SAMSUNG

Leveraging LevelDB for Unit-level Replay of Top-level Stimulus in UVM

Nick Jones

Samsung Austin R&D Center

September 10th, 2019

DVClub Europe

SAMSUNG

Agenda

High-Level Overview

Use Case

Evaluating Data Stores

Testbench Changes

Conclusions

Q&A

SAMSUNG

Leveraging DBs for Replay

- Some interactions (aka bugs) only occur at high levels of integration. ☹
- Small unit testbenches can run **orders of magnitude faster** than cluster/SoC testbenches.
- Writing directed tests to model the observed real behavior can take considerable time and effort.

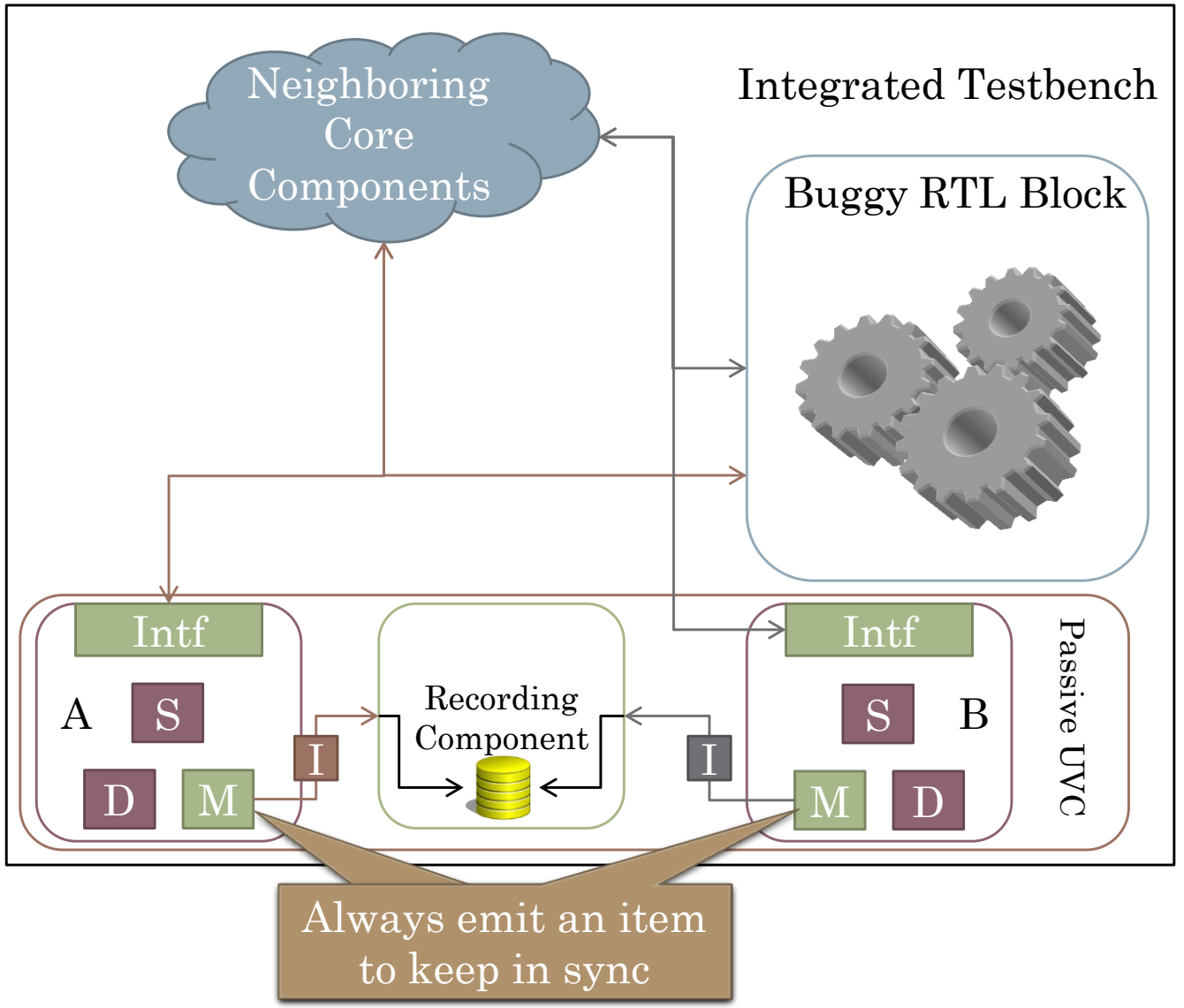
➔ Record the ports and replay it on the unit testbench.

