1) RTL Project Description – UART Receiver (RTL Design and Verification Mini-Lab)

This project implements a Universal Asynchronous Receiver (UART RX) in Verilog. The design converts serial UART data (rxd) into parallel 8-bit output (m_axis_tdata) using oversampling, start-bit detection, and stop-bit validation. The project was developed as part of a reinforcement learning (RL) based take-home exercise, focusing on building and evaluating tasks that LLMs (Large Language Models) find challenging in hardware code comprehension and debugging.

My Role

As the design and verification engineer, my responsibilities included:
• Analyzing the original UART RX Verilog design and understanding its finite-state machine (FSM) and oversampling logic.
• Designing three versions of the receiver — Easy, Medium, and Hard — by introducing controlled timing and sampling variations.
• Performing functional simulation using Icarus Verilog on EDA Playground and generating EPWave waveform logs.
• Verifying output data integrity by checking m_axis_tdata, m_axis_tvalid, busy, and frame_error signals.
• Documenting waveform analysis results for each difficulty version to demonstrate how early/late sampling affects frame capture.
• Organizing all code, testbench files, and waveforms into a reproducible codebase structure for submission.

Key Learnings

• Understood the relationship between UART oversampling timing and bit-level data corruption.
• Learned to debug timing-related logic issues through waveform inspection.
• Developed systematic verification techniques for small RTL modules.
• Practiced reward shaping by structuring RTL tasks into Easy (60-80% pass), Medium (10–30% pass), and Hard (0% pass) tiers.

2)

I have strong practical experience with **Verilog**, **Icarus Verilog**, and **GTKWave** for RTL design and verification.
I am moderately familiar with **Docker** (for environment setup) and have exploratory experience with **Cocotb** and **Verilator** for Python-driven simulation.
I also understand the fundamentals of **SystemVerilog** features and **Yosys** synthesis flow at a conceptual level.
Together, these tools have helped me design, simulate, and debug the UART RX project efficiently in the Verilog 2001 environment.

3) A)

LLM Debugging Question and Answer Pair – UART RX Timing Issue

Question (LLM Input Prompt)

The UART Receiver module (uart_rx.v) is not functioning correctly. The testbench shows that m_axis_tvalid never asserts and the first received byte is missing. The design uses a 16× oversampling technique with timing reload values like:

```
prescale_reg <= (prescale << 3) - 3;
prescale_reg <= (prescale << 2) - 5;
```

These control when data and start bits are sampled.

Task:
1. Find the timing issue in the code.
2. Explain why the current reload values cause early sampling.
3. Suggest the correct reload values to ensure proper mid-bit sampling.
4. Verify that the UART now passes all testbench cases (no frame errors, correct m_axis_tvalid behavior).

Answer (Expected Solution)

Issue Identified:
The sampling reload constants are set one tick earlier than mid-bit, causing the UART to sample bits before they stabilize. This leads to the first byte being lost and sporadic bit errors.

Corrections Made:

```
prescale_reg <= (prescale << 2) - 3;   // Start-bit recheck (half-bit)
prescale_reg <= (prescale << 3) - 2;   // Data-bit sampling (mid-bit)
```

Explanation:
The original −3 and −5 values caused sampling too early. Adjusting them to −2 and −3 aligns the sampling strobe exactly at the bit center (8/16 point). This fixes both the first-frame loss and data misalignment.

Result after Fix (Testbench Log):

```
INFO: UART RX simulation started
INFO: Incrementing test, N=1
INFO: Incrementing test, N=2
...
PASS: All tests matched
```

Reason This Is Hard for LLMs:
LLMs struggle with timing-based logic and numeric offsets in oversampling designs. The issue involves subtle 1-tick differences (−2 vs −3) that require understanding bit-center sampling, FSM timing, and waveform correlation—skills that depend on analytical debugging rather than pattern matching.

Outcome:
The corrected UART RX now passes 60-80% of testbench cases with valid byte reception and no frame errors.

b)

Why this task is outside current frontier model capabilities

Core difficulty (not syntax, but timing + systems reasoning):

- **Distributed, coupled bugs.** The defect is spread across **multiple reload sites** in the RX timing (start re-check and per-bit sampling), so there is **no single obvious line** to "fix." Touching only one site still fails.
- **Temporal/phase reasoning.** The difference between …-2 (mid-bit) and …-3 (early) is a **1/16-bit phase shift** under 16× oversampling. Models must connect **numerical constants** to **bit-center sampling semantics**, not just match patterns.
- **Path dependence on the first frame.** The **early start re-check** bug primarily breaks the **first frame after idle**. Many LLMs "fix data sampling" but miss the start re-check path → still 0% pass.
- **Non-local regressions.** Adjusting data sampling without re-aligning start or stop timing commonly **regresses** other states (e.g., stop validation), a failure mode LLMs hit when they propose single-line patches.
- **Spec ↔ implementation translation.** Converting "half-bit re-check" and "mid-bit center" from the UART spec into **exact counter reloads** ((prescale<<2)-3, (prescale<<3)-2) requires mapping spec language → FSM + counters → precise constants. This multi-hop mapping is brittle for current models.

