

Design of Embedded Hardware and Firmware *Simulation, Verification & Debugging*

Andres Upegui

Laboratoire de Systèmes Numériques
hepia/HES-SO
4, rue de la Prairie, CH-1202 Genève

Andres.uegui@hesge.ch

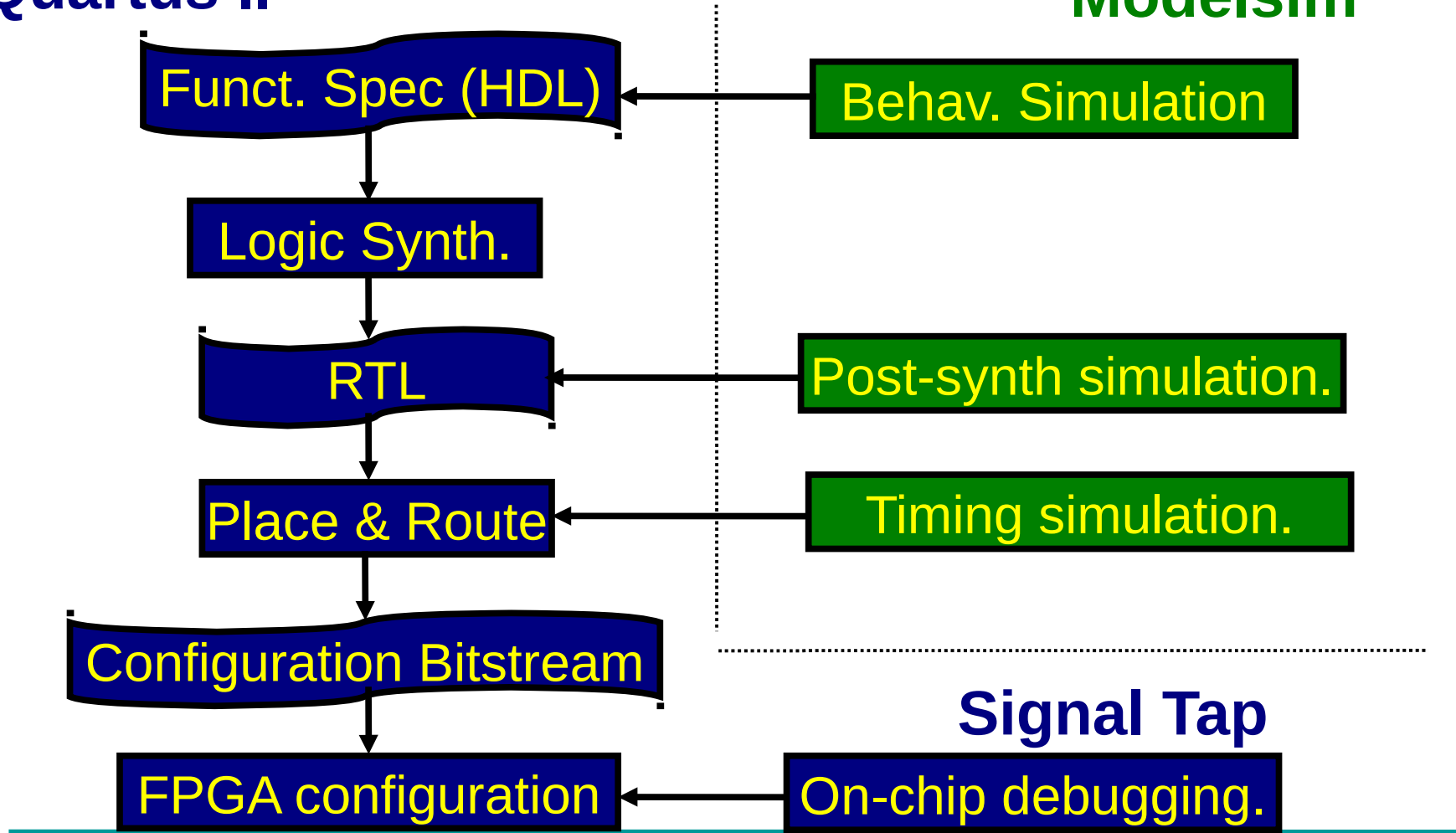
Simulation, Verification & Debugging... Why ?

