Getting Started with Requirements Based Verification

DAC2008

Dr. David Robinson
david.robinson@verilab.com
Today’s Topics

- Overview of the Requirements Based Verification Flow

- Requirements gathering
  - Brainstorming faults and failures

- Risk analysis and prioritisation
“Verification is a big job. Treat it like one. Sweeping it under the rug won't make it go away”
### What are we Trying to Solve?

<table>
<thead>
<tr>
<th>Category</th>
<th>Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Costs</strong></td>
<td>Out of control, fire fighting, and cancellation</td>
</tr>
<tr>
<td><strong>Quality Costs</strong></td>
<td>Low quality and unverified designs shipped</td>
</tr>
<tr>
<td><strong>Personnel Costs</strong></td>
<td>High stress and high staff turnover</td>
</tr>
</tbody>
</table>

*Just because you don’t know about something doesn’t mean it doesn’t exist. You will find out at some point!*  

*Whether or not you will have the time to deal with it is a different matter.*
The Planning Phase

1. Work out what you *would* verify given infinite resources
The Planning Phase

1. Work out what you **would** verify given infinite resources

2. Decide what you **will** verify
The Planning Phase

1. Work out what you **would** verify given infinite resources

2. Decide what you **will** verify

3. Decide **how** you will verify it, and what’s involved in that

---

**Verification requirements**

**Prioritised verification requirements**

**Testbench requirements**
The Planning Phase

1. Work out what you would verify given infinite resources

2. Decide what you will verify

3. Decide how you will verify it, and what’s involved in that

4. Estimate how long it will take, and what resources you need
The Execution Phase

Unsatisfied Requirement: One which hasn’t been fully verified
Satisfied Requirement: One which has been fully verified
The Payoffs

**Better Schedule**
Starts accurately, and stays there due to less rework

**Better Quality**
Early risk mitigation, and detailed verification

**Better Visibility**
Results (not effort), remaining work and risk

*It’s better to get bad news early than late*
“People talk about requirements gathering as if they’re just lying around, waiting for you to pick them up”
Example Verification Requirement

**bus-if.read.1**

**Check:** Check that we can only read from the registers when the slave is actually selected.

**Conditions:** htrans in [NONSEQ, SEQ, BUSY] x hsize in [8, 16, 32] x hburst in [SINGLE, INCR] x hsel in [0, 1];

**Importance:** 1

**Risk:** 3

**Status:** Reviewed

**%Covered:** 10%
Finding Verification Requirements

1. Understand the design
2. Work out how could it fail
3. Work out why could it fail
4. Write the requirements

In great detail!
Finding Verification Requirements

1. Understand the design

2. Work out **how** could it fail

3. Work out **why** could it fail

4. Write the requirements

In great detail!
Operation: Something that the design is meant to do
Operation Outputs

**Operation** : Something that the design is meant to do

**Result** : Something produced by an operation

- Transmit Data Frame
  - A data frame
**Operation Outputs**

**Operation**
- Transmit Data Frame
  - A data frame
    - Start bit
    - Data
    - Parity
    - Stop bits

**Result**
- **Operation**
  - Something that the design is meant to do
- **Result**
  - Something produced by an operation
- **Field**
  - An identifiable part of a result

---

Copyright © 2008 Verilab Ltd. v1.5
**Operation Outputs**

- **Operation**: Something that the design is meant to do
- **Result**: Something produced by an operation
- **Field**: An identifiable part of a result
- **Rules**: How the data is stored, encoded or interpreted in a field

---

**Transmit Data Frame**

- **A data frame**
  - **Start bit**
  - **Data**
  - **Parity**
  - **Stop bits**
    - **Presence**
    - **Even/Odd**
    - **# bits**
    - **Direction**
    - **# bits**
What can the inputs be?
Finding Verification Requirements

1. Understand the design
2. Work out how could it fail
3. Work out why could it fail

In great detail!

4. Write the requirements

It’s not enough to show the design works. You have to show that it doesn’t fail.
Failure Modes and Effects Analysis (FMEA)

For each operation

- How could it fail?
- Why could it fail?
- Do we care?

Brainstorm
FMEA Output

What would a failure look like?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Failure Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit Data</td>
<td>Parity value</td>
</tr>
<tr>
<td>Frame</td>
<td>incorrect</td>
</tr>
</tbody>
</table>
### FMEA Output

#### What could cause it to fail?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Failure Mode</th>
<th>Failure Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit Data Frame</td>
<td>Parity value incorrect</td>
<td>Parity calculated over entire frame and not just data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type configuration register misinterpreted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type misunderstood</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stick parity not implemented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parallel parity calculation &amp; dataBits changes between frames (e.g. 8 to 5).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The now unused bits corrupt the new value</td>
</tr>
</tbody>
</table>
### FMEA Output

#### What’s the risk of it failing?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Failure Mode</th>
<th>Failure Cause</th>
<th>Risk Exposure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit Data Frame</td>
<td>Parity value incorrect</td>
<td>Parity calculated over entire frame and not just data</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type configuration register misinterpreted</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type misunderstood</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stick parity not implemented</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parallel parity calculation &amp; dataBits changes between frames (e.g. 8 to 5). The now unused bits corrupt the new value</td>
<td>4</td>
</tr>
</tbody>
</table>
# Ignore low risk items