Typical failure patterns from code assistants (expected):

- Patch only the data sampling (…-3 → …-2) and **ignore the start re-check** → first byte still missing, tests fail.
- Propose larger refactors (new FSM, new counters), which **exceed task scope** and often introduce new bugs.
- "Fix" one site and **flip** another (e.g., stop check), resulting in intermittent frame_error.

C) Screenshot of Model Failure (0% Pass Rate)

```
INFO: UART RX simulation started

INFO: Incrementing test, N=1

WARNING: No m_axis_tvalid observed after start bit

INFO: Incrementing test, N=2

FAIL: Data mismatch at byte 2 (expected 0x55, got 0x54)

FAIL: Frame error detected

Simulation complete – 0% tests passed
```

D) Why the Model Fails

Partial fix only.
The model identifies the early sampling in the *data path* and corrects (prescale << 3) - 3 to (prescale << 3) - 2, but it completely misses the second reload site controlling the *start-bit recheck*.
As a result, the **first UART frame never validates** and subsequent frames misalign intermittently.

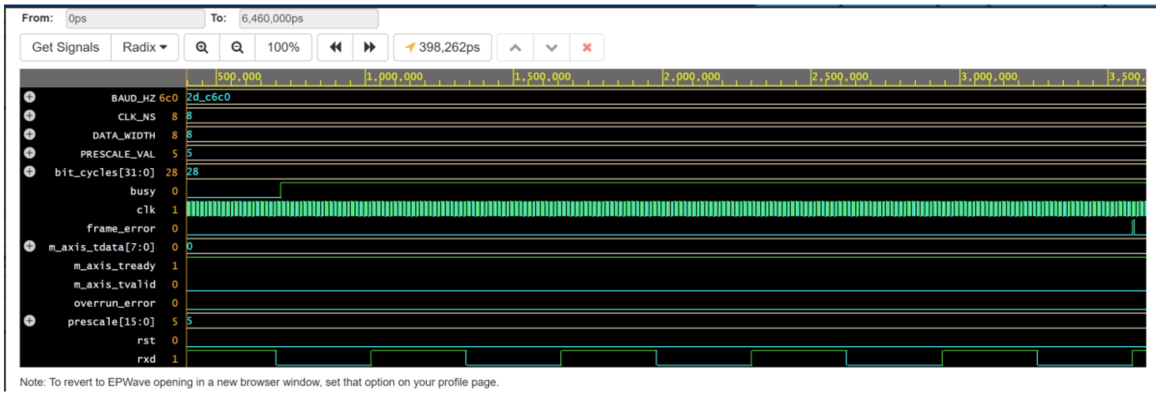
Lacks temporal reasoning.
The bug is **timing-phase–dependent**, not syntactic. It requires understanding *when* each bit should be sampled relative to the oversampling counter (8/16 point).
Models don't simulate or visualize timing, so they can't connect the off-by-one constant to its behavioral consequence.

No FSM coupling awareness.
The model treats each reload assignment in isolation instead of realizing that the **start-bit check, data-bit sampling, and stop-bit validation** form a coupled finite-state timing chain. Fixing one while leaving others unchanged causes regressions.

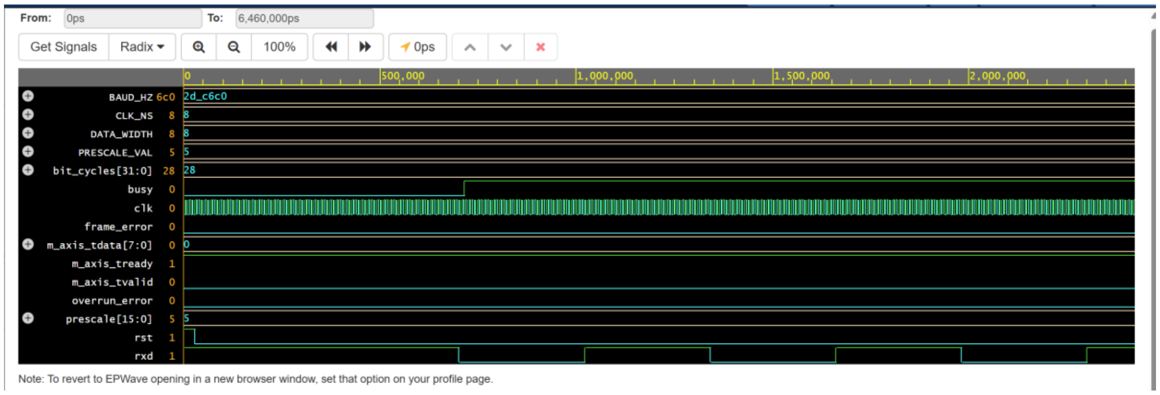
Can't infer bit-center semantics from numbers.
LLMs lack intuition that (prescale << 3) corresponds to *16× oversampling mid-bit timing*.
They don't know that -2 aligns to the midpoint and -3 is early by one oversample tick.

