Using SystemC in the RTL Design Flow

February 22, 2006

Lawrence Case and Sergey Sokol

freescale™ semiconductor

Agenda

- Objectives
- Benefits
- Design Overview
- Testbench Overview
- Conclusion
- Question

Prioritized Objectives

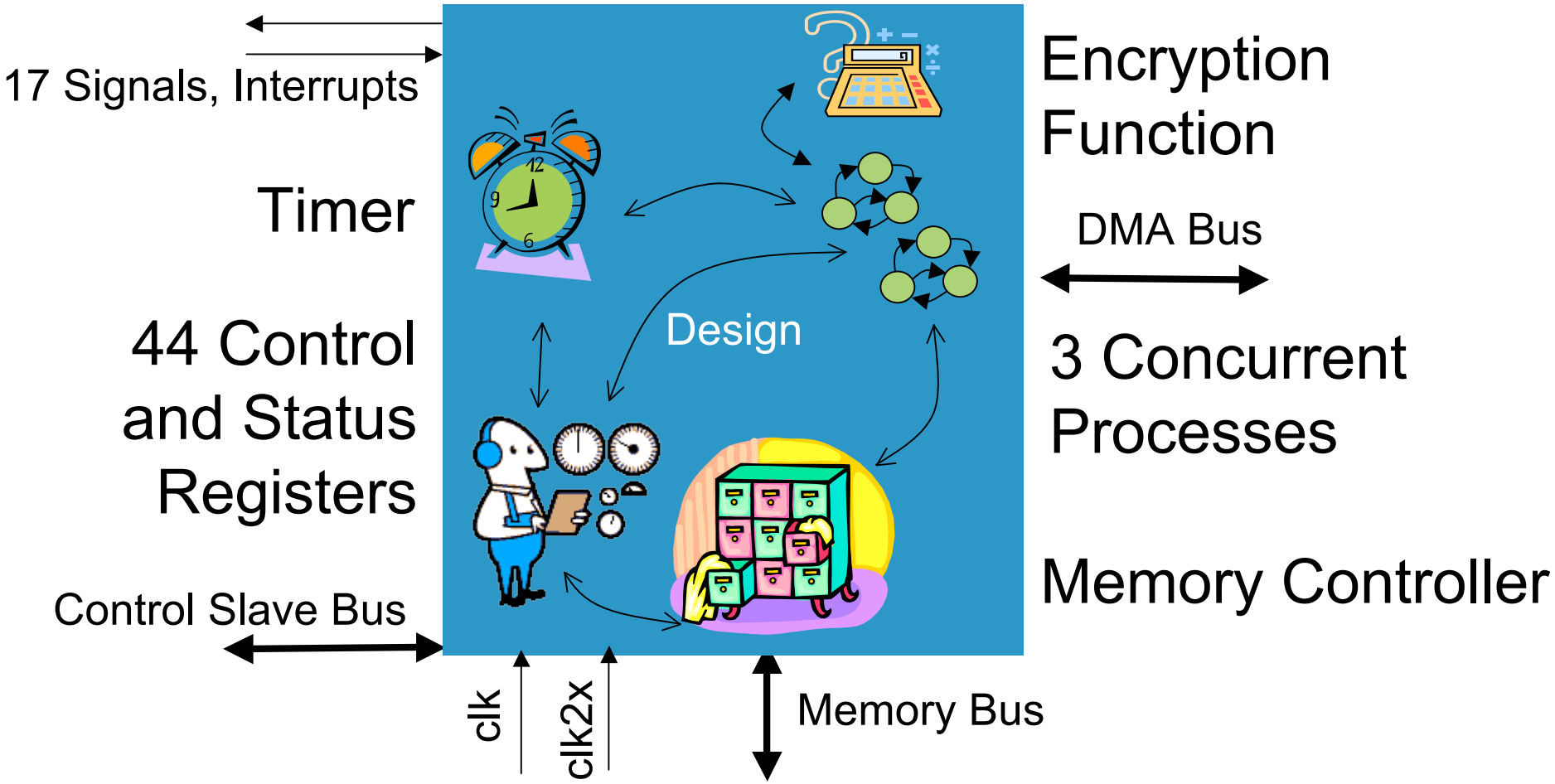
1. Deliver fully verified RTL
2. Deliver model for software driver development
3. Automate stimulus generation.
4. Validate architectural enhancements prior to modifying RTL.

