Error Injection in a Subsystem Level Constrained Random UVM Testbench

Jeremy Ridgeway and Hoe Nguyen
Broadcom, Ltd.
Agenda

• Introduction

• Selective Error Report Demotion

• Managing Error Injection

• Expecting Errors

• Responding to Interrupts
Constrained Random UVM Testbench
Constrained Random UVM Testbench

UVC
  RAND
  Sequence
  Sequencer

Driver
  PKT

Monitor
  PKT

DUT
  PKT

Scoreboard
  PKT

FAIL
Constrained Random UVM Testbench
Reporting Layer: Functional Coverage

Diagram showing the flow of information:

- UVC
- RAND
- Sequence
- Sequencer
- Driver
- PKT
- Monitor
- DUT
- PKT
- OK
- fcov
- fco
Random Scenario Cross Coverage

\[ \text{fcov} \times \text{fcov} \times \text{fcov} \times \text{fcov} = \text{OK} \]

- Important for verification status
Random Scenario Cross Coverage

\[ f_{cov} \times f_{cov} \times f_{cov} \times f_{cov} = \text{FAIL} \]

- Equally important
Random Scenario WITH Error Injection

UVC

RAND

Sequence

PKT

Sequencer

PKT

PKT

Monitor

DUT

PKT

fcoy

INJECT ERROR

OK

PKT Monitor

fcoy
Agenda

• Introduction

• Selective Error Report Demotion

• Managing Error Injection

• Expecting Errors

• Responding to Interrupts
Reporting Layers: Test PASS/FAIL

UVC
RAND

Sequence
Sequencer

Driver
Monitor

DUT

Monitor

OK

ALL OK = PASS
Reporting Layers: Test PASS/FAIL

Need to Ensure Error Reports are accurate

JUST 1 ERROR = FAIL
Error Report Message Demotion

- Focus on the Report Type and Report ID
- Catch all report messages
- Demote known Report Error Report ID messages to WARNING
UVM Report Catcher Extension

• Error Demoter maintains an array keyed by Report Message ID
• Demote when caught report message matches and:
  – Always, Within (absolute/relative) time window, Count

![Diagram showing the structure of UVM callbacks and report catcher classes](image)

```plaintext
function new(...);
  super.new(...);
  uvm_report_cb::add(.obj(null),
    .cb(this), ordering(UVM_PREPEND);
endfunction
```

```plaintext
uvm_callback

uvm_report_catcher

err_demoter_basic

- err_dem_req m_tbl[string]
+ void error_demote(ID, ...)  
+ void error_undemote(ID)
+ int get_cmdline_demotes()
+ virtual action_e catch()
```

```plaintext
```
UVM Report Catcher Flow

- Error demotion becomes warning when expected
- Error demotion becomes info when managed by error injection services
Agenda

• Introduction

• Selective Error Report Demotion

• Managing Error Injection

• Expecting Errors

• Responding to Interrupts
Error Injection Services (EIS)

• Single object to manage everything*
  – Inject the error
  – Expect the error
  – Recover from the error – reset?

• May be instantiated any time == uvm_object

• Randomly selected and started

*As with most functional verification, there are no absolutes.
Error Injection Services (EIS) Object

- Contain common functionality in base class
- Extend for specific error injection type
- Randomly select and start error injections

```
my_eis
+ task start_injection(uvm_phase)
eis_base
+ virtual task start_injection(uvm_phase)
test_base
- eis_base m_eis_tbl[$]
uvm_object

```

Works for homegrown injection
VIP Callbacks == inheritance is difficult

<table>
<thead>
<tr>
<th>eis_base</th>
<th>vip_callback_class</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ virtual task start_injection(uvm_phase)</td>
<td>+ void pkt_pre_xmt(ref pkt p)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>my_eis</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ task start_injection(uvm_phase)</td>
</tr>
</tbody>
</table>

- C++-style multiple inheritance is not supported in SystemVerilog

interface classes | opaque type-parameter
Opaque Type Parameter Extension with an Interface Class

- Opaque type param extension won’t solve the problem of maintaining a table of EIS objects:

<table>
<thead>
<tr>
<th>test_base</th>
</tr>
</thead>
<tbody>
<tr>
<td>- eis_pif m_eis_tbl[$]</td>
</tr>
</tbody>
</table>

```plaintext
interface class eis_pif
+ pure virtual task start_injection(…)
```

```plaintext
eis_base#(T) extends T implements eis_pif
+ virtual task start_injection(…)
```

```plaintext
my_eis
+ task start_injection(uvm_phase)
```

```plaintext
eis_base#(T) extends T implements eis_pif
```
Managing Error Injection: Common API

- Start injection – task taking `uvm_phase` allows for blocking if necessary
- Instrumentation
  - Set expected errors
  - Set expected interrupts
- Response via callback access
  - Error report ID is active
  - Contextual Interrupt is active
- Infrastructure to handle connections

