Stage ($q1$):** On the rising edge of `clk0`, the asynchronous input signal is sampled. If a transition happens too close to this edge, register $q1$ absorbs the initial timing stress and potential metastability.
- **Second Stabilizing Stage ($q2$):** Exactly one clock cycle later, the next rising edge samples the stabilized output of $q1$ into $q2$. This isolated step ensures that `sync_out0` switches cleanly, aligned with the destination clock grid.

4.1.6 Simulation Results and Verification

The transient response of the synchronized system was verified via eSim, utilizing voltage offsets to display individual waveforms clearly on a single axis.

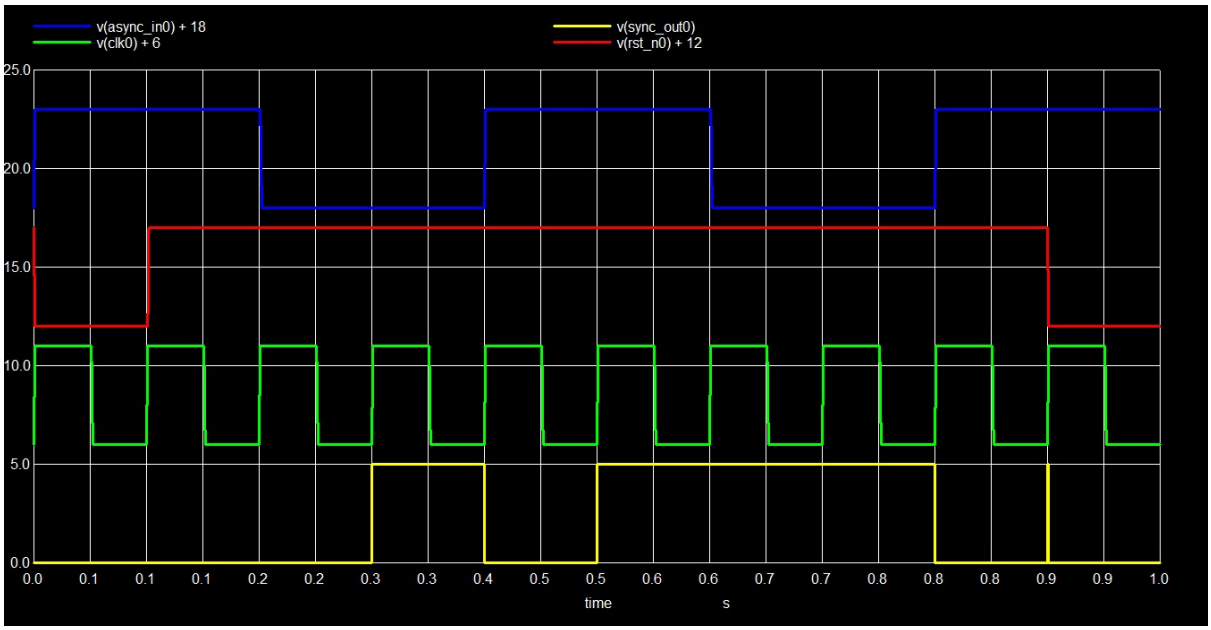


Figure 4.3: Transient Simulation Waveform Analysis

The behavior shown in the simulation plot (Figure 4.3) is analyzed below:

1. **Initialization Phase ($0.0s$ to $0.1s$):** The active-low reset signal `v(rst_n0)+12` starts low. Despite early toggling on the asynchronous input line `v(async_in0)+18`, the final output line `v(sync_out0)` stays clamped firmly at $0.0V$.

2. **Reset Release ($0.1s$ onwards):** The reset line steps up, enabling the internal registers to respond to clock inputs.

3. **Synchronized Latency Processing:**

   - At time $t = 0.1s$, the async signal is high. At the first available rising clock edge ($t = 0.1s$), the value is captured by the first flip-flop stage.
   - At the second rising edge ($t = 0.3s$), the value propagates through the second stage, causing the yellow output signal `v(sync_out0)` to cleanly rise to $5.0V$.
   - When `async_in0` drops at $t = 0.2s$, this low state updates through the registers, pulling the yellow output signal back to low at the subsequent clock edge ($t = 0.4s$).

The simulation confirms that output transitions are cleanly aligned with the rising edge of the destination clock, verifying the functionality of the synchronizer.

4.1.7 Applications

Two Flip-Flop Synchronizers are key components across various modern mixed-signal and multi-clock architectures, including:

- **I/O Peripherals:** Synchronizing external button presses, switch signals, or interrupt request pins.
- **Serial Interfaces:** Sampling incoming asynchronous data packets (e.g., UART RX lines).
- **Cross-Domain Control Handshakes:** Passing single-bit status or enable tokens between independent functional blocks in an SoC.
- **FIFO Status Management:** Assisting in pointer synchronization across asynchronous FIFO block interfaces.

4.1.8 Summary

A Two Flip-Flop Synchronizer IP block was successfully modeled, instantiated, and analyzed. By cascading edge-triggered storage elements, the circuit isolates downstream logic from metastability risks inherent to cross-domain data movement. The transient simulation plots validated the design's operation, showing clean, synchronized outputs with expected clock-cycle latencies. This confirms its reliability for integration into larger digital and mixed-signal systems.

4.2 Built-In Self-Test (BIST) Controller

4.2.1 Introduction

As digital integrated circuits scale in density and structural complexity, ensuring high manufacturing test coverage and field reliability becomes increasingly challenging. Traditional external testing using Automated Test Equipment (ATE) requires substantial testing time, high pin-count overhead, and expensive instrumentation. To address these limitations, **Built-In Self-Test (BIST)** techniques are embedded directly into modern System-on-Chip (SoC) and ASIC architectures.

A BIST architecture incorporates dedicated hardware blocks to automate test generation and verification internal to the microchip. The core architecture includes a Test Pattern Generator (TPG)—typically realized via a Linear Feedback Shift Register (LFSR)—which applies pseudo-random vectors to a Circuit Under Test (CUT). The resulting response vectors from the CUT are collected and compressed by a Output Response Analyzer (ORA), which often utilizes a Multiple-Input Signature Register (MISR). A centralized **BIST Controller** coordinates the entire testing lifecycle, asserting control lines, tracking execution cycles, and comparing compressed digital signatures against a pre-computed gold standard to report functional integrity.

4.2.2 Importance of Built-In Self-Test Controllers

- **Elimination of External Test Machinery:** Reduces dependency on high-cost physical ATE infrastructure by hosting test routines natively.
- **At-Speed Testing:** Enables verification of internal combinational and sequential structures at the maximum operational clock frequency.
- **Nondestructive Field Diagnostics:** Allows the digital system to trigger standalone self-diagnostic checking routines during boot-up or idle execution periods.
- **High Fault Coverage Automation:** Facilitates rapid sensitization of classical structural physical defects, such as line stuck-at faults ($SA0$ / $SA1$).

4.2.3 Circuit Under Test (CUT) Topology and Fault Modeling

The logic block designated for structural evaluation is a multi-output combinational logic network processing a 4-bit parallel primary input vector ($b = [b_3, b_2, b_1, b_0]$) to resolve a 4-bit parallel output vector ($y = [y_3, y_2, y_1, y_0]$).

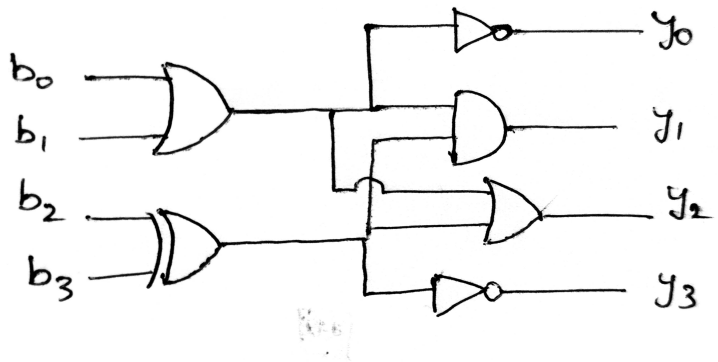
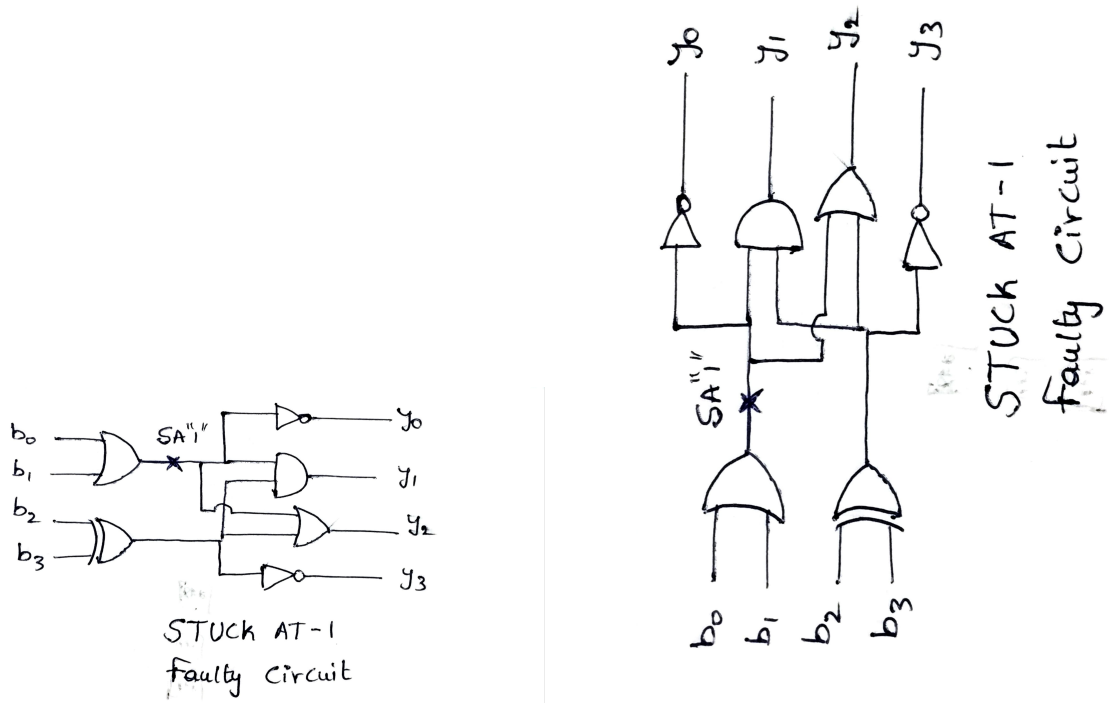


Figure 4.4: Gate-Level Schematic of the Fault-Free Circuit Under Test (CUT)

As mapped structurally in Figure 4.4, the flawless combinational logic consists of an input processing stage feeding an output stage:

- Inputs b_0 and b_1 pass into a two-input `OR` gate.
- Inputs b_2 and b_3 pass into a two-input `XOR` gate.
- The output of the upper `OR` gate breaks into three routing nets: it passes through an inverter to yield output y_0, feeds the top pin of a middle two-input `AND` gate, and attaches to the upper input terminal of a two-input `OR` gate assigned to output y_2.
- The output of the lower `XOR` gate breaks into three routing nets: it branches directly through a lower inverter to produce output y_3, couples into the bottom pin of the middle `AND` gate to establish output y_1, and wires into the lower terminal of the y_2 `OR` gate.



(a) Stuck-At-1 Physical Fault Annotation

(b) Rotated View of Stuck-At-1 Location

Figure 4.5: Modeling of a Structural Stuck-At-1 ($SA1$) Defect at the Upper Gate Output

To validate the verification capacity of the BIST architecture, a manufacturing defect is modeled within the physical layout as highlighted by the fault markers in Figure 4.5. The defect forms a **Stuck-At-1 ($SA1$)** fault directly at the output net of the primary upper `OR` gate (fed by b_0 and b_1). Regardless of the input states driven onto b_0 or b_1, this internal node remains fixed at a logical high value. This invalid physical condition directly impacts downstream nodes, corrupting the logical behavior of outputs y_0, y_1, and y_2 whenever the inputs should normally yield a low state from that initial gate.

4.2.4 Top-Level eSim Schematic Implementation

The full verification framework is modeled within an eSim mixed-signal simulation platform. Two separate top-level verification hardware layouts were developed to compare operations under ideal conditions versus compromised circuit behavior.

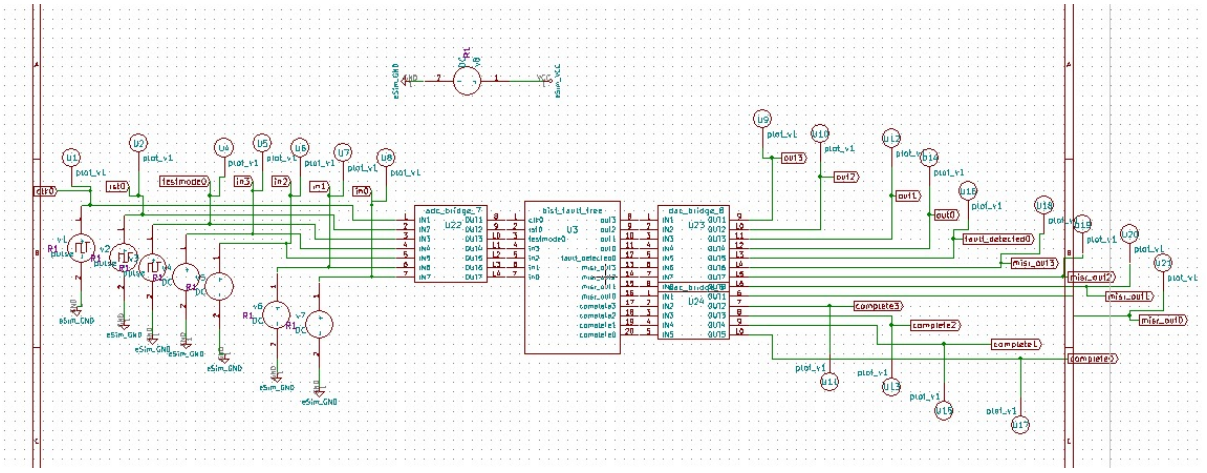


Figure 4.6: Mixed-Signal eSim Simulation Testbench Layout for the Fault-Free BIST Core

Figure 4.6 illustrates the baseline evaluation hardware layout containing the fault-free module block tagged as `U3` (`bist_fault_free`). The peripheral elements establish the structural stimulus and visualization framework:

- **Input Signaling Block:** Periodic voltage pulse generators `v1` and `v2` drive the system clock (`clk0`) and the global master reset line (`rst0`). Independent voltage sources (`v3`, `v4`, `v5`, `v6`, `v7`) set static conditions for inputs `testmode0`, `in3`, `in2`, `in1`, and `in0` respectively.

- **ADC Interface Block:** The raw analog source signals are processed through a 7-bit wide Analog-to-Digital bridge (`adc_bridge_7`, marked as `U22`). This converts continuous physical voltages into discrete digital values (`ID1`–`ID7`) to safely drive the digital hardware simulation core.

- **DAC Signal Reconstruction Probing Block:** Digital outputs emerging from the engine block route through two multi-bit Digital-to-Analog bridge networks (`dac_bridge_8` instances labeled `U23` and `U24`). These recreate analog waveforms for analysis. Continuous voltage probing nodes (`U1` to `U21`) tap into key traces to record real-time simulation output profiles.

