HEIG VD
HAUTE ÉCOLE
D'INGÉNIERIE
ET DE GESTION
DU CANTON
DE VAUD

Open source formal verification with SymbiYosis

Yann Thoma, 13/06/2024, FPGA developers Forum, yann.thoma@heig-vd.ch

Hes·so
Haute École Spécialisée
de Suisse occidentale

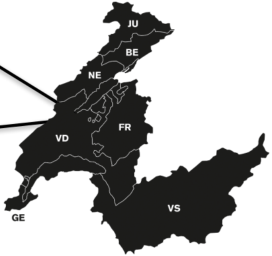
HEIG VD

HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

TIC Department

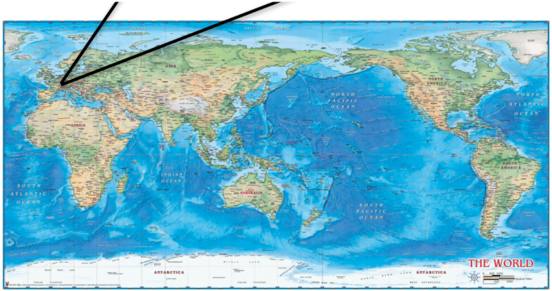
HEIG VD

REDS
Institut Reconfigurable and Embedded Digital Systems



HES-SO
70 bachelor and master programs
>20'000 students
6 faculties

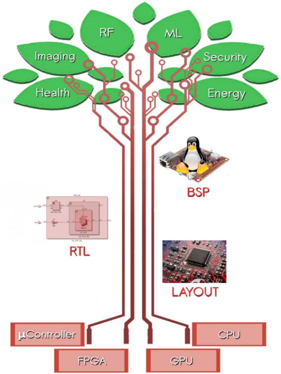
- Design and Visual arts
- Business and Services
- Engineering and Architecture
- Music and Performing Arts
- Health Sciences
- Social Work



REDS institute

Reconfigurable and Embedded Digital Systems

- Competencies
  - Embedded software
  - FPGA design and verification
  - PCB design
- Team
  - 8 professors
  - 5 senior assistants
  - 15 junior assistants



Assumption

You followed the previous talk

That many slides ? Are you crazy ?

Well, we'll stop at 45 :-)

Goal of digital systems verification

Answering two questions

- Does it work ?
- Are you sure ?
- Really ?
- Well, really really sure ?
- No kidding ?

Digital systems verification

- Verification of a digital system shall allow to assert that the realization of a system corresponds to its specification

⇒ Proof of its proper behavior

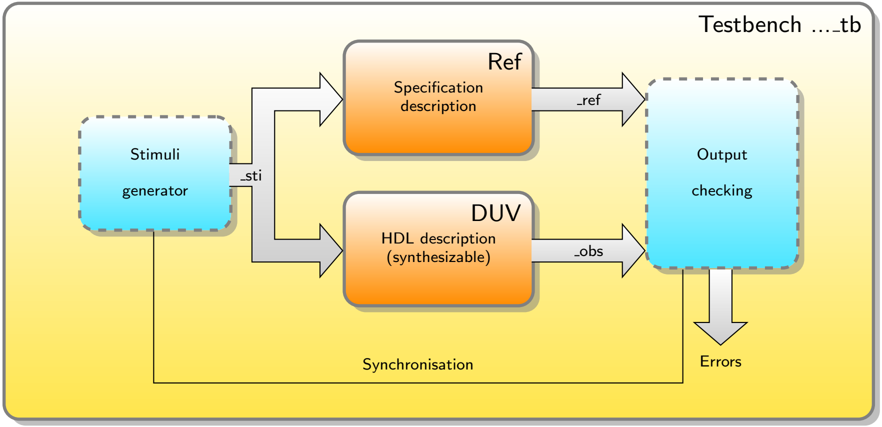
Digital systems verification

- Methodologies
  - Fonctional verification
    - Simulation
    - Emulation
  - Formal verification
  - Real tests

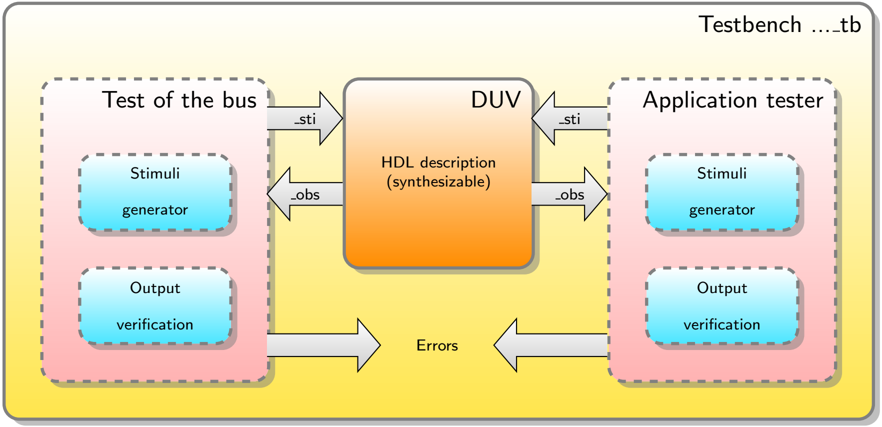
Functional verification

- Simulation/emulation of the system
- Design of testbenches
- Simulation
    - Directed
        - Hard to think about all interesting/challenging cases
    - Random-based coverage driven
        - Not easy to guarantee a good coverage
    - Mix
        - Good compromise, but time-consuming