- Complexity enhancement increases the probabilities of inserting bugs in your system.
- After describing a hardware architecture using VHDL we need to make sure it works. Several options are possible:
 - ➔ Trial and error... really an option?
 - ➔ Simulation or manual verification
 - ➔ Automatic verification
 - ➔ On-chip debugging

Conventional design flow

Quartus II

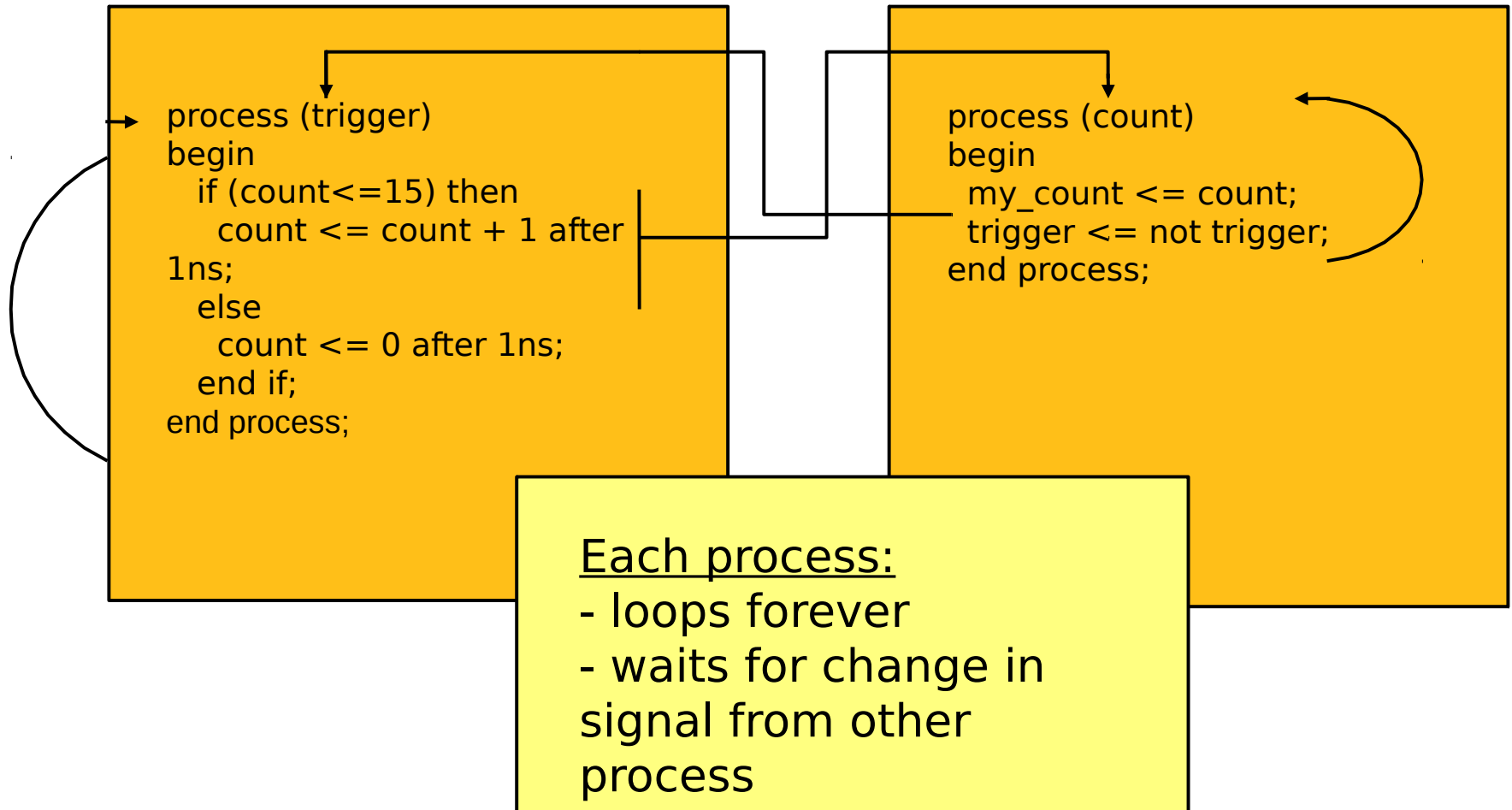
Modelsim



Simulation or manual verification

- Modelsim allows you to perform a simple initial functional verification of your system with a few simple steps:
 - Compile your VHDL code.
 - Write some input stimuli from the command line.
 - Select the signals you want to observe.
 - Run simulation.
 - Analyze the resulting time-diagram.