### eis_base#(T)

- `virtual task start_injection(phase)`
- `virtual void set_expect_error(ID, ...)`
- `virtual int error_active(ID, ...)`
- `virtual void set_expect_interrupt(...)`
- `virtual task interrupt_active(...)`

### my_eis

- `task start_injection(...)`
Agenda

• Introduction

• Selective Error Report Demotion

• Managing Error Injection

• Expecting Errors

• Responding to Interrupts
Errors Expected during Error Injection

- Virtual sequencer:
  - Acts as intermediary between active EIS objects and demoter
  - Allows EIS object to start a virtual sequence

Expecting 1 or 2 EIS objects to be active at one time
Expected Error Caught

- Expected errors are not warnings – they are demoted to INFO
Demoted Error Caught

- Unexpected but demoted errors are still warnings

```vhdl
'UVM_ERROR("MYERR"")
```
Unexpected Error Reported

• Unexpected errors are still errors

```
`uvm_error("MYERR"...)
```

```
catch()
```

```
if(m_vseqr.error_active("MYERR") == 0)
```

```
if(m_tbl.exists("MYERR"))
```

```
foreach(m_tbl["MYERR"] [pif])
  result |= pif.error_active("MYERR",...)
```

```
UVM_ERROR test_env.sv(46) @ 0.0ns: uvm_test_top.ENV [MYERR] Error seen.
```
Agenda

• Introduction

• Selective Error Report Demotion

• Managing Error Injection

• Expecting Errors

• Responding to Interrupts
Error Injected at Driver

Probably not going fail in this manner
Error Injection Cleanup

- Need to clean-up the scoreboard
- Need to handle the interrupt
### Example Register Description

<table>
<thead>
<tr>
<th>Register</th>
<th>Field</th>
<th>[msb:lsb]</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>top_int</td>
<td>rsvd</td>
<td>[31:2]</td>
<td>0</td>
<td>RO</td>
</tr>
<tr>
<td></td>
<td>PXPATH</td>
<td>[1]</td>
<td>0</td>
<td>W1C</td>
</tr>
<tr>
<td></td>
<td>RXPKT</td>
<td>[0]</td>
<td>0</td>
<td>RO</td>
</tr>
<tr>
<td>pkterr</td>
<td>rsvd</td>
<td>[31:1]</td>
<td>0</td>
<td>RO</td>
</tr>
<tr>
<td></td>
<td>CRC</td>
<td>[0]</td>
<td>0</td>
<td>W1C</td>
</tr>
</tbody>
</table>

- RXPATH interrupt is autonomous and handled directly
- RXPKT interrupt is dependent on pkterr.CRC register field
Interrupt Service Routine

- At interrupt:
  - Monitor starts virtual sequence on virtual sequencer

Diagrame description:
- DUT
- Reactive UVC
  - Monitor
  - VSequencer
  - VSequence
ISR Vseqr and Register Vseq

- Single top-level vseq exists per interrupt
- isr_reg_vseq table contains this register’s field names
- vseqr table is full string register path
Example ISR: RXPKT=1/pkterr.CRC=1

**Diagram:***

- **DUT**
- **Monitor**
- **VSequencer**
  - `m_tbl [string]`
  - `top_int.RXPKT fld_vseq`
  - `pkterr reg_vseq`
  - `pkterr.CRC fld_vseq`

**Table:**

<table>
<thead>
<tr>
<th>Register</th>
<th>Field</th>
<th>[msb:lsb]</th>
<th>Value</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>top_int</code></td>
<td>RXPKT</td>
<td>[0]</td>
<td>1</td>
<td>RO</td>
</tr>
<tr>
<td><code>pkterr</code></td>
<td>CRC</td>
<td>[0]</td>
<td>1</td>
<td>W1C</td>
</tr>
</tbody>
</table>

**Legend:** Handles and returns
Example ISR: RXPKT=1/pkterr.CRC=1

- **DUT**

  - **Monitor**

  - **VSequencer**
    - `m_tbl [string]`

  - **top_int reg_vseq**
    - `top_int.RXPKT fld_vseq`
    - `pkterr reg_vseq`
    - `pkterr.CRC eis_vseq`

- **eis_base#(T)**
  - Virtual task `start_injection(phase)`
  - Virtual void `set_expect_interrupt(...)`
  - Virtual task `interrupt_active(...)`

  - **my_eis**

  - Calls back EIS object
  - Handles interrupt
  - Cleans up
Summary

• Random Error Injection Selection

• Manage Error Injection

• Manage Expected Errors

• Manage and Respond to Interrupts

```
my_eis
+ task start_injection(…)
  ...
```

```
eis_base#(T)
+ virtual task start_injection(phase)
+ virtual void set_expect_error(ID, …)
+ virtual int  error_active(ID, …)
+ virtual void set_expect_interrupt(…)
+ virtual task interrupt_active(…)
```
THANK YOU!

Questions Please