🧩 TLM, Monitor, Scoreboard & Environment Integration

🎯 Objective

Build a clear understanding of how **data moves inside a UVM testbench** from

sequences → drivers → monitors → TLM connections → scoreboard —
and how all of them fit together inside the environment.

🧠 TLM Basics & Monitor Integration

🧩1️⃣Transaction-Level Modeling (TLM)

🔹 **What is TLM?**

- UVM's communication mechanism for **class-based components**.
- Instead of connecting **signals**, components exchange **transactions (objects)**.

🔹 **Why use TLM?**

- Easier to connect UVM classes (driver, monitor, scoreboard).
- Promotes **modularity** and **reuse** — no signal-level dependencies.

🧱 TLM Port Types

Port Type	Direction	Blocking?	Typical Use Case
`put_port`	→	Blocking	Producer → Consumer
`get_port`	←	Blocking	Consumer → Producer
`peek_port`	←	Blocking	Observe without removal
`analysis_port`	→	Non-Blocking	Broadcast from monitor → scoreboard/coverage

---

⚙️ Common TLM Classes in UVM

Class	Meaning	Used In
`uvm_analysis_port`	Publisher → calls `write()`	Monitor
`uvm_analysis_export`	Middleman bridge	Scoreboard
`uvm_analysis_imp`	Implementation sink	Scoreboard

✅ Direction Rule

`analysis_port` → `analysis_export` → `analysis_imp`

🔹 Virtual Interface vs TLM Connections

Feature	Virtual Interface	TLM Connection
Connects to	DUT signals (pins)	UVM class components
Type	Physical signal-level	Transaction-level
Used By	Driver, Monitor	Monitor, Scoreboard, Coverage

Data Type	Logic bits	Objects (`counter_item`, etc.)

🧩2️⃣Monitor Implementation (Observer in UVM)

💭 Why Monitor is Passive

- It **never drives** signals; only **watches** them.
- Works like a **CCTV camera** observing DUT activity.

🧱 Typical Steps

1. Get virtual interface from config DB.
2. Sample DUT signals every clock.
3. Create a transaction (`counter_item`).
4. Publish it using `ap.write(tr);`.

⚙️ Example

```
class counter_monitor extends uvm_monitor;
  `uvm_component_utils(counter_monitor)
  virtual counter_if vif;
  uvm_analysis_port#(counter_item) ap;

  function new(string name, uvm_component parent);
    super.new(name, parent);
    ap = new("ap", this);
  endfunction

  task run_phase(uvm_phase phase);
    counter_item tr;
    forever begin
      @(posedge vif.clk);
      tr = counter_item::type_id::create("tr");
      tr.reset  = vif.reset;
      tr.enable = vif.enable;
      tr.count  = vif.count;
      ap.write(tr);
    end
  endtask
endclass
```

🧩3️⃣TLM FIFO (`uvm_tlm_analysis_fifo`)

💡 Purpose

A **buffer** between monitor and scoreboard when outputs arrive later than inputs.

⚙️ Example

```
uvm_tlm_analysis_fifo#(counter_item) fifo;
mon.ap.connect(fifo.analysis_export);
fifo.get(sb.tr);
```

🧠 When to Use

Scenario	Recommended Connection
Real-time checks (immediate)	Direct `ap.write()`
Multi-cycle delays / async DUT	`uvm_tlm_analysis_fifo` buffer

🧠 Scoreboard Design & Environment Integration

🧩1️⃣Scoreboard Architecture & Checker Logic

🧭 Role

> Verify that the DUT output matches the expected behavior.

⚙️ Typical Flow

```
Input Monitor  → Scoreboard (expected_q)
Output Monitor → Scoreboard (actual_q)

counter_item expected_q[$];
counter_item actual_q[$];
```

🧩 Compare Logic

```
function void compare_results();
  if (expected_q.size() && actual_q.size()) begin
    counter_item exp = expected_q.pop_front();
    counter_item act = actual_q.pop_front();
    if (exp.count == act.count)
      `uvm_info("SB", $sformatf("MATCH count=%0d", act.count), 
UVM_LOW)
    else
      `uvm_error("SB", $sformatf("MISMATCH exp=%0d act=%0d", 
exp.count, act.count))
  end
endfunction
```

⚙️ Reference Model vs Simple Checker

Type	Description	Example
Simple Checker	Direct compare expected ↔ actual	Counter
Reference Model	Generates expected outputs algorithmically	ALU, FIFO, DSP

🧩2️⃣Integrating Multiple Agents (e.g., UART + FIFO)

🧭 Why

Multi-interface SoCs have more than one data source.
All monitors must feed the same scoreboard through different TLM exports.

⚙️ Connection Example

```
uart_agt.mon.ap.connect(sb.uart_export);
fifo_agt.mon.ap.connect(sb.fifo_export);
```

Each export maps to its own `write(<type>)` method inside the scoreboard.

🧩3️⃣Environment Integration with TLM

🧭 Goal

Combine everything → **sequence** → **driver** → **monitor** → **TLM** → **scoreboard**.

🧱 Hierarchy

```
uvm_test_top
 └── counter_test
       └── counter_env
             ├── counter_agent
             |     ├── counter_driver
             |     ├── counter_monitor
             |     └── counter_sequencer
             └── counter_scoreboard
```

⚙️ Phases Overview

Phase	Purpose	Example
build_phase	Instantiate components	`agt = counter_agent::create("agt", this);`
connect_phase	Wire TLM connections	`agt.mon.ap.connect(sb.analysis_export);`

run_phase	Execute stimulus & time flow	`seq.start(env.agt.seqr);`

🧩 Full Data Flow

```
Sequence → Sequencer → Driver → DUT
                          ↓
                       Monitor
                          ↓
                    Scoreboard (via TLM)
```

✅ Key Takeaways

Concept	Summary
TLM	Class-to-class communication using transactions
Monitor	Passive observer sampling DUT pins
Scoreboard	Compares expected vs actual results
FIFO	Buffers data when DUT output is delayed
Environment	Brings everything together (build → connect → run)

✅ Conclusion:

TLM connects the UVM world. Monitors observe, drivers stimulate, and scoreboards check - all linked by TLM in the environment through build, connect, and run phases.