Event-driven simulation



Simulation or manual verification (2)

- Command lines can also be scripted as a .do file.
- Script exemple:

```
restart
```

```
force clk 1 0, 0 10ns -repeat 20ns
```

```
force reset 1 0, 0 100ns
```

```
force avalon_address XX 0, 00 205ns, XX 225ns, 01 405ns,  
XX 425ns
```

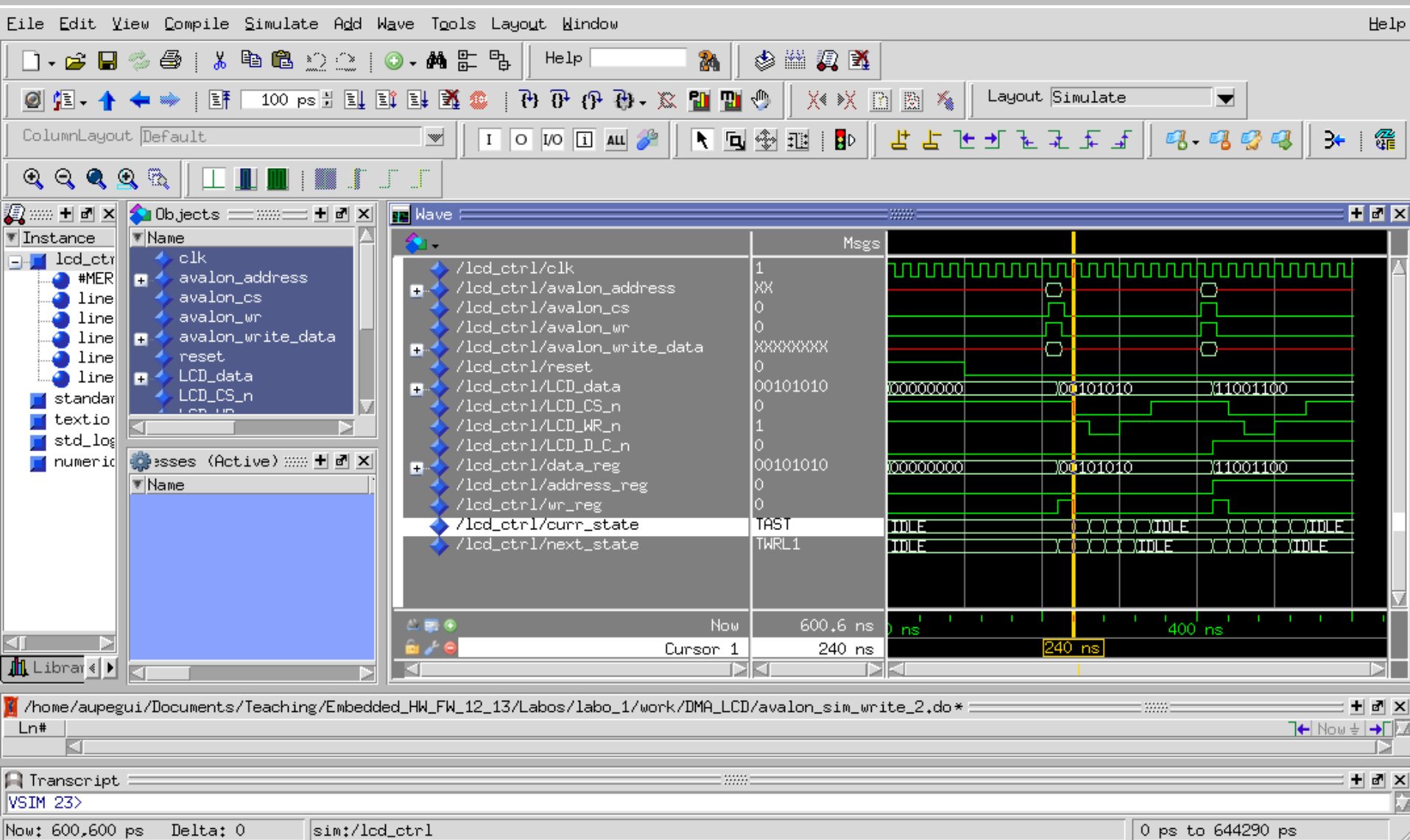
```
force avalon_write_data 16#XX 0, 16#2A 205ns, 16#XX 225ns,  
16#CC 405ns, 16#XX 425ns
```

```
force avalon_wr 0 0, 1 205ns, 0 225ns, 1 405ns, 0 425ns
```

```
force avalon_cs 0 0, 1 208ns, 0 228ns, 1 405ns, 0 425ns
```

```
run 600ns
```

Simulation or manual verification (3)



Simulation or manual verification (4)

Advantages:

- fast and easy to setup and use.