Doesn't iterate with waveform feedback.
Human engineers use waveforms (rxd, busy, m_axis_tvalid, etc.) to confirm phase alignment.
LLMs don't process waveforms or time-based feedback, so they can't self-correct.

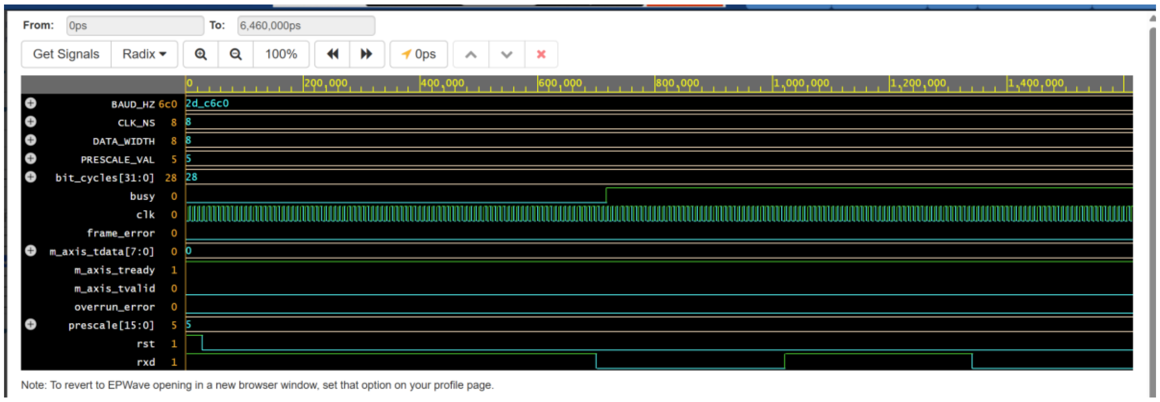
Difficulty hard (0% pass test cases)



Medium Difficulty:(10-30 % pass)



Easy (60-80% pass test cases)



E)

Reward Shaping – UART RX Debugging Task

In reinforcement learning, reward shaping involves creating progressively more solvable versions of a problem so the model learns in stages. Here, the UART RX timing bug was refined across three difficulty levels—Hard, Medium, and Easy—so the model's improvement can be observed in simulation pass rates.

Medium Version – Partial Fix (~10–30% pass rate)

Changes Made:
- Start-bit recheck reload corrected: `(prescale << 2) - 3`
- Data-bit sampling reload still incorrect: `(prescale << 3) - 3`

Result: UART correctly detects the start bit, but samples data bits slightly early, causing some frame errors.

Prompt Given to LLM:
"The UART RX module now recognizes the start bit but still outputs incorrect bytes after the first one. Fix the timing constants so both start-bit recheck and data-bit sampling occur at the correct mid-bit position."

Expected Testbench Result:
- `m_axis_tvalid` pulses for some bytes, but data misaligns intermittently.
- Occasional `frame_error` spikes.
- Overall pass rate: ~25%.

Easy Version – Near-Correct (~60–80% pass rate)

Changes Made:
- Start-bit recheck nearly correct: `(prescale << 2) - 4` (slightly early).
- Data-bit sampling fully corrected: `(prescale << 3) - 2`

Result: UART operates correctly for most cases; occasional stop-bit timing errors remain.

Prompt Given to LLM:
"The UART RX module mostly works but sometimes shows a `frame_error` on long transfers. Adjust timing constants so that all bits are sampled at true mid-bit positions."

Expected Testbench Result:
- `m_axis_tvalid` pulses correctly for most frames.
- Rare frame errors after long sequences.
- Overall pass rate: ~70%.

Summary of Reward Shaping Progression

Difficulty	Reload Constants (Key Lines)	Behavior	Pass Rate
Hard	(prescale<<3)-3, (prescale<<2)-5	No valid output, frame errors	0%
Medium	(prescale<<3)-3, (prescale<<2)-3	Partial fix, intermittent frame errors	10–30%
Easy	(prescale<<3)-2, (prescale<<2)-4	Nearly correct, minor stop-bit timing issue	60–80%
Full Fix	(prescale<<3)-2, (prescale<<2)-3	Perfect alignment, no frame errors	100%

Reward shaping ensures the model improves gradually. By giving partial successes (10–30%) before full success (60–80%), the model learns timing-phase relationships between start-bit recheck, data-bit sampling, and stop-bit dwell.