Benefits From Experience

- Constant signal monitoring.
- Accurate Driver Developer's model.
- Golden and Guidance references both help.
- "Golden-izing" model leads to randomizing vectors.
- Strengths of each representation supports the other.
- 3 votes/views (Spec, Model, RTL)
- Model view provides "higher" perspective.

Design Section

Challenging Design Features for Model



Model Requirements

- Bit Accurate Control and Status Registers.
- Correct Sequence of Commands and Status
- Correct Sequence of I/O Signals and Interrupts.
- Data Accurate Functions.
- Expected Responses Generated during Simulation
- Model Integrates into SW and Verification Environments

Model Techniques – Interrupt-able Thread

```
function thread
while (1) {
        wait (function event);
         try (
                setup ();
                 ...
                execute ();
                do wait (delay, event);
                finish ();
        )
        catch failure (
                set failure condition ();
                failure notify;
        )
}
```

```
do wait (delay, event) {
   wait (delay_evt | event);
    if (failure is asserted)
        throw failure event;
}
```

Model Techniques – Reset Thread

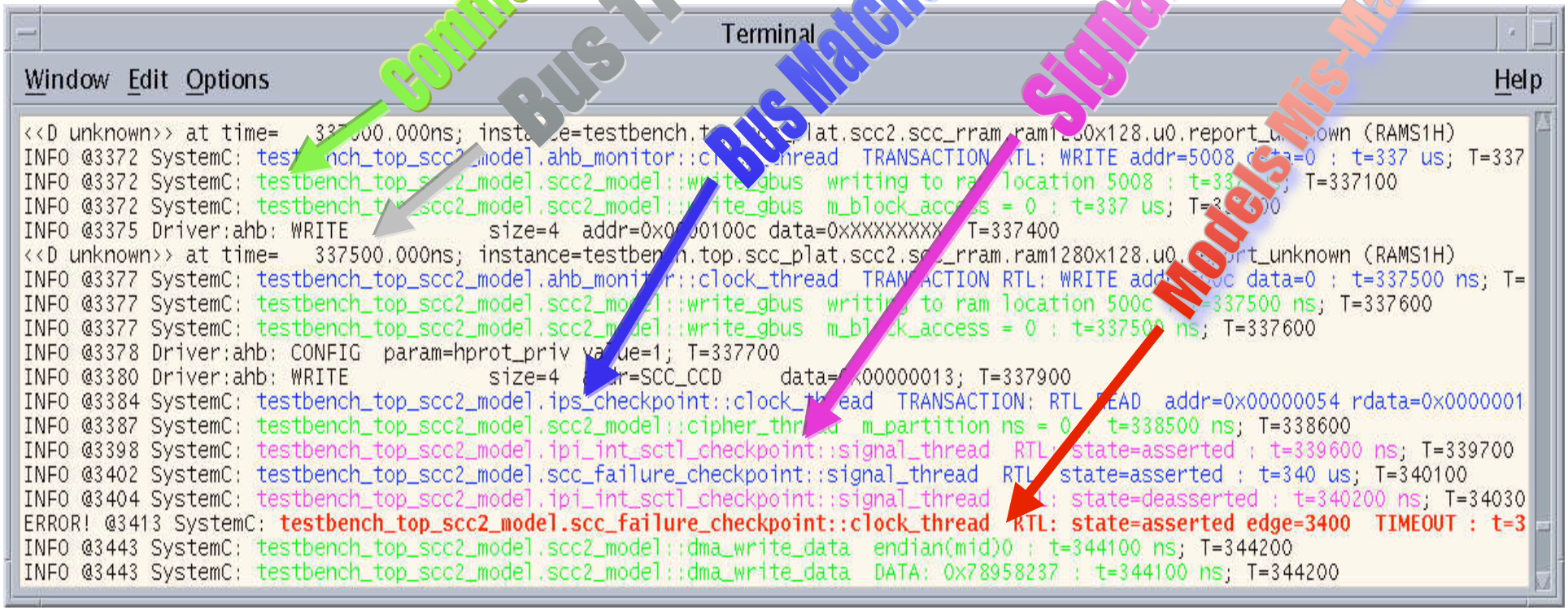
```
reset thread
while (1) {
        wait (reset asserted);
        clear registers ();
        wait (reset deasserted);
        notify startup threads
        ...
}
```