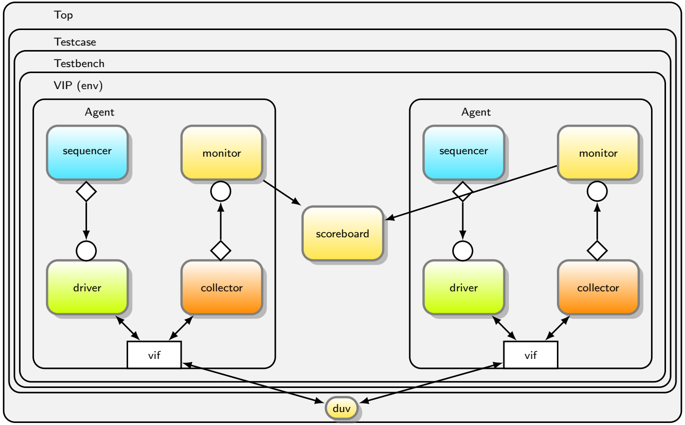
Testbench with reference



Testbench with multiple interfaces

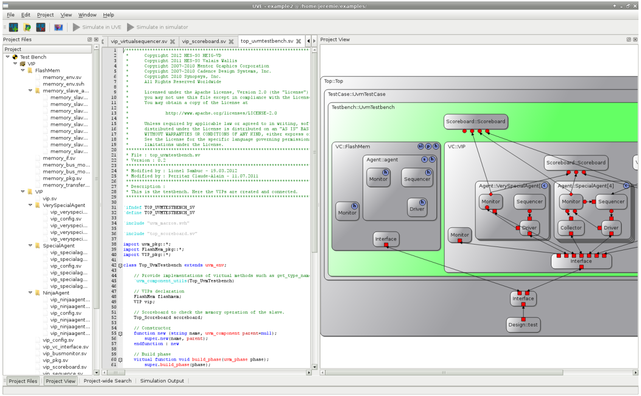


TLM / UVM testbench



UVM testbench : UVE project

http ://www.systemverilog.ch/



```
/**************************************************************************
*       Copyright 2012 HES-SO HEIG-VD
*       Copyright 2011 HES-SO Valais Wallis
*       Copyright 2007-2010 Mentor Graphics Corporation
*       Copyright 2007-2010 Cadence Design Systems, Inc.
*       Copyright 2010 Synopsys, Inc.
*       All Rights Reserved Worldwide
*
*       Licensed under the Apache License, Version 2.0 (the "License")
*       you may not use this file except in compliance with the Licens
*       You may obtain a copy of the License at
*
*               http://www.apache.org/licenses/LICENSE-2.0
*
*       Unless required by applicable law or agreed to in writing, sof
*       distributed under the License is distributed on an "AS IS" BAS
*       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express o
*       See the License for the specific language governing permission
*       limitations under the License.
**************************************************************************
* File : top_uvmtestbench.sv
* Version : 0.2
**************************************************************************
* Modified by : Lionel Sambuc - 19.03.2012
* Modified by : Perritaz Claude-Alain - 11.07.2011
**************************************************************************
* Description :
* This is the testbench. Here the VIPs are created and connected.
**************************************************************************

`ifndef TOP_UVMTESTBENCH_SV
`define TOP_UVMTESTBENCH_SV

`include "uvm_macros.svh"

`include "top_scoreboard.sv"

import uvm_pkg::*;
import FlashMem_pkg::*;
import VIP_pkg::*;

class Top_UvmTestbench extends uvm_env;

    // Provide implementations of virtual methods such as get_type_nam
    `uvm_component_utils(Top_UvmTestbench)

    // VIPs declaration
    FlashMem flashmem;
    VIP vip;

    // Scoreboard to check the memory operation of the slave.
    Top_Scoreboard scoreboard;

    // Constructor
    function new (string name, uvm_component parent=null);
        super.new(name, parent);
    endfunction : new

    // Build phase
    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
```

https://github.com/uve-project/uve

A testbench generator

TbGenerator

- Within QuestaFormal, a wrapper instantiating the DUV is required to bind the assertions modules with the DUV
  - Uninteresting code

  ⇒ Automate this
- `https://reds-gitlab.heig-vd.ch/reds-public/TbGenerator`
  - Generation of code skeletons via Python scripts
    - VHDL testbench
    - SystemVerilog testbench
    - Wrapper for formal proof
    - Scripts for command line execution
- Soon : TB with a UVM-like structure (but in VHDL)

Get involved !