- Bonus: Keep seeded random unit tests as directed tests.

Replay stimulus with 5x better simulation performance!

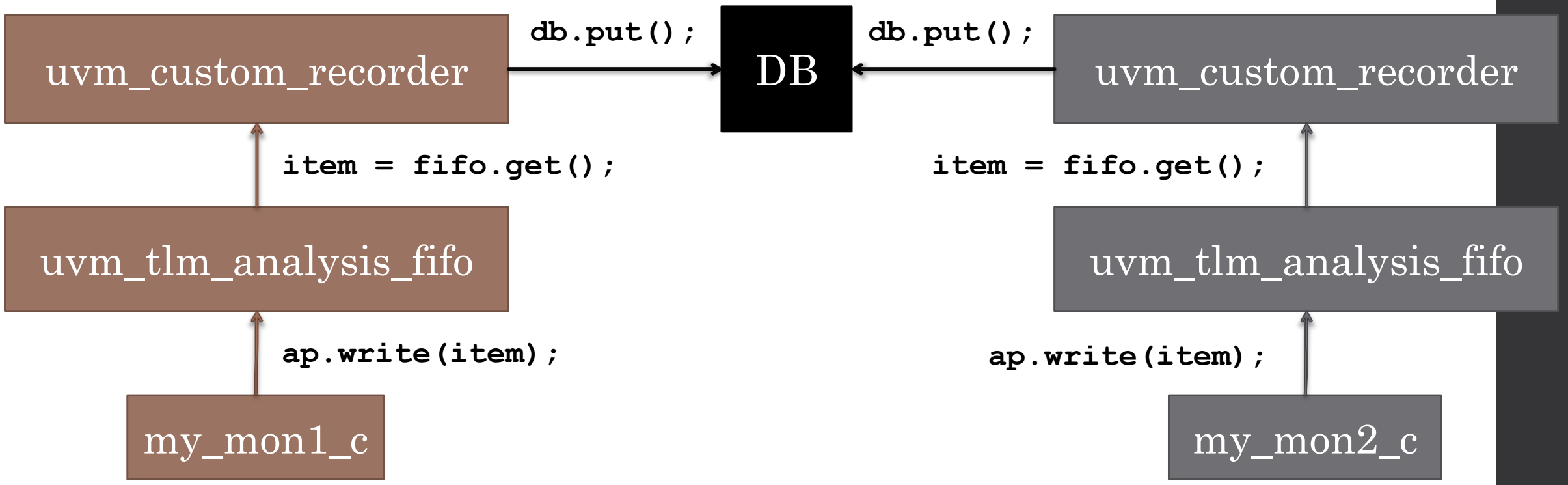
Capturing

- Integrated unit UVC
- All relevant ports are monitored
- Sequence items are emitted **every cycle**
- Collected items leverage the UVM **`pack`** method for storage



