MARIE CURIE IAPP: FAST TRACKER FOR HADRON COLLIDER EXPERIMENTS

1ST SUMMER SCHOOL: VHDL BOOTCAMP

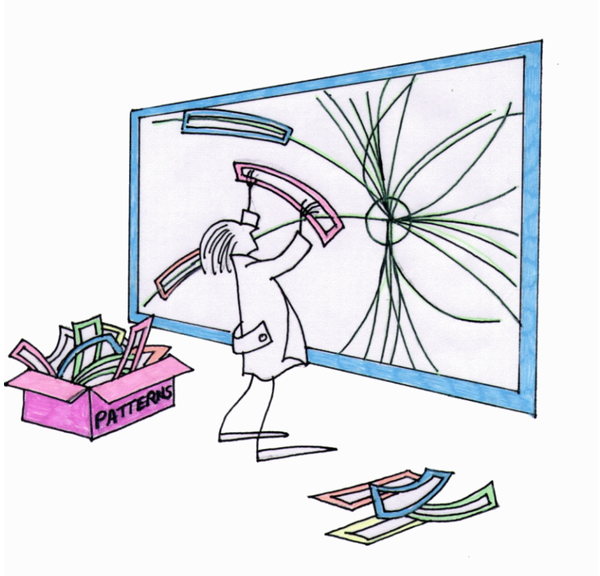
PISA, JULY 2013

Simulation with Testbenches

Calliope-Louisa Sotiropoulou

PhD Candidate/Researcher

Aristotle University of Thessaloniki



AUTH e-LAB

Aristotle University of Thessaloniki-Electronics Laboratory

IN SUPREMÆ DIGNITATIS 1343

UNIVERSITÀ DI PISA

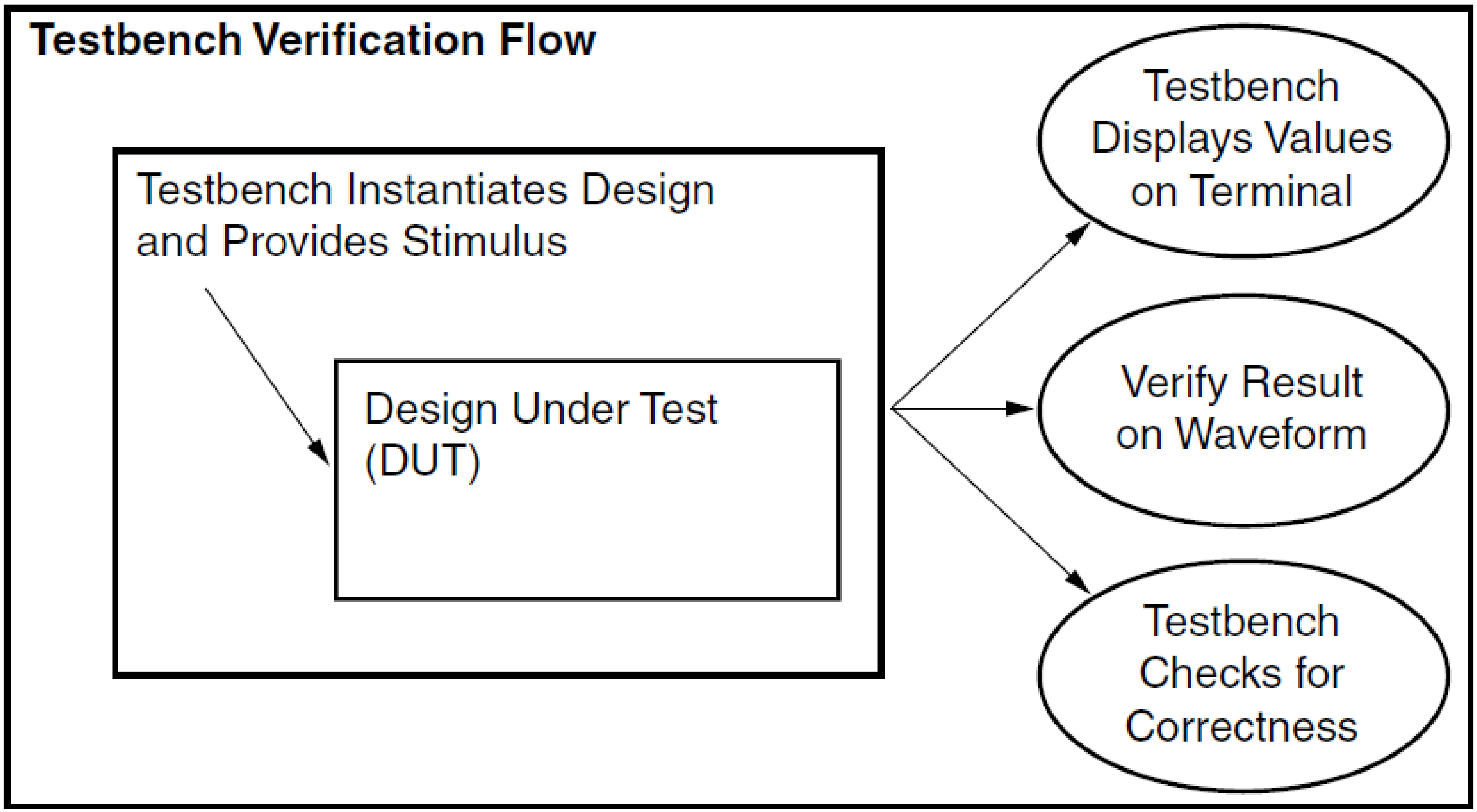
Simulation with Testbenches

- Testbenches
- Modeling Memory

Verification Flow Using Testbenches

- Instantiate the design under test (DUT)
- Stimulate the DUT by applying test vectors to the model
- Output results to a terminal or waveform window for visual inspection
- Compare actual results to expected results

Verification Flow Using Testbenches



Testbenches

- Testbenches can be written in VHDL or Verilog.
- Testbenches are used for simulation only, they are not limited by semantic constraints that apply to RTL language subsets used in synthesis.
- All behavioral constructs can be used.
- Testbenches can be written more generically, making them easier to be reused.

Testbench Structure

VHDL
Entity and Architecture Declaration
Signal Declaration
Instantiation of Top-level Design
Provide Stimulus

Simulation Time

- There is one simulation time for the entire simulated system (one **master clock**).
- Simulation is indexed by an integer Tc.
- Units are in seconds.
- A default resolution is specified when the simulator is invoked
- Simulation time begins at 0 s.
- Simulation time is advanced by the execution of the simulation cycle to the time of the next queued event.
- A transaction can be scheduled "immediately" using **delta time**.