Disadvantages:

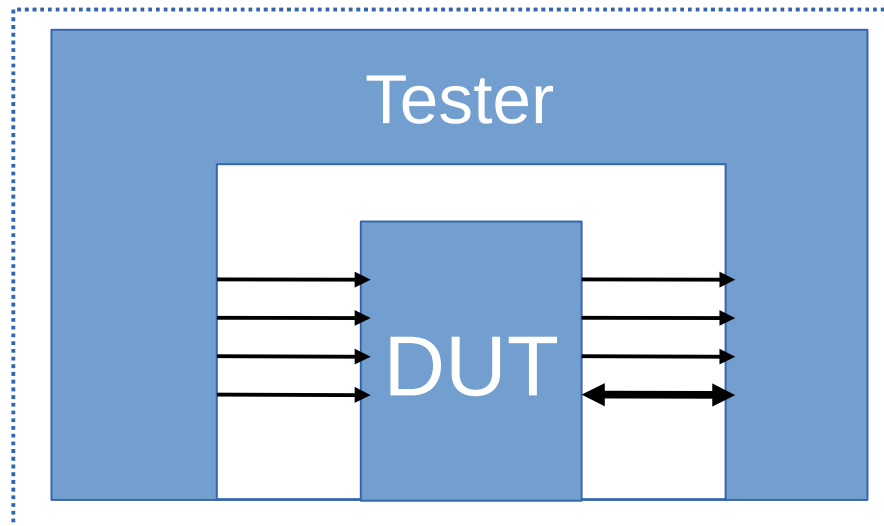
- “human-in-the-loop” verification process.
- It can be very tedious for manually generating every test to be evaluated.
- It is up to you to verify behavior correctness.
- Complex designs may have many signals to analyze... and should be run for a long time.

A smarter approach is needed...

Automatic verification : Testbench

- A testbench can be used for generating input stimuli and verifying responses through output signals.

Testbench



DUT = Design Under Test

Testbench : signal generation

- Input signals for DUT are force by the tester.
- Using the `wait` and `after` commands is almost mandatory.
- Example:

```
constant CLK_PERIOD: time:=40 ns;
```

```
a<='0', '1' after 10 ns, '0' after 20 ns, '1' after 60 ns;  
b<='0', '1' after CLK_PERIOD, '0' after 2*CLK_PERIOD;
```

```
generateur: process  
begin  
    c<='0';  
    wait for 10ns;  
    c<='1';  
    wait for 10ns;  
    c<='0';  
    wait for 40ns;  
    c<='1';  
    wait;  
end process;
```

Testbench : signal generation (2)

- Example: reset and clock generation

```
constant CLK_PERIOD: time:=40 ns;  
signal nreset: std_logic;  
...
```

```
reset_process: process  
begin  
    nreset<='0';  
    wait for CLK_PERIOD;  
    nreset<='1';  
    wait;  
end process;
```

```
constant CLK_PERIOD: time:=40 ns;  
signal clk: std_logic;  
...
```

```
clk_process: process  
begin  
    clk<='0';  
    wait for CLK_PERIOD/2;  
    clk<='1';  
    wait for CLK_PERIOD/2;  
end process;
```

Testbench : assertions

- The `assert` command permits to evaluate conditions and display a diagnostic message during a simulation. It is thus possible to verify whether an affirmation is true or false. If the affirmation is false, we refer to it as an assertion violation and a message is generated.

Syntaxis

```
assert boolean_expression  
report expression  
severity level; -- the severity_level type is defined in the STD library.
```

```
type severity_level is (note, warning, error, failure);
```

Exemple :

```
assert result=15 report "incorrect result" severity error;
```

Andres

Exemple 1: Testbench for ALU

-- Testbench for ALU able to perform +, -, **xor** and **and** operations

library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_arith.all;

use ieee.std_logic_unsigned.all;

entity simple_alu_tb is -- empty entity, no inputs neither outputs

end simple_alu_tb;

architecture behavioral of simple_alu_tb is

component simple_alu -- component declaration

port(

A : in std_logic_vector(3 downto 0);

B : in std_logic_vector(3 downto 0);

operation : in std_logic_vector(1 downto 0);

output : out std_logic_vector(3 downto 0);

equal : out std_logic

);

end component;

Exemple 1: ALU (2)

-- signal declaration

```
signal A : std_logic_vector(3 downto 0);  
signal B : std_logic_vector(3 downto 0);  
signal operation : std_logic_vector(1 downto 0);  
signal output : std_logic_vector(3 downto 0);  
signal equal : std_logic;
```

begin

```
simple_alu_to_test: simple_alu  
port map(  
    A => A,  
    B => B,  
    operation => operation,  
    output => output,  
    equal => equal  
);
```

-- component instantiation

Exemple 1: ALU (3) – stimuli generation

```
-- stimuli generation
```

```
process
```

```
begin
```

```
    for op in 0 to 3 loop
```

```
        for i in 0 to 15 loop
```

```
            for j in 0 to 15 loop
```

```
                A<=conv_std_logic_vector(i,4);
```

```
                B<=conv_std_logic_vector(j,4);
```

```
                operation<=conv_std_logic_vector(op,2);
```

```
            wait for 10 ns;
```

