bitvis

11000101101001111010011101101101001111 0011

Improve your VHDL testbench - A practical approach

FPGA™ world

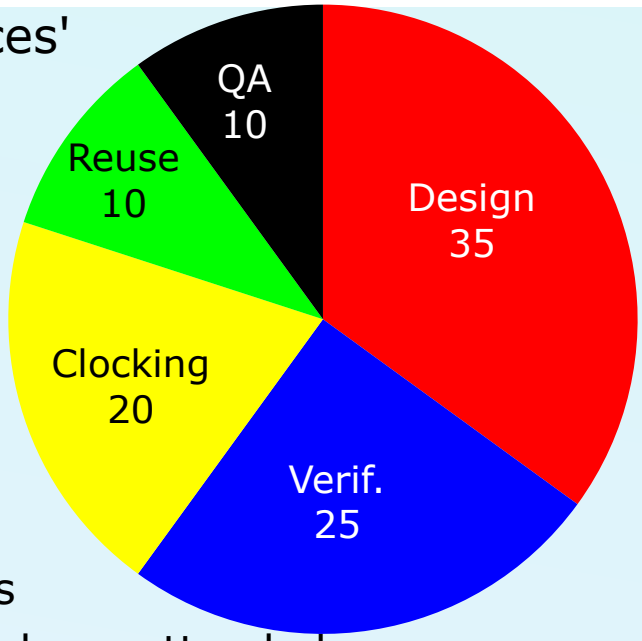
Note: Send an email to info@bitvis.no to request the actual Powerpoint file

Bitvis

- Independent Design Centre for SW and FPGA (& ASIC)
- 12 designers – and increasing….
- Building on the Digitas legacy
  - Efficiency/Quality → Methodology
  - Customer partnership relation
- Specification, Design, Verification, Implementation, Test
- **Methodology partner**
- Sparring and review partner
- **Verification IP provider**
- **Course provider "FPGA Development Best Practices"**

FPGA Development Best Practices

- Two-day course: 'FPGA Development Best Practices'
  - Based on the Digitas (and Data Respons) course presented in DK, SE, NO
  - Focus on improving your FPGA-based projects
    - Design Structure and Methodology
    - Verification Structure and Methodology
    - Focus on Efficiency and Quality improvement
    - Focus on a Best Practices approach that you can apply today
  - Practical and Pragmatic approaches
  - Very good feedback from participants in all countries
  - Approximately 50% of all FPGA designers in Norway have attended....
  - Quotes from last three courses:
    - 'The only bad thing about this course - is that we didn't do it earlier'
    - 'An eye opener' & 'Most issues apply directly in our organisation'
    - 'A very good course on very relevant improvement potentials'



Today - In track C5 - @14:30

Faster, Cheaper and Safer FPGA Development - A practical approach

Find out why you should attend this course...

Seminar tomorrow

FPGAs:
Establishing a 'Fit for Purpose' Design Flow

- 11th September 2013, 8:30-13:30, Stockholm, Sweden
  (at Broadband Technology AB, STOCKHOLM)

- My Presentation (60 minutes):
  'The Use of Open Source Utilities
  for Simple, Efficient and Structured Testbenches'
  - *More details. Going through example DUT & TB. Demo.*
- *See www.bitvis-no / events for link to flyer and registration*

"Be afraid, be very afraid"

- From 'Design Warrior's Guide to FPGAs':
  - "SW designers look at other code with horror"
  - "RTL sets new standards for awfulness"
- From my 25 years of experience:
  - RTL is heaven compared to Testbenches
  - Testbench behavioural code is often pure horror

State of the FPGA community - for Verification

- Most Testbenches:
  - are Unstructured
  - are Undocumented
  - are Not very understandable
  - are Terrible to modify
  - produce very little alert and debug information
  - produce very little progress information
  - produce close to no positive acknowledge on most checks

There seems to be a strong focus on fast TB code writing

Yet - readability and understanding is far more important...

Main Testbench scope

- Purpose: To verify DUT requirements
- Focus: Sufficient functional coverage with a minimum effort
- **Test Sequencer Requirements:**
  - ➔ **Simple to write**
  - ➔ **Simple to understand and modify - by anyone**
  - ➔ **Simple to execute, debug and understand reports & results**

The development stages

- Ideally:
  - Specify → Design → Simulate → Synthesize → Test
  - Verification should start as early as possible
- But: Specification changes all the time…
  - ➔ Design must be extended or modified
    - ➔ Testbench or test cases must be extended or modified
      - ➔ Simulations must be re-executed
        - ➔ and the result must be checked again

➔ TB Readability, Flexibility and Extendibility is important

➔ Simulation result checking and readability is important

How do we achieve this?

Achieving flexibility, readability, extendibility, ...

- What is always required for a good testbench?
  - Logging - with good messages
  - Alert handling - with good messages
  - Checking values
  - Checking signal stability
  - Waiting for something to happen
  - Maybe some randomisation
- Required for both simple and advances testbenches
  - Advanced TB architectures need additional advanced structures, - but these are also building on the basic infrastructure.

➔ Requires a basic TB infrastructure
- as a platform for any TB

Bitvis utility library

- No cost; Open source VHDL (previously 9.000 euro)
- Extremely simple to use *(acc. to feedback)*
- Advanced options when you need them
- Good documentation:
  - Simple TB step-by-step (PPT)
  - Library concepts and details (PPT)
  - Quick Reference (PDF)
  - Example test sequencer for a simple interrupt controller
- Improve quality and efficiency

Purpose of Bitvis Utility Library

- Standardise and qualify a set of good TB support procedures
- Improve TB readability, modifiability and extendability
- Significantly reduce TB code size
- Allow a more uniform TB methodology
- Improve reuse
- Make it easy to generate good simulation transcripts
  - Force a more uniform log of events
- Promote more single source (and less work)
  Same code used as e.g.:
  - Verification spec.
  - Testbench comments
  - Transcript to log

Concepts

- Three main parts
  - ✓ Logging mechanism and verbosity control concept

Logging mechanism is intended for informative messages that require no attention.
Main purpose: Simulation progress reporting.

Alerts are intended for messages that need or may need attention. Could be any severity.

- Additional features
  - ✓ Randomisation – as simple as possible
  - ✓ String handling
  - ✓ BFM support
- Adaptations in a separate customer/project package

➔ First focus on concepts

Verbosity control

- Method to control amount of information
  - To allow only selected groups of messages
  - Allows reduction/increase of information – without modifying the code (comment/uncomment)
  - Makes it possible to get a different set of messages depending on current simulation focus - e.g.:
    - Debugging testcase or verification component
    - Debugging a specific interface on DUT
    - Debugging a data flow through DUT
    - General simulation progress report.
  - Most verbosity control systems are priority based
    - ID based is better

ID based verbosity control

- ID based verbosity allows log messages with different IDs to be shown or blocked depending on whether a given ID is enabled or disabled.
- Every message has a given ID
  - Bitvis Utility Library: ID is an Enumerated 'ID_<name>'
    E.g. ID_BFM, ID_PACKET, ID_PACKET_HDR, etc...
  - E.g. **`log(ID_PACKET_HDR, "Packet header received")`**
- Verbosity control will determine which messages to show
  - Enable an ID using **`enable_log_msg(ID_BFM)`**
  - Disable an ID using **`disable_log_msg(ID_BFM)`**
  - May dynamically enable or disable from the test sequencer!
- ID-based verbosity is far more flexible and controllable

Using the log method


```
log(msg_id, msg, [scope])  -- Simple version
```

- Where? → Anywhere!
  (Transcript printout depends on verbosity control)

```
-- In test sequencer as a normal progress msg (single param overload)
log
-- In BFM (procedure) to detect header in received data
log(ID_PACKET_HDR, "Packet header received for packet 1", C_SCOPE);
```

```
BV: 160 ns   irqc_tb   Checking Registers in UART 4
BV: 1260 ns   P_PCK_RECEIVER   Packet header received for packet 1
```

Alerts and severities


```
tb_error("address value does not fit target");
```

```
BV: TB_ERROR:
BV:     192 ns. irqc_tb
BV:             address value does not fit target
```

- All alert levels (severity levels) are counted separately
- set_alert_stop_limit(alert_level, N>=0)
- set_alert_attention(alert_level, IGNORE|REGARD)
- increment_expected_alerts(alert_level, N)
- report_alert_counters(VOID)

check_value()

check_value(val, exp, severity, msg, [scope]) -- Simple version

- checks value against expected (or boolean)
  - Triggers alert if fail – and reports mismatch + message
  - Positive acknowledge depending on verbosity control
- Overloads for sl, slv, u, s, int, bool, time
- With or without a return value (boolean OK)

```
-- E.g. in test sequencer as a section header
check_value(dout, x"00", ERROR, "dout must be default inactive", "irq_tb");
```

```
BV: 60 ns  irqc_tb  check_value(slv x00)=> OK.
                    dout must be default inactive
```

```
BV:=============================================================
BV: ERROR:
BV:     192 ns. irqc_tb
BV:             value was: 'xFF'.  expected 'x00'.
BV:             dout must be default inactive
BV:=============================================================
```

other checks

- check_value_in_range
  - minimum <= value <= maximum
  - overloads for u, s, int, time, real
- check_stable
  - checks signal stable for minimum the given time
  - overloads for sl, slv, u, s, bool, int

await_*

- await_change()
  - expects (and waits for) a change on the given signal
    - inside the given time window
    - otherwise timeout
  - a real change (event) is required on the signal
- await_value()
  - expects (and waits for) a given value on the signal
    - inside the given time window
    - otherwise timeout
  - accepts value if already present and min = 0ns

Other functions/procedures

- Various string handling
  - justify, find_leftmost, to_upper, fill_string, replace, to_string
  - Additional to ieee_proposed/2008 (and for missing variants)
- Randomisation
  - my_slv := random(7); -- 7-bit std_logic_vector
  - my_int := random(2,8); -- integer >=2 and <=8
  - More advanced variants available with controlled seeds
- normalise() - for any unconstrained vector

Makes it very easy to make good messages in a structured manner - for log, alert, check, await,.....