Model Techniques – Timer Status

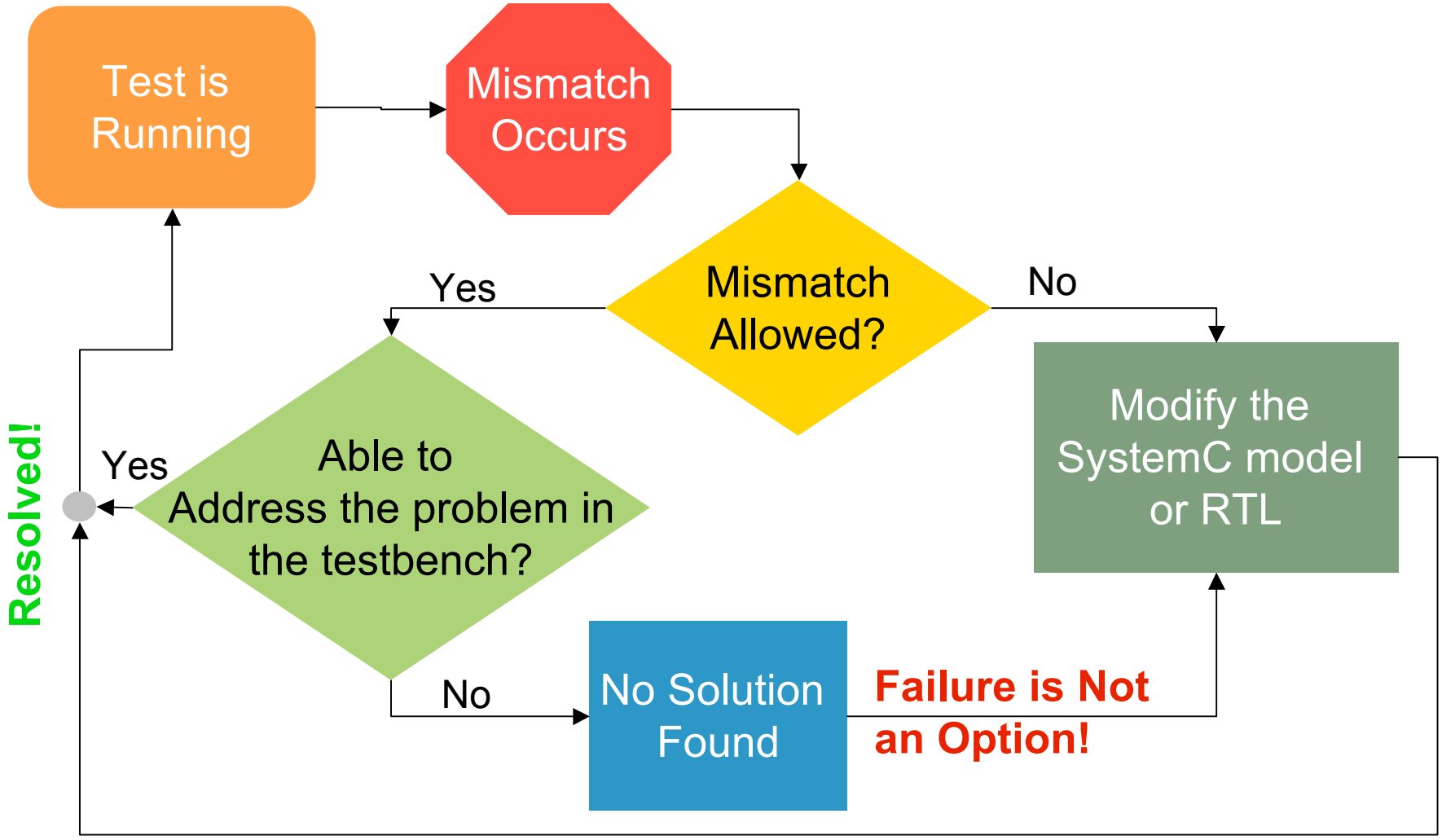
```
case Timer Write
...
previous cycles = clock ->get cycle();
}
```

```
case Timer Read
...
time = clock -> get cycle() - previous cycles;
}
```