Exemple 1: ALU (4) – assertion verification

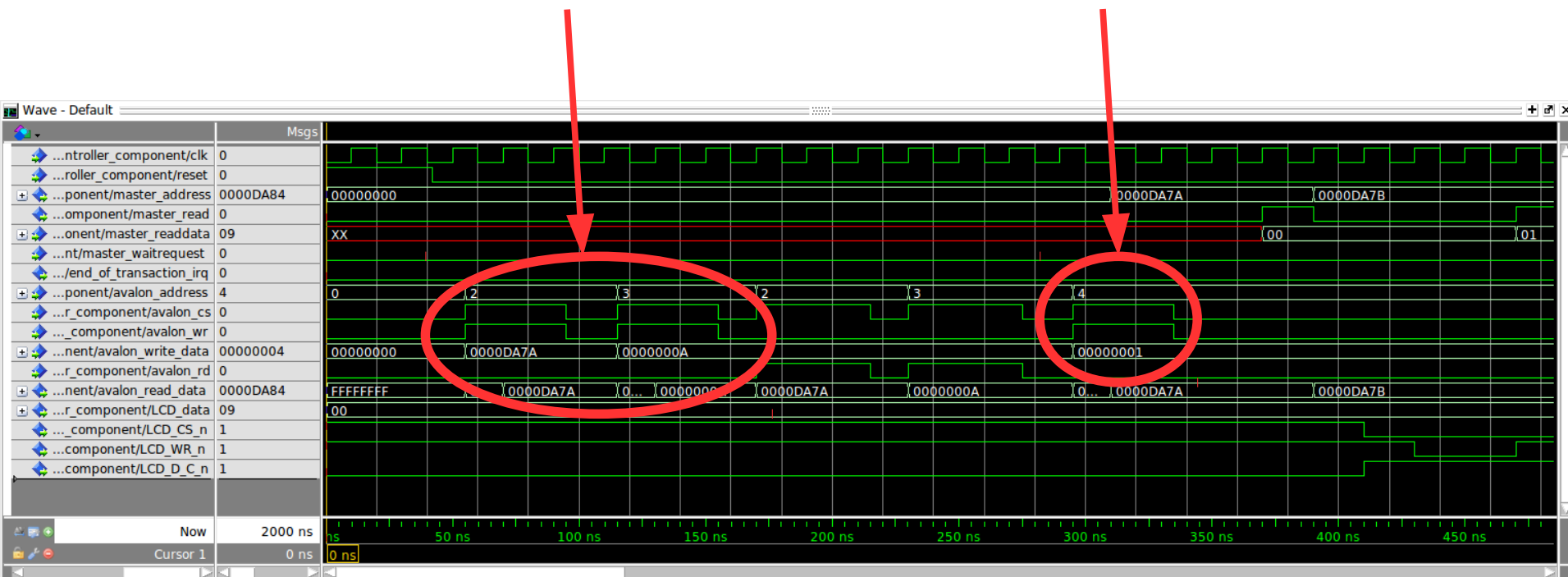
```
case op is          -- assertion verification
  when 0 =>
    assert output=A+B report "Error in addition" severity error;
  when 1 =>
    assert output=A-B report "Error in subtraction" severity error;
  when 2 =>
    assert output=(A xor B) report "Error in xor" severity error;
  when 3 =>
    assert output=(A and B) report "Error in and" severity error;
  when others=>
    report "Error in operation";
end case;
assert (A=B) xor (equal='0') report "Equality error" severity error;
end loop;
end loop;
end loop;
wait;
end process;

end behavioral;
```