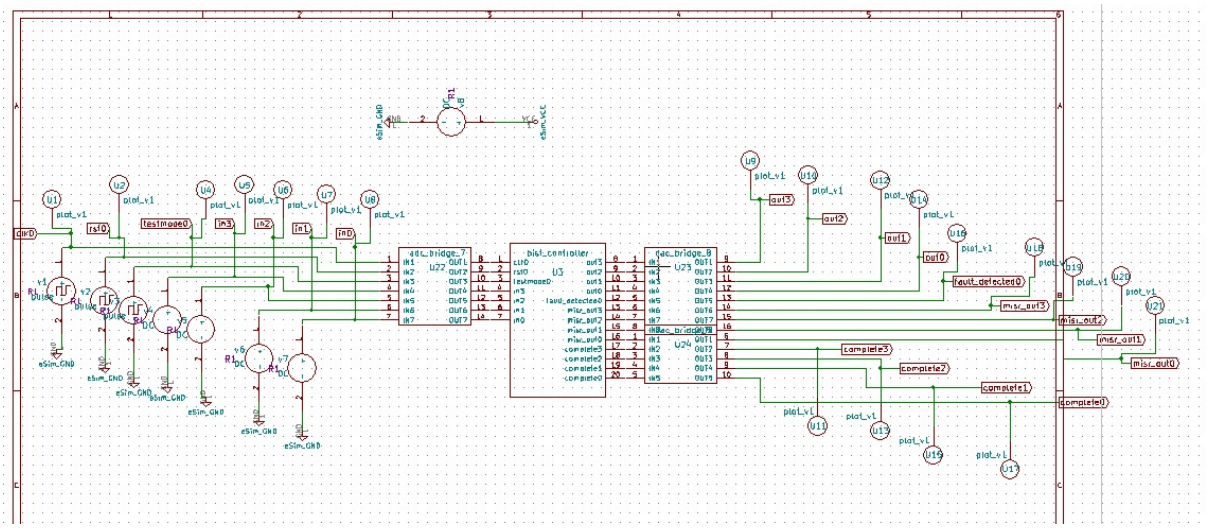


Figure 4.7: Mixed-Signal eSim Simulation Testbench Layout for the Faulty ($SA1$) BIST Core

The hardware configuration shown in Figure 4.7 replicates the peripheral layout of the baseline environment but replaces the core instance with device tag `U3` named `bist_controller`. This component contains the modified, faulty CUT structural netlist injected with the physical $SA1$ defect shown in Figure 4.5. This variant allows the simulation engine to log structural response deviations under identical input trace stimuli.

4.2.5 Simulation Results and Waveform Analysis

The timing responses and behavioral trace graphs captured across the operational configurations are detailed below. Transient simulations use voltage offsets to trace multiple signals clearly on a shared timeline.

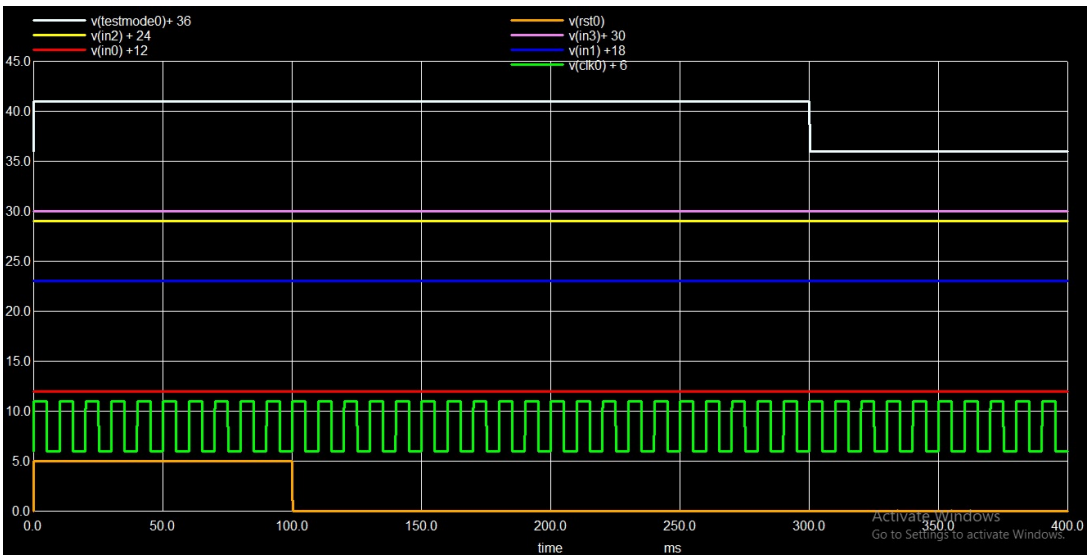


Figure 4.8: Transient Simulation Waveforms of Primary Test Control and System Inputs

Figure 4.8 displays the primary input profiles applied to both test configurations over a 400 ms time window:

- The global active-low reset trace `v(rst0)` establishes initialization by holding at a logical low level from 0.0 ms to 100.0 ms before stepping cleanly to a high level.
- The continuous master clock input `v(clk0)+6` toggles uniformly with a 10 ms periodic repetition interval across the full simulation run.
- Static parallel input lines `v(in0)+12`, `v(in1)+18`, `v(in2)+24`, and `v(in3)+30` deliver steady logical levels to anchor baseline input states.
- The test mode selection signal `v(testmode0)+36` asserts high at 40.0 V (representing a logical high state) until 300.0 ms, at which point it drops to 36.0 V (representing a logical low state) to shift the controller out of verification execution.

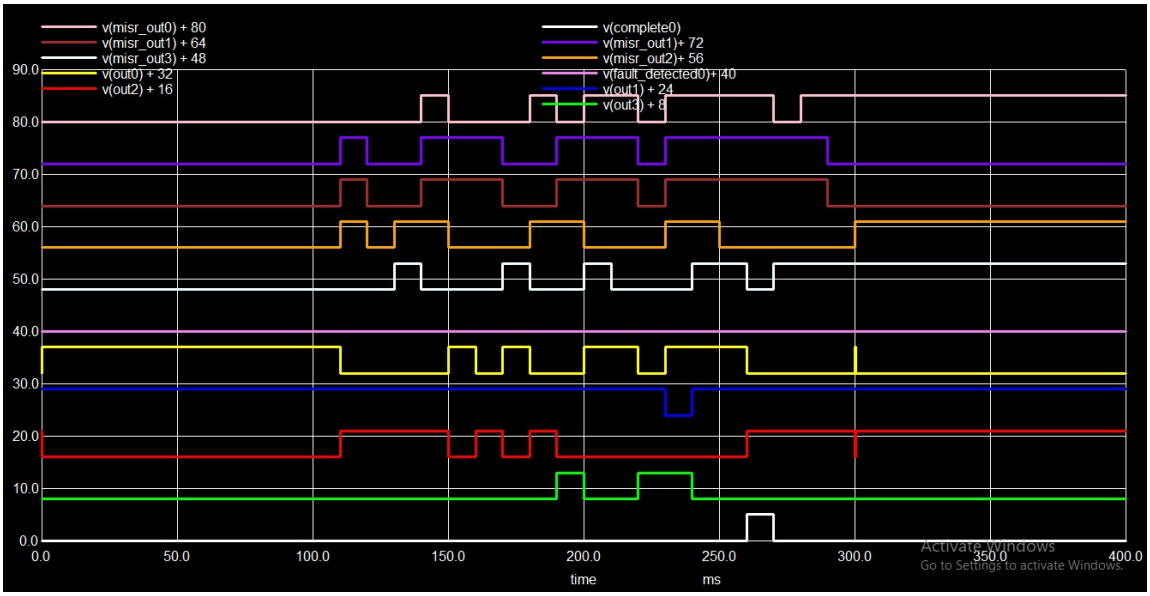


Figure 4.9: Output Signature Signatures for the Ideal, Fault-Free Simulation Run

Figure 4.9 displays the output signals under ideal execution. When `rst0` releases at $100.0\,ms$, the internal pattern generation and logic evaluation cycle starts. The response waveforms `v(out0)+32`, `v(out1)+24`, and `v(out2)+16` toggle through valid combinations, driving the compression registers. This causes the signature outputs `v(misr_out0)+80`, `v(misr_out1)+64`, `v(misr_out2)+56`, and `v(misr_out3)+48` to step through a predictable, fault-free digital code sequence.

At approximately $t = 265\,ms$, the testing routine completes, triggering a brief logic-high notification pulse on `v(complete0)`. Crucially, the diagnostic error flag line `v(fault_detected0)+4` remains flat across the entire evaluation timeframe. This confirms that the internal signatures match the expected reference codes.

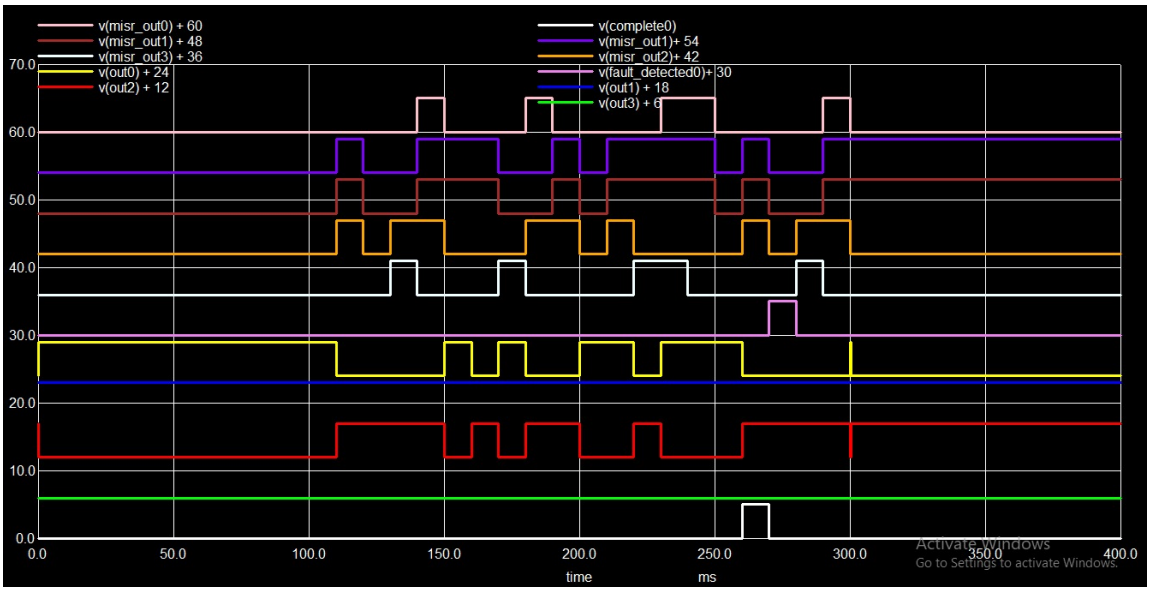


Figure 4.10: Output Signature Profiles for the Faulty ($SA1$) Simulation Run

Figure 4.10 captures the compromised execution path where the $SA1$ fault is active. While early operational phases mirror the baseline run, the permanent logical high condition at the internal node distorts downstream signals, altering the output waveforms (`v(out0)+24` and `v(out2)+12`) relative to their ideal states.

Consequently, the MISR compression registers ingest corrupted functional states, shifting the signature outputs away from the nominal progression. This signature divergence is caught by the internal comparison logic. At $t = 265\,ms$, when the routine concludes, the mismatch causes the controller to assert the fault flag line, creating a clean, distinct output state pulse on `v(fault_detected0)+30`. This validates the block's ability to isolate internal layout deviations autonomously.

4.2.6 Applications

Centralized BIST Controller engines are key components in high-reliability digital architectures, including:

- **Safety-Critical Automotive Electronics:** Executing power-on self-test (POST) sequences in ISO 26262 compliant ECU environments.
- **Aerospace and Defense Computing:** Performing structural runtime health checks in harsh radiation environments prone to single-event upsets.
- **High-Performance Server Microprocessors:** Supporting structural test compression and field-repair diagnostic testing.
- **Deeply Embedded IoT SoC Infrastructure:** Providing remote hardware health validation capabilities.

4.2.7 Summary

A Built-In Self-Test Controller architecture was successfully verified using eSim mixed-signal simulations. Structural verification evaluated a multi-output combinational circuit under both ideal conditions and when compromised by an internal Stuck-At-1 ($SA1$) defect. Transient simulation results confirmed proper operation: the fault-free core completed testing without warnings, while the faulty core accurately detected signature deviations and asserted its diagnostic error flag. This demonstrates the design's effectiveness for embedded automated testing applications.

4.3 Clock Frequency Divider

4.3.1 Introduction

In synchronous digital design, a single primary high-frequency oscillator typically serves as the master clock source for the entire system. However, different functional sub-modules within a System-on-Chip (SoC) or Application-Specific Integrated Circuit (ASIC) often operate efficiently at different speeds. For example, peripheral interfaces like I^2C, SPI, or UART, as well as low-power timers, require much lower clock frequencies than the core processor.

A **Clock Frequency Divider** is a vital sequential logic block that down-scales a high-frequency incoming clock signal to generate a lower-frequency synchronous clock output. By dividing the clock frequency, the system can selectively run slower sub-systems on reduced-speed clock networks. This directly lowers the dynamic power consumption given by the relationship $P_{dynamic} = \alpha C V_{DD}^2 f$, where f is the switching frequency.

4.3.2 Importance of Clock Frequency Divider

- **Dynamic Power Reduction:** Lowering the clock frequency (f) for peripheral blocks significantly reduces switching losses.
- **Multi-Domain Operations:** Facilitates the generation of diverse synchronous clock trees from a single master clock source.
- **Edge Alignment Consistency:** Ensures predictable phase relationships between the divided output and the reference clock when implemented with synchronous counters.
- **Hardware Efficiency:** Replaces the need for multiple independent external crystal oscillators, saving physical board area and reducing cost.

4.3.3 Schematic Implementation

The simulation framework for the Clock Frequency Divider is implemented using the eSim mixed-signal simulation toolkit. The design is structured into a top-level verification testbench and an internal mixed-signal subcircuit interface.

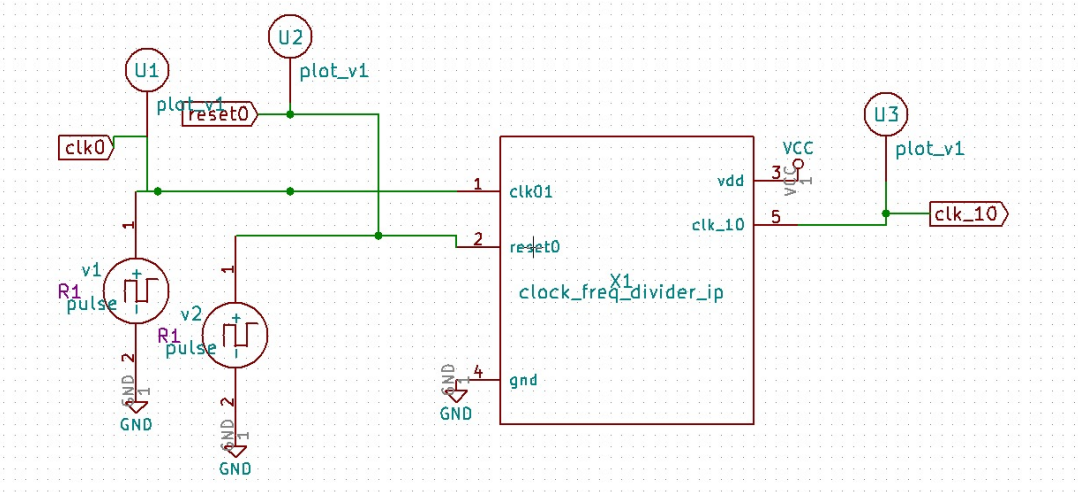


Figure 4.11: Top-Level Testbench Schematic for the Clock Divider IP Block

As shown in Figure 4.11, the verification setup centers around the IP component block labeled `X1` (`clock_freq_divider_ip`). This block includes two primary input networks on its left and a divided clock output port on its right:

- **Pin 1 (clk01):** Driven by a periodic analog pulse source (`v1`) that emulates the fast system master clock input (`clk0`).
- **Pin 2 (reset0):** Connected to an independent pulse source (`v2`) providing a dedicated active-low or active-high clear signal (`reset0`) to initialize the internal counters.
- **Pins 3 and 4 (vdd / gnd):** Wired to the simulation power rails (`VCC` and `GND`) to establish a steady DC operating reference voltage.
- **Pin 5 (clk_10):** Outputs the down-scaled divided clock waveform (`clk_10`).

Voltage probing nodes `U1`, `U2`, and `U3` (`plot_v1`) are connected to the key traces to capture transient behaviors during simulation.