Snapshot of Output Log



Mismatch Handling



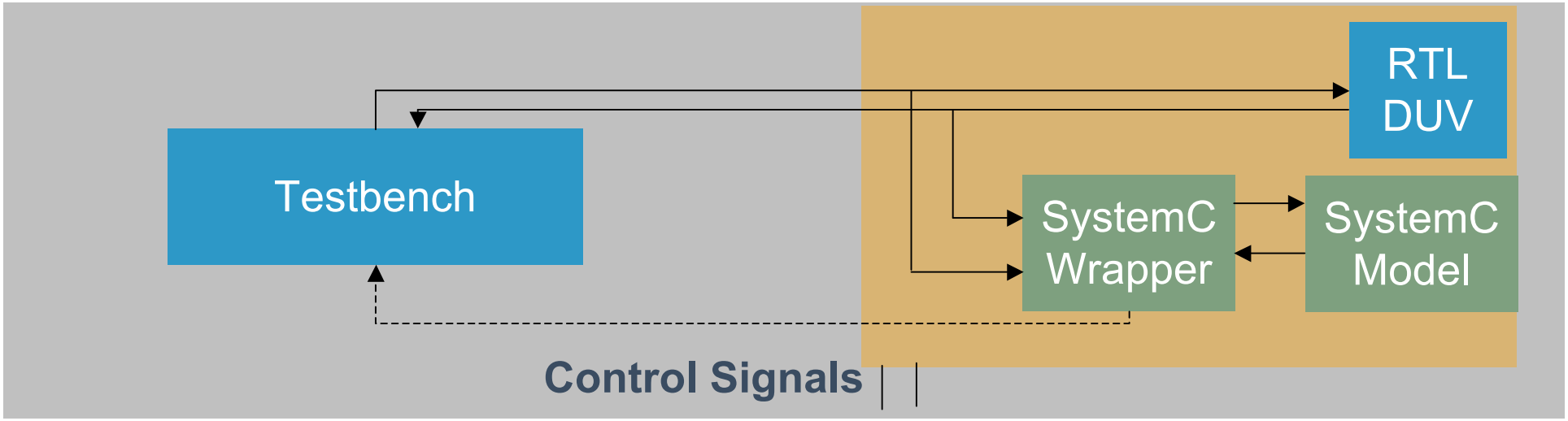
Observations

- 2 model method takes extra communication, TB overhead, and of course, code.
- Golden reference not always model, even when model component designed first.
- At this level, model tended to be buggier than RTL
- Lines: RTL ~10000, Model ~5000 (ex. Encryption)
- Yet, at least one bug found that would not have been found in traditional flow.
- Tens of thousands of randomized vectors passed.