Transaction Level Modeling VHDL Methodology

- What a big name
- Actually just a bunch of interesting stuffs for testbenches :
  - Objections to smoothly end a simulation
  - Heartbeats to detect inactivity
  - FIFOs (bounded and unbounded) for TLM
- Open Source (since yesterday evening)
- `https://reds-gitlab.heig-vd.ch/reds-public/tlmvm`

Formal verification

- Checking the system against properties
- Proof that the system follows its specifications
- Advantages
  - Formal proof, so irrefutable if properties are well written
  - Allows to validate part of a system that can then be used in simulation without further tests
- Disadvantages
  - Maybe not adequate for all systems
  - Thinking about all the properties is not easy
  - Writing good properties is not easy
- Could need to be complemented by functional verification

Formal verification : Software tools

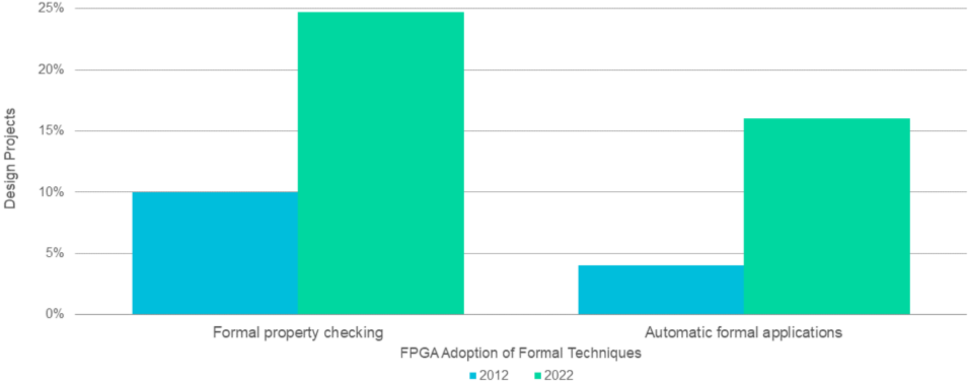
- QuestaFormal, from Mentor Graphics
  - Validates properties
  - Or finds a counterexample and shows a waveform
  - Very close to model checking technics
- SymbiYosis (now Sby)
  - Validates properties
  - Or finds a counterexample and shows a waveform
  - Open Source
  - Various engines (model checking of proof)

Formal verification

Languages

- Formal verification needs properties to be written
  - Useful for :
    - Assertions
    - Assumptions
    - Coverage
- Languages
  - PSL (Property Specification Language) : *language-agnostic*
  - SVA (SystemVerilog assertions) : Inherent to the language
  - OVL (Open Verification Library) : Built on top of VHDL/SVA

FPGA adoption of formal techniques

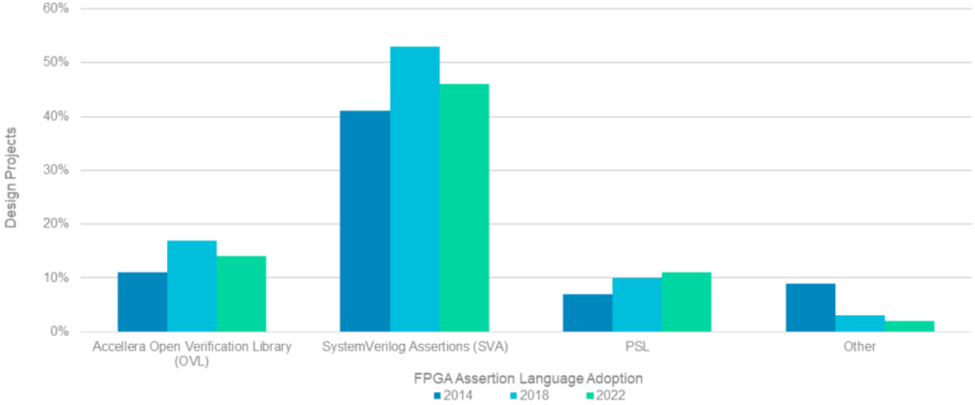


Source: Wilson Research Group and Siemens EDA, 2022 Functional Verification Study

Unrestricted | © Siemens 2022 | Functional Verification Study

SIEMENS

FPGA assertion language adoption



Source: *Wilson Research Group and Siemens EDA, 2022 Functional Verification Study*

* Multiple replies possible

PSL/SVA

LTL or ?

- PSL
  - LTL-style : close to Linear Temporal Logic (until, next, eventually, ...)
  - CTL-style : close to Conditional Temporal Logic (existence operator), called Optional Branching Extension (OBE), but it is not available in QuestaFormal nor Sby
  - SERE-style : exploits regular expressions
- SVA
  - LTL-style
  - SERE-style
- Differences
  - At the end you can use one or the other

Some use cases

Examples available at the end of the slides

- Counter
- FIFO (simple)
- FIFO (multi-channel)
- Control/datapath calculator

Testbench vs assertions

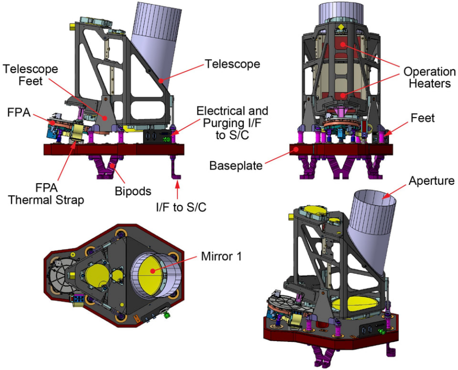
Lines of code

Project	Lines TB	Lines Assertions
Counter	200	4
FIFO	200	30
FIFO multi	300	45
Calculator	350	50