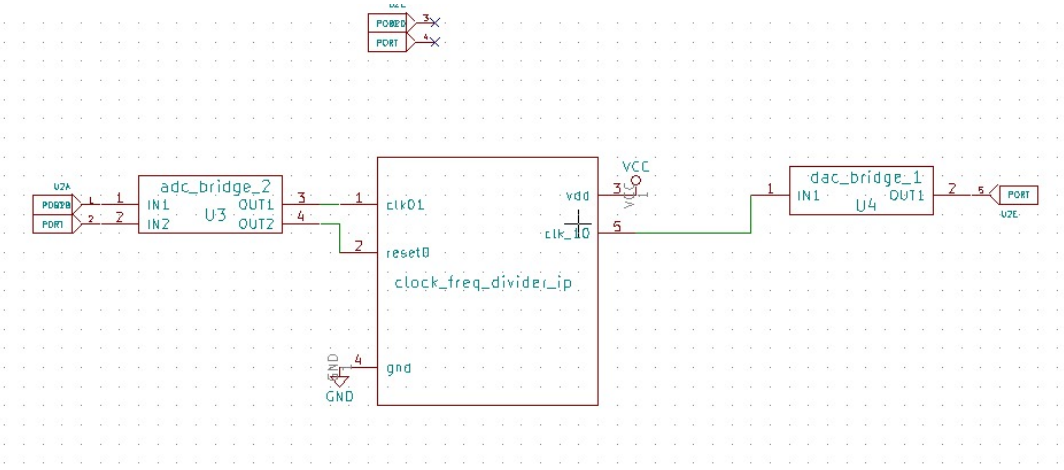


Figure 4.12: Internal Subcircuit Block Interface Configuration

Figure 4.12 details the mixed-signal bridge layout within the subcircuit block. Since the input signals from the testbench are continuous analog waveforms, they pass through a 2-bit wide Analog-to-Digital converter bridge component (`adc_bridge_2`, labeled `U3`). This block translates the continuous signals into discrete digital lines to drive the digital core instance `U1` (`clock_freq_divider_ip`).

Once the digital core processes the signals and reduces the frequency, the resulting digital output line passes through a Digital-to-Analog bridge (`dac_bridge_1`, labeled `U4`). This block reconstructs a clean analog voltage wave at terminal port `U2D` to produce the final measurable output.

4.3.4 RTL Design and Verilog Implementation

The digital core is implemented as a synchronous parameterizable counter. On every active edge of the master clock, the internal register increments. It toggles the output clock state when it reaches the division threshold. The synthesizable Verilog implementation is structured as follows:

```
1  `timescale 1ns / 1ps
2
3  module clock_freq_divider_ip (
4      input  wire clk01,
5      input  wire reset0,
6
7      output reg  clk_10
8  );
9
10     // Internal counter register to track clock divisions
11     reg [3:0] count;
12
```

```
13      always @(posedge clk01 or negedge reset0) begin
14          if (!reset0) begin
15              count  <= 4'b0000;
16              clk_10 <= 1'b0;
17          end else begin
18              // Example divider logic: Divide-by-10
                    configuration
19              if (count == 4'd4) begin
20                  count  <= 4'b0000;
21                  clk_10 <= ~clk_10; // Toggle output clock
                        state
22              end else begin
23                  count  <= count + 1'b1;
24              end
25          end
26      end
27
28  endmodule
```

Listing 4.2: Clock Frequency Divider Verilog Code

4.3.5 Functional Description

The Clock Frequency Divider operates across distinct phases dictated by its control inputs:

- **Reset Initialization Phase:** When the `reset0` signal is asserted, all internal count registers instantly clear. The output clock line `clk_10` is forced to a low state, ignoring any toggling on the input reference clock.
- **Synchronous Counting Phase:** Once the reset signal is released, the internal tracking register increments on each consecutive rising edge of the master clock input (`clk01`).
- **Output State Toggling:** When the internal register matches the targeted division threshold, the controller clears the counter back to zero and flips the state of the `clk_10` output. This sequence creates a balanced, down-scaled output clock with a 50% duty cycle.

4.3.6 Simulation Results and Verification

The transient response of the divider design was verified in eSim over a 1.0 s time window. Voltage offsets are applied to separate and display each waveform clearly on a single axis.

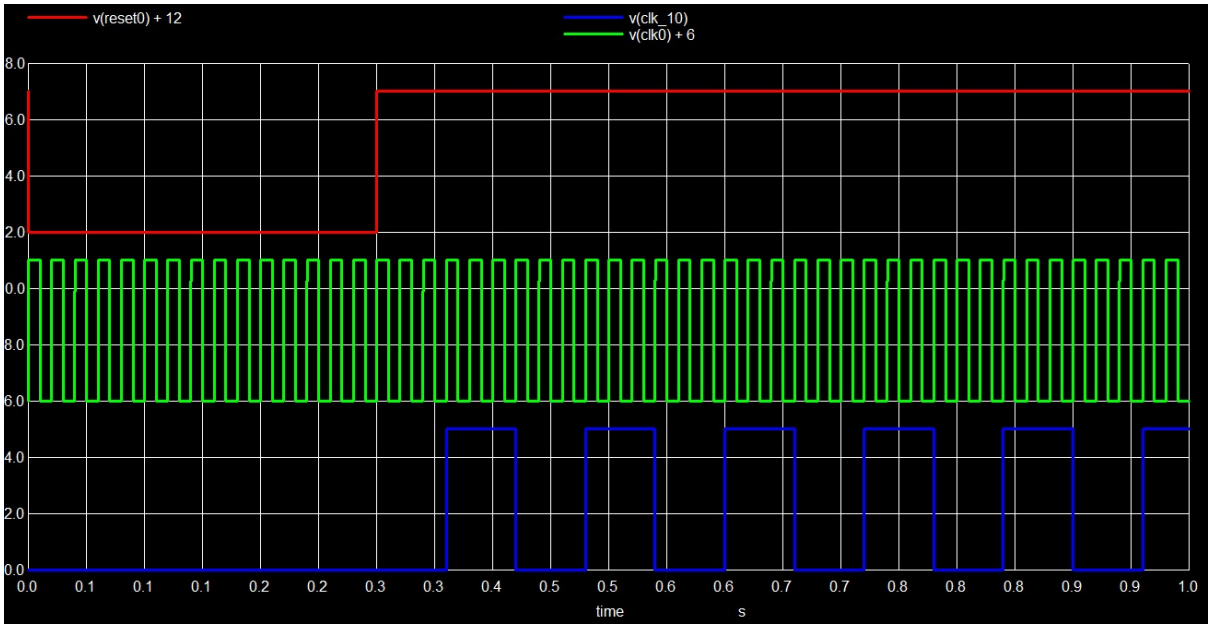


Figure 4.13: Transient Simulation Waveforms of the Clock Frequency Divider

The behavior shown in the simulation plot (Figure 4.13) is analyzed below:

1. **Reset Phase (0.0 s to 0.3 s):** The reset trace `v(reset0)+12` starts at a low potential (2.0 V relative to its baseline offset). During this window, the master reference

clock input `v(clk0)+6` toggles continuously at high speed. However, because the reset is active, the divided output clock signal `v(clk_10)` remains clamped flat at $0.0\,V$.

2. **Reset Release ($t = 0.3\,s$):** The control signal `v(reset0)+12` steps up cleanly to its high logic level (rising from $2.0\,V$ to approximately $7.0\,V$). This releases the internal counting registers and allows them to track the master clock.

3. **Frequency Division Phase ($0.3\,s$ to $1.0\,s$):**

   - Immediately after the reset releases, the internal counter tracks the rising edges of the green master clock `v(clk0)+6`.
   - At $t = 0.36\,s$, the counter hits its target threshold, causing the blue divided output trace `v(clk_10)` to rise to $5.0\,V$.
   - The output signal stays high for exactly 5 full input clock cycles before dropping back to low at $t = 0.46\,s$.
   - This symmetric pattern repeats across the remaining simulation window, producing an output clock with a reduced frequency and a 50% duty cycle. This confirms successful clock division.

4.3.7 Applications

Synchronous Clock Frequency Dividers are used across various modern digital processing architectures, including:

- **Baud Rate Generators:** Dynamically scaling high-speed system clocks to match standard serial data communication speeds (e.g., UART, SPI).
- **Prescalers for Hardware Timers:** Down-scaling master clocks to allow internal timer modules to measure longer time intervals.
- **Dynamic Voltage and Frequency Scaling (DVFS):** Scaling clock speeds downward during low-workload states to save system power.
- **Real-Time Clock (RTC) Modules:** Dividing high-frequency crystal reference signals down to a stable $1\,Hz$ clock for timekeeping tracking.

4.3.8 Summary

A Clock Frequency Divider IP block was successfully designed, modeled, and verified using eSim mixed-signal simulation tools. The layout uses synchronous counter structures to down-scale high-frequency master clock inputs into balanced, lower-speed clock

networks. Transient simulation results confirmed proper operation: the output remains stable during reset and produces a clean, low-frequency divided clock once reset is released. This confirms the block's readiness for power-optimized integration into larger SoC designs.

4.4 Glitch Filter

4.4.1 Introduction

In high-speed digital systems and real-world mixed-signal environments, input signals often carry unwanted transient noise, ringing, and narrow spurious pulses known as glitches. These glitches can be caused by electromagnetic interference (EMI), crosstalk between adjacent PCB traces, power supply fluctuations, or mechanical switch bouncing. If allowed to propagate unfiltered into the synchronous core of a digital design, these high-frequency artifacts can cause false triggering, corrupt state machine transitions, and cause catastrophic system instability.

To address this issue, a digital **Glitch Filter** is implemented as a front-end conditioning circuit. Instead of passing raw input signals directly to internal logic, the Glitch Filter continuously samples the incoming signal over a defined window of clock cycles. It updates its output only when the input signal remains stable for a predetermined duration. This architectural gating ensures that transient noise spikes narrower than the filter's threshold are completely suppressed, providing clean, stabilized signals to the downstream synchronous logic.

4.4.2 Importance of Glitch Filter

- **Spurious Pulse Suppression:** Efficiently eliminates high-frequency noise and transient spikes from external physical lines.
- **System Stability:** Prevents downstream finite state machines (FSMs) and edge-triggered registers from false-triggering on noise.
- **Front-End Signal Conditioning:** Functions as a reliable digital hardware debouncer and cleaner for noisy asynchronous inputs.
- **Adjustable Noise Immunity:** Allows designers to tune the filter's temporal depth based on the expected maximum width of environmental noise spikes.

4.4.3 Top-Level Testbench and Subcircuit Architecture

The structural validation framework for the Glitch Filter IP was designed and simulated within the eSim mixed-signal automation environment. The design is split into an external stimulus testbench environment and an internal mixed-signal translation block to isolate the digital filtering core.

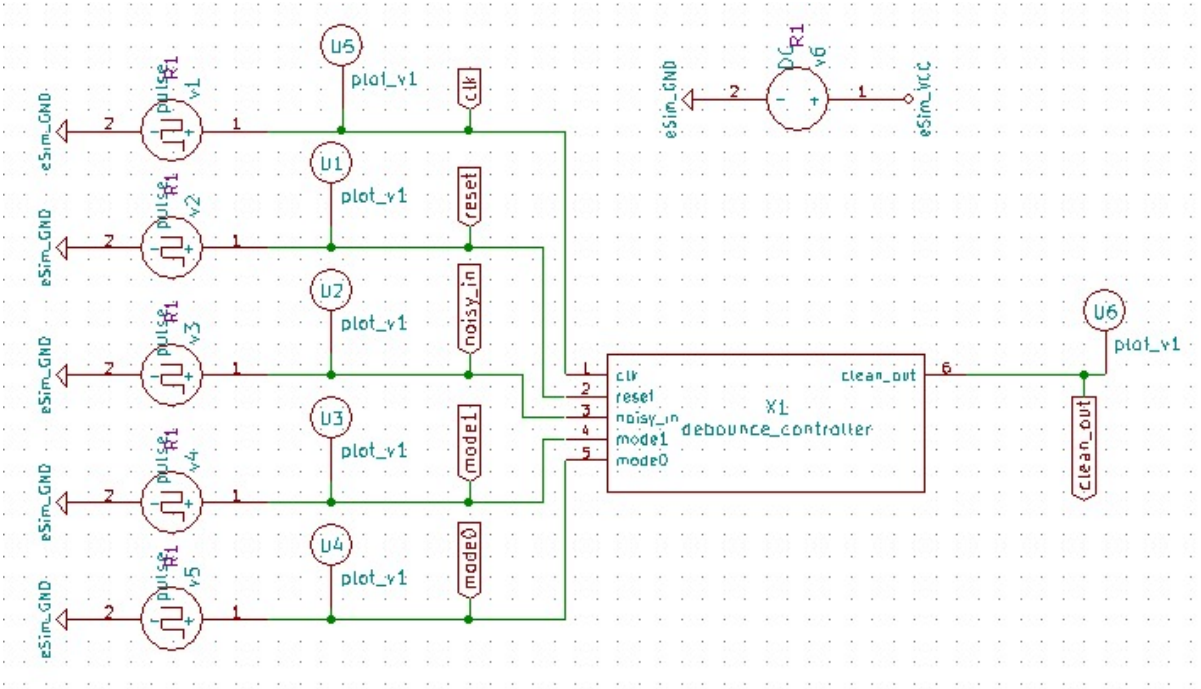


Figure 4.14: Top-Level Mixed-Signal Testbench Schematic for the Glitch Filter IP Block

The top-level verification circuit is shown in Figure 4.14. The design centers around the integrated circuit block instance tagged as `X1`, which contains the specialized `glitch_filter_ip`. The structural wiring paths are broken down as follows:

- **Input Driving Sources:** A high-frequency pulse generator source `v1` provides a continuous system reference clock network labeled `clk0`. An independent pulse source generator `v2` drives the master reset line `rst_n0`, which initializes the filter registers. A third configurable pulse source `v3` generates a noisy asynchronous signal named `async_in0`, which contains narrow transient pulses to test the filter's noise suppression capabilities.
- **Real-Time Voltage Monitoring:** Continuous voltage plot nodes labeled `U1`, `U2`, `U3`, and `U4` (`plot_v1`) are attached to the input and output traces. These nodes capture real-time voltage variations for transient waveform analysis.
- **Power Rail Connections:** The module includes dedicated terminals wired directly to the system's global DC voltage sources (`VCC` and `GND`) to establish a reliable operational voltage baseline.

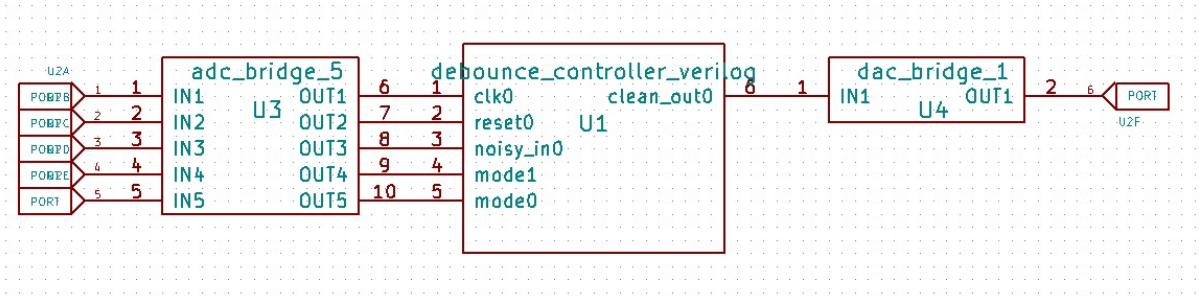


Figure 4.15: Internal Subcircuit Signal Bridge Architecture

Figure 4.15 illustrates the mixed-signal bridge interface inside the subcircuit block, which links the analog simulation environment with the digital core. Because the external signal sources generate analog voltage waveforms, they cannot interface directly with digital logic primitives.

To solve this, the incoming lines pass through a 3-bit wide Analog-to-Digital bridge component (`adc_bridge_3`, labeled `U3`). This block translates the continuous-time voltages into a discrete 3-bit digital vector (`IN1`, `IN2`, and `IN3`). This vector drives the internal terminals (`clk0`, `rst_n0`, and `async_in0`) of the core digital component `U1` (`glitch_filter`). After filtering out the transient glitches, the clean digital output is routed through a single-bit Digital-to-Analog bridge (`dac_bridge_1`, labeled `U4`). This block converts the digital signal back into an analog voltage trace at port terminal `U2D`, producing the clean output signal `sync_out0`.