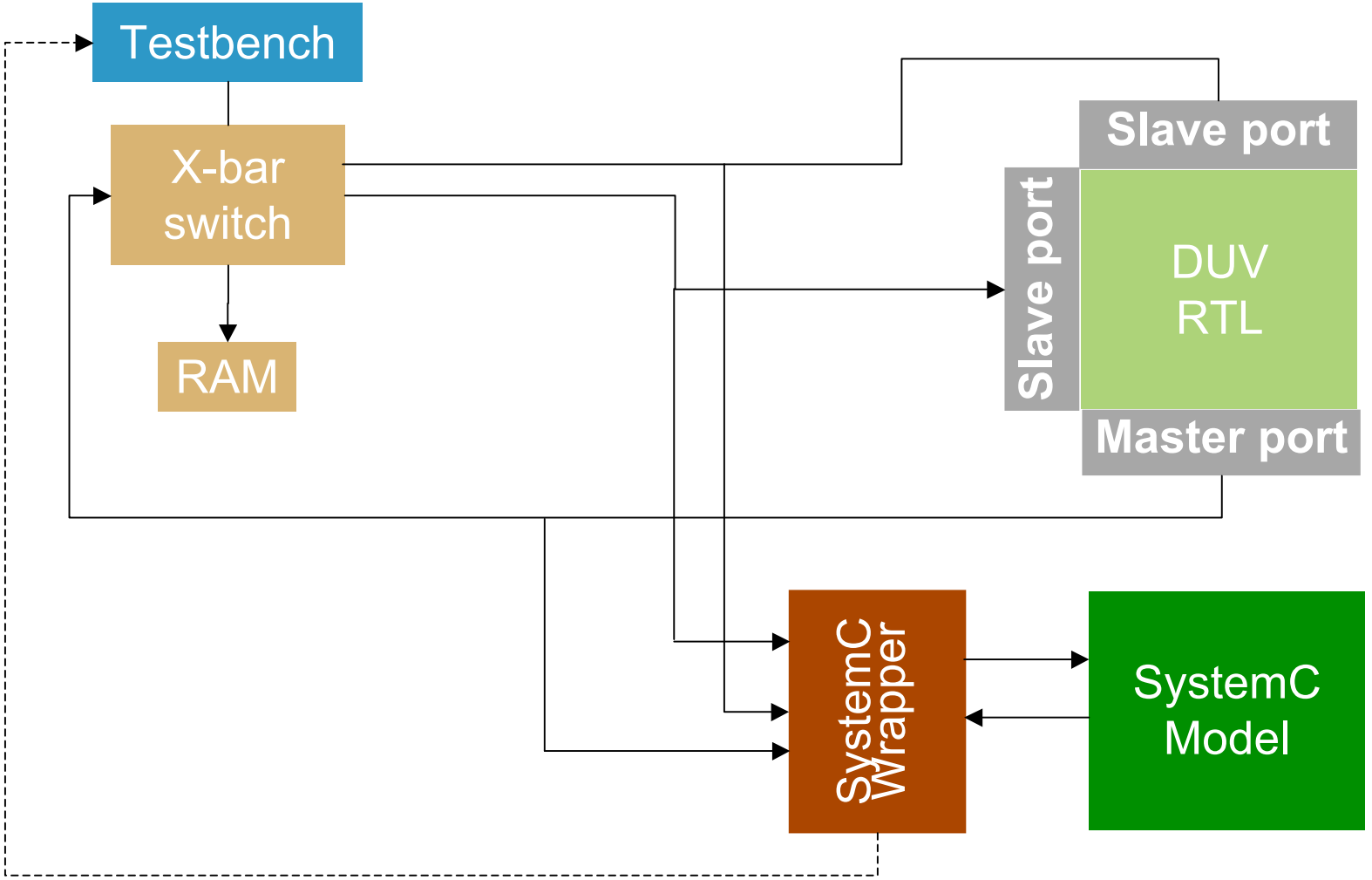
Testbench Section

Dynamic Response Checking

- RTL – developed by designer
- SystemC model ideally developed independently
- Dynamic response checking on interfaces