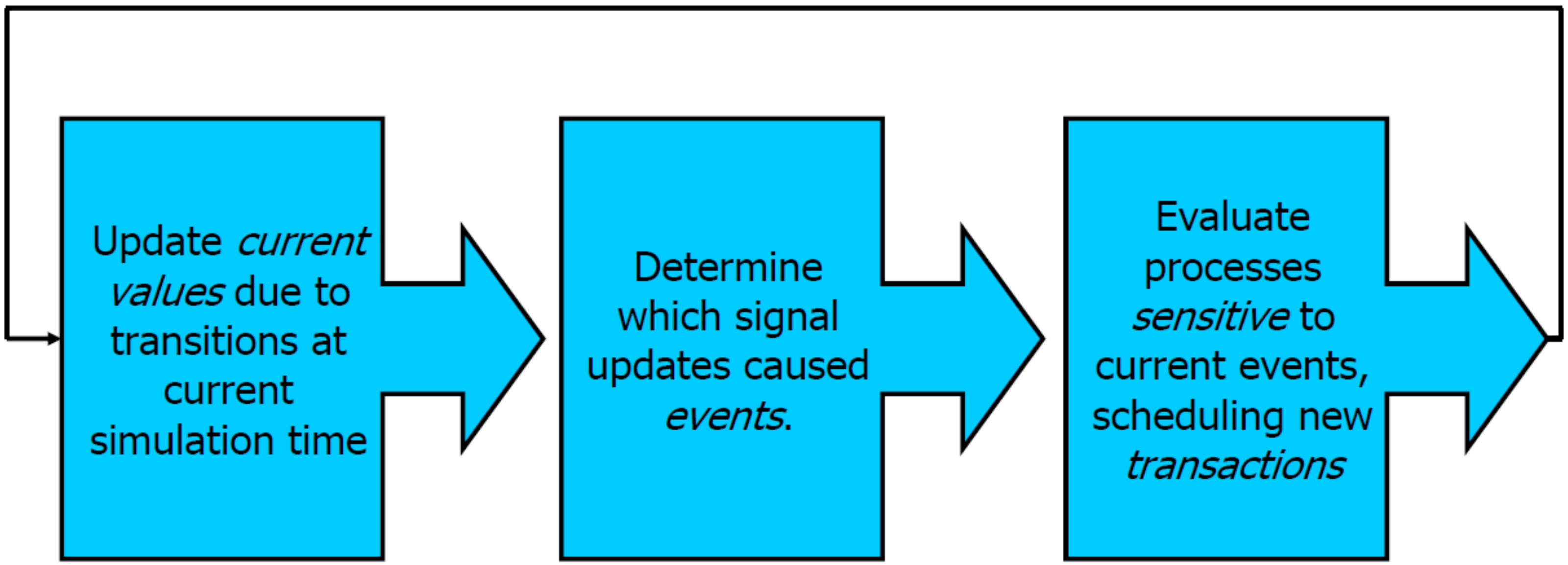
Simulation Terminology

- All signals have a current value associated with them at all times during a simulation
- A **transaction** is an update of the current value of a signal
- During a simulation cycle in which a signal experiences a transaction, the signal is said to be **active**
- An **event** is a transaction that results in a change of value
- Processes can schedule transactions to take place on their output signals, both at future times or "immediately".

Simulation Cycle

- The simulation cycle governs execution of the simulation.
- The simulation cycle is repeated until simulation terminates.
- Steps in the simulation cycle.
  - The current time Tc is assigned the next schedule simulation time Tn.
  - Each active signal is updated with its new value.
  - Processes that are sensitive to signals that have just experienced events are marked to resume during the current simulation cycle, as well as processes scheduled to resume at the current time.
  - Each process that is marked to resume is executed (in no defined order of processes) until (if) it suspends.
  - The next simulation time Tn is calculated according to the next time a signal is schedule to become active or a process is scheduled to resume.
- If Tn=Tc then the next simulation cycle is called a delta cycle.