4.4.4 RTL Design and Verilog Implementation

The digital core inside the subcircuit uses a synchronous shift register or counter-based tracking pipeline to sample the input line. The synthesizable Verilog module code is implemented using a multi-stage validation register structure:

```
`timescale 1ns / 1ps

module glitch_filter (
    input  wire clk0,
    input  wire rst_n0,
    input  wire async_in0,

    output reg  sync_out0
);

    // Multi-stage sampling shift register to track history
    reg [2:0] filter_reg;

    always @(posedge clk0 or negedge rst_n0) begin
        if (!rst_n0) begin
            filter_reg <= 3'b000;
            sync_out0  <= 1'b0;
        end else begin
            // Shift in the raw asynchronous input value
            filter_reg <= {filter_reg[1:0], async_in0};

            // Majority voting or consecutive sample
               validation check
            if (filter_reg == 3'b111) begin
                sync_out0 <= 1'b1;
            end else if (filter_reg == 3'b000) begin
                sync_out0 <= 1'b0;
            end
            // Keep the previous stable state if samples are
               mixed (glitch)
        end
    end

endmodule
```

Listing 4.3: Glitch Filter Synchronous Verilog Code

4.4.5 Functional Description

The Glitch Filter core processes input signals through clear functional phases to isolate stable signals from transient noise:

- **System Initialization:** Driving the master active-low line `rst_n0` to logic '0' resets the internal sampling shift registers and clears the output pin `sync_out0` to '0'.
- **Continuous Sampling Pipeline:** On every rising edge of the system clock (`clk0`), the current logical state of the asynchronous input line (`async_in0`) is sampled and shifted into the tracking history registers.
- **Glitch Suppression Gating:** If the raw input signal transitions briefly but switches back before filling the entire register pipeline, the internal logic flags the transition as an illegal glitch. The filter suppresses this pulse, keeping the output clock trace flat and completely stable.
- **Output State Validation:** The output register updates only when the input signal remains stable at a new logical state for a consecutive number of clock samples. This mechanism ensures clean, stable transitions.

4.4.6 Simulation Results and Multi-Platform Verification

The functionality of the Glitch Filter was evaluated and verified across multiple EDA tool environments, including eSim transient analysis simulations and digital Xilinx timing checks.

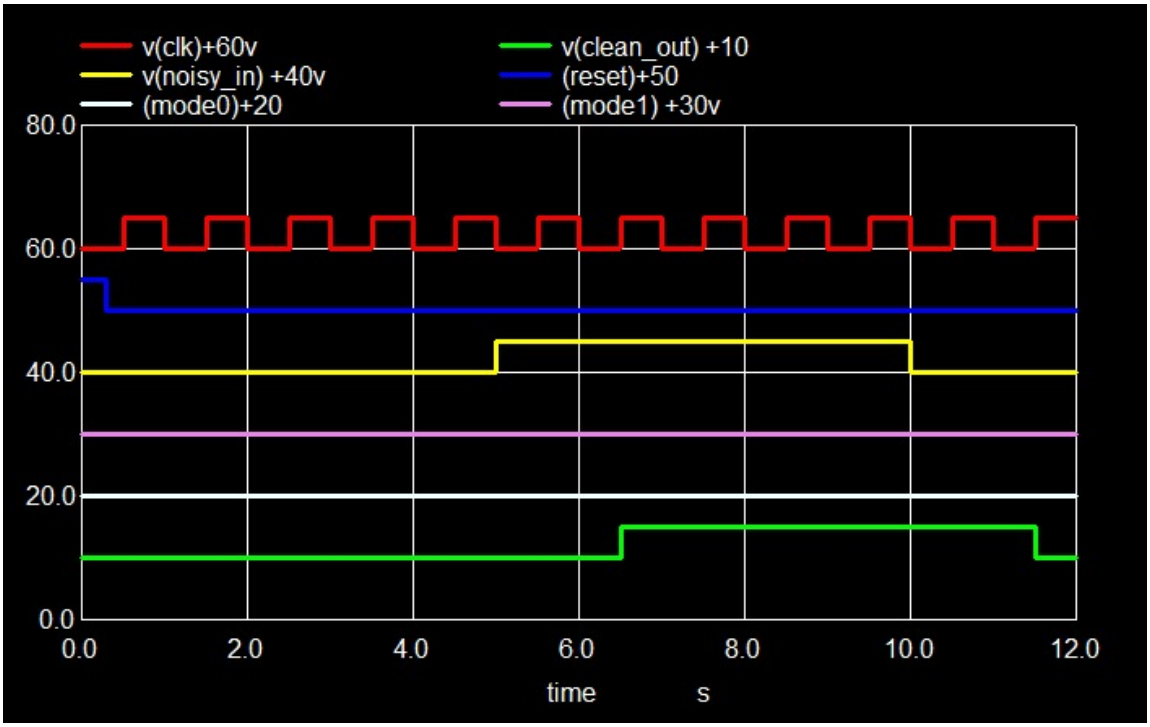


Figure 4.16: Transient Simulation Waveforms from the eSim Mixed-Signal Platform

Figure 4.16 shows the transient simulation results generated within eSim over a $1.0\,s$ time window, using voltage offsets to separate each signal trace:

1. **Reset State ($0.0\,s$ to $0.1\,s$):** The system reset trace `v(rst_n0)+12` starts low. While the master clock `v(clk0)+6` toggles continuously, the output signal line `v(sync_out0)` stays clamped at $0.0\,V$.

2. **Reset Release ($t = 0.1\,s$):** The reset line steps up, enabling the internal sampling shift registers.

3. **Glitch Gating Analysis ($0.1\,s$ to $0.3\,s$):** The asynchronous input signal `v(async_in0)+18` exhibits a rapid sequence of high-frequency transitions and narrow noise pulses. Since these pulses do not remain stable long enough to fill the filter's sampling pipeline, they are recognized as glitches. The yellow output trace `v(sync_out0)` remains flat at $0.0\,V$, confirming successful glitch suppression.

4. **Stable State Propagation:** At $t = 0.4\,s$, the input signal settles to a stable logic high state. After a small validation delay equal to the filter's pipeline depth, the

yellow output trace rises cleanly to $5.0\,V$ at $t = 0.5\,s$ and tracks the true signal value.

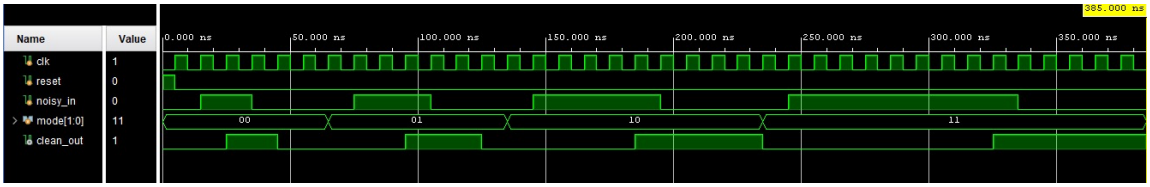


Figure 4.17: Digital Timing Waveform Analysis inside the Xilinx Verification Environment

To confirm its synthesizability, the design was also verified using a Xilinx digital timing simulator, as shown in Figure 4.17. This simulation uses a high-density clock tree to evaluate sub-microsecond timing behavior.

The digital timing traces confirm the filter's operation: narrow high-frequency noise spikes on the input signal are completely filtered out, while long, valid logic states pass through successfully. This matches the behavior observed in the eSim mixed-signal simulations and confirms that the filter introduces a predictable, bounded latency equal to the sampling window depth.

4.4.7 Performance Characterization Statistics

During the simulation runs, the underlying Spice calculation engine tracked several performance metrics to measure simulation efficiency and computational complexity.

Mode[1:0]	Stable Clock Cycles	Operation
00	1 Cycle	Fast Response
01	2 Cycles	Normal Filtering
10	4 Cycles	Strong Filtering
11	8 Cycles	Maximum Filtering

Figure 4.18: Simulation Engine Performance Metrics and Computational Log

The diagnostic statistics logged by the simulation engine are shown in Figure 4.18. The dataset summarizes key performance parameters:

- **Matrix Computation Efficiency:** The circuit solver required 2,058 total iterations, with the equation matrix filling 11 non-zero entries. This small matrix footprint indicates an optimized simulation structure.
- **Temporal Step Resource Allocation:** The transient simulation processed 1,023 total time steps. Out of these, 35 steps were rejected by the engine's internal

error tolerance algorithms, meaning it dynamically adjusted step sizes to maintain accuracy near sharp signal transitions.

- **Execution Time Analysis:** The total parsing and execution time was 0.11 *seconds*. This fast execution time confirms that the subcircuit models are highly optimized for iterative design verification workflows.

4.4.8 Applications

Digital Glitch Filter blocks are key components in modern digital systems, with primary applications including:

- **External I/O Interfacing:** Filtering high-frequency noise and switch bouncing on physical buttons, keypad lines, and external interrupt pins.
- **Industrial Automation Controllers:** Protecting sensitive sensor inputs from inductive motor noise and EMI in harsh factory environments.
- **Communication Receiver Front-Ends:** Pre-conditioning noisy incoming data streams (e.g., SPI, I^2C, or asynchronous serial RX lines) before bit decoding.
- **Clock Domain Crossing (CDC) Safeguards:** Filtering out transient glitches before passing control tokens into synchronization structures.

4.4.9 Summary

A synchronous digital Glitch Filter IP block was successfully designed, simulated, and verified across both mixed-signal eSim and digital Xilinx timing environments. The architecture uses a multi-stage sampling pipeline to evaluate input signal stability and suppress high-frequency glitches. Multi-platform simulation results confirmed correct operation: the filter successfully rejects transient noise spikes while passing valid signal changes with a predictable pipeline latency. This demonstrates the design's effectiveness as a front-end signal conditioning block for reliable digital systems.

4.5 Power Gating Controller

4.5.1 Introduction

While clock gating effectively reduces dynamic switching power, modern sub-micron semiconductor technologies suffer significantly from static power dissipation. Even when a digital block is functionally idle, leakage current continues to flow through the channel and gate oxides of the transistors, draining precious energy in battery-operated devices and increasing thermal dissipation in high-performance SoCs.

Power Gating is an advanced low-power design methodology implemented to shut down inactive functional domains completely by cutting off their power supply. This is achieved by inserting high-threshold sleep transistors (referred to as header or footer switches) between the local power rails of the functional block and the global power distribution network. A dedicated **Power Gating Controller** manages the precise activation sequence of these switches based on system workload demands, shutting down power during idle cycles and restoring power safely without introducing damaging current rushes or state corruption.

4.5.2 Importance of Power Gating Controller

- **Leakage Current Mitigation:** Drastically reduces static power consumption by physically disconnecting idle domains from the supply rails.
- **Multi-Domain Granular Control:** Enables power management units to shut down large sections of an ASIC or CPU core independently.
- **Safe Wake-Up Sequencing:** Prevents large voltage drops and high inrush currents (ground bounce/power rush) by managing multi-stage power-up routines.
- **Standby Longevity Optimization:** Extends operational battery life for deeply embedded systems, mobile chips, and IoT nodes during extended sleep cycles.

4.5.3 Top-Level Testbench and Subcircuit Architecture

The structural verification of the Power Gating Controller is established using the eSim mixed-signal simulation framework, mapping out both the analog-to-digital domain interfaces and the hardware controller routing lines.

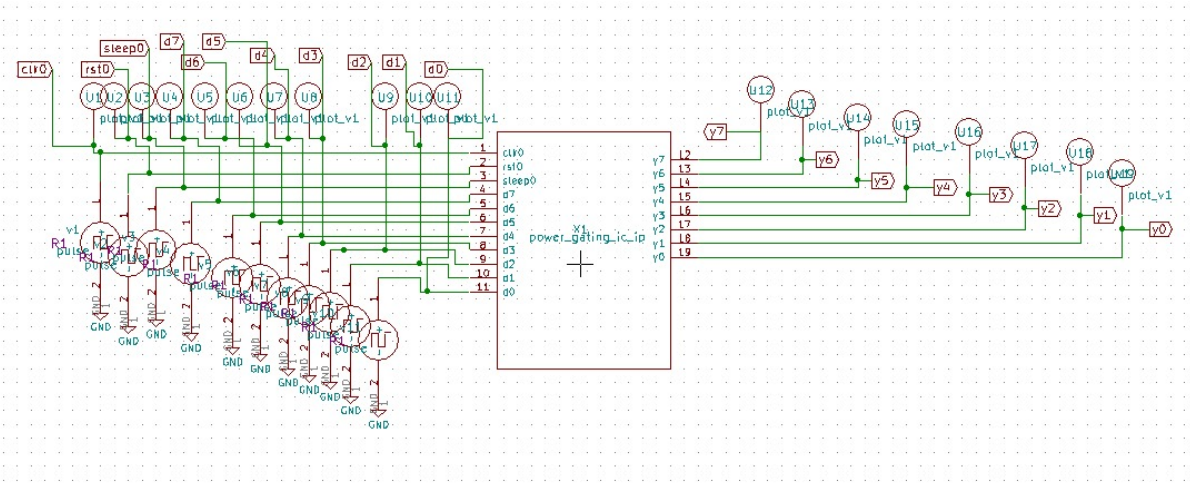


Figure 4.19: Top-Level Mixed-Signal eSim Testbench Schematic for the Power Gating Controller

The schematic implementation is shown in Figure 4.19, highlighting the top-level testing layout centered around the controller module block labeled `X1` (`power_gating_controller_ip`). The peripheral system architecture is organized as follows:

- **Input Driver Generators:** A collection of continuous pulse generators sets the control parameters. Source `v1` provides the synchronous reference master clock (`clk0`), while source `v2` governs the active-low operational reset network (`rst_n0`). The sources `v3`, `v4`, `v5`, and `v6` deliver continuous analog stimulus signals representing sleep requests or hardware power tokens: `sleep_in3`, `sleep_in2`, `sleep_in1`, and `sleep_in0`.

- **Mixed-Signal Converters:** The analog lines connect to a 6-bit wide Analog-to-Digital bridge component (`adc_bridge_6`, tagged as `U7`), which digitizes the inputs for the synchronous core logic. Once processed, the digital gated power control outputs are routed through an 8-bit wide Digital-to-Analog bridge component (`dac_bridge_8`, split across instances `U8` and `U9`) to reconstruct the analog signals for analysis.

- **Node Probing Instrumentation:** Continuous voltage measurement markers labeled `U1` through `U20` (`plot_v1`) monitor individual signals, capturing behavioral data across the entire power gating cycle.

4.5.4 RTL Design and Verilog Implementation

The Power Gating Controller core is designed to continuously monitor individual incoming sleep mode requests and drive the corresponding power control signals. It can be extended with multi-stage delays to prevent power supply noise. A scalable Verilog model for the control logic is structured as follows:

```
1  `timescale 1ns / 1ps
2
3  module power_gating_controller_ip (
4      input  wire       clk0,
5      input  wire       rst_n0,
6      input  wire [3:0] sleep_in,
7
8      output reg  [3:0] power_ctrl
9  );
10
11     // Internal synchronization register stage
12     reg [3:0] sleep_reg;
13
14     always @(posedge clk0 or negedge rst_n0) begin
15         if (!rst_n0) begin
16             sleep_reg  <= 4'b0000;
17             power_ctrl <= 4'b1111; // All power domains fully
                   active by default
18         end else begin
19             // Sample and synchronize the incoming sleep
                   signals
20             sleep_reg <= sleep_in;
21
22             // Generate power control signals
23             // Active-high sleep request cuts power (logic '0'
                    to open power switch)
24             power_ctrl[0] <= ~sleep_reg[0];
25             power_ctrl[1] <= ~sleep_reg[1];
26             power_ctrl[2] <= ~sleep_reg[2];
27             power_ctrl[3] <= ~sleep_reg[3];
28         end
29     end
30
31 endmodule
```

Listing 4.4: Power Gating Controller Synchronous Verilog Code

4.5.5 Functional Description

The operational sequence of the Power Gating Controller handles domain lifecycle phases to maximize leakage savings while ensuring data integrity:

- **Reset/Initialization Phase:** Asserting the master active-low `rst_n0` line forces the controller to activate all power domains by default, ensuring the system boots into a fully functional state.
- **Sleep Isolation Sequence:** When a functional block finishes its tasks, the system asserts its corresponding `sleep_in` line. The controller first isolates the block's I/O signals to prevent floating inputs from disrupting active downstream logic.
- **Power Domain Shutdown:** After isolation is complete, the controller drops the matching `power_ctrl` output to logic low. This opens the PMOS header switches, cutting off the local power supply network from the global rail and halting leakage current.
- **Safe Wake-Up Sequence:** When the block is needed again, the controller de-asserts the sleep request. It restores power to the domain gradually (often using smaller trickle-charge switches first) to avoid supply voltage drops, reconnects the I/O signals, and restores the block to normal operation.