RTL Platform with SystemC Model

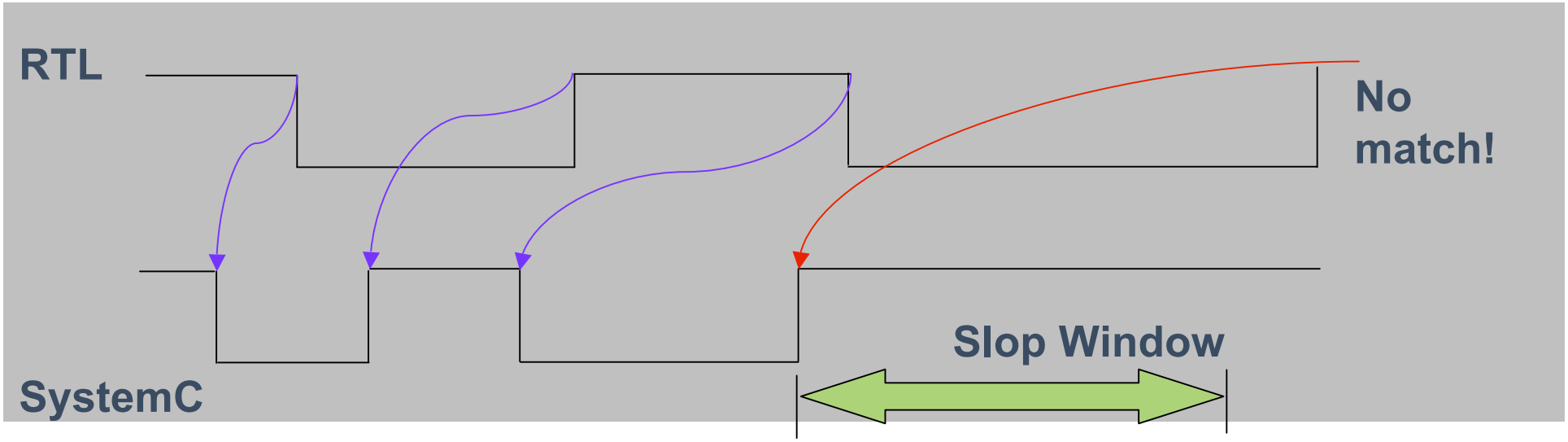


Test Bench and the DUV

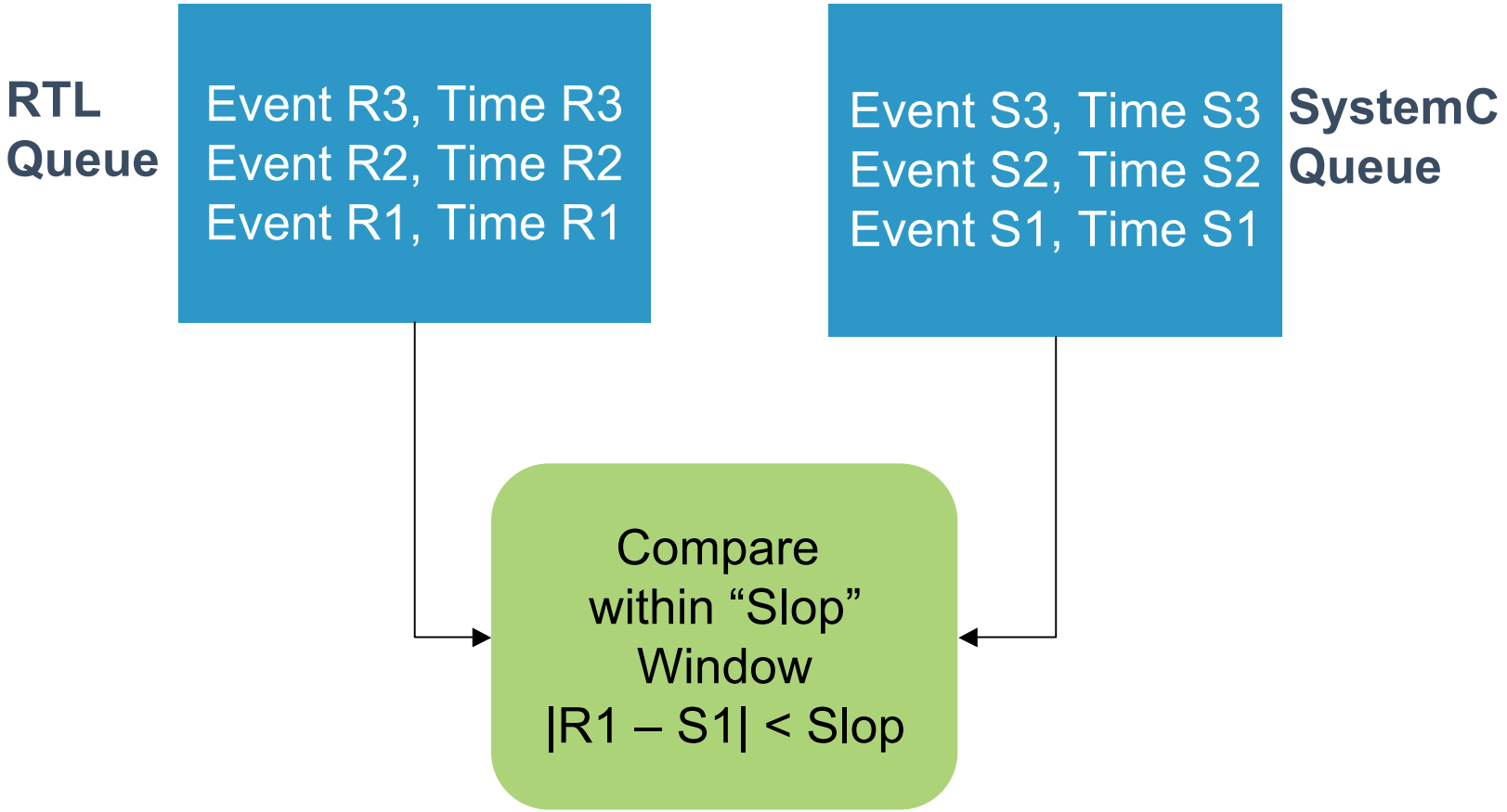
- Testbench provides stimulus to RTL
- SystemC wrapper of design
  - Snoops stimulus and presents it to Model
  - Compares responses with “Slop” window – Coming Up!!
  - Forwards “Matching” information to Test Bench
- Test Bench Logs output and discrepancies

