Introduction to Requirements Based Verification

ClubT 2008

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

- Overview of Requirements Based Verification
- Process 1: Requirements gathering
  - Brainstorming faults and failures
- Process 2: Risk based prioritisation
“Hope” is not a strategy
<table>
<thead>
<tr>
<th>Processes</th>
<th>Terminology</th>
<th>Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement Extraction</td>
<td>Operations</td>
<td>FMEA</td>
</tr>
<tr>
<td>Prioritisation</td>
<td>Faults and Failures</td>
<td>5 Whys</td>
</tr>
<tr>
<td>Risk Management</td>
<td>Failure Meta-classes</td>
<td>FTA</td>
</tr>
<tr>
<td>Traceability</td>
<td>Features</td>
<td>Planning Poker</td>
</tr>
<tr>
<td>Workflow</td>
<td>Soap Opera Use Cases</td>
<td>Path &amp; Event Analysis</td>
</tr>
<tr>
<td>Reporting</td>
<td>Equivalence Classes</td>
<td>Complexity Points</td>
</tr>
<tr>
<td>Status Tracking</td>
<td>Wildcard Requirements</td>
<td>100 Points</td>
</tr>
</tbody>
</table>
Problems Caused by Unstructured Approaches

**Project Problems**
- Out of control, fire fighting, and cancellation

**Quality Problems**
- Low quality and unverified designs shipped

**Personnel Problems**
- High stress and high staff turnover

“A cancelled project has an overhead of 100 percent”

McConnell, IEEE Computer, May 1998

http://www.stevemcconnell.com/articles/art09.htm
Focussed on the Human Problems

“What can we verify?”

“What should we verify?”

“How do we decide what’s important”

“How do we decide what to do next?”

“How do we ‘brainstorm verifiable features’?”

“Have we identified the risks?”

“Where are we really?”
Help Answer the Following Questions

- “What schedule and resources do we need to verify this?”
  - alt. “can we verify this with the time and resources we have”

- “What’s the best way to organise the workflow?”

- “What would we deliver if we taped out today?”
Something we are required to verify before we are “done”

A verification requirement is
- something we want to check
- plus the conditions we want to check it under
- plus some administrative information
**Example Verification Requirement**

### bus-if.read.1

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

**Conditions**: \( \text{htrans in [NONSEQ, SEQ, BUSY]} \times \text{hsize in [8, 16, 32]} \times \text{hburst in [SINGLE, INCR]} \times \text{h.sel in [0, 1]} \);

**Importance**: 1

**Risk**: 3

**Status**: Unsatisfied

**%Satisfied**: 10%

**%Implemented**: 100%
Unsatisfied Requirement: One which hasn’t been fully verified or discarded
Satisfied Requirement: One which has been fully verified or discarded
Requirements Help with Schedule

How long will it take to do all of these?

Unsatisfied requirements

What’s the average time to satisfy a requirement?

Having something that’s “countable” makes estimation easier.
Calibration is also possible
Requirements Help with Decision Making

Do we need more **time** or **people** to do all of these?

We can’t be finished if we have unsatisfied requirements.

What **would** (or might not) **work** if we finished today?
Requirements Help Organise Workflow

Organise workflow around **completely** satisfying requirements

- **Satisfied requirements**
- **Functional Coverage Results**

- **Good**
- **Not so Good**

Time
People talk about requirements gathering as if they’re just lying around, waiting for you to pick them up
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!
Specifications are much bigger

Hard to understand

Multiple sources
- Functional Requirements
- Industry Protocols
- Marketing Material
- Compliance
- Algorithms
- Constraints
- Architecture

Hard to write

Modifiers
- Delta specifications
- Corrigendums (list of errors)
- Change Requests
- Versions
Designs are much Bigger

Specifications are much smaller(!)

Information not readily available

Doesn’t exist

Distributed formats

Emails
Sticky notes
Issue tracker entries
People’s brains
RTL comments
Personal logbooks
etc

Too much hassle to even write
Which Means

- Easy to miss information, or for information to be missing
- Increased chance of bugs
- Increased chance of missing bugs
- Some sort of framework for navigating the information would be helpful

Note: Only time for the basic concepts today
**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

---

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

**Result**: Something produced by an operation

**Field**: An identifiable part of a result
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
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
4. Write the requirements

In great detail!

It's not enough to show the design works. You have to show that it doesn't fail.
A failure occurs when the design’s behaviour deviates from the expected, or specified behaviour
- what you see when a simulation fails

Failures are not bugs!
- just the visible symptoms of bugs

We’re not trying to find and fix failures
- we’re trying to find and fix the cause of those failures
- they are called faults (or defects or bugs)
Failure Modes and Effects Analysis (FMEA)

For each operation

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

Brainstorm
Output of an FMEA Session

What would a failure look like?

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

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>
## Output of an FMEA Session

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>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type configuration register misinterpreted</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type misunderstood</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stick parity not implemented</td>
<td>5</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>21</td>
</tr>
</tbody>
</table>
Output of an FMEA Session

**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>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type configuration register misinterpreted</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type misunderstood</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stick parity not implemented</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parallel parity calculation &amp; dataBits changes between frames (e.g. 8 to 5).</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The now unused bits corrupt the new value</td>
<td></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!
Requirements to Show the Operation Works

**Check**

Check that packet_error is asserted when an error is detected in an incoming packet.