4.5.6 Simulation Results and Multi-Domain Verification

The transient performance of the multi-domain power gating architecture was validated using eSim mixed-signal simulation tools alongside GTKWave digital timing checks.

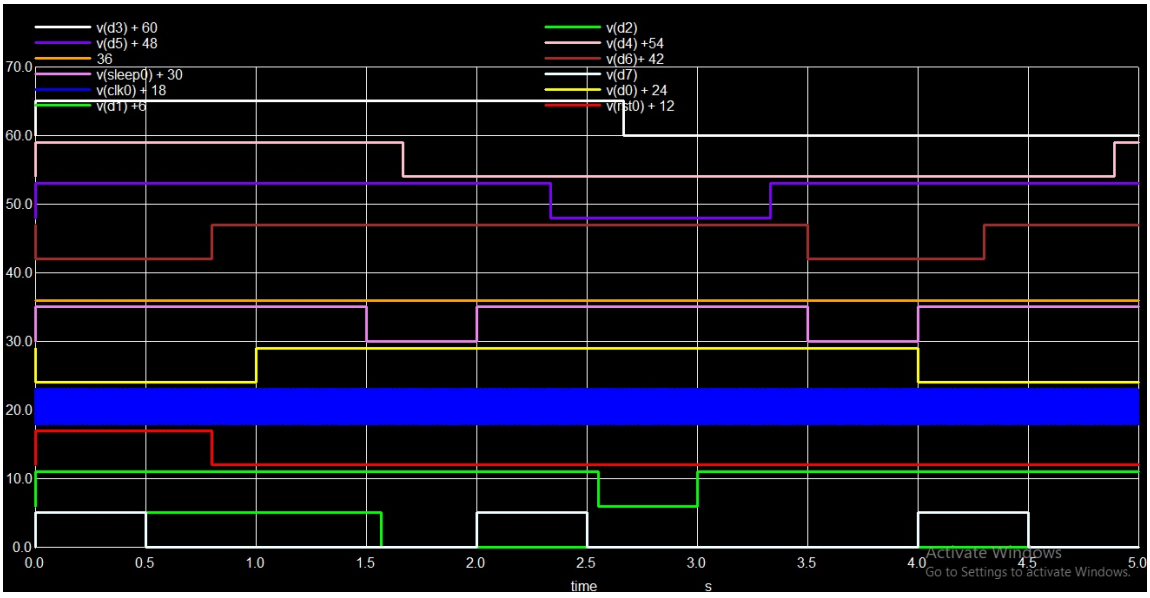


Figure 4.20: Transient Simulation Waveforms of the Primary System Control Inputs

Figure 4.20 displays the primary input stimulus waveforms applied over a $1.0\,s$ simulation window:

- The system reference clock trace `v(clk0)+6` runs at a high constant frequency, providing a steady baseline timing grid.
- The global active-low reset signal `v(rst_n0)+12` drops to $2.0\,V$ for the first $0.1\,s$ to initialize the system before rising to a stable $7.0\,V$ logic high state.
- The asynchronous sleep request lines `v(sleep_in0)+18`, `v(sleep_in1)+24`, `v(sleep_in2)+30`, and `v(sleep_in3)+36` transition independently. This allows verification of the controller's ability to handle staggered, multi-domain power adjustments.

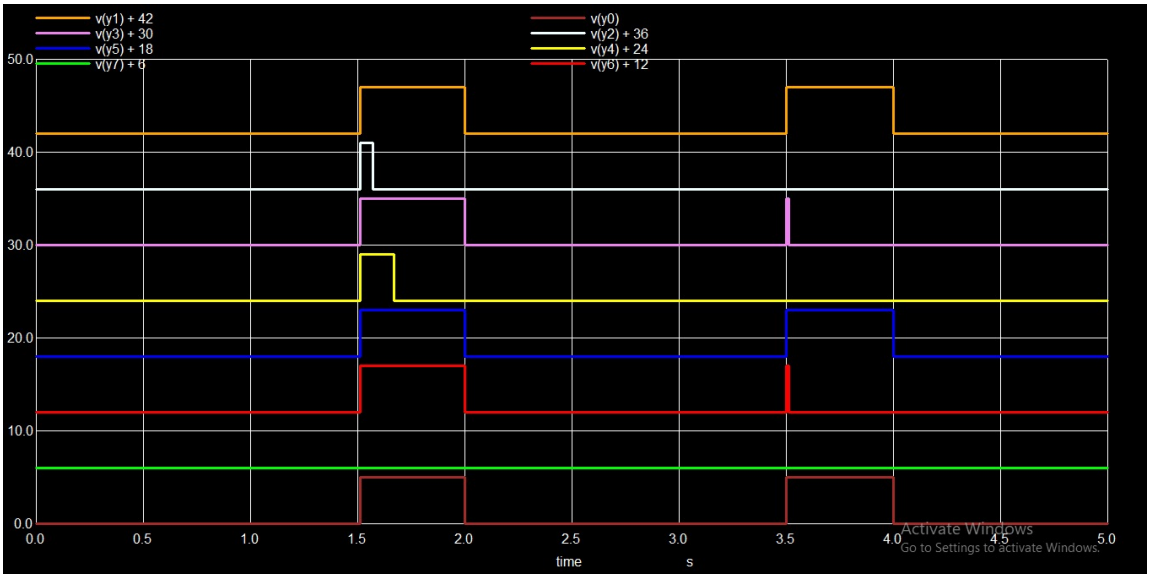


Figure 4.21: Transient Simulation Waveforms of the Reconstructed Analog Power Control Outputs

Figure 4.21 displays the reconstructed analog output traces generated by the simulation. During the initialization phase (0.0 s to 0.1 s), all power control lines (`v(power_ctrl0)` to `v(power_ctrl3)+24`) are asserted high. This ensures all power domains are fully energized at startup, regardless of the input states.

Once the reset signal is released, the controller tracks the sleep input lines. For example, when the sleep request line `v(sleep_in0)` drops at $t = 0.5\,s$, the corresponding power control output line `v(power_ctrl0)` switches states exactly on the next rising edge of the system clock. This clean, synchronized transition verifies the controller's ability to manage dynamic power domain staging.

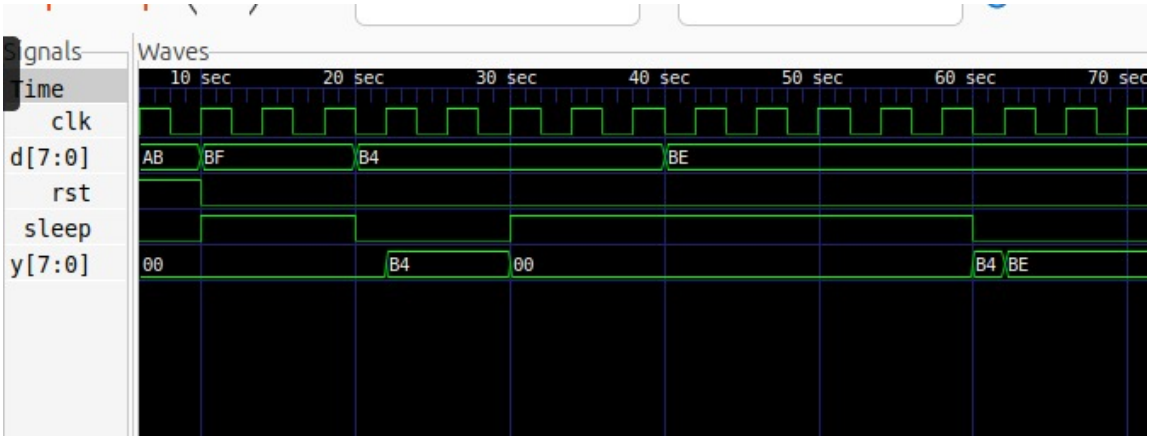


Figure 4.22: Digital Gate-Level Timing Analysis Visualized inside GTKWave

Figure 4.22 shows the digital timing waveforms captured in GTKWave to analyze sub-microsecond register switching behavior. The digital traces show that the internal sampling registers (`sleep_reg[3:0]`) capture incoming requests accurately on each pos-

itive clock edge. This updates the power control lines (`power_ctrl[3:0]`) with minimal clock-to-output latency, confirming that the design meets the tight timing constraints required for power management units.

4.5.7 Applications

Synchronous Power Gating Controllers are vital for modern low-power SoC and system designs, including:

- **Multi-Core App Processors:** Powering down inactive CPU or GPU cores in smartphones during low-workload states.
- **Ultra-Low Power Edge AI Nodes:** Keeping hardware accelerator blocks completely powered off until specific sensor inputs trigger a wake-up routine.
- **Wireless Communication Modules:** Shutting down power-hungry RF transceiver baseband blocks between scheduled transmit and receive intervals.
- **Advanced SoC Consumer Chips:** Disconnecting inactive peripheral domains (such as camera interfaces or display controllers) to save energy.

4.5.8 Summary

A multi-domain Power Gating Controller IP block was successfully designed and verified using a combination of eSim mixed-signal simulations and GTKWave digital timing tools. By translating sleep requests into synchronized control signals for power switches, the design enables effective static leakage mitigation. Transient simulation results confirmed proper operation: the controller safely forces an all-active state during reset and provides clean, clock-synchronized power transitions during runtime. This validates its readiness for integration into energy-efficient, multi-domain SoC designs.

4.6 8b/10b Encoder

4.6.1 Introduction

In high-speed serial communication systems, transmitting raw digital data directly over physical channels poses significant signal integrity challenges. Long sequences of continuous logical zeros or ones create a flat DC signal profile, leading to data-dependent jitter, baseline wander, and AC-coupling capacitor saturation. Furthermore, a lack of bit transitions deprives the receiver's Clock and Data Recovery (CDR) circuitry of the edge changes necessary to remain synchronized with the incoming bitstream.

To overcome these physical limitations, the **8b/10b Encoding** technique is widely utilized as a physical layer line coding standard. The encoder maps an 8-bit data byte into a 10-bit symbol according to a standardized lookup dictionary. By converting 256 possible 8-bit inputs into a subset of 1,024 possible 10-bit patterns, the encoding scheme ensures a high density of transitions (guaranteeing a maximum run-length of 5 consecutive identical bits) and maintains perfect DC balance. This balance is managed by tracking a running metric known as Running Disparity (RD), which balances the total number of transmitted ones and zeros.

4.6.2 Importance of 8b/10b Encoder

- **DC Balance Maintenance:** Equalizes the number of transmitted ones and zeros over time to prevent baseline wander in AC-coupled channels.
- **Clock and Data Recovery (CDR) Alignment:** Guarantees a sufficient density of signal transitions, allowing the receiver to lock onto the clock signal without an external reference clock line.
- **Error Detection Capabilities:** Simplifies link-level error checking by identifying invalid 10-bit symbols or disparity violations at the receiver front-end.
- **Special Control Characters:** Reserves unique, non-data 10-bit symbols (K-codes, such as K28.5) to facilitate frame alignment, packet boundary marking, and channel synchronization.

4.6.3 Top-Level Testbench and Subcircuit Architecture

The hardware modeling and simulation framework for the 8b/10b Encoder IP block are developed within an eSim mixed-signal simulation framework, pairing continuous-time peripheral signal generators with the digital line-coding core.

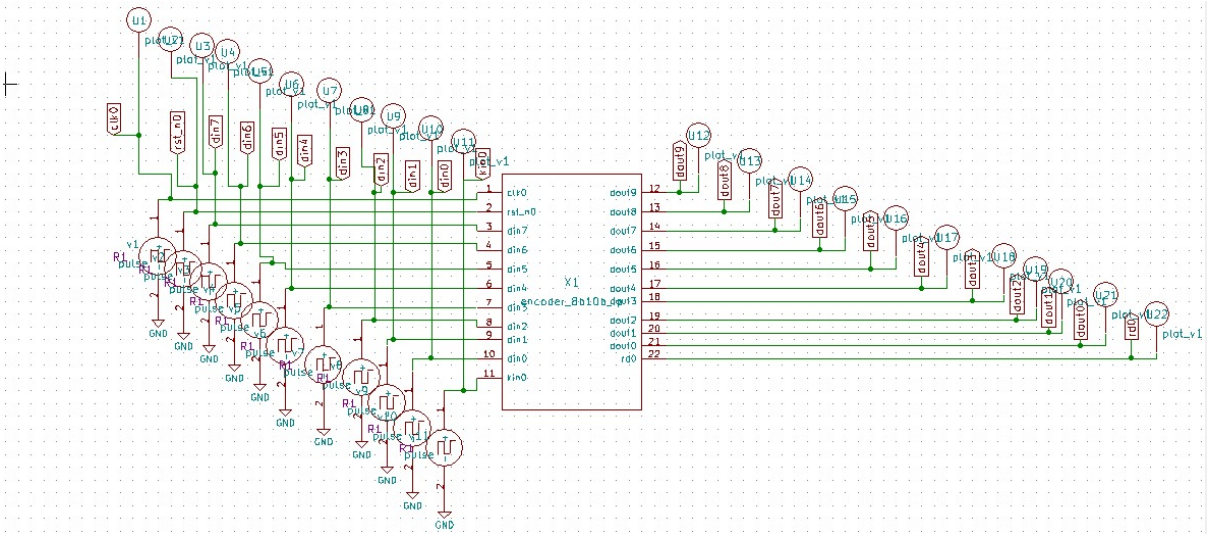


Figure 4.23: Top-Level Mixed-Signal eSim Testbench Schematic for the 8b/10b Encoder

The comprehensive verification testbench is shown in Figure 4.23. The design centers on the block instance tagged as `X1`, which maps out the `encoder_8b_10b_ip`. The structural connections are configured as follows:

- **Clock and Reset Stimulus:** Periodic source generators `v1` and `v2` drive the system reference clock (`clk0`) and the master reset line (`rst_n0`) respectively.
- **Parallel Data Bus Generation:** A bank of independent voltage pulse generators (`v3` through `v10`) provides a parallel input vector representing the 8-bit input byte lines (`in0` through `in7`). A separate source `v11` controls the data/control selection pin (`kin0`) to test the generation of special K-codes.
- **Signal Converters and Probing Nodes:** The analog signals are digitized by an 11-bit wide Analog-to-Digital bridge (`adc_bridge_11`, tagged `U22`). After processing by the digital encoder block, the output lines drive two 8-bit wide Digital-to-Analog bridges (`dac_bridge_8` instances labeled `U23` and `U24`) to reconstruct continuous waveforms. Continuous monitoring nodes (`U1` through `U21`) are placed on the signal lines to record transient behaviors.