Relevant lines

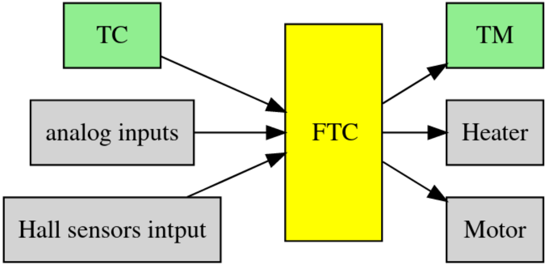
Context : Comet interceptor

- ESA mission
- Specifically : Comet Camera

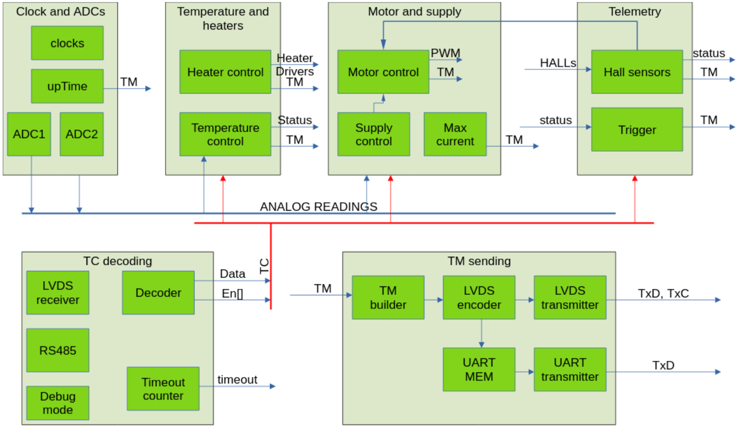


Filter wheel and Temperature Control FPGA (FTC)

- Various interfaces
- Telecommands packets (in)
- Telemetry packets (out)



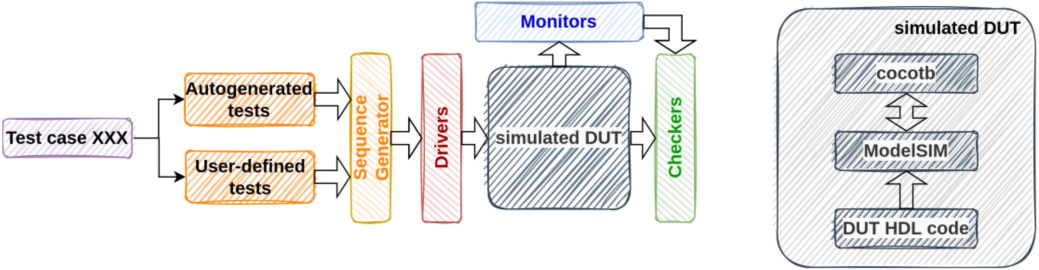
FTC architecture



How to test it ?

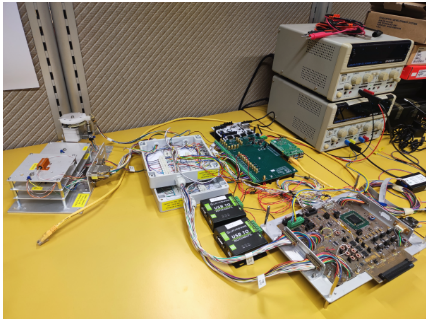
Python testbench with cocotb

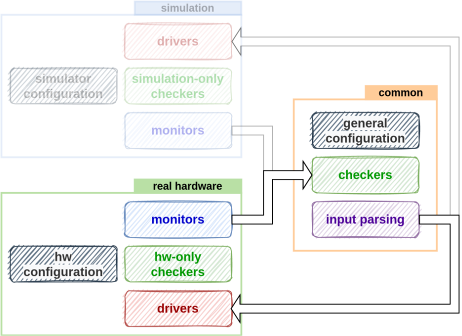
- Easy integration of VHDL DUV within a python framework
- All tests, scenarios, checking, writing in python



Python testbench with the real hardware

- Efficient reuse for testing the real hardware





Why adding formal verification ?

