UVM Sequence Item Based Error Injection

Jeff Montesano
Verilab, Canada
Agenda

• Introduction
• When to Implement Error Injection
• Types of Error Injection
• Types of Error Detection
• Planning for Error Injection
• A UVM Error Injection Strategy
  – Proactive Masters
  – Reactive Slaves
• Conclusion and Questions
Introduction

• Generic communications protocol example:

- Errors in Preamble/SOF/Payload/CRC/EOF/
- What exactly should an “EOF” error be?
Introduction

• Examples of EOF Errors

```
PREAMBLE  SOF  PAYLOAD  EOF  PL  CRC  EOF
```

```
PAYLOAD  CRC  EOF  IDLE  EOF  IDLE  PREAMBLE
```

```
PREAMBLE  SOF  PAYLOAD  CRC  EOF w/ error
```
When to Implement Error Injection

• Planning should start early
  – Verification plan
  – Code hooks/comments (e.g. “TBD”)

• Implementation should start later
  – After basic functionality is verified
  – Before comprehensive testing is finished
Types of Error Injection

• Protocol errors
  – Important for interoperability
  – Violate each protocol rule in at least one way

• Physical errors
  – Important for certain designs
  – Only emulate likely operating scenarios
Types of Error Detection

• Stimulus monitors do error detection
  – Publish transactions to scoreboards and coverage

• Classifiable errors
  – Individual error flags in transaction

• Unclassifiable errors
  – “Aggregate” error flag in transaction
  – Stimulus coverage is appropriate
Planning for Error Injection

• For the approach outlined in this presentation:

1. Sequence item modifications
   - add error fields
2. Sequence modifications
   - add error sequence
3. Driver modifications
   - inject the errors
4. Transaction modifications
   - flag the errors
5. Monitor Modifications
   - detect the errors
A UVM Error Injection Strategy
Proactive Masters

• What is a proactive master?
Planning for Error Injection

• For the approach outlined in this presentation:

1. Sequence item modifications
   - add error fields

2. Sequence modifications
   - add error sequence

3. Driver modifications
   - inject the errors

4. Transaction modifications
   - add error field

5. Monitor modifications
   - detect the errors
A UVM Error Injection Strategy
Proactive Masters

• Error injection fields added to sequence items

```cpp
class protocol_seq_item extends uvm_seq_item;
    bit [7:0] payload[];
    bit [15:0] crc;
    ...
    rand bit transmission_err;
    rand bit latency_err;
    rand bit crc_err;
    rand bit eof_err;
    ...
    constraint c_errors {
        transmission_err == 0;
        latency_err == 0;
        crc_err == 0;
        eof_err == 0;
    }
    ...
endclass
```
Planning for Error Injection

• For the approach outlined in this presentation:

  1. Sequence item modifications
     - add error fields
  2. Sequence modifications
     - add error sequence
  3. Driver modifications
     - inject the errors
  4. Transaction modifications
     - add error field
  5. Monitor modifications
     - detect the errors
A UVM Error Injection Strategy
Proactive Masters

class reg_seq extends uvm_sequence;
    bit [7:0] ss_payload[];
    ...
endclass

class error_seq extends uvm_sequence;
    bit [7:0] ss_payload[];
    ... 
    bit ss_crc_err;
    bit ss_eof_err;
    ...
`uvm_create(s_item)
    if (!randomize(s_item)) `uvm_warning
        ("", "")
        s_item.crc_err  = ss_crc_err;
        s_item.eof_err  = ss_eof_err;
`uvm_send(s_item)
endclass
Planning for Error Injection

• For the approach outlined in this presentation:

  1. Sequence item modifications
     - add error fields
  2. Sequence library modifications
     - add error sequence
  3. Driver modifications
     - inject the errors
  4. Transaction modifications
     - add error field
  5. Monitor modifications
     - detect the errors
A UVM Error Injection Strategy
Proactive Masters

• Driver reacts to error flags

```verbatim
class protocol_driver extends uvm_driver;

    task run();
        forever begin
            seq_item_port.get(s_item);
            drive(s_item);
        end
    endtask

    task drive();
        ...
        if (s_item.transmission_err) begin
            // flip a payload bit
            ...
        end
    endtask

endclass
```
Planning for Error Injection

For the approach outlined in this presentation:

1. Sequence item modifications
   - add error fields
2. Sequence modifications
   - add error sequence
3. Driver modifications
   - inject the errors
4. Transaction modifications
   - add error field
5. Monitor modifications
   - detect the errors
A UVM Error Injection Strategy
Proactive Master

- Aggregated error field added to transaction

```verilog
class protocol_trans extends uvm_transaction;
  bit [7:0] payload[];
  bit [15:0] crc;
  ...
  bit error_detected;
  ...
}
endclass
```
Planning for Error Injection

• For the approach outlined in this presentation:

1. Sequence item modifications
   - add error fields
2. Sequence modifications
   - add error sequence
3. Driver modifications
   - inject the errors
4. Transaction modifications
   - add error field
5. Monitor modifications
   - detect the errors
A UVM Error Injection Strategy
Proactive Masters

• Monitor detects errors and publishes flag

class protocol_monitor extends uvm_monitor;
    analysis_port #(protocol_trans) ap;
    protocol_trans trans;

    task run();
        trans = new();
        bit eof_err, crc_err;

        if (eof_err | crc_err) `uvm_warning("", "detected error");

        trans.payload = observed_payload;
        trans.crc = observed_crc;
        trans.error_detected = eof_err | crc_err;
        ap.write(trans)

        endtask
endclass
A UVM Error Injection Strategy
Reactive Slaves

• What is a reactive slave?
A UVM Error Injection Strategy
Reactive Slaves

- Error injection counters added to configuration

```verilog
class agent_config extends uvm_object;
  ...
  int latency_err_cnt;
  int eof_err_cnt;
  ...
endclass
```
A UVM Error Injection Strategy

Reactive Slaves

• Error sequence added to virtual sequencer to increment counters

```verbatim
class incr_err_seq extends uvm_sequence;
  ...
  bit incr_latency_err;
  bit incr_eof_err;
  ...
  task body();
    if (incr_latency_err)
      p_sequencer.cfg.latency_err_cnt++;
    if (incr_eof_err)
      p_sequencer.cfg.eof_err_cnt++;
  ...
  endtask
  ...
endclass
```
A UVM Error Injection Strategy
Reactive Slaves

• What is a reactive slave?
A UVM Error Injection Strategy
Reactive Slaves

- Error sequence added to agent sequencer

```gherkin
class agent_err_seq extends uvm_sequence;

...  
  task body();
  forever begin
  ...
    `uvm_create(s_item)
    if (!randomize(s_item)) `uvm_warning("","")

    if (p_sequencer.cfg.latency_err_cnt > 0) begin
      s_item.latency_err = 1;
      p_sequencer.cfg.latency_err_cnt--;
    end

    ...  
    `uvm_send(s_item)
  ...
```

Jeff Montesano
A UVM Error Injection Strategy
Reactive Slaves

• Base test overrides default sequence

```verilog
class error_test extends uvm_test;
    ...
    task build();
    ...
    set_config_string("agent.sequencer","default_sequence",
    "agent_err_seq");
    ...
    endtask
    ...
endclass
```
Conclusion and Questions

• Error injection
  • Opens up meaningful discussion
  • Stresses the design
  • Is a necessary part of any environment

• A sequence item approach
  • Straightforward to implement and understand
  • Applies to both proactive masters and reactive slaves