“Slop” Window

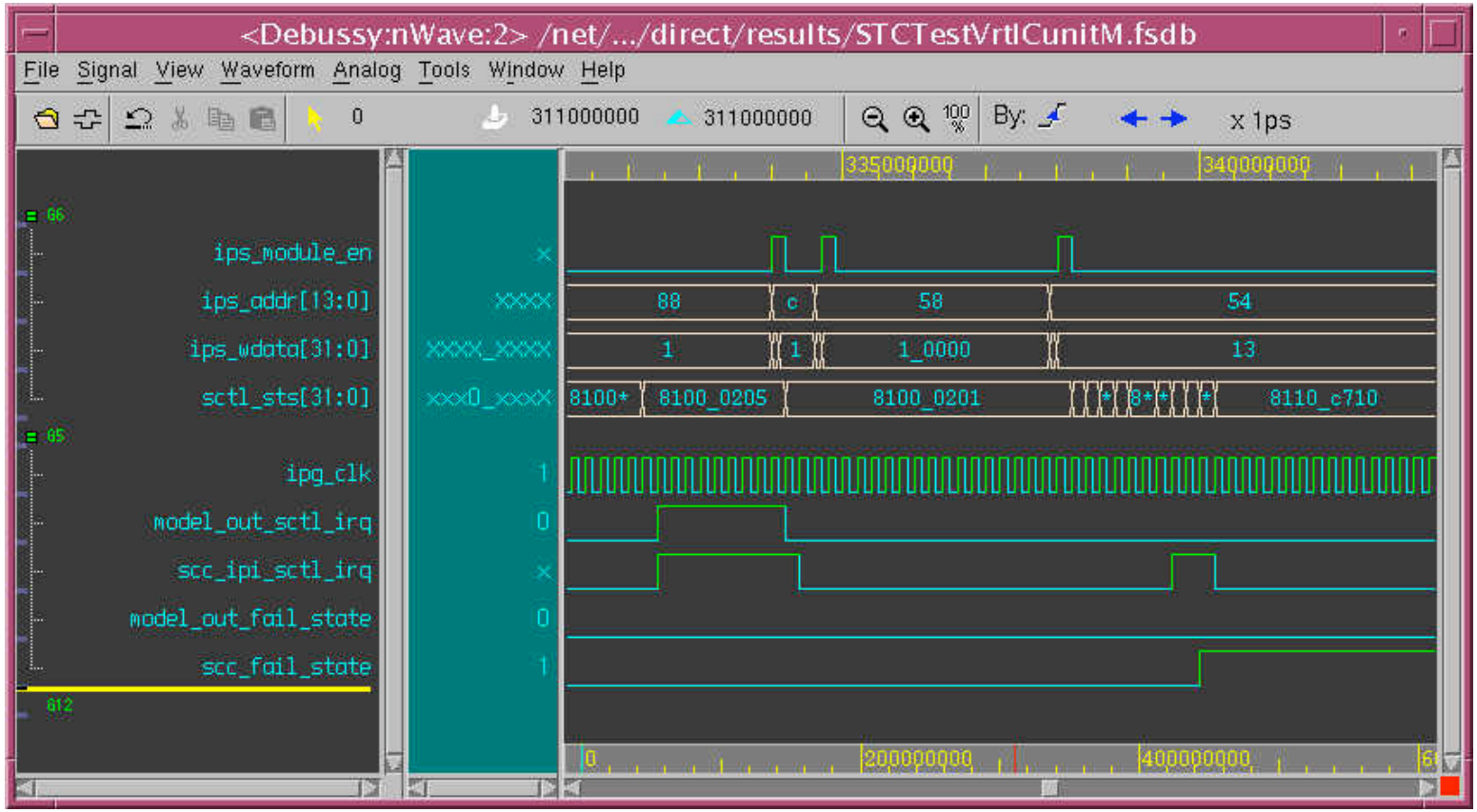
- Why “Slop” Handling?
- Testbench queues response events from RTL and model
- Testbench compares within “Slop” window