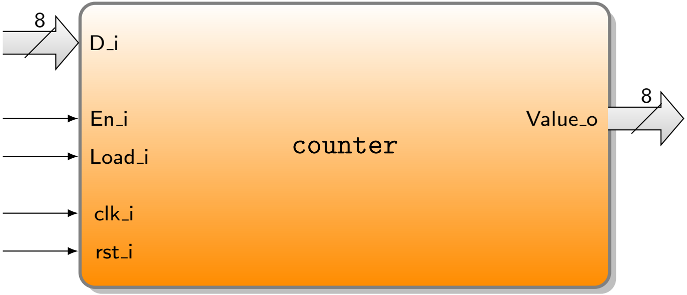
- In that context, the opportunity to try SymbiYosis (Sby)
- As a complementary approach to cocotb

Setup

- Tools needed :
  - GHDL and SymbiYosis
    - CAD suite ready to use
    - `https://github.com/YosysHQ/oss-cad-suite-build/releases/`
  - GTKWAVE
    - `https://gtkwave.sourceforge.net/gtkwave-gtk3-3.3.119.tar.gz`
- Both are easy to install
- PSL as the property language

VHDL - PSL : vunit

Example for a counter

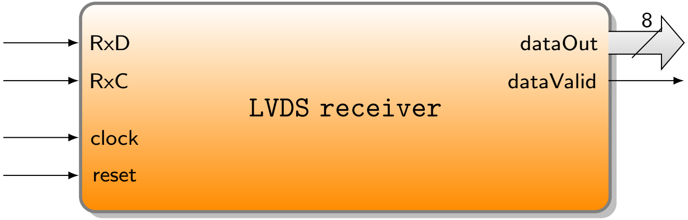


VHDL - PSL : vunit

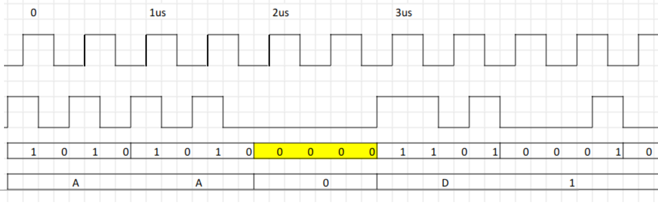
Example for a counter

```
vunit counter_psl(counter) {
    default clock is rising_edge(clk_i);
    property prop_incr is always
        ({(not Load_i) and En_i} |=>
        {Value_o=std_logic_vector(unsigned(prev(Value_o))+1)});
    assert prop_incr report "Increment error";

    property prop_hold is always
        ({(not Load_i) and (not En_i)} |=> {Value_o=prev(Value_o)});
    assert prop_hold report "Hold error";

    property prop_load is always ({Load_i} |=> {Value_o=prev(D_i)});
    assert prop_load report "Load error";

    assert always Value_o/="XXXXXXXX" report "Error, value is XXXXXXXX";
}
```

Example : LVDS receiver



A bit is received on every edge of `RxC`



Example : LVDS receiver

checking control signals

Detection of 8 edges of RxC, ensuring dataValid goes high at least 5 clock cycles after the 8th.

```
assert always(
    (not(stable(RxC)))[->8] |=> next_e[0 to 5](dataValid)
) abort(reset or dataValid);
```

Example : LVDS receiver

checking data

```
assert always(
    dataValid |-> dataOut = realDataOut
) abort(reset);
```

- Issue here : `realDataOut`
- Some pieces of VHDL code to build the 8-bit vector during transmission
  - It would be elegant to build a property for that
  - Feasible in theory, but not available with the Open Source tools

Example : Address decoder

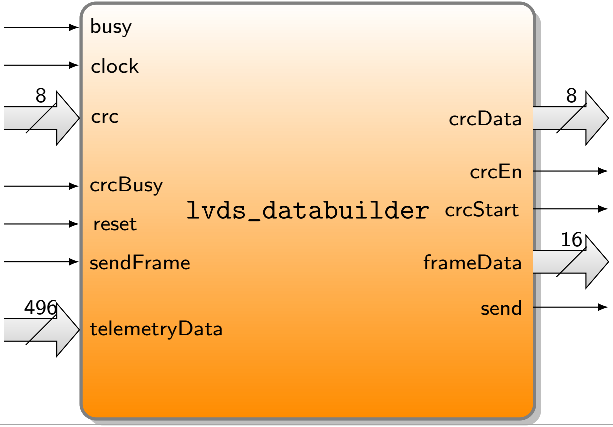
```
sequence start_transmission is {not(receiving);
    (rxDataValid and unsigned(rxData) = TC_START_OF_FRAME)};

sequence data_valid_clk is {not(rxDataValid)[*2];rxDataValid};

sequence end_transmission is
    {start_transmission;data_valid_clk[*7]};

assert always start_transmission |-> startCRC before enByte;
```

Example : Address decoder

- When a new transmission ends, ack_tc is up only if the telecommand is valid, otherwise it is down. (requires a reference)
- The output ack_tc is stable while no new transmission ended

```
assert_valid_ack: assert always
    {end_transmission;[*2]}
        |=>
    {ack_tc = ref_valid_telecommand};
assert_stable_ack: assert always {
    fell(reset) | {end_transmission;[*2]}}
        |=>
    stable(ack_tc) until_ end_receiving;
```