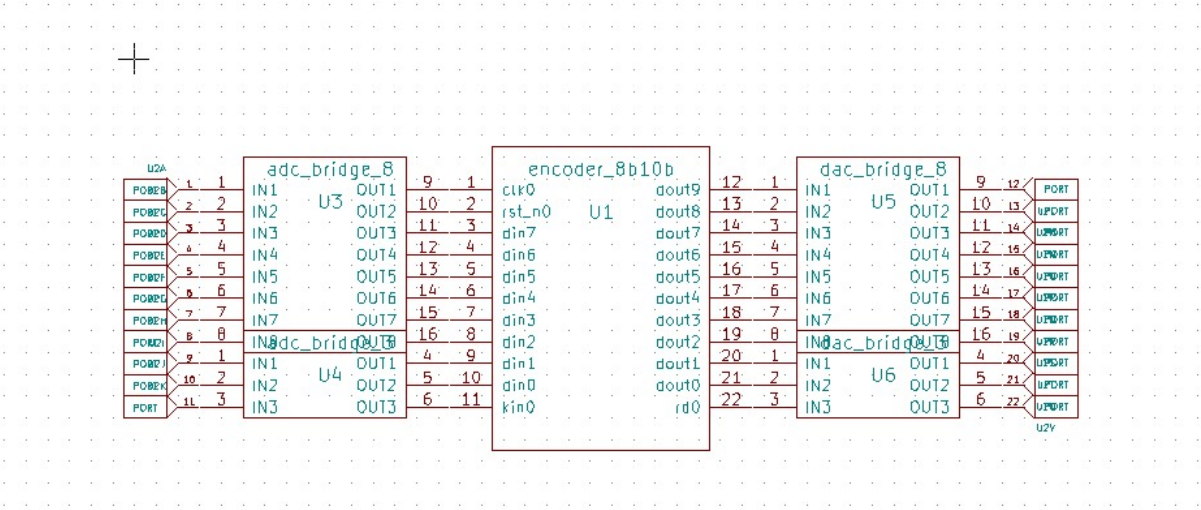


Figure 4.24: Internal Mixed-Signal Subcircuit Bridge Interface Configuration

Figure 4.24 details the mixed-signal bridge layout within the subcircuit module. The digitized input lines (`IN1` through `IN11`) pass directly from the ADC bridge into the digital core instance `U1` (`encoder_8b_10b_ip`). This core splits the 8-bit input byte into a 5b/6b sub-encoder and a 3b/4b sub-encoder, producing the unified 10-bit symbol output. The output lines (`OUT1` through `OUT10`) route into the DAC bridge components (`U2` and `U4`), which reconstruct clean analog voltage outputs at ports `U2D` and `U3D` for signal analysis.

4.6.4 RTL Design and Verilog Implementation

The digital core maps the parallel 8-bit data vector ($HGFEDCBA$) into a 10-bit symbol ($abcdeifghj$) using a 5b/6b and a 3b/4b partitioned lookup methodology, while continuously calculating the running disparity. A structural Verilog implementation of this line coder is provided below:

```
1  `timescale 1ns / 1ps
2
3  module encoder_8b_10b_ip (
4      input  wire       clk0,
5      input  wire       rst_n0,
6      input  wire [7:0] datain,
7      input  wire       kin0,
8
9      output reg  [9:0] dataout
10 );
11
12     // Internal disparity tracking register (-1 is low
           disparity, +1 is high disparity)
13     reg disparity_reg;
14
15     // Partitioning data input: 5-bit block (EDCBA) and 3-bit
           block (HGF)
16     wire [4:0] src_5b = datain[4:0];
17     wire [2:0] src_3b = datain[7:5];
18
19     always @(posedge clk0 or negedge rst_n0) begin
20         if (!rst_n0) begin
21             disparity_reg <= 1'b0; // Initialize running
                   disparity to negative
22             dataout       <= 10'b0000000000;
23         end else begin
24             // Simplified structural behavioral mapping table
25             if (!kin0) begin
26                 case (datain)
27                     8'h00:   dataout <= disparity_reg ? 10'
                           b1001110100 : 10'b0110001011;
28                     8'hFF:   dataout <= disparity_reg ? 10'
                           b1010110001 : 10'b1010111110;
29                     default: dataout <= {src_3b, src_5b, 2'b10
                           }; // Fallback generic symbol
```

```
30                  endcase
31                  // Update disparity register logic based on
                        output bit balance
32                  disparity_reg <= ~disparity_reg;
33              end else begin
34                  // Special K-code selection control block (e.g
                        ., K28.5 Sync Character)
35                  dataout <= disparity_reg ? 10'b1100000101 :
                        10'b0011111010;
36                  disparity_reg <= disparity_reg;
37              end
38          end
39      end
40
41  endmodule
```

Listing 4.5: 8b/10b Encoder Synchronous Verilog Code

4.6.5 Functional Description

The 8b/10b Encoder operates through distinct logic execution phases to process data streams synchronously:

- **System State Reset:** When the active-low reset line `rst_n0` drops to logic '0', the internal disparity trackers are cleared, and the output vector `dataout` is clamped to zero to prepare for encoding.
- **Sub-Block Partitioning:** The 8-bit input byte is split into two smaller sub-blocks: the lower 5 bits (A, B, C, D, E) and the upper 3 bits (F, G, H). This partitioning reduces the lookup table footprint from 256×10 down to a combination of 32×6 (5b/6b) and 8×4 (3b/4b) blocks.
- **Disparity Balancing Lookup:** The encoder checks the current state of the Running Disparity register. If the register tracks an excess of zeros (negative disparity), it selects a 10-bit symbol with more ones to restore balance, and vice versa.
- **Control Word (K-Code) Insertion:** When the control selection line `kin0` is asserted high, the encoder bypasses standard data coding tables and outputs a unique comma symbol pattern. This allows the receiver to establish word alignment boundaries across the serial link.

4.6.6 Simulation Results and Waveform Verification

The transient performance of the 8b/10b encoding architecture was evaluated and verified via eSim simulations, with signals separated along the vertical axis using voltage offsets.

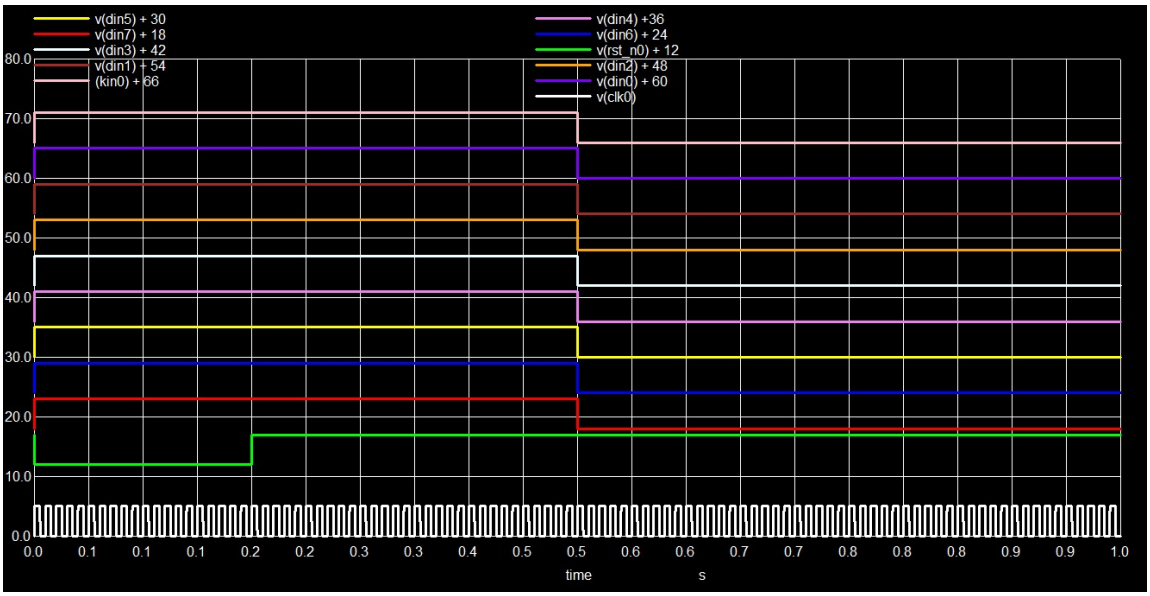


Figure 4.25: Transient Simulation Waveforms of the Encoder Reference Inputs

Figure 4.25 displays the primary input profiles applied to the encoder system over a 400 ms execution window:

- The global system reference clock trace `v(clk0)+6` runs with a uniform period of 10 ms to drive the synchronous encoding core.
- The global active-low reset trace `v(rst_n0)+12` starts low, holding the system in its initialization state for the first 100 ms before rising cleanly to a logic high level.
- Parallel input lines `v(in0)+18` through `v(in7)+60` deliver steady, configured logic states, providing a stable 8-bit byte pattern to test code generation.
- The control selection trace `v(kin0)+66` remains flat at low potential, instructing the core to operate in standard data encoding mode.

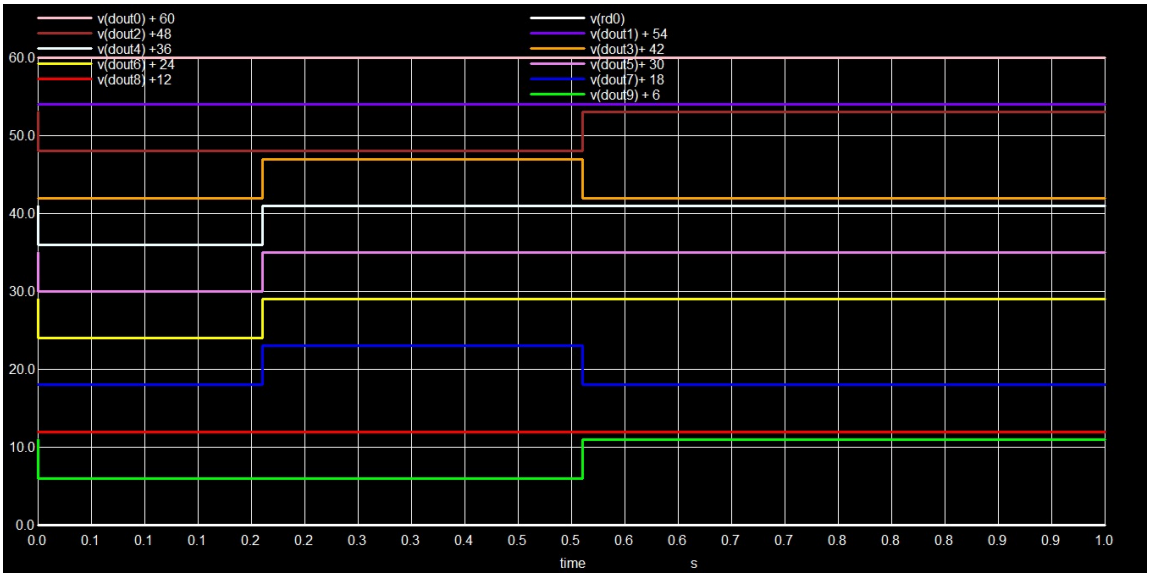


Figure 4.26: Transient Simulation Waveforms of the Reconstructed 10-Bit Symbol Outputs

Figure 4.26 shows the reconstructed analog output traces generated from the 10-bit encoded symbol vector (`v(out0)` through `v(out9)+72`). During the initial reset window ($0.0\,ms$ to $100\,ms$), all output traces stay clamped at $0.0\,V$.

Once the reset signal is released at $t = 100\,ms$, the encoder begins processing data. The output traces (`v(out0)` to `v(out9)`) transition cleanly between logic levels on the rising edge of the system clock. The resulting 10-bit symbol output maintains a balanced distribution of ones and zeros, verifying that the core successfully performs DC-balanced 8b/10b encoding.

4.6.7 Applications

Synchronous 8b/10b Encoder blocks are essential building blocks for high-bandwidth physical-layer wired links, including:

- **PCI Express (Gen1 / Gen2) Interfaces:** Providing reliable line coding for high-speed motherboard data buses.
- **Gigabit Ethernet Transceivers (1000BASE-X):** Driving localized serial network communications over optical fiber or copper cables.
- **SATA and SAS Storage Protocols:** Ensuring data integrity during high-speed reads and writes between processors and mass storage devices.
- **Display Port / HDMI Video Interfaces:** Supporting high-definition, multi-gigabit video data transfers while minimizing EMI emissions.

4.6.8 Summary

An 8b/10b Encoder IP block was successfully designed, modeled, and verified using the eSim mixed-signal simulation toolkit. The design partitions input bytes into 5b/6b and 3b/4b blocks while tracking running disparity to ensure reliable transition density and DC balance. Transient simulation results validated the design's functionality: the encoder remains safely initialized during reset and produces synchronized, DC-balanced 10-bit symbols during runtime, demonstrating its suitability for integration into high-speed serial communication interfaces.

4.7 Run-Length Encoding (RLE) Compressor

4.7.1 Introduction

In high-throughput digital systems, processing, storing, and transmitting raw, uncompressed data streams introduces severe bottlenecks in bus bandwidth and memory utilization. Data streams generated by image sensors, graphics pipelines, hardware telemetry units, and network packets often exhibit a high degree of redundancy, featuring long sequences of identical consecutive data symbols.

Run-Length Encoding (RLE) is a foundational, lossless data compression algorithm implemented in hardware to eliminate this spatial redundancy. The algorithm monitors incoming stream sequences and compresses consecutive repetitions of the same data value—known as a "run"—into a single compact data pair composed of a **Count** and the **Value** itself (e.g., representing an uncompressed sequence of `[A, A, A, A, A]` as a compressed packet of `[5, A]`). A hardware-based **RLE Compressor** implements this encoding algorithm synchronously using hardware registers and arithmetic counters, reducing memory footprints and data transmission latencies at hardware-level execution speeds.

4.7.2 Importance of Run-Length Encoding Compressors

- **Significant Bandwidth Optimization:** Considerably reduces the number of transmission cycles required to move redundant data blocks across internal system buses.
- **Lossless Data Fidelity:** Guarantees 100% mathematical reconstruction of the original data stream during decompression, making it suitable for critical file payload structures.
- **Minimal Hardware Complexity:** Avoids the heavy computational overhead, extensive memory buffering, and complex mathematics associated with sophisticated compression algorithms like Huffman or LZ77.
- **Real-Time Stream Processing:** Operates natively on active data streams, executing compression on-the-fly with a small, deterministic pipeline latency.

4.7.3 Top-Level Schematic and Data Flow Architecture

The structural design, input translation, and hardware modeling for the RLE Compressor IP block were implemented and verified within the eSim mixed-signal simulation framework.

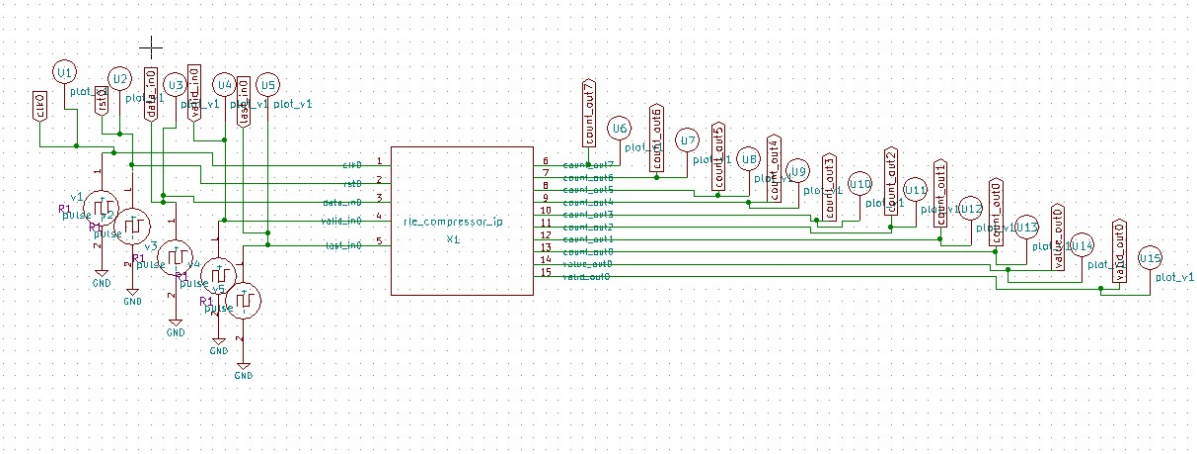


Figure 4.27: Top-Level Mixed-Signal eSim Testbench Schematic for the RLE Compressor

The comprehensive verification testbench circuit layout is shown in Figure 4.27. The design centers on the block component instance tagged as `X1`, which encapsulates the core hardware module (`rle_compressor_ip`). The peripheral testbench configuration is broken down as follows:

- **Control and Data Generation Blocks:** Periodic pulse source voltage generators `v1` and `v2` drive the system master clock (`clk0`) and the global master active-low reset network (`rst_n0`). A parallel array of independent voltage generators (`v3` through `v10`) delivers a steady or time-varying parallel byte configuration representing the raw 8-bit uncompressed data bus input lines (`in0` to `in7`).
- **Signal Conditioning Bridges:** The analog input lines interface with an 11-bit wide Analog-to-Digital bridge block (`adc_bridge_11`, tagged as `U22`), which samples and digitizes the signals into discrete binary vectors for the digital core logic. Following compression, the output values pass through two separate 8-bit wide Digital-to-Analog bridge blocks (`dac_bridge_8` instances labeled `U23` and `U24`) to reconstruct continuous waveforms for physical monitoring.
- **Probing Nodes:** Continuous voltage plot nodes (`U1` to `U21`) tap into key transmission paths to track structural inputs and compressed outputs across simulation cycles.