SAMSUNG

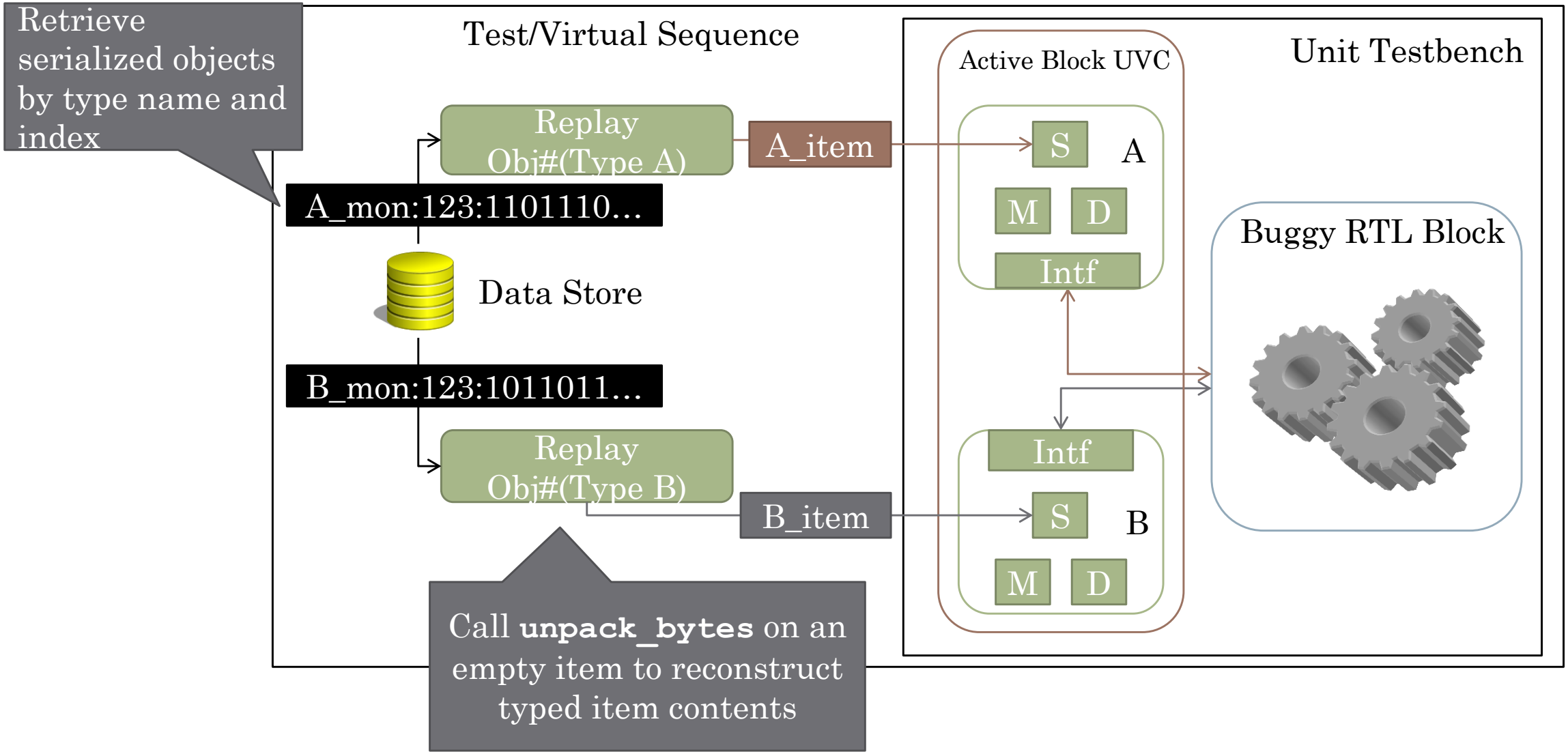
Monitor Connections for Capture



- Replicate FIFOs and recorders per agent
- Support multiple monitors recording to single DB

SAMSUNG

Replay



SAMSUNG

Use Case

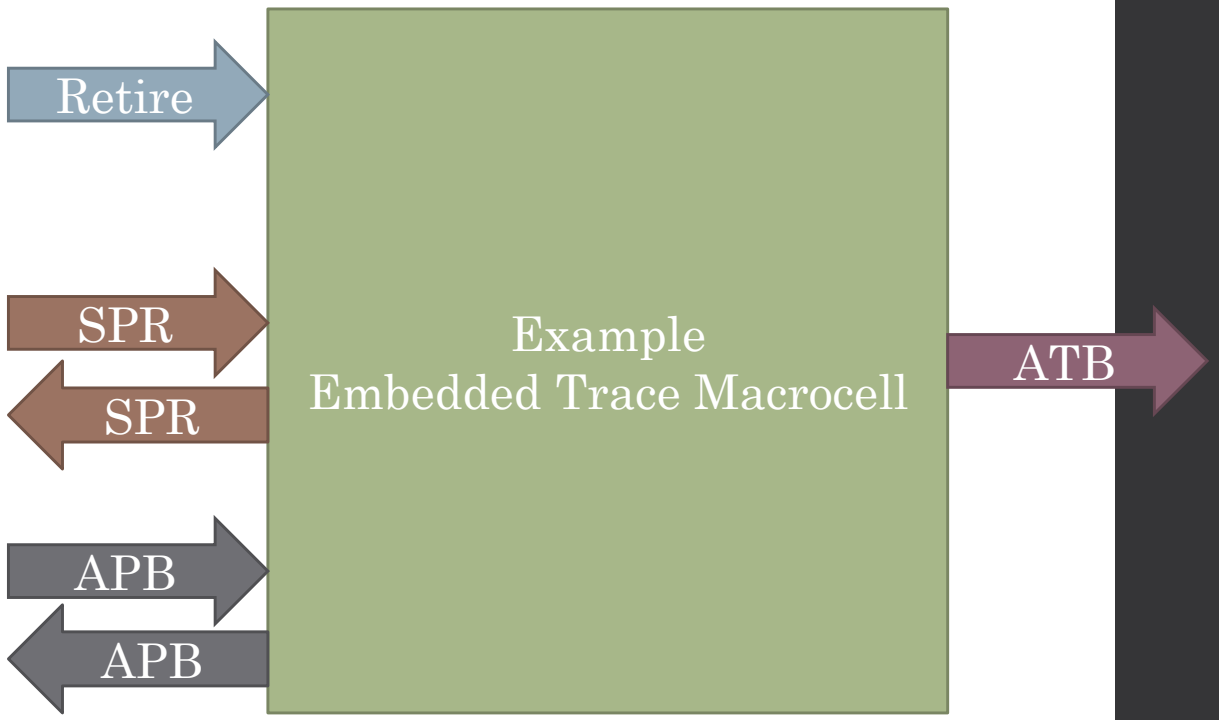
Porting a random assembly ETM failure at cluster