Example : LVDS data builder



Example : LVDS data builder

Sending starts with `AAAA`, then 31 data + CRC

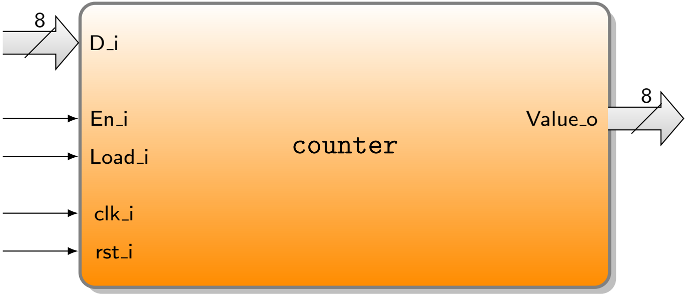
Example : LVDS data builder

```
property frameData_send_eq_telemetryData is
    always(
        {send and not(send_aaaa_s) and not(send_crc_s)}
        |->
        {frameData = telemetryData(
                         telemetryData’high-cnt_tm_bit_send_s downto
                         telemetryData’high-cnt_tm_bit_send_s-15)});
assert property frameData_send_eq_telemetryData;

f_input_data_stay_stable : assume
    always({stable(telemetryData)});
```

- cnt_tm_bit_send_s is a counter generated by a process
- send_aaaa_s and send_crc_s are also generated

SymbiYosis wins one set

Against QuestaFormal



If the counter decrements once after 1'000'000 clock cycles !

Conclusion

- Formal is a good complementary approach to functional simulation
- A good way for thinking differently with respect to design
  - Excellent for control paths
  - More difficult for data paths
- Open source tools and PSL are quite OK compared to commercial solutions
  - Less powerful, specially with no local variables in properties
- Always try to have generic parameters instead of hardcoded constants
  - Modular and reusable code (standard)
  - Allows to reduce the search space for formal proofs

Thanks

- Alberto Dassatti
- Enrico Petraglio
- Roberto Rigamonti
- Clément Dieperink
- Loïc Fournier

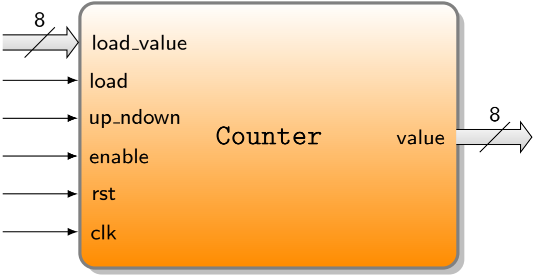
Questions ?

Supplementary material

Verification of a counter

DUV

- 8-bit counter
  - Parallel load
  - Incrementing
  - Decrementing
  - Hold



Verification of a counter

Testbench

1. Directed tests
    - Defined scenarios with borderline cases
2. Random generation of inputs
    - Coverage for verifying all borderline cases have been covered
        - Max to 0, or 0 down to Max

Verification of a counter

Properties to check

- `p_load` : If `load` is active, at the next clock cycle the output shall be the current input
- `p_hold` : If `load` and `enable` are active, the counter shall keep its value
- `p_incr` : If `load` is inactive, `enable` is active and `up_ndown` is active, the counter shall be incremented
- `p_decr` : If `load` is inactive, `enable` is active and `up_ndown` is inactive, the counter shall be decremented

Verification of a counter

Assertions

```
// load operation
assert_load: assert property (@( posedge clk) disable iff (rst==1)
    (load==1) |=> value == $past(load_value));

// maintain operation
assert_maintain: assert property (@( posedge clk) disable iff (rst==1)
    (load==0) & (enable==0) |=> value == $past(value));

// increment operation
assert_increment: assert property (@( posedge clk) disable iff (rst==1)
    (load==0) & (enable==1) & (up_ndown==1) |=> value == ($past(value)+1)%256);

// decrement operation
assert_decrement: assert property (@( posedge clk) disable iff (rst==1)
    (load==0) & (enable==1) & (up_ndown==0) |=> value == ($past(value)-1)%256);
```

Verification of a counter

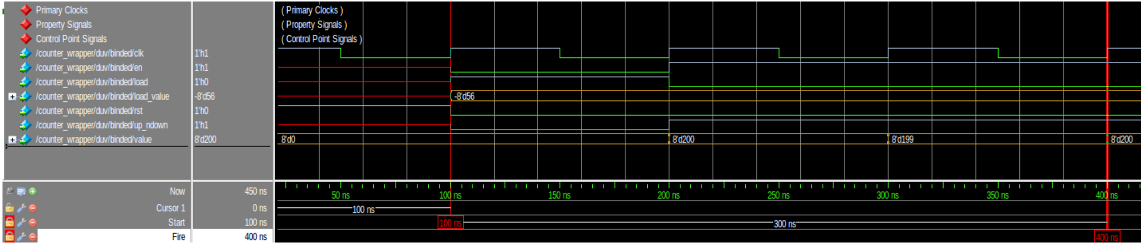
Assertions

```
‘define ASSERT_PROP(p) assert property (@( posedge clk) disable iff (rst==1) p );

// load operation
assert_load: ‘ASSERT_PROP( (load==1) |=> value == $past(load_value) )

// maintain operation
assert_maintain: ‘ASSERT_PROP( (load==0) & (enable==0) |=> value == $past(value) )

// increment operation
assert_increment: ‘ASSERT_PROP( (load==0) & (enable==1) & (up_ndown==1) |=>
                                value == ($past(value)+1)%256 )

// decrement operation
assert_decrement: ‘ASSERT_PROP( (load==0) & (enable==1) & (up_ndown==0) |=>
                                value == ($past(value)-1)%256 )
```