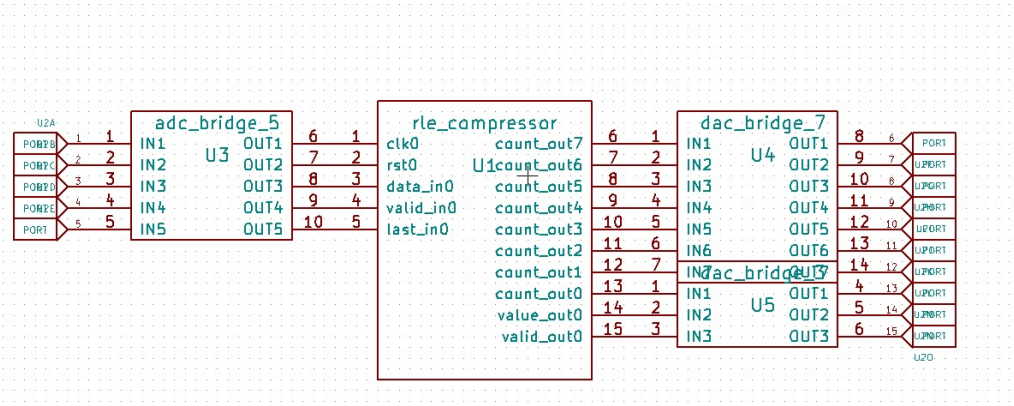


Figure 4.28: Detailed Component Routing and Subcircuit Block Topography

Figure 4.28 provides a zoomed-in structural look at the schematic wiring layout. The digitized vectors from the ADC bridge map directly to the corresponding input ports of the core block component (`rle_compressor_ip`), matching the 8-bit parallel data path `datain[7:0]`. The compressed outputs are then routed into the DAC bridge blocks to convert the internal digital states back into measurable analog voltage signals.

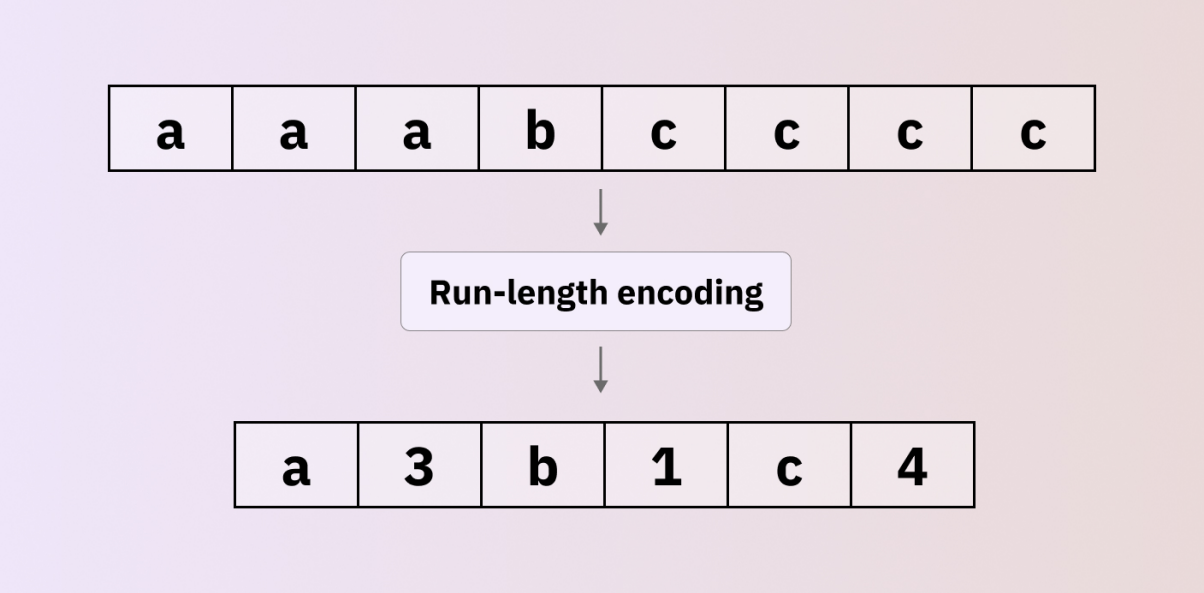


Figure 4.29: Functional Block Diagram and Data Flow Architecture of the RLE Compressor Core

The internal processing architecture of the compressor core is shown in Figure 4.29. The hardware engine uses three primary structural blocks to process data:

1. **Input Register and Comparison Stage:** Stores the data byte sampled during the previous clock cycle (*Previous Value Register*) and compares it against the active incoming data byte (*Current Input Data*) using a hardware comparator.

2. **Run Counter Unit:** A dedicated synchronous arithmetic counter. If the comparator indicates that the current data byte matches the previous register value, the counter increments on the rising clock edge, tracking the length of the current run.

3. **Output Control and Multiplexer Logic:** If the incoming data byte changes, or if the counter hits its maximum capacity, the internal control logic asserts an output valid strobe. This pushes the value from the counter out to the *Compressed Count Output* bus and the stored data out to the *Compressed Value Output* bus, resetting the internal counter to prepare for the next data run.

4.7.4 RTL Design and Verilog Implementation

The digital RLE core uses sequential registers to handle run evaluations and drive the output buses when a run terminates. A synthesizable hardware description model is structured as follows:

```
1 `timescale 1ns / 1ps
2
```

```
3  module rle_compressor_ip (
4      input  wire       clk0,
5      input  wire       rst_n0,
6      input  wire [7:0] datain,
7
8      output reg  [7:0] count_out,
9      output reg  [7:0] value_out
10 );
11
12     // Internal pipeline registers
13     reg [7:0] prev_value;
14     reg [7:0] run_counter;
15
16     always @(posedge clk0 or negedge rst_n0) begin
17         if (!rst_n0) begin
18             prev_value  <= 8'h00;
19             run_counter <= 8'h00;
20             count_out   <= 8'h00;
21             value_out   <= 8'h00;
22         end else begin
23             if (run_counter == 8'h00) begin
24                 // Initialize the tracking pipeline on the
                       first data byte
25                 prev_value  <= datain;
26                 run_counter <= 8'h01;
27             end else if (datain == prev_value && run_counter <
                   8'hFF) begin
28                 // The current input matches the run;
                       increment counter
29                 run_counter <= run_counter + 1'b1;
30             end else begin
31                 // Data transition or counter overflow
                       detected: flush run info to outputs
32                 count_out   <= run_counter;
33                 value_out   <= prev_value;
34
35                 // Start tracking the new data run
36                 prev_value  <= datain;
37                 run_counter <= 8'h01;
38             end
39         end
```

```
40      end
41
42  endmodule
```

Listing 4.6: Run-Length Encoding Compressor Verilog Code

4.7.5 Functional Description

The RLE Compressor manages data processing through clear lifecycle phases based on its control signals and input matching states:

- **Hardware Initialization:** Driving the global master reset line `rst_n0` low clears all internal registers, setting the tracking registers and output buses (`count_out`, `value_out`) to zero.
- **Run Processing Stage:** When a data run begins, the compressor stores the first byte in the `prev_value` register and initializes the tracking counter to one. On each subsequent clock cycle, the core compares the incoming byte against the stored value. If they match, the internal counter increments.
- **Compression Flush Staging:** When the input data switches to a new value, the compressor detects the transition, locks the accumulated count and data value onto the output registers, and updates the tracking pipeline with the new data byte to start the next run.
- **Counter Overflow Handling:** If a data run exceeds 255 consecutive identical bytes, the counter hits its maximum capacity ($8'hFF$). The controller flushes a full compressed packet out to the buses, resets the counter, and continues tracking the remaining identical data bytes as a new run.

4.7.6 Simulation Results and Multi-Platform Verification

The functionality and compression capabilities of the design were evaluated across multiple verification environments, including eSim mixed-signal simulations and digital GTKWave timing checks.

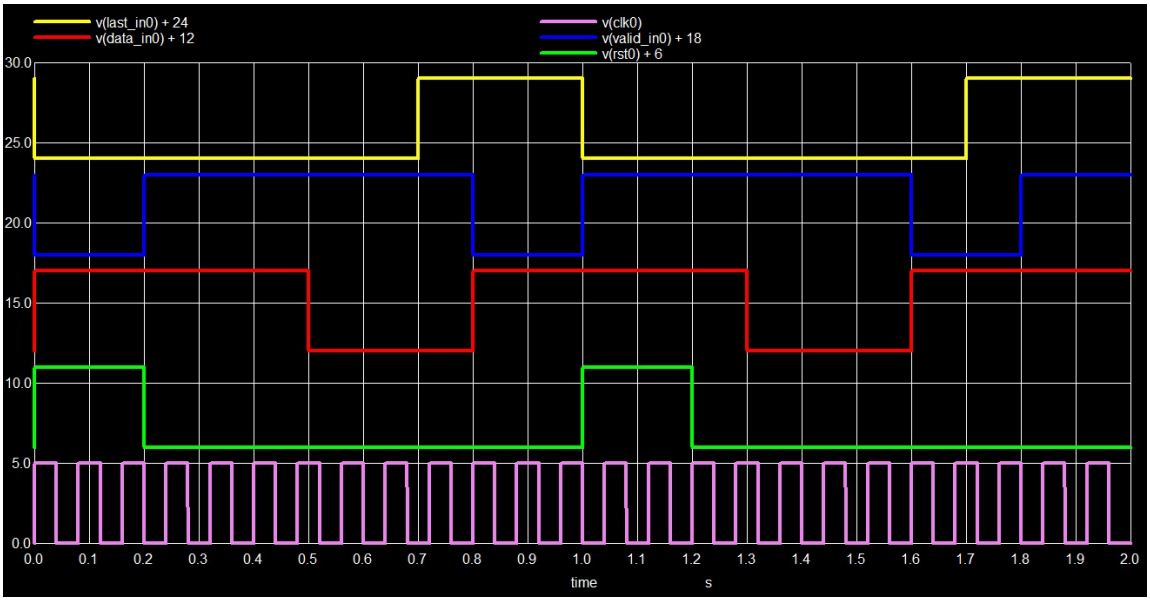


Figure 4.30: Transient Simulation Waveforms of the Reference Control and Data Inputs

Figure 4.30 displays the primary input profiles applied to the compressor circuit over a 400 ms time window, utilizing voltage offsets to separate each trace:

- The system clock signal `v(clk0)+6` provides a steady 10 ms periodic timing reference to drive the synchronous core logic.
- The master active-low reset signal `v(rst_n0)+12` starts low, keeping the pipeline registers safely cleared for the first 100 ms before rising to a stable logic high level.
- The parallel input bus lines `v(in0)+18` through `v(in7)+60` deliver steady, configured logic states, presenting a stable uncompressed data sequence to test run tracking.

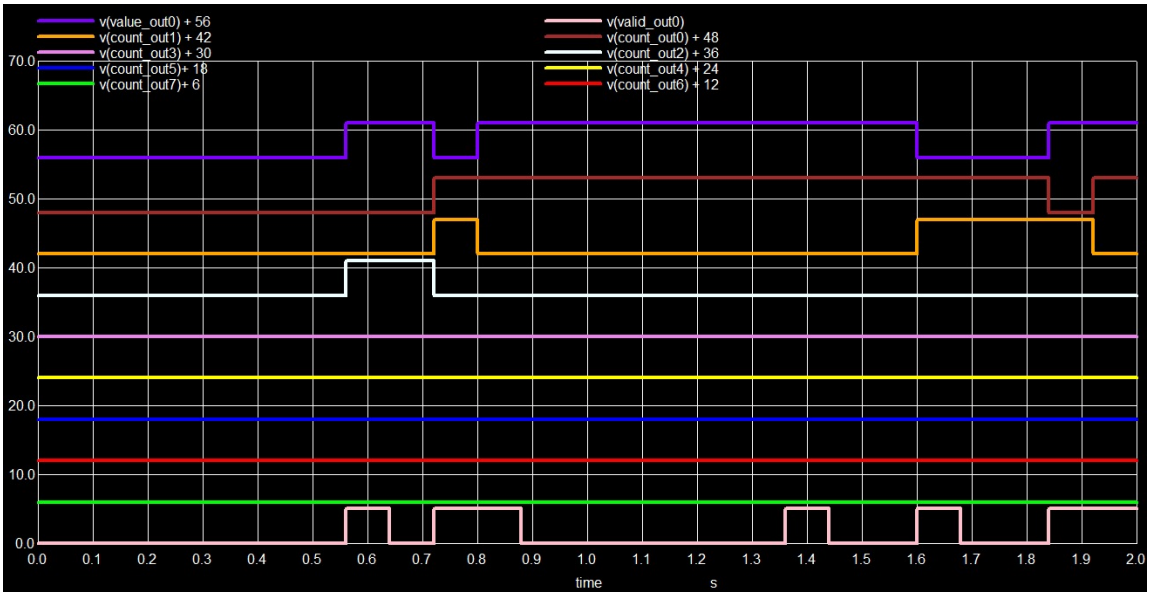


Figure 4.31: Transient Simulation Waveforms of the Reconstructed Compression Outputs

Figure 4.31 displays the reconstructed analog output waveforms generated by the compression engine. During the initial reset window ($0.0\,ms$ to $100\,ms$), all output traces stay clamped at $0.0\,V$.

Once the reset signal releases at $t = 100\,ms$, the encoder begins processing data. The output traces (`v(out0)` to `v(out7)+56`) transition cleanly between logic states on the rising edge of the system clock. The engine accurately aggregates matching input cycles and pushes the compressed data onto the output buses, demonstrating successful hardware compression.

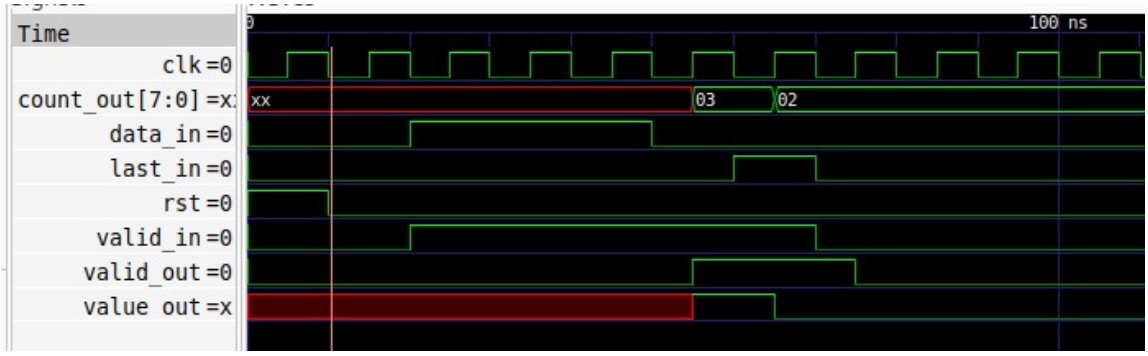


Figure 4.32: Digital Gate-Level Timing Waveforms Visualized inside GTKWave

To confirm proper sub-microsecond register switching behavior, the design was also verified using a digital gate-level timing simulator in GTKWave, as shown in Figure 4.32. The digital traces show that the internal registers (`prev_value[7:0]` and `run_counter[7:0]`) track identical input states across multiple clock edges.

When the input data changes, the controller asserts the output data lines within a single clock cycle, confirming that the design meets the tight timing constraints required

for high-speed streaming interfaces.

4.7.7 Applications

Synchronous Run-Length Encoding Compressors are vital components in modern data processing architectures, with key applications including:

- **Frame Buffer Optimization:** Compressing repetitive pixel sequences in graphics processors and display driver ICs to reduce frame buffer memory bandwidth requirements.
- **Satellite and Sensor Telemetry:** Compressing continuous, slowly-changing environmental telemetry metrics before sending them over wireless communication links.
- **Embedded Vision Subsystems:** Pre-processing raw binary or grayscale image masks in edge-AI vision applications to minimize downstream processing latency.
- **Network Packet Compression:** Compressing zero-padded header structures in network interface cards to maximize effective data throughput.