Simulation Cycle

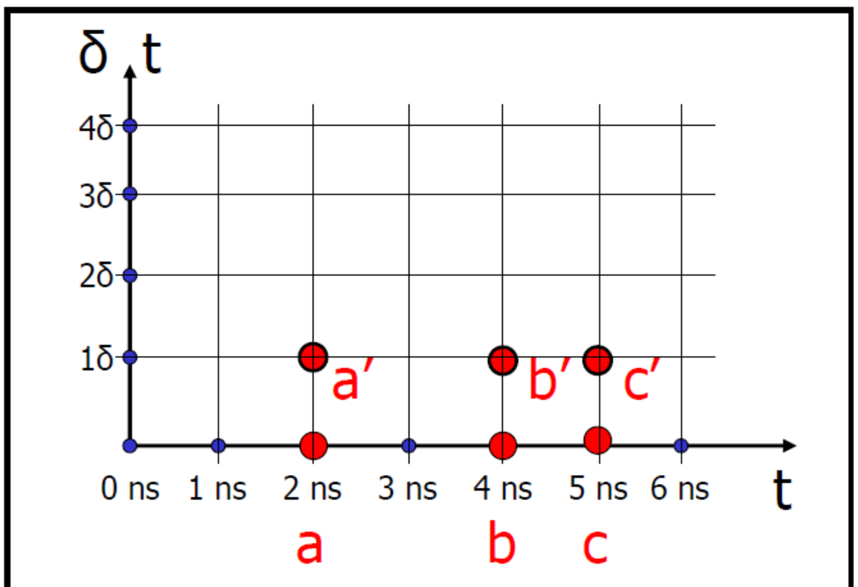


Simulation Delta Time

- Infinitesimally small (δ) advance in time
- The current time Tc does not advance
- An infinite number of delta time steps can occur between tags in the current time Tc
- "Which delta cycle" is not accessible in the language
  → "do something immediately" means do it during the next cycle, which will be a delta cycle
- Provides a means of ordering an event and the events that result from it

Simulation Delta Time

.
.
a b c
x <='1' after 2 ns, '0' after 4ns, '1' after 5 ns
y<=x after 0 ns
.
.



Scheduled Transactions

- y
- x

Simulation Time – Clock Generation

```
-- Declare a clock period constant.
Constant ClockPeriod : TIME := 10 ns;

-- Clock Generation method 1:
Clock <= not Clock after ClockPeriod / 2;

-- Clock Generation method 2:
GENERATE_CLOCK: process
Begin
wait for (ClockPeriod / 2)
Clock <= '1';
wait for (ClockPeriod / 2)
Clock <= '0';
end process;
```

Simulation Time

- “after”:
  Used in concurrent VHDL (signal assignment)

```
sig1 <= sig2 after 10 ns;
clk <= '1' , '0' after TimePeriod/2 ;
sig3 <= transport sig4 after 3 ns;
sig4 <= reject 2 ns sig5 after 3 ns;        -- increasing time order
sig6 <= inertial '1' after 2 ns, '0' after 3 ns , '1' after 7 ns;
```

- “wait”:
  Used in sequential VHDL

```
wait for 10 ns;                   -- timeout clause, specific time delay.
wait until clk='1';               -- condition clause, Boolean condition
wait until A>B and S1 or S2;      -- condition clause, Boolean condition
wait on sig1, sig2;               -- sensitivity clause, any event on any
                                  -- signal terminates wait
```

Providing Stimulus

```
MainStimulus: process begin
Reset <= '1';
Load <= '0';
Count_UpDn <= '0';
wait for 100 ns;
Reset <= '0';
wait for 20 ns;
Load <= '1';
wait for 20 ns;
Count_UpDn <= '1';
end process;
```

Providing Stimulus

```
Process (Clock)
Begin
If rising_edge(Clock) then
TB_Count <= TB_Count + 1;
end if;
end process;
SecondStimulus: process begin
if (TB_Count <= 5) then
Reset <= '1';
Load <= '0';
Count_UpDn <= '0';
Else
Reset <= '0';
Load <= ‘1’;
Count_UpDn <= ‘1’;
end process;
```

Verification Techniques

- Testbenches
- Modeling Memory

Modeling Memory

- When memory is part of a system, we'd like a way to get some contents into our memory model in a fast easy way.
- The contents of memory are often created by some other tool, and thus already exist in electronic form.

↓

FILE I/O

File I/O: Automated Testing

- Instead of creating a testbench which explicitly tries all possible alternatives, a testbench can use FILE I/O to read a "vector file" which contains a list of all the stimulus in tabular form.

- The VHDL testbench can also contain statements that make sure your design is working automatically, and thus doesn't count on you observing the output of the wave window for verification.

File I/O: VHDL as Test Generator

- A VHDL model can export transitions that occur on all signals to a file in any format for use by automated testing equipment
- The system level model can also be used to create stimulus and test assertions for another design tool altogether.

- Testbench can record informative messages as events happen

File I/O: Package Declaration

<u>Package std.TextIO:</u>

- Built in File I/O facility of VHