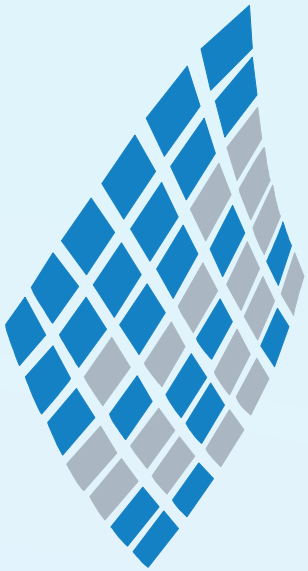
bitvis

1100010110100111101001110110110100111110011



The critically missing VHDL testbench feature

- Finally a structured approach

FPGAworld
Sept. 8, 2015

About Bitvis

- Independent Design Centre for SW & FPGA/ASIC
- 14 designers (Embedded SW: 5, FPGA: 9) *(April 2015)*
- Specification, Design, Implementation, Verification, Test

- Methodology partner
- Sparring and review partner
- Verification IP provider
- 2-3 day course:
  'Accelerating FPGA Development'

The leading independent design centre in our field in Norway

Added after the presentation at FPGAworld

This presentation has a lot of animation.

If you would like to watch the original PowerPoint presentation, just send us an email requesting this.

Send the email to info@bitvis.no

and include your name (mandatory)

and your company (optional)

What is missing?
- And What is the improvement potential?

Missing:

- A good Testbench Architecture and Structure
  - For Protocol and Control oriented Design in particular

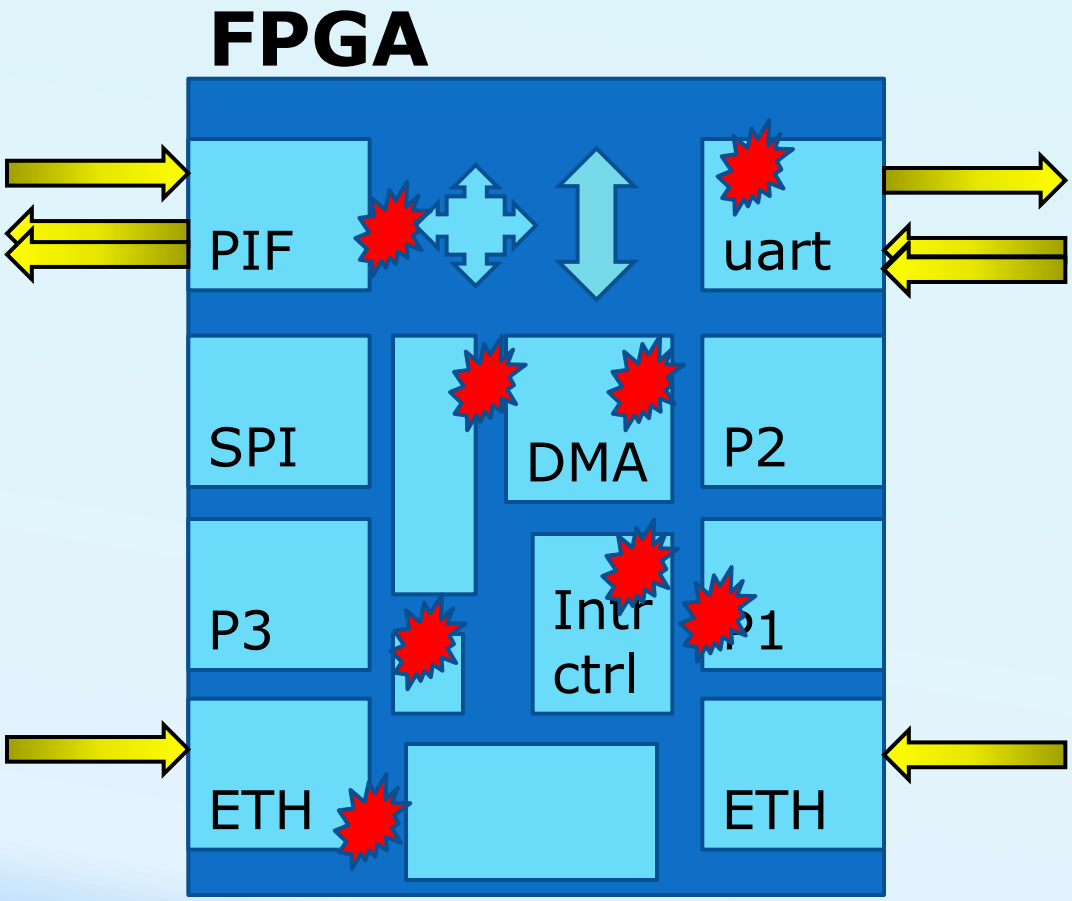
Consequences:

- Inefficient testbench development, extensions, modifications
- Difficult to reuse TB parts in a project - or to share the TB itself
- High risk of missing critical corner cases

Improvement Potential using a well structured approach:

- Major time saving (many MWs for medium complexity FPGA)
- Significant quality improvement for end product
- A new world for overview, maintainability, extendibility and reuse

FPGA functional scenario



Sequential testing

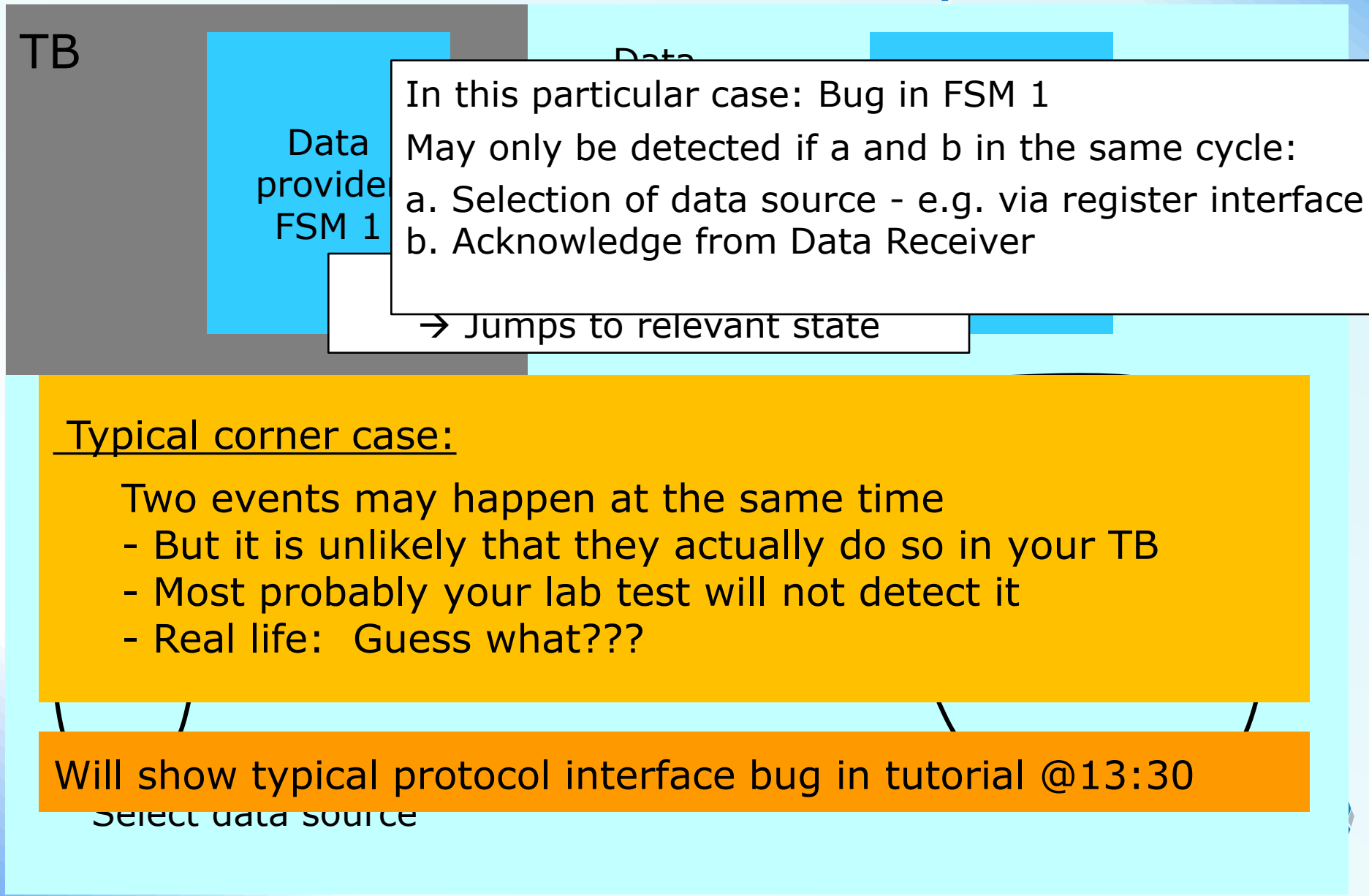
➔ Find some bugs?

➔ Then behaves fine

Parallell operation

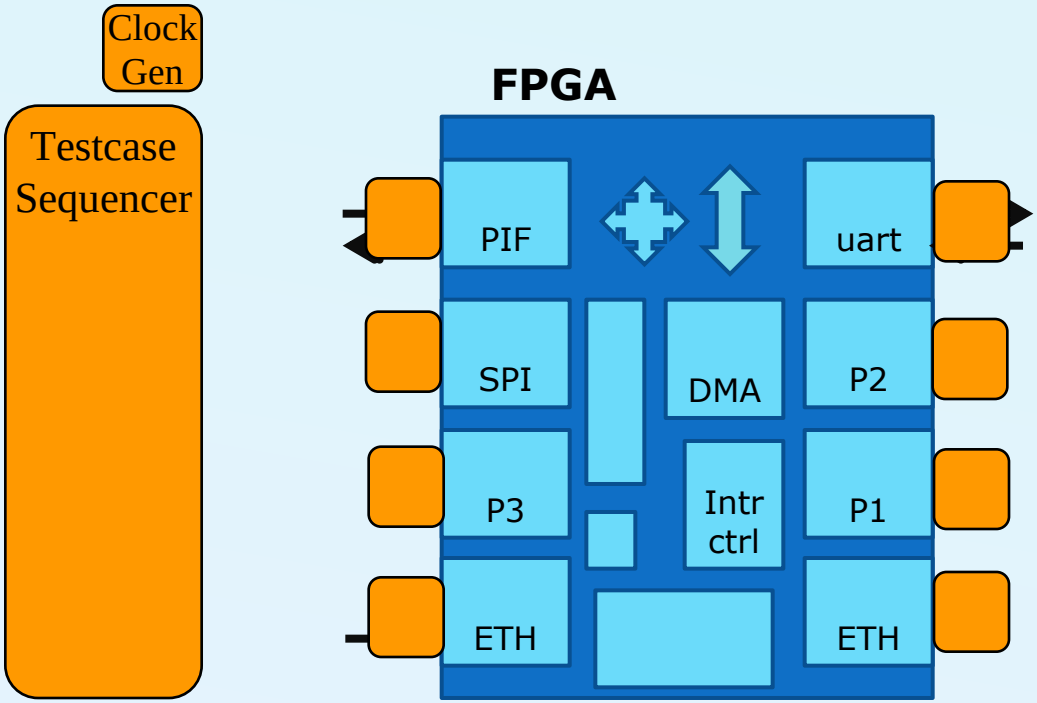
➔ Corner case bugs

Corner case example



Typical testbench

Typical testbench
➔ Sequential



Sequential testing
➔ No bug found

Parallel operation
➔ Corner case bugs

Cycle related corner cases?

Adding threads | Lab ? | CCL | ?

Ad hoc "structure"

Structured, but far too complex

Structured with good overview

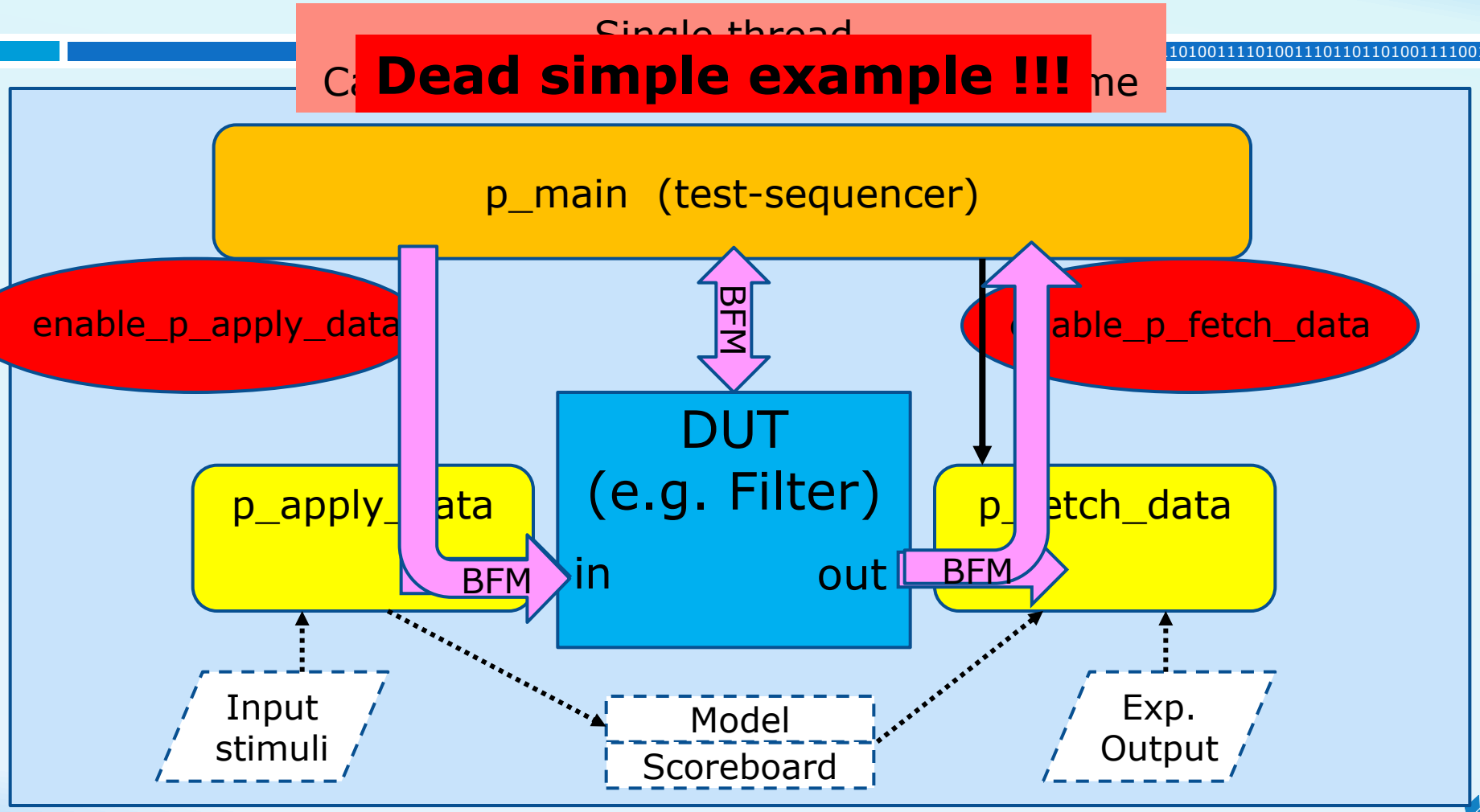
Simple problem → ad-hoc solution

Example: A regular data stream

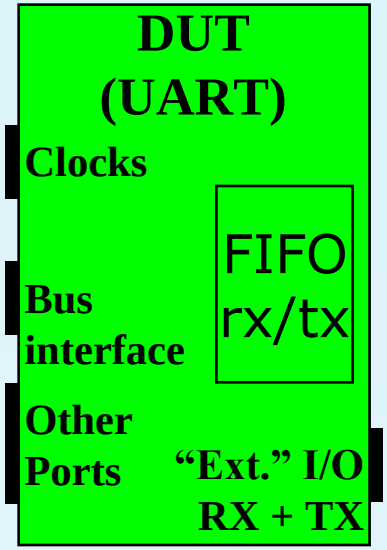
- E.g. a filter or simple algorithm

- Simultaneous access on two interfaces
- Contiguous data input & contiguous data output

Regular Data Stream - Example



The UART



- Still a VERY simple module!
- BUT
  - May have lots of corner cases
    - often never simulated at all
    - often never tested in the lab
- So
  - Need to control RX, TX and PIF independently
  - Must be tightly controlled from a sequencer
  - Must allow full flexibility in data, access times, etc

Will show typical protocol interface bug in tutorial @13:30

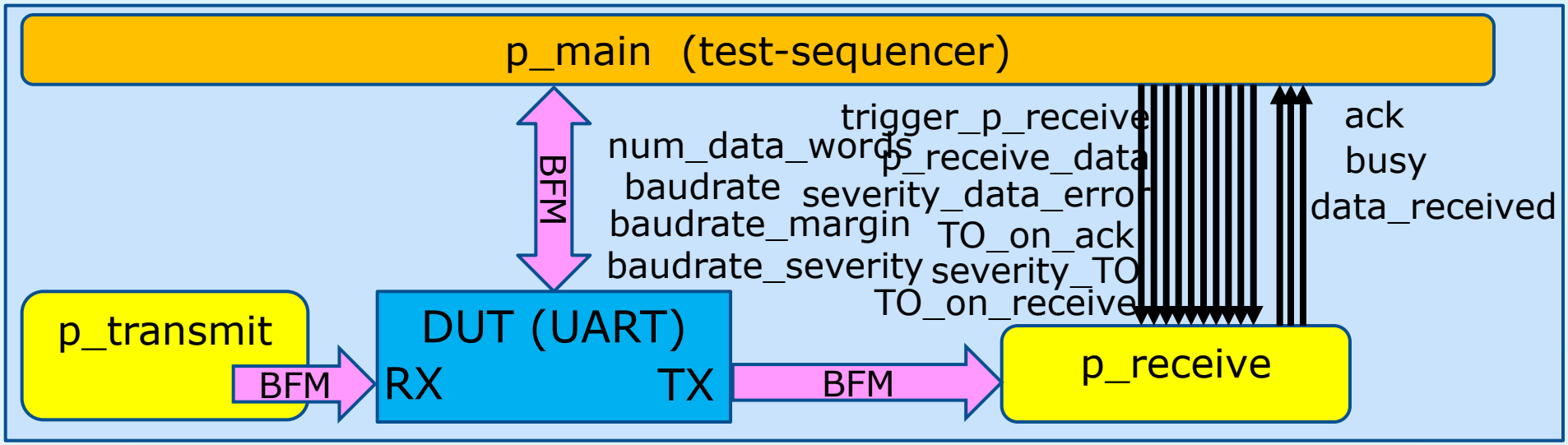
Wishful thinking for a testbench?

Wouldn't it be nice if we could ...

- handle any number of interfaces in a structured manner?
- reuse major TB elements between module TBs?
- reuse major module TB elements in the FPGA TB?
- read the test sequencer almost as simple pseudo code?
- recognise the verification spec. in the test sequencer?
- understand the sequence of event
  - just from looking at the test sequencer

Is this feasible at all?

Scenario: Check data out of UART (DUT) TX



Problem is growing for every detected need

- Need to set number of data words
- Need to see the actual data received in the sequencer
- Need to set baudrate
- Need to set margins for baudrate, and severity
- etc, etc .....

Problem keeps growing...

- More and more control signals must be entered
- New issues detected during test case development
- Old code lines must be updated

After "finishing" the control and data reception of UART TX

→

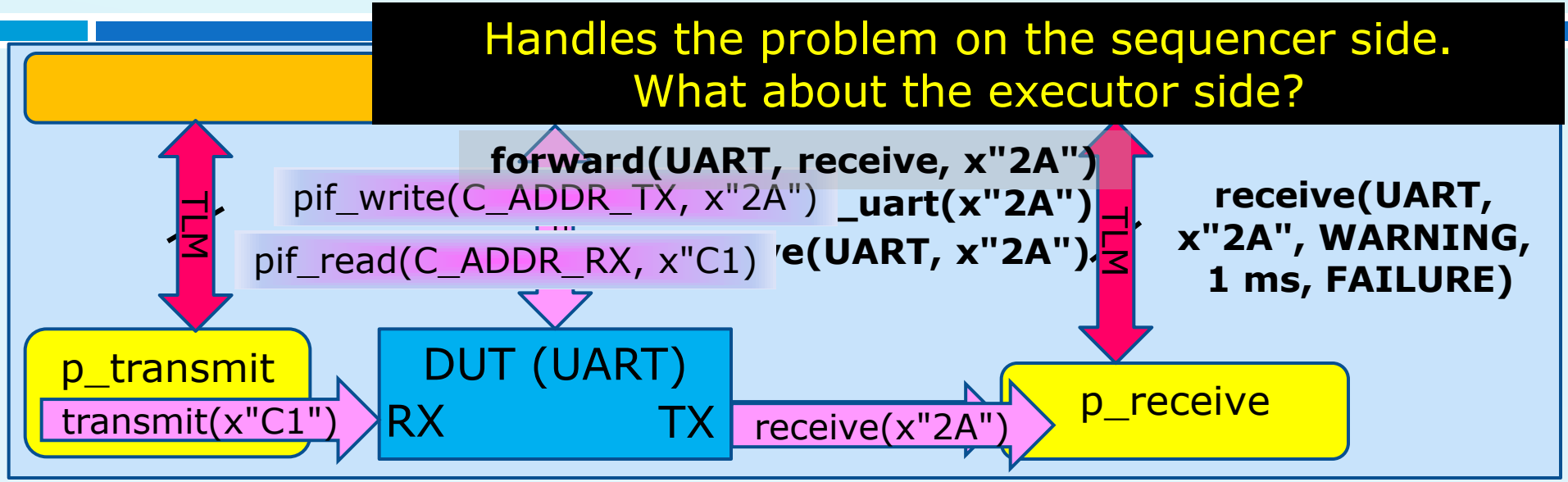
Same issues for control and data to UART RX

Same issues for other modules

Same issues at top level

For every new interface and feature
"old" code may have to be updated

Let's Solve the problem



Need BFM-like solution

- Procedural
- Signals between Sequencer and p_receive / p_transmit
- No physical signals, but used to control transactions

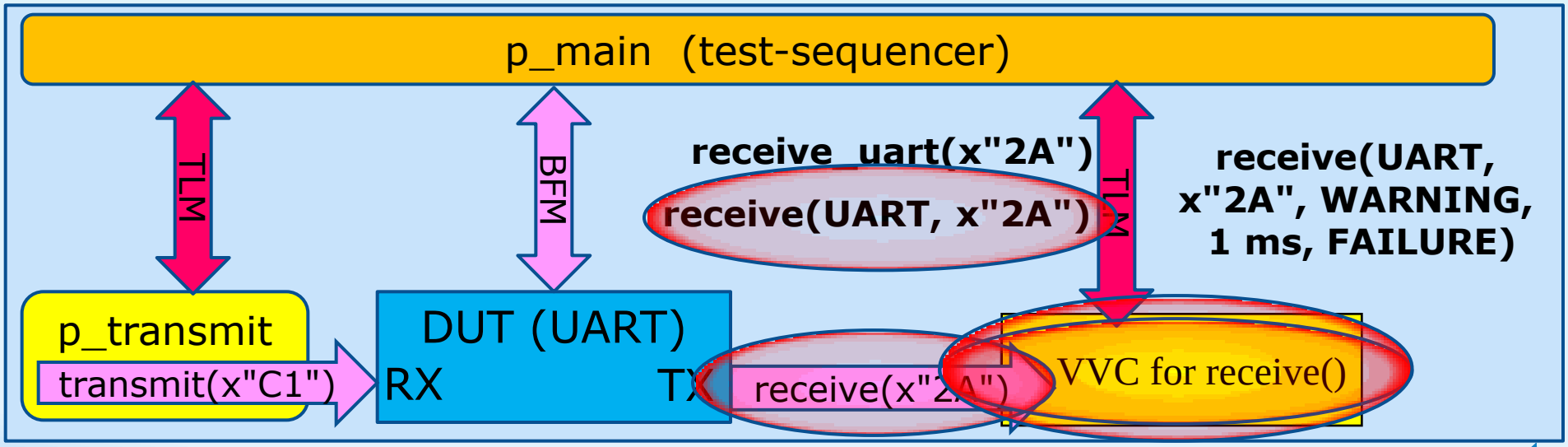
➔ **Transaction level models/commands**

- Allows painless modifications of signals, protocol, features

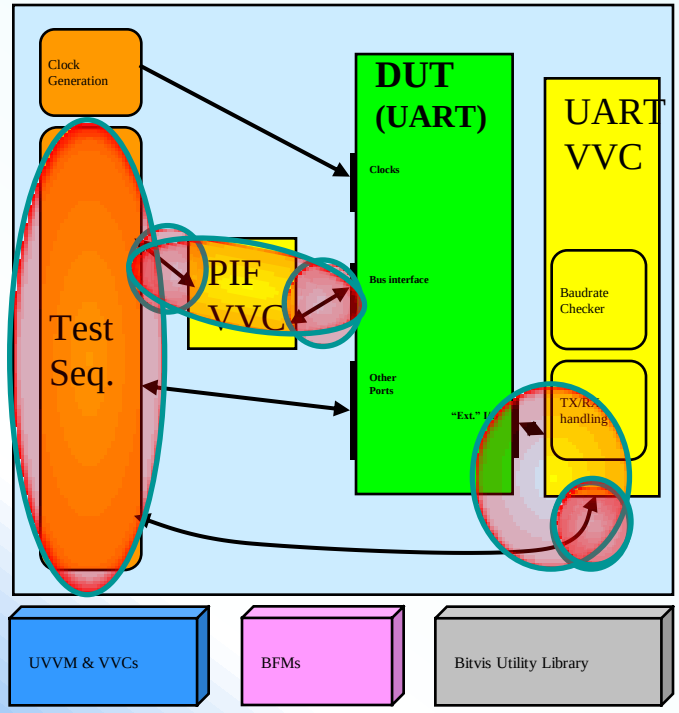
The executor side - for Receive

- Must detect command from central test sequencer
- Then execute BFM receive() towards the DUT

From UART BFM and VVC template to UART VVC : 1 hour



Hitting the corner cases



Simple test sequencer example:

```
. . .
write(PIF, C_ADDR_TX, x"2A", "Uart TX");

expect(UART, RX, x"2A", "From DUT TX");
transmit(UART, TX, x"C1");-- into DUT RX

insert_delay(UART, TX, 2*C_BIT_PERIOD);
transmit(UART, TX, x"C2");
await_completion(UART);

check(PIF, C_ADDR_RX, x"C1", "Uart RX");
check(PIF, C_ADDR_RX, x"C2", "Uart RX");
await_completion(PIF);
.........
report_simulation_summary;
```

Making a comprehensive test (1)

UVVM provides an enhanced version of OSVVM Random & Coverage

In central test sequencer

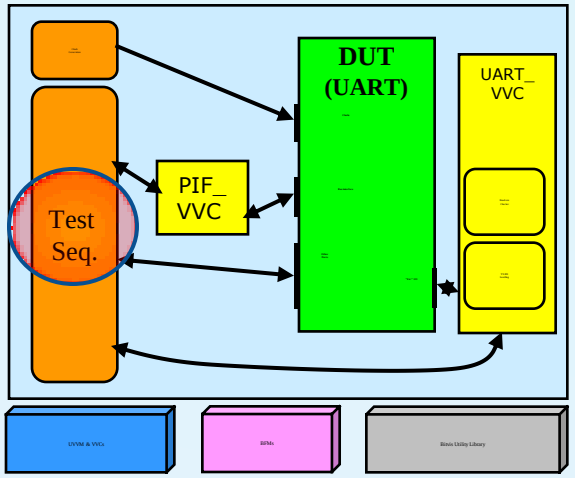
```
test_uart_transmit_with_delay(<data_tx>, <data_rx>, <delay>);
```

xN

```
test_uart_transmit_random_with_skewed_delay(<num_times>);
-- random data and random delay
```