SAMSUNG

Embedded Trace Macrocell

- Goal: Instruction stream represented in trace packets

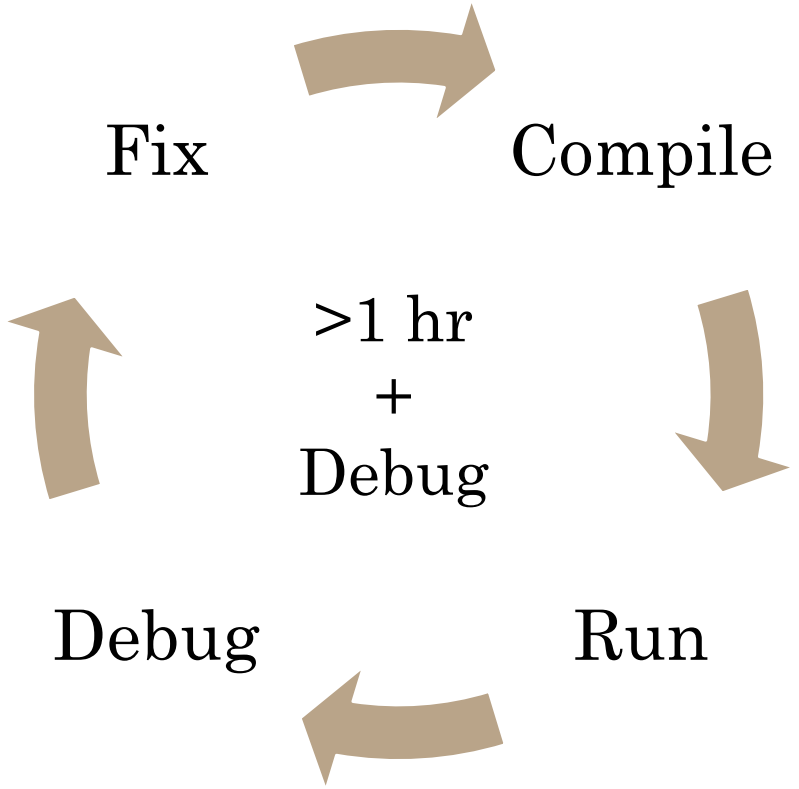
- Interfaces
  - Enabled via register bus(es)
  - Incoming instruction retire
  - Pack and push trace packets on AMBA Trace Bus (ATB)



SAMSUNG

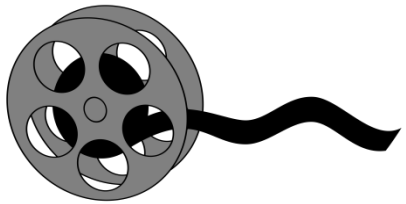
What Broke

- Random generated stream of instructions
- Quirky behavior of instruction execution and
  retirement hit an ETM bug
- Not yet hit at unit with constrained random
- Iterative debug and fixing bugs for one test