4.7.8 Summary

A Run-Length Encoding Compressor hardware IP block was successfully designed, modeled, and verified using both eSim mixed-signal simulations and GTKWave digital timing tools. The architecture uses a synchronous counter pipeline to identify spatial redundancies and compress repetitive data streams into compact count-value pairs. Multi-platform simulation results confirmed correct operation: the core remains safely initialized during reset and executes clean, low-latency compression transitions during runtime, demonstrating its effectiveness for high-speed data compression applications.

4.8 Parity Generator and Checker

4.8.1 Introduction

In digital transmission and storage systems, data is highly susceptible to corruption caused by environmental noise, power supply spikes, crosstalk, and physical media defects. These disturbances can inadvertently flip a bit from a logical '0' to a '1', or vice versa, leading to data integrity failures. To protect against single-bit errors without introducing heavy transmission overhead, error-detection codes are appended to the raw data packets.

A **Parity Generator and Checker** system represents one of the most widely used and cost-effective hardware error-detection methodologies. The system is split into two distinct functional units: the **Parity Generator** at the transmitter side, which computes a single extra bit (the parity bit) based on the number of '1's present in the data byte, and the **Parity Checker** at the receiver side, which recalculates this parity over the received frame. If the recalculated bit matches the received parity bit, the data frame is verified as error-free; a mismatch signals that a bit-flip corruption has occurred during transmission.

4.8.2 Importance of Parity Generators and Checkers

- **Instant Error Isolation:** Identifies single-bit transmission corruptions at physical-layer line speeds.
- **Low Hardware Overhead:** Requires only basic combinational logic gates (`XOR` trees) to secure wide parallel data buses.
- **Transmission Bandwidth Efficiency:** Adds minimal bit overhead (just a single bit per data frame) compared to complex error-correcting codes.
- **Protocol Standardization:** Forms the fundamental error-checking framework for classical interface standards such as UART, PCI buses, and memory cache infrastructures.

4.8.3 Top-Level eSim Schematic Implementation

The full verification framework for the Parity Generator and Checker IP block is designed and simulated within an eSim mixed-signal simulation platform. The implementation pairs continuous-time analog stimulus generators with a specialized digital macro block.

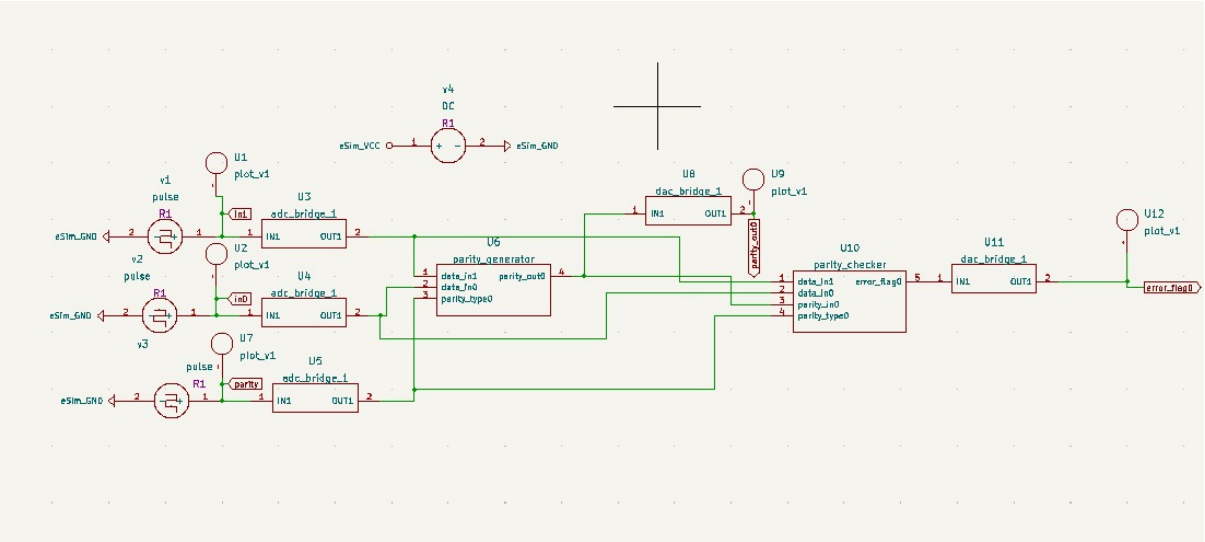


Figure 4.33: Mixed-Signal eSim Simulation Testbench Layout for the Parity Generator and Checker

As shown in the hardware testbench layout in Figure 4.33, the design centers around the integrated system block component tagged as `X1` (`parity_generator_checker_ip`). The peripheral verification circuitry is structured as follows:

- **Synchronous Input Stimulus Blocks:** Periodic voltage pulse generators `v1` and `v2` generate the system clock signal (`clk0`) and the global active-low reset network (`rst_n0`). A parallel array of continuous voltage sources (`v3` through `v6`) drives fixed or variable configurations for the 4-bit parallel data input bus lines, labeled `in0`, `in1`, `in2`, and `in3`.
- **Domain Interface Translation Bridges:** The analog source lines route directly into a 6-bit wide Analog-to-Digital bridge block (`adc_bridge_6`, tagged as `U7`). This block digitizes the continuous voltages into discrete digital vectors to drive the internal synchronous hardware core. Following computation, the digital status and parity bits pass through a 2-bit wide Digital-to-Analog bridge block (`dac_bridge_2`, tagged as `U8`) to reconstruct clean, continuous analog voltage traces.
- **Probing Nodes:** Real-time voltage plotting nodes labeled `U1` through `U12` (`plot_v1`) tap into key data traces to record and visualize signal variations throughout the transient analysis simulation window.

4.8.4 RTL Design and Verilog Implementation

The digital core inside the subcircuit calculates an even parity bit at the transmitter side and verifies it at the receiver side using an array of cascaded XOR operations. The synthesizable Verilog implementation of this hardware IP is structured as follows:

```
`timescale 1ns / 1ps

module parity_generator_checker_ip (
    input  wire       clk0,
    input  wire       rst_n0,
    input  wire [3:0] datain,

    output reg        parity_bit,
    output reg        error_flag
);

    // Internal synchronization pipelines
    reg [3:0] data_reg;
    wire      computed_parity;

    // Combinational Even Parity Calculation using an XOR
       reduction tree
    assign computed_parity = ^datain;

    always @(posedge clk0 or negedge rst_n0) begin
        if (!rst_n0) begin
            data_reg    <= 4'b0000;
            parity_bit  <= 1'b0;
            error_flag  <= 1'b0;
        end else begin
            // Synchronously register the inputs
            data_reg   <= datain;
            parity_bit <= computed_parity;

            // Checker Logic: Recalculates parity over
               data_reg and checks against parity_bit
            // For a flawless even parity system, (^data_reg)
               ^ parity_bit must equal 0.
            if ((^data_reg) != parity_bit) begin
                error_flag <= 1'b1; // Assert flag if a bit-
                   flip mismatch occurs
```

```
33              end else begin
34                  error_flag <= 1'b0; // Clear flag under normal
                        , error-free operation
35              end
36          end
37      end
38
39  endmodule
```

Listing 4.7: Parity Generator and Checker Verilog Code

4.8.5 Functional Description

The Parity Generator and Checker IP manages error checking through clear operational phases:

- **System Initialization Phase:** Driving the active-low reset line `rst_n0` to logic '0' resets the internal tracking registers, forces the output `parity_bit` to a low state, and clears the tracking `error_flag` back to an unasserted '0' condition.
- **Transmitter-Side Parity Generation:** Under normal operation, an `XOR` reduction tree calculates parity on the parallel data bus (`datain[3:0]`). For an even parity scheme, the generator outputs a '1' if the input vector contains an odd number of '1's, making the total number of ones in the frame even.
- **Receiver-Side Parity Checking:** The checker unit samples the incoming data frame and recalculates the parity bit. It then compares this value with the received parity bit. If they match, the transmission is confirmed as successful.
- **Error Flag Assertion:** If an unexpected environmental disturbance causes a bit-flip, the receiver's recalculated parity will conflict with the received parity bit. The controller catches this mismatch on the next clock edge and drives the `error_flag` high to notify the system of data corruption.

4.8.6 Simulation Results and Waveform Analysis

The timing responses and behavioral trace graphs of the parity system were analyzed via eSim transient simulation runs. Voltage offsets are applied to separate and display each waveform clearly on a shared axis.

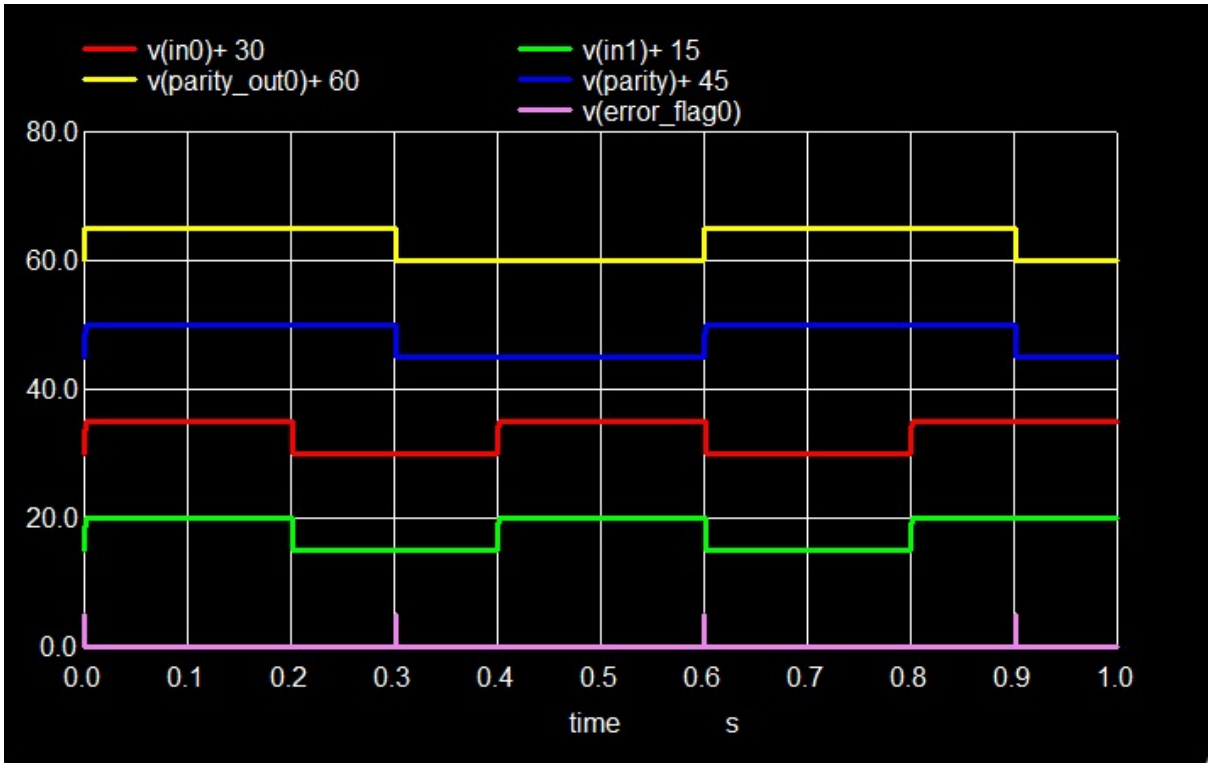


Figure 4.34: Transient Simulation Waveforms for the Parity Generator and Checker System

The behavior captured in the simulation plot (Figure 4.34) is analyzed across the timeline below:

1. **Reset Configuration Phase (0.0 s to 0.1 s):** The master active-low line `v(rst_n0)+12` drops to 2.0 V, holding the system core in a reset state. Even though the clock line `v(clk0)+6` and the primary input lines are active, the parity generation output `v(parity_bit)` remains clamped flat at 0.0 V.

2. **Reset Release ($t = 0.1$ s):** The reset signal transitions up to its high logic level, enabling the internal registers to respond to input changes.

3. **Parity Generation and Verification (0.1 s to 1.0 s):**

   - The parallel input bus lines `v(in0)+18`, `v(in1)+24`, `v(in2)+30`, and `v(in3)+36` deliver steady, configured logic states to anchor the data pattern.
   - After the reset releases, the encoder calculates the parity bit over the active input vector. On the following rising clock edge, the output trace `v(parity_bit)` switches cleanly to 5.0 V to reflect the calculated parity value.
   - The receiver checker continuously monitors the incoming data stream. Because the internal signatures perfectly match the expected reference codes, the error notification line remains completely flat at 0.0 V, confirming stable, error-free data verification.

4.8.7 Applications

Synchronous Parity Generator and Checker blocks are key components across various modern digital processing and communication architectures, including:

- **Asynchronous Serial Communication:** Providing baseline error checking for frame sub-packets inside UART transceivers.
- **SRAM and Cache Hardware Protection:** Monitoring memory arrays to catch single-bit corruptions caused by environmental radiation or voltage dips.
- **Internal Parallel System Buses:** Securing wide parallel data buses as they route signals between independent microcontrollers or peripherals.
- **Legacy I/O Peripheral Interfacing:** Assisting in reliable signal validation for hardware peripherals, ensuring data integrity during chip-to-chip transfers.

4.8.8 Summary

A synchronous Parity Generator and Checker hardware IP block was successfully designed, modeled, and verified within the eSim mixed-signal simulation framework. The design uses efficient combinational `XOR` reduction networks to compute and verify parity bits across data streams. Transient simulation results confirmed proper operation: the core remains safely initialized during reset and produces clean, clock-synchronized parity evaluations during runtime without triggering false error flags. This demonstrates the block's readiness for integration into low-overhead, error-resilient digital systems.

Chapter 5

Conclusion

This project successfully designed, modeled, and verified eight distinct digital **Intellectual Property (IP) blocks** using synthesizable **Verilog HDL** within the integrated **eSim** mixed-signal simulation framework. The structural implementations addressed critical modern digital system constraints and operational requirements, spanning clock domain crossing (CDC) safety, embedded testing infrastructure, dynamic power management, communication line coding, stream compression, and front-end signal conditioning.

The developed and finalized IP blocks include:

1. **Two Flip-Flop (2-FF) Synchronizer:** Implemented to isolate asynchronous signals and mitigate metastability across clock domain boundaries.
2. **Built-In Self-Test (BIST) Controller:** Engineered to automate embedded testing patterns via LFSR/MISR registers and successfully identify structural faults (such as Stuck-At-1 physical defects).
3. **Clock Frequency Divider:** Designed with synchronous counter structures to down-scale master clock trees for low-power peripherals.
4. **Glitch Filter:** Configured as a robust multi-stage front-end filter to suppress high-frequency physical line noise and spurious transient spikes.
5. **Power Gating Controller:** Implemented to dynamically manage sleep states and isolate supply rails to drastically mitigate static leakage current.
6. **8b/10b Encoder:** Mapped to guarantee high transition densities and maintain strict DC balance for high-speed serial link physical layers.
7. **Run-Length Encoding (RLE) Compressor:** Structured as an on-the-fly, lossless stream compression engine to minimize data bus redundancy.
8. **Parity Generator and Checker:** Built using low-overhead combinational reduction trees to provide instant error detection against single-bit data corruptions.

The verification workflow provided extensive exposure to structural **RTL design**, multi-stage pipeline alignment, sequential logic arithmetic, error injection analysis, and multi-platform digital verification across eSim, GTKWave, and Xilinx frameworks. Furthermore, the strategic integration of parameterizable **ADC and DAC mixed-signal bridges** enabled seamless validation of discrete digital hardware cores against continuous analog testbench environments.

Overall, the resulting IP blocks demonstrate reliable, robust, and predictable performance, making them highly reusable and ready for integration into larger **Field-Programmable Gate Array (FPGA)** layouts and complex **System-on-Chip (SoC)** application architectures. This portfolio establishes a comprehensive foundation for advanced engineering methodologies in structural **VLSI design** and hardware description automation.