Example of an error

Injected in the design

- Injected error : If value == 200, then decrement instead of increment

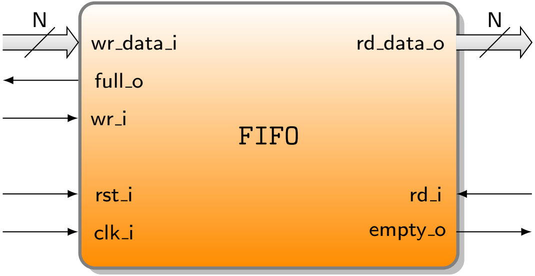


- Scenario : Load 200, then increment

Verification of a FIFO

DUV

- Simple FIFO
- Synchronous read/write
- Two outputs for the FIFO status (empty/full)



Verification of a FIFO

Testbench

1. Directed tests
   - Checks what happens when the FIFO is empty
   - Checks what happens when the FIFO is full
2. Random tests
   - Random commands
   - Coverage to be sure every relevant condition has been observed
     - Simultaneous read/write

Verification of a FIFO

Formal proof

- Properties to verify
  - P1. If the number of writes equals the number of reads, then `empty_o` shall be active
  - P2. If the difference between the number of writes and the number of reads equals the FIFO size, then `full_o` shall be active
  - P3. The nth data written shall be the nth to go out
  - P4. `full_o` and `empty_o` shall never be active at the same time

Verification of a FIFO

Assumptions

```
// The following disables reads when the FIFO is empty
assume property ( @(posedge clk_i) (!(rd_i & empty_o)));

// The following disables writes when the FIFO is full
assume property ( @(posedge clk_i) (!(wr_i & full_o)));
```

Verification of a FIFO

- *Naive* assertion

```
assert_not_empty_after_write :
    assert property ( @(posedge clk_i)  (
        (wr_i) |=> !empty_o));
```

Verification of a FIFO

Useful counters

```
int wcnt = 0;
int rcnt = 0;

always @(posedge clk_i or posedge rst_i)
    if (rst_i)
        wcnt = 0;
    else if (wr_i)
        wcnt = (wcnt + 1);

always @(posedge clk_i or posedge rst_i)
    if (rst_i)
        rcnt = 0;
    else if (rd_i)
        rcnt = (rcnt + 1);
```

Verification of a FIFO

```
property p_full;
    @(posedge clk_i)
        ( full_o == (wcnt == rcnt + FIFOSIZE) );
endproperty

property p_empty;
    @(posedge clk_i)
        ( empty_o == (wcnt == rcnt) );
endproperty

property p_data_integrity;
    int cnt;
    logic[DATASIZE-1:0] data;
    @(posedge clk_i)
        (wr_i, cnt=wcnt, data=data_i) |=>
            (([0:$] (rd_i & (rcnt==cnt))) |->
             (data_o==data));
endproperty
```

Verification of a FIFO

State explosion

```
assume property ( @(posedge clk_i) (wcnt < 4*FIFOSIZE));
```

Errors found

- Signals initialized at declaration

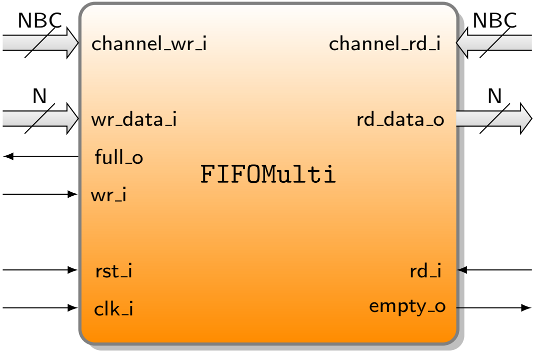
```
signal mon_signal : std_logic := '0';
```

- Problems with delay of data at the output

Verification of a multi FIFO

DUV

- Multi FIFO (multi-channel)
- Synchronous read/write commands
- Two outputs for the FIFO status (empty/full)
- A channel number to define which FIFO to access
    - Write on channel `channel_wr_i`
    - Read on channel `channel_rd_i`



Verification of a multi FIFO

Formal proof

- Quite similar to the simple FIFO
  - Use of *generate* for channels assertions

Verification of a multi FIFO

```
generate

    genvar channel_var;

    for (channel_var = 0; channel_var < NBFIFOS; channel_var = channel_var + 1)
    begin : channel

        property p_full;
        @(posedge clk_i)
          (channel_wr_i == channel_var) |->
            ( full_o == (wcnt[channel_var] == rcnt[channel_var] + FIFO_DEPTH_G) );
        endproperty

        assert_full : assert property (p_full);

    end

endgenerate
```