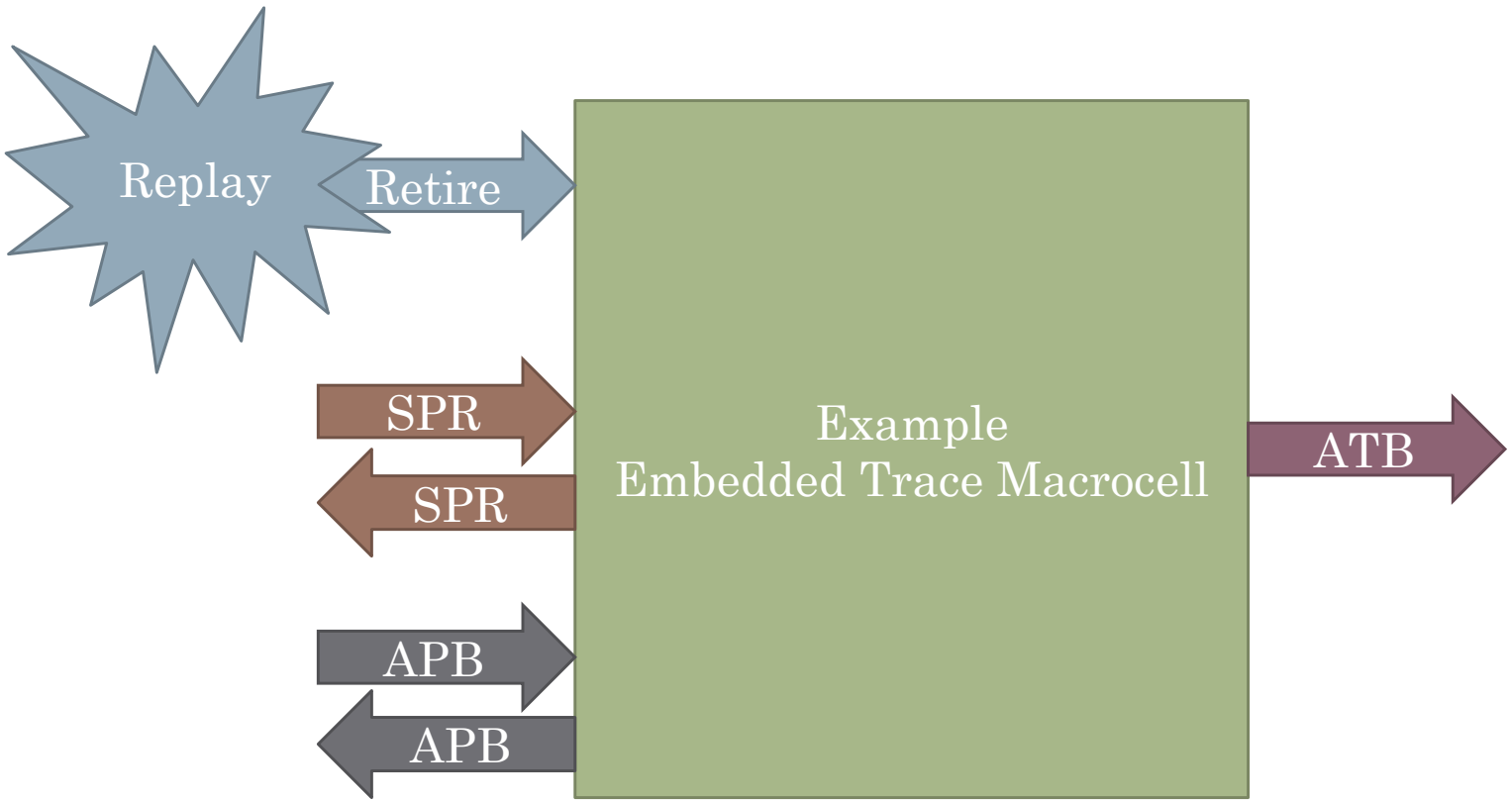


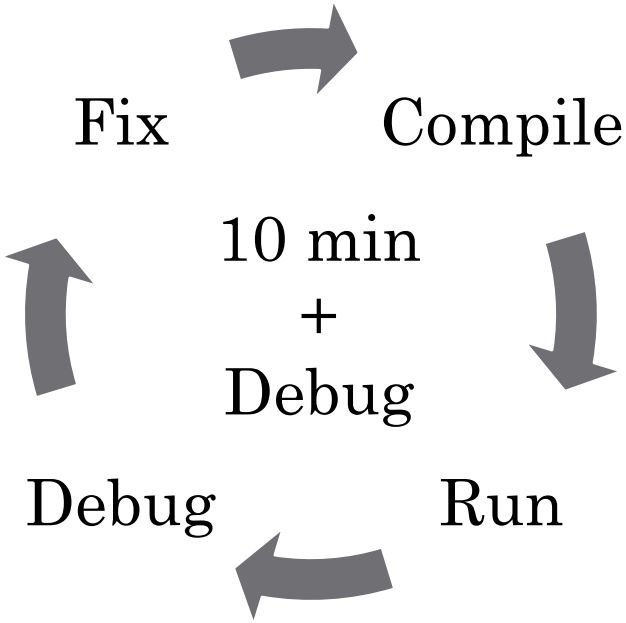
SAMSUNG

Replay



- Reproduced the failures!
- Fixed and confirmed with cluster rerun





SAMSUNG

Persistent Storage at Speed

Choosing and implementing a storage solution

SAMSUNG

Data Store Options

- Decoupled from other simulations
  - Embedded, no networking!
  - Independent and parallel interaction between sims
- Support 32-bit and 64-bit simulator runs
  - ~10% perf uplift in 32b for cluster and unit
  - External library dependencies