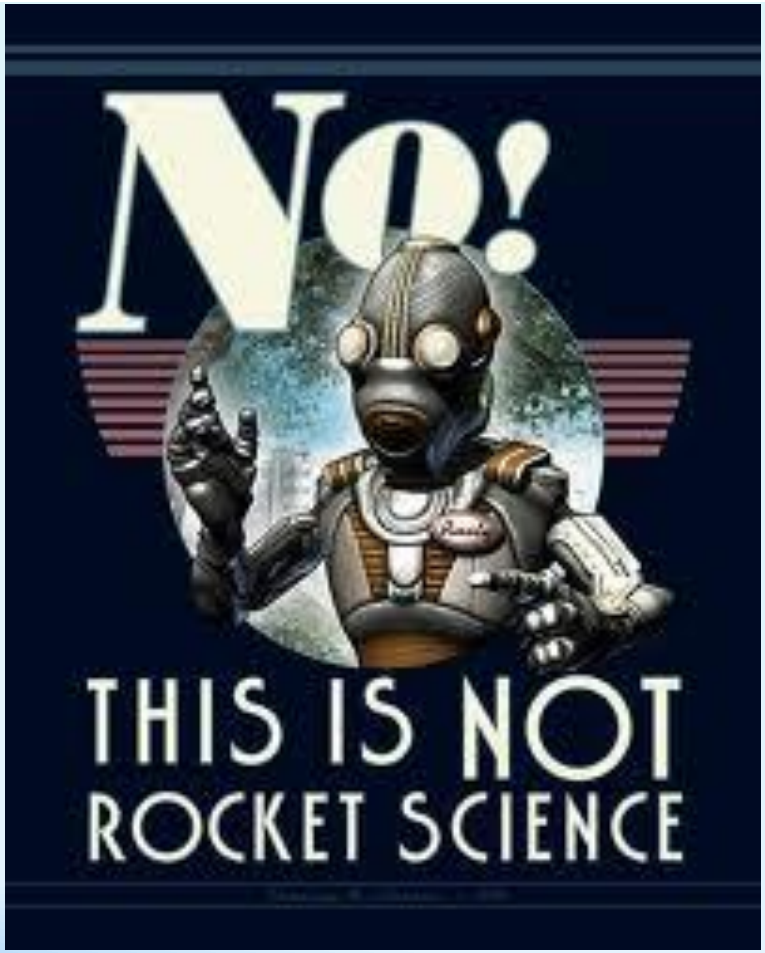
Very simple to use

Incl. Sanity check and flexibility + check

Getting the full picture

- Not sufficient time to show real examples or demo
- Get the full picture by downloading
  '*Making a Simple, Structured and Efficient Testbench, Step-by-step*' from www.bitvis.no
  (Previous version presented at FPGAworld 2009)
  - or even better:
  Watch the one hour webinar for Aldec (see www.bitvis.no)
- Example TB for IRQC is included in library download
  Use this to see the real benefits of the library and applying a structured verification approach.
- Quick reference gives a good overview of all commands
  - incl. all overloads and all parameters

Rocket science?



- This is NOT rocket science, BUT....
- This is
  - Simple
  - Structured
  - Consistent
  - Flexible
  - A working system
  - A platform
  - Documented
  - Built on a well defined Philosophy

Getting started

All you have to do is:

- Download library from www.bitvis.no 5 min
- Compile the library (as described) 10 min
- Add library in your TB 2 min
- Add log command 1 min
- Add check value(s) 1 min
- Add alert summary 1 min
- Run simulation & check transcript

```
Add log command;
Add check value(s);
Add alert summary;
Run simulation & check transcript
```

```
log(ID_LOG_HDR, "Testing a log header");
check_value(my_sig, x"00", ERROR, "Provoking error");
```

Total of 20 min to get started

Conclusions

- All TBs should have proper logging, checking, alerts,....
  → Saves time. Improves quality, flexibility, readability, extendibility
- Bitvis Utility Library provides a good solution:

Well documented

Consistent throughout

No cost

Low user threshold

Faster development

Better Quality

A platform for any TB structuring

Are there any similar or better libraries around?

Not using this **Bitvis Utility Library** would be rather strange - unless you have something even better...

Improve your VHDL testbench - A practical approach

Bitvis
Utility Library

www.bitvis.no

Thank you

Quality in every bit