“Slop” Window

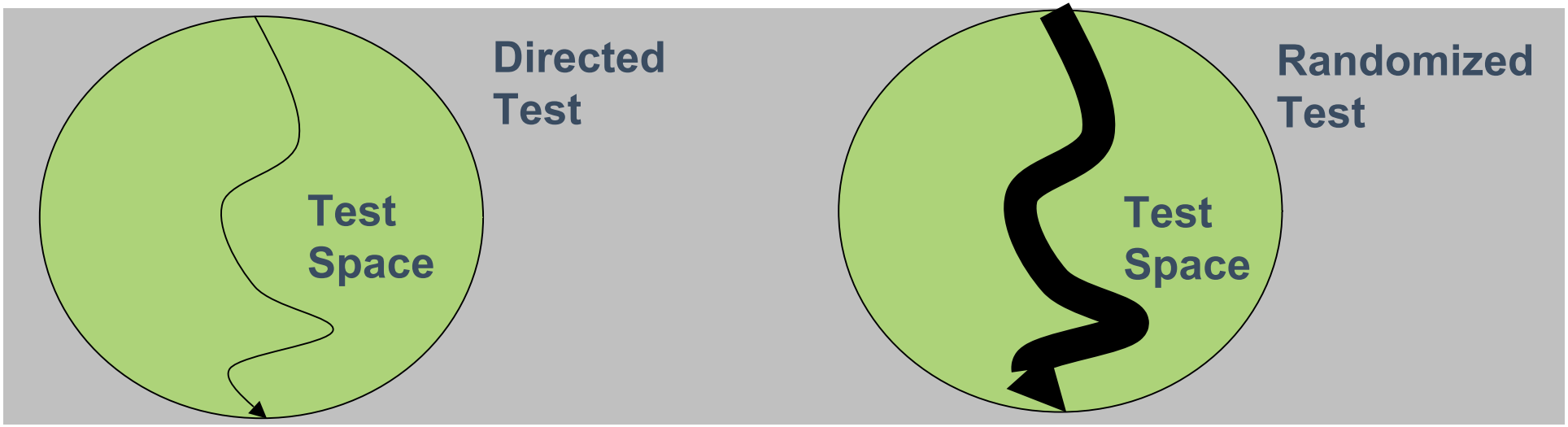


Real World Example 1



Getting Good Coverage

- Develop a set of “Directors” that...
  - Emulate a software driver
  - Randomize over all possible parameters
- Corner cases not covered by Directors



Resolved in Testbench

- Ignoring discrepancies during Reset.
- Matching events within time frame.
- Re-synchronization after missed events.
- Slop Window handling *some* drift issues.

Need more mismatch handling and order-enforcement techniques.

Status

- Testbench is staying the course on current project despite challenges in schedule.
- Plan to continue testbench on next project with new ideas incorporated.