Exemple 2: DMA-LCD (1) – Configuring DMA transfer

Configuring transfer : setting source address to 0xDA7A and transfer size to 0xA

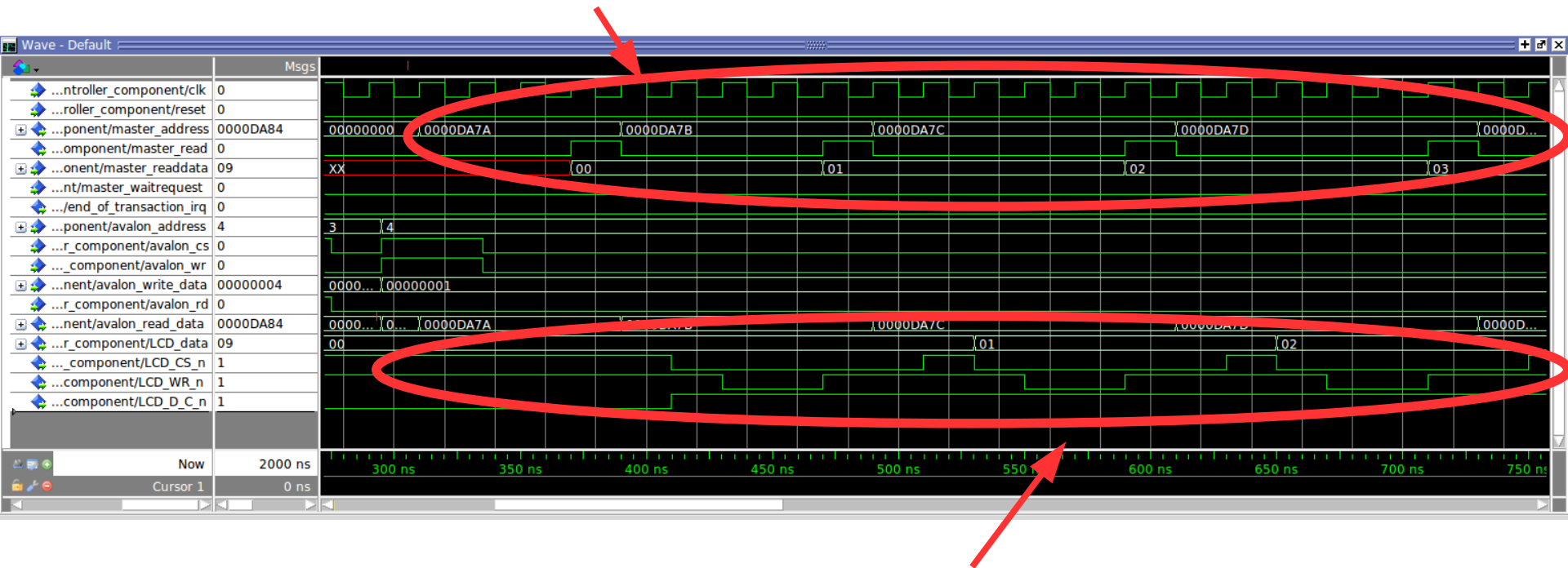
Enabling transfer: setting bit 0 of reg 4 to 1



NOTE : The code for this testbench is available on the [moodle webpage](#)

Exemple 2: DMA-LCD (2) – Performing DMA transfer

Avalon master : Consecutive read access from the memory.

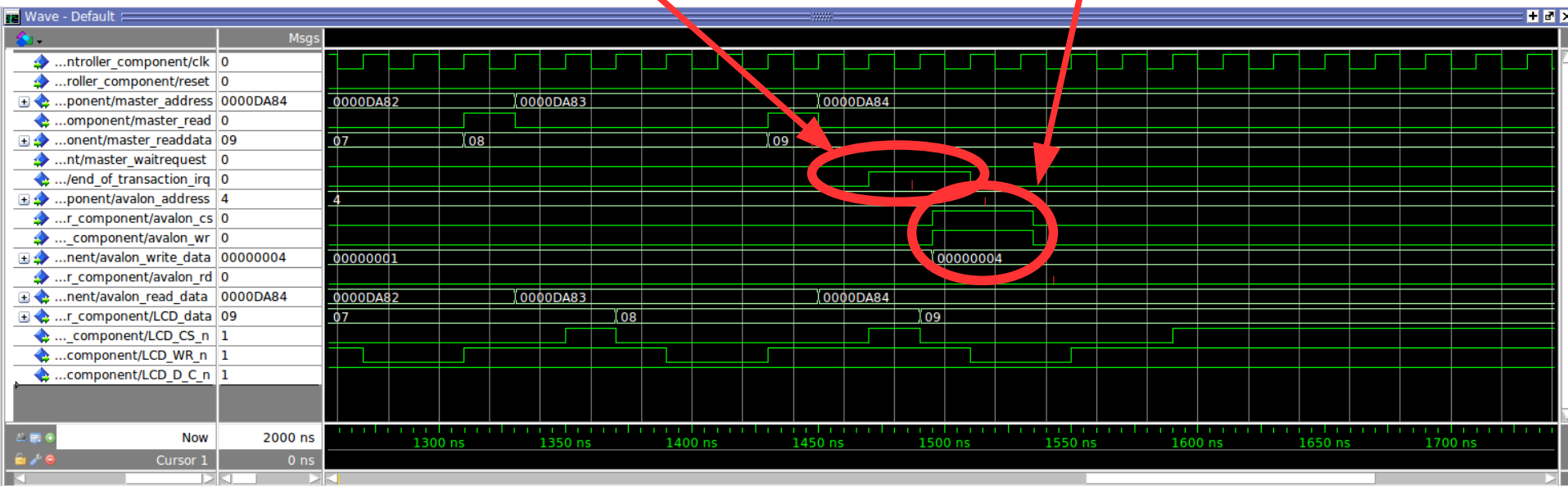


Consecutive writings to the LCD

Exemple 2: DMA-LCD (3)– Completing DMA transfer

IRQ generation at the end of the transfer.