SAMSUNG

SQL vs. Key/Value Stores

Storing

- INSERT INTO my_mon_c (‘id’, ‘data’) VALUES(1234, ‘010101’);
  - Table per monitor, fields for records

Key/Value

- Put(“my_mon_c:1234”, “010101”);
  - Compose key of both monitor and index
- Put(“my_mon_c:total”, “120000”);
  - Store total since no COUNT() SQL function

Retrieving

SQL

- SELECT data FROM my_mon_c WHERE id = 1234; => ‘01010…’
- SELECT COUNT(*) FROM my_mon_c; => 120000

- Get(“my_mon_c:total”); => 120000
- Get(“my_mon_c:1234”); => ‘01010…’

SAMSUNG

The Embedded DB Lineup

- Use tables for monitor segregation
- Simple insert/select
- Pure C
- Tiny footprint – Just 500KB

Terrible Performance ☹

- Key/Value API
- Composite keys
- C++
- Very fast get/put operations

Good Performance & Meets Requirements

- Similar Key/Value API as LevelDB
- C++
- Also fast get/put

Lacks 32b Compile & C++11 API

SAMSUNG

Packing Captured Items

- Connect “always” monitored port through a **`uvm_tlm_analysis_fifo`**
- Get, pack to bytes, put
- Decouple packing and data store insertion from monitor write method

```
task run_phase(uvm_phase phase);
    forever begin
        …
        fifo_port.get(item);
        void'(item.pack_bytes(bytestream));
        for(int i = 0; i < bytestream.size(); i++) begin
            $sformatf(
                packed_bytes, "%2h%s",
                bytestream[i], packed_bytes
            );
        end

        db.put(
            $sformatf("%s:%0d", type_name, txn_count),
            packed_bytes
        );
        txn_count++;
    end
endtask : run_phase
```

SAMSUNG

Unpacking Objects

- Retrieve incoming key
- Convert string types
- Recreate dynamic byte array from returned value
- UVM library call to reconstruct object member variables

```
function void read_uvm_obj(string key, ref T obj);

    str_handle = leveldb_get(db, key);

    // Convert void * to const char *, free original
    str_bytes  = leveldb_to_string(str_handle);
    leveldb_free_string(str_handle);

    bytestream = new[str_bytes.len/2];

    for(int i = str_bytes.len-1, j = 0; i > 0; i-=2, j++)
    begin
        substr         = str_bytes.substr(i-1, i);
        bytestream[j] = substr.atohex();
    end

    void'(obj.unpack_bytes(bytestream));
endfunction : read_uvm_obj
```

SAMSUNG

Replaying in a Sequence

- Read total stored objects from a special key written at the end of recording
- Create, retrieve, drive items
- Driver must be in do-as-you-are-told mode – no smarts!

```
task replay_my_unit_seq_items();
    my_unit_seq_item_c my_unit_item;

    for(int i = 0; i < my_unit_item_count; i++) begin
        `uvm_create_on(my_unit_item, p_sequencer.my_seqr)

        start_item(my_unit_item);

        my_unit_db.read_uvm_obj(
            $sformatf("my_unit_mon_c:%0d", i),
            my_unit_item
        );

        finish_item(my_unit_item);
    end
endtask : replay_my_unit_seq_items
```

SAMSUNG

Results

- Reproduced original series of failures!
- 30x performance improvement
- Accuracy
  - Clock-to-clock on single interface
  - Slightly staggered between different interfaces

SAMSUNG

Future Work

- Perfect coordination between interfaces
- Recording specific input agent to reduce recording/replay complexity
- Multiple clock domain environments untested

SAMSUNG

C++ Environments

- SystemVerilog and C++ lack reflection

- Implement custom serialization and de-serialization methods on objects
  - Boost “archive” http://www.boost.org/doc/libs/1_61_0/libs/serialization/doc/tutorial.html

- Put/Get the same way

SAMSUNG

Thank You