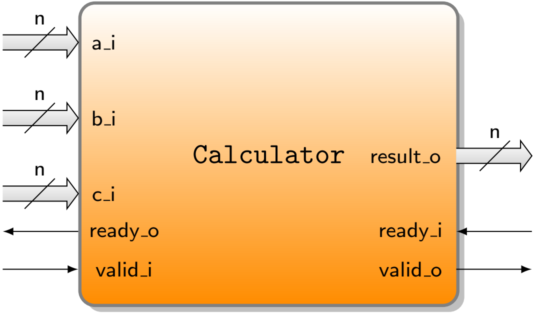
Errors found

- Error if a read command and a write command occured on the same channel at the same time

Calculator

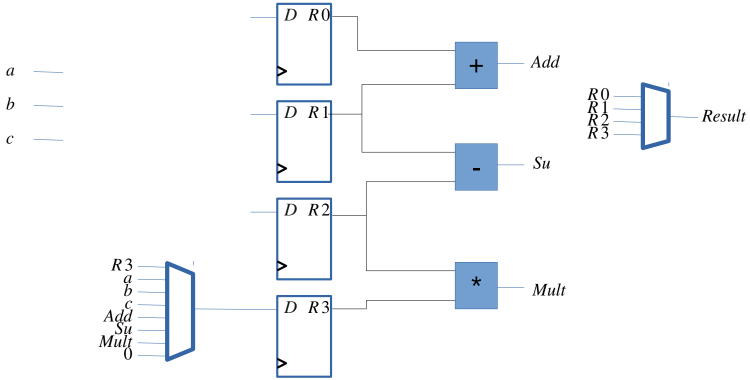
DUV

- A module able to compute the following function :
- $F(a, b, c) = a + a + b - c$
    - Could be any kind of arithmetic computation
- Different architectures :
    - Datapath-control
    - Pipelined
- A single testbench to validate different architectures



Verification of a calculator

Datapath



Verification of a calculator

Formal proof

- Properties to verify
  - P1. When a computation is launched, then after a certain time the result shall be available at the output
  - P2. The result of a computation shall correspond to the calculation (verification of data)
    - The nth result shall correspond to the nth input data
  - P3. When no computation is under way, then `ready_o` shall be active

Property 1

```
property p_valid_eventually;
    @(posedge clk_i)
    (valid_i && ready_o)
        |=>
        ##[0:100] (valid_o);
endproperty
```

2 counters

```
int wcnt = 0;
int rcnt = 0;

always @(posedge clk_i or posedge rst_i)
    if (rst_i)
        wcnt = 0;
    else if (valid_i && ready_o)
        wcnt = (wcnt + 1);

always @(posedge clk_i or posedge rst_i)
    if (rst_i)
        rcnt = 0;
    else if (valid_o && ready_i)
        rcnt = (rcnt + 1);
```

Property 2

```
property p_ready_once;
    @(posedge clk_i)
    (rcnt == wcnt) |-> (ready_o);
endproperty
```

Property 3

```
property p_data_integrity;
    int cnt;
    logic[DATASIZE-1:0] a;
    logic[DATASIZE-1:0] b;
    logic[DATASIZE-1:0] c;
    @(posedge clk_i)
        (valid_i && ready_o,
        cnt = wcnt,
        a = a_i,
        b = b_i,
        c = c_i
        ) |=>
            ((##[0:$] (valid_o && ready_i & (rcnt==cnt))) |->
            (result_o == a + a + b - c));
endproperty
```

Some limitations

Issues with QuestaFormal

```
always_ff @(posedge clk, posedge rst) begin
    if (rst == 1) begin
        count <= 0;
        internal_count <= 0;
    end
    else begin
        internal_count <= internal_count + 1;
        if (load == 1)
            count <= load_value;
        else begin
            if (en == 1) begin
                if (up_ndown == 1)
                    count <= count + 1;
                else
                    count <= count - 1;
                if (internal_count == 10000000)
                    count <= count -1;
            end
        end
    end
end
```

Synthesis of properties

- Some properties can be synthesized
  - For verification
    - Embedded in an emulation
  - For design
    - Generates a design compliant with the properties

Synthesis of a design

Automata approach

- European project PROSYD (`http://www.prosyd.org/`), 2004-2006
- From PSL properties, build a Mealy machine that matches the properties
  1. A two-player game played between a system and an environment
  2. The game structure is a multi-graph that represents the input variables and the state variables
     - Each node represents a combination of input and state variables that are valid
     - Each edge represents the transitions from a set of input and state variables to another set of valid input and state variables.
  3. If the environment wins, then the specification is *unrealizable*, and if the system wins, they extract a BDD representing this system
  4. In case the specification is *realizable*, the corresponding BDD is synthesized.
- Use cases : A generalized buffer and an AMBA bus arbiter
- Issues : Size of the design (and applicability)

Synthesis of a design

Modular approach

- Dominique Borrione and her team (Grenoble, France)
- Based on a library of monitors and generators
  - Corresponding to PSL operators
- Monitors detect correct or wrong behavior
- Generators produce sequences
- Allows to interpret LTL-style and SERE-style assertions
- Way more efficient in terms of size