IRQ ack : writing 1 to the bit 2 on register 4



NOTE : The IRQ signal must be maintained until an acknowledge is done by the processor

Exemple 2: DMA-LCD (3)– Assertions

```
assert not(dma_access_cnt < DMA_TEST_IMG_SIZE) report "LCD DMA  
test: number of dma accesses is lower than image size"  
severity error;
```

```
assert end_of_transaction_irq_s = '0' report "LCD DMA test:  
IRQ flag couldn't be cleared " severity error;
```

```
assert lcd_cs_n_s = '0' report "LCD bus monitor: timing error,  
lcd_cs_n_s should be low " severity error;
```

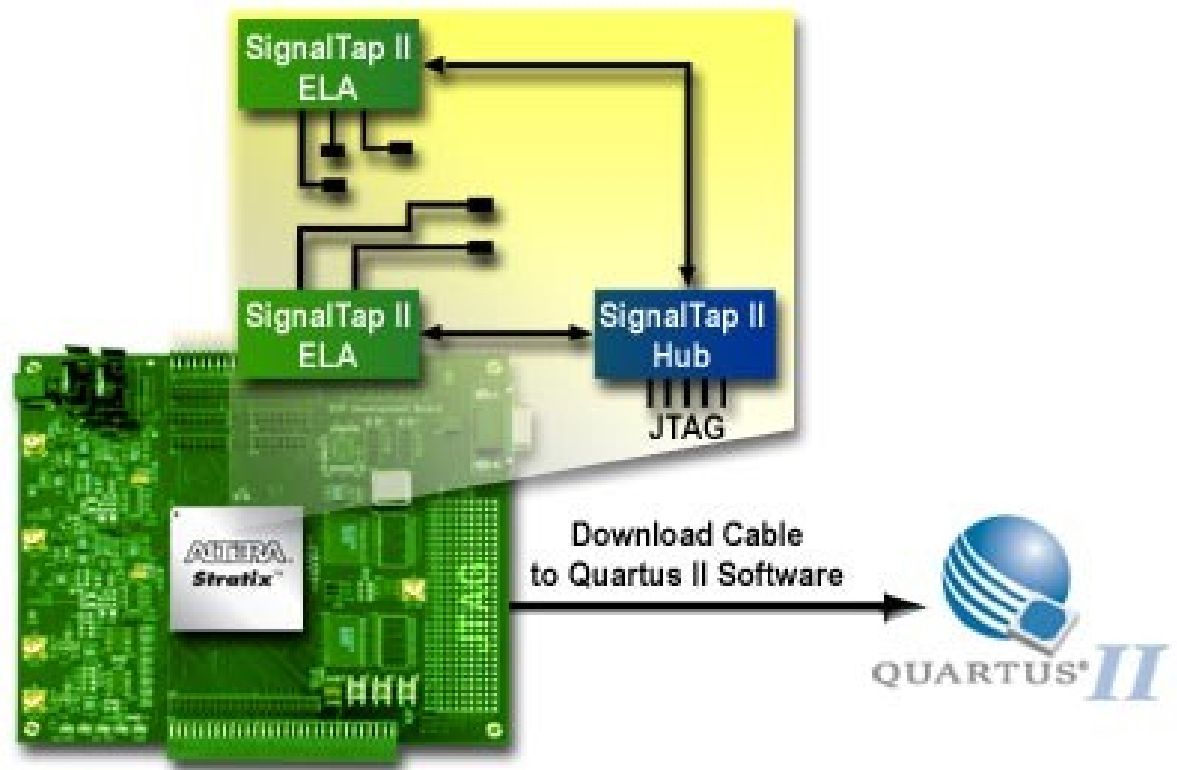
```
assert lcd_wr_n_s = '1' report "LCD bus monitor: timing error,  
lcd_wr_n should be high" severity error;
```

On-chip debugging : Signal Tap logic analyzer

- Captures the Logic State of FPGA Internal Signals Using a Defined Clock Signal
- Gives Designers Ability to Monitor Buried Signals
- Connects to Quartus II through FPGA JTAG Pins
- Captures Real-Time Data
 - Up to 200 Mhz
- Is Available for Free
 - Installed with Full Subscription or Web Edition
 - Installed with Stand-Alone Programmer

On-chip debugging : Signal Tap logic analyzer

- Embedded Logic Analyzer
- Downloads into Device with Design
- Captures State of Internal Nodes
- Uses JTAG for Communication



STP sampling and trigger setup

Select sampling clock
(usually system clk)

Select sampling buffer size and
memory resources
(how long is the time window you need?)

Trigger options
(default use to be ok...)