<table>
<thead>
<tr>
<th>Operation</th>
<th>Failure Mode</th>
<th>Failure Cause</th>
<th>Risk Exposure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit Data Frame</td>
<td>Parity value incorrect</td>
<td>Parity calculated over entire frame and not just data</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type configuration register misinterpreted</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type misunderstood</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stick parity not implemented</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parallel parity calculation &amp; dataBits changes between frames (e.g. 8 to 5).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The now unused bits corrupt the new value</td>
<td>4</td>
</tr>
</tbody>
</table>
Finding Verification Requirements

1. Understand the design
2. Work out how could it fail
3. Work out why could it fail
4. Write the requirements

In great detail!
From FMEA to Requirements

- Add one requirement to check that the operation works

- Add one requirement per failure mode
  - if not covered above

- Add a new requirements for any failure-mode/failure-cause combinations that requires special attention
## Verification Requirements

<table>
<thead>
<tr>
<th>Req</th>
<th>Check</th>
<th>Conditions</th>
<th>Risk Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx.frame.1</td>
<td>Check that the UART transmits a data frame correctly: start → data → [parity] → stop</td>
<td>( \text{dataBits} = [5, 6, 7, 8] \times \text{data} = [0..255] \times \text{parity} = [\text{OFF, ON}] \times \text{pType} = [\text{ODD, EVEN}] \times \text{stopBits} = [1, 1.5, 2]; )</td>
<td>1</td>
</tr>
<tr>
<td>tx.frame.2</td>
<td>Check that previous data values do not affect the parity calculation</td>
<td>( \text{parity} = [\text{ON}] \times \text{pType} = [\text{ODD, EVEN}] \times \text{data} = [0..255]^* \times (\text{dataBits} = [N] \rightarrow \text{dataBits} = [M (&lt;N)]); )</td>
<td>4</td>
</tr>
</tbody>
</table>

* Data must be chosen carefully so that legacy unused bits would affect new calculation
“Brainstorming is easier if you know what you’re looking for”
Brainstorming Failure Modes

- Know what a “correct” result looks like, and consider all deviations from it

- 4 failure classes to get you started:
  1. Unscheduled execution
  2. Failure to start when required
  3. Failure to stop when required
  4. Failure during execution
Temporal Failures

Consider how the field can vary temporally within the result.
Value Failures

Address 0xF000 accessed. reg_sel should be 0001

reg_sel field must be one hot when active

Invalid Value

0000 1000 1000 0000

Illegal Value

0000 1010 1010 0000

Consider how the value of the field can fail
Data = -5, encoded as 2’s complement

- 1011
- 1010 1’s complement
- 1101 Sign and magnitude

Consider how the field can be incorrectly encoded
What if your testbench is accidentally tolerant to these failures?

- consider an illegal value on reg_sel

1. Uses “if” statements with reg_sel
2. 0011 treated as xxx1
3. No failure detected

1. Uses “case” statements with reg_sel
2. 0011 causes deadlock (infinite wait states)
3. Customer isn’t very happy
Too much Detail?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Report errors on incoming packets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Req</strong></td>
<td><strong>Check</strong></td>
</tr>
<tr>
<td>rx.error.1</td>
<td>Check that packet_error is asserted when an error is detected in an incoming packet</td>
</tr>
</tbody>
</table>

```
packet  | 1FF4 | D173 | ERROR | AB00 |
```

```
packet_error | 1 | 1 | 1 | 1 |
```

1. *packet_error* wrong almost all of the time
2. *No checker in place to catch that failure mode*
3. *No failure detected*
Use “5 Whys” for root cause analysis

It might help to:

- understand the design’s implementation
- consider external failures:
  - workflow management
  - change control
  - revision control
  - scripts
  - spreadsheets
“Not all requirements are created equal. Why waste time on the unimportant ones?”
Implementation Priority

- Implementation Priority lets the team specify the order in which to verify requirements
  - useful for phased releases
  - uncovers the hidden value of each requirement

1. Verify this in release 1
2. Verify this in release 2
N. Verify this in release N
Risk Exposure

Risk Exposure = likelihood of failure \times impact of failure

Likely (1) \rightarrow Not likely (5)

Unacceptable (1) \rightarrow Acceptable (5)

1

Extremely risky – lots of effort needed

25

Not worth any effort at all
What’s the likelihood of:
- the operation being wrong?
- the inputs being wrong?
- the operation being used?
Likelihood of a Failure Occurring

How complex is the design for this operation?

<table>
<thead>
<tr>
<th>Major Mode</th>
<th>Very Complex/Repetitive</th>
<th>Moderately Complex</th>
<th>Trivial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Minor Mode</th>
<th>Very Complex/Repetitive</th>
<th>Moderately Complex</th>
<th>Trivial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Where does the stimuli for this operation come from?

- External (non-user)
- External (user)
- Internal
Impact of a Failure Occurring

What is the worst that will happen?

- Something will be damaged
- The chip won’t work
- A major mode won’t work
- A minor mode won’t work

Is there a workaround?

- No
- User behaviour
- A hardware one
- A software one
where next?
Requirement Based Verification

Verification requirements

Prioritised verification requirements

Testbench requirements

Schedule and Allocation

Traceability

 Unsatisfied requirements

 Satisfied requirements

Execution

Time

Planning

→ + Other requirement management processes

DUT

SB

Copyright © 2008 Verilab Ltd.

v1.5
The End

- david.robinson@verilab.com