Simple test sequencer example:

```
. . .
write(PIF, C_ADDR_TX, x"2A", "Uart TX");
expect(UART, RX, x"2A", "From DUT TX");
transmit(UART, TX, x"C1");-- into DUT RX
insert_delay(UART, TX, 2*C_BIT_PERIOD, "Between 2 transmits");
transmit(UART, TX, x"C2", "2nd of two transmits. Delay between");
await_completion(UART);
check(PIF, C_ADDR_RX, x"C1", "Uart RX");
check(PIF, C_ADDR_RX, x"C2", "Uart RX");
await_completion(PIF);
.....
report_simulation_summary;
```



Making a comprehensive test (2)

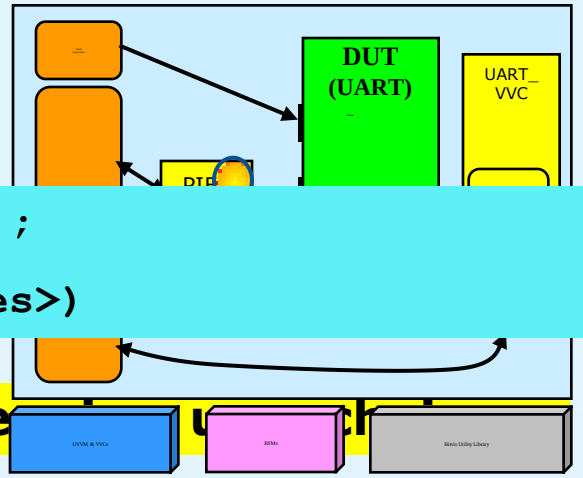
Using VVC local test sequencers

Inside UART_VVC (in executor):

```
uart_transmit_buffer(<buffer_idx>, <num_bytes>);
```

Inside PIF_VVC (in executor):

```
pif_check_buffer(C_ADDR_RX, <buffer_idx>, <num_bytes>);
```



Calls from central sequencer:

```
uart_transmit (UART_VVC,1,TX,   <num_bytes>);
pif_check (PIF_VVC,1,   C_ADDR_RX, <num_bytes>)
```

Name of procedures and use of parameter

Solving the multi-interface problem

- **Central Sequencer with TLM and VVCs:**
  - ✓ One single brain in the system
  - ✓ Inherent synchronisation between sequencer and VVCs
  - ✓ No process is starting by "pure magic"
  - ✓ Full control of complete env. from the main sequencer
  - ✓ Allows simple queuing of commands
  - ✓ Lego-like VVC connections
- **Global connections and proper Verbosity control:**
  - ✓ Automatic connection between sequencer and VVC
  - ✓ Allows built-in VVCs (+ checkers and monitors)
- Excellent transcript of test sequence
  - ✓ Allows selected progress information at all levels

Wishful thinking? - Recaptured

Wouldn't it be nice if we could ...

- handle any number of interfaces in a structured manner? ☑
- reuse major TB elements between module TBs? ☑
- reuse major module TB elements in the FPGA TB? ☑
- read the test sequencer almost as simple pseudo code? ☑
- recognise the verification spec. in the test sequencer? ☑
- understand the sequence of event ☑
  - just from looking at the test sequencer ☑

More info in tutorial @13:30 today

- A typical corner case - found everywhere
- Verification components
- Experience data on making a new VVC
- Simulation progress information
- Debug friendliness
- Low user threshold
- Constrained random and functional coverage

Must register
for the tutorial
in the reception.

Note:
Limited seating.

A 90 min. course - for free ☺

UVVM will be freeware

Using a structured TB like UVVM for handling multi-interfaces issues

Savings on a mainly control or protocol oriented design:

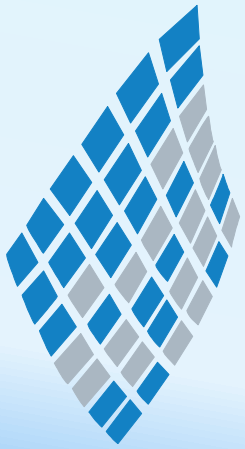
a) 2000-hour FPGA Development Project: **100-500 MH**
b) 5000-hour FPGA Development Project: **500-1500 MH**
Increasing from project to project

➔ Additionally saving a lot of time for SW developers
➔ Significantly improving quality on final product
➔ Drastically improving LCC and TTM

The critically missing VHDL testbench feature

- Finally a structured approach

Thank you



bitvis

Your partner for Embedded SW and FPGA

www.bitvis.no