The screenshot shows the 'Signal Configuration' dialog box with the following settings:

- Clock:** CLOCK
- Data:**
 - Sample depth:** 128
 - Nodes allocated:** Auto (selected), Manual: 12
 - RAM type:** M4K
- Buffer acquisition mode:**
 - Circular:** Pre trigger position
 - Segmented:** 128 1 bit segments
- Trigger:**
 - Trigger levels:** 1
 - Nodes allocated:** Auto (selected), Manual: 8
 - Trigger in:**
 - ☒ Trigger in:
 - Source:** auto_stp_trigger_in_1
 - Pattern:** Don't Care
 - Trigger out:**
 - ☒ Trigger out:
 - Target:** auto_stp_trigger_out_1
 - Level:** Active High
 - Latency delay:** 4 cycles

Arrows from the text blocks point to the following elements in the dialog:

- Arrow from 'Select sampling clock' points to the 'Clock' field.
- Arrow from 'Select sampling buffer size and memory resources' points to the 'Sample depth' field.
- Arrow from 'Trigger options' points to the 'Trigger in' section.

STP File Waveform Viewer

counter.stp*

Instance Manager: Ready to acquire

Instance	Status	Incremental Compilation	LEs: 1998	Memory:
instance_one	Not running	<input type="checkbox"/>	729 cells	716
instance_ten	Not running	<input type="checkbox"/>	1269 cells	1536

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0] Setup...

Device: @1: EP1S25/_HARDCOPY_FPGA_PRC Scan Chain

SOF Manager: counter_stratix_es.sof

Signal Configuration:

Buffer acquisition mode:

☐ Circular: Center trigger position

trigger: 2005/05/17 20:33:27 #0 Lock mode: Allow all changes

Type	Alias	Node	Incremental Route	Debug Port Out	Data Enable 7/Auto	Trigger Enable 7/Auto	Trig...
		ONE_SEG	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	zer
		ONE_SEG[6]	<input type="checkbox"/>	-K3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ONE_SEG[5]	<input type="checkbox"/>	-K3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ONE_SEG[4]	<input type="checkbox"/>	-K3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ONE_SEG[3]	<input type="checkbox"/>	-K3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ONE_SEG[2]	<input type="checkbox"/>	-K3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ONE_SEG[1]	<input type="checkbox"/>	-K3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ONE_SEG[0]	<input type="checkbox"/>	-K3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Setup

Hierarchy Display: counter

instance_one instance_ten

counter.stp*

Instance Manager: Ready to acquire

Instance	Status	Incremental Compilation	LEs: 1998	Memory:
instance_one	Not running	<input type="checkbox"/>	729 cells	716
instance_ten	Not running	<input type="checkbox"/>	1269 cells	1536

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0] Setup...

Device: @1: EP1S25/_HARDCOPY_FPGA_PRC Scan Chain

SOF Manager: counter_stratix_es.sof

log: 2005/05/17 20:36:16 #0 click to insert time bar

Type	Alias	Node	0	1	2	3	4	5	6	7	8
		ONE_SEG	zero_ones	zero_ones	zero_ones	zero_ones	zero_ones	zero_ones	zero_ones	zero_ones	zero_ones
		ONE_SEG[6]									
		ONE_SEG[5]									
		ONE_SEG[4]									
		ONE_SEG[3]									
		ONE_SEG[2]									
		ONE_SEG[1]									
		ONE_SEG[0]									

Data

Hierarchy Display: counter

Data Log: instance_one

instance_one instance_ten

Setup Tab

Data Tab

STP Triggering

All Signals Must Be
True for Level to
Cause Data Capture

trigger: 2003/10/01 18:44:12 #1 Lock mode: Allow all changes

Node			Incremental Route	Debug Port Out	Data Enable 11/15	Trigger Enable 7/15	Trigger Levels				
Type	Alias	Name					1 ✓ Basic	2 ✓ Basic	3 ✓ Basic	4 ✓ Basic	5 ✓ Basic
		TEN_SEG	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	zero_tens	one	two	three	four
		TEN_SEG[6]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	1	0	0	1
		TEN_SEG[5]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	0	0	0	0
		TEN_SEG[4]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	0	1	0	0
		TEN_SEG[3]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	1	0	0	1
		TEN_SEG[2]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	1	0	1	1
		TEN_SEG[1]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	1	1	1	0
		TEN_SEG[0]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	1	1	0	0
		...ix:wysi_counter[safe_q[3]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>					
		...ix:wysi_counter[safe_q[2]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>					
		...ix:wysi_counter[safe_q[1]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>					
		...ix:wysi_counter[safe_q[0]	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>					

Right-Click to Set Value

Don't Care

Low

Falling Edge

Rising Edge

High

Either Edge

Insert Value...

STP data acquisition

Run, Autorun, Stop



Format in Sample Number