**Conditions**

- packet_type = [A, B, C]
- packet_error = [TRUE, FALSE]

---

**Packet**

<table>
<thead>
<tr>
<th>Packet</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

**packet_error (expected)**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>packet_error (expected)</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

**packet_error (actual)**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>packet_error (actual)</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Targeting this failure.
Requirements to Show the Operation Doesn’t Fail

Check
Check that packet_error is deasserted when no error is detected in an incoming packet, and when no detection is taking place

Conditions
- packet_type = [A, B, C]
- packet_error = [TRUE, FALSE]
Requirements to Show the Operation Doesn’t Fail

**Check**

Check that packet_error is deasserted when no error is detected in an incoming packet, and when no detection is taking place.

**Conditions**

- `packet_type = [A, B, C]`
- `packet_error = [TRUE, FALSE]`

**This was a real bug that made it to silicon**

**Targeting these failures**
<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>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type configuration register misinterpreted</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parity type misunderstood</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stick parity not implemented</td>
<td>5</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>21</td>
</tr>
</tbody>
</table>
## Verification Requirements (post FMEA)

<table>
<thead>
<tr>
<th>Req</th>
<th>Check</th>
<th>Conditions</th>
<th>Risk Exposure</th>
</tr>
</thead>
</table>
| tx.frame.1 | Check that the UART transmits a data frame correctly:  
start → data → [parity] → stop 
Data must be sent LSB first with the correct # of bits... | dataBits = [5, 6, 7, 8]  
x data = [0..255]  
x parity = [OFF, ON]  
x pType = [ODD, EVEN]  
x stopBits = [1, 1.5, 2]; | 25 |
| tx.frame.2 | Check that previous data values do not affect the parity calculation | parity = [ON]  
x pType = [ODD, EVEN]  
x data = [0..255]*  
x (dataBits = [N]  
→ dataBits = [M (<N)]); | 21 |

* 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
Consider how the field can vary temporally within the result.

**Temporal Failures**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Access Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Premature Activation</strong></td>
<td>0001 0000 0000 0000</td>
</tr>
<tr>
<td><strong>Delayed Activation</strong></td>
<td>0000 0000 0000 0001</td>
</tr>
<tr>
<td><strong>Partial Activation</strong></td>
<td>0000 0001 0000 0000</td>
</tr>
</tbody>
</table>

The `reg_sel` field must only be active between these points.
Value Failures

Address 0xF000 accessed. reg_sel should be 0001

Consider how the value of the field can fail

---

reg_sel field must be one hot when active

Invalid Value: 0000 1000 1000 0000

Illegal Value: 0000 1010 1010 0000

---

Copyright © 2008 Verilab Ltd. v1.5
Data = -5, encoded as 2’s complement

1011
1010  1’s complement
1101  Sign and magnitude

Consider how the field can be incorrectly encoded
But that’s too much Detail!

- “We don’t have time for that level of detail”

- That’s ok, but you might end up writing a fault tolerant testbench
  - where you cannot detect an error no matter how many simulations you run

- Consider another *real* example...
Consider an illegal value on `reg_sel` driven by RTL

```vhdl
case (haddr[3:2])
  2'b00  : reg_sel = 4'b0011;
  2'b01  : reg_sel = 4'b0010;
  2'b10  : reg_sel = 4'b0100;
  2'b11  : reg_sel = 4'b1000;
  default : reg_sel = 4'b0000;
endcase
```

Bug in the RTL
Too much Detail?

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

```c
if (reg_sel[0] == 1){ // Access reg 0
} else if (reg_sel[1] == 1){ // Access reg 1
} else if (reg_sel[2] == 1){ // Access reg 2
} else if (reg_sel[3] == 1){ // Access reg 3
}
```
Too much Detail?

```
    case (reg_sel)
      4'b0001: // Access reg 0
      4'b0010: // Access reg 1
      4'b0100: // Access reg 2
      4'b1000: // Access reg 3
    default: // No access. Don’t drive data_valid
    endcase
```

1. Uses “case” statements with reg_sel
2. 0011 causes deadlock (infinite wait states)
3. Customer isn’t very happy
“The indispensible first step to getting what you want is this: Decide what you want”

Ben Stein
What’s the technical risk involved in this operation?

If we don’t explicitly verify it, what’s the risk that it

- doesn’t work
- is not found by us
- is found by a customer
Determining Risk – Risk Exposure

Risk Exposure = likelihood of failure \times impact of failure

- Not Likely (1) → likely (N)
- Acceptable (1) → Unacceptable (N)

1
- Not worth any effort at all

N^2
- Extremely risky – lots of effort needed

- 5 and 10 are common values for N
What’s the likelihood of:
- the operation being wrong?
- the inputs being wrong?
- the operation being used at all?
Likelihood of a Failure Occurring

How complex is the design for this operation?

Major Mode
- Very Complex/Repetitive: 3, 4, 5
- Moderately Complex: 2, 3, 4
- Trivial: 1, 2, 3

Minor Mode
- Very Complex/Repetitive: 2, 3, 4
- Moderately Complex: 1, 2, 3
- Trivial: 1, 1, 2

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
The End

david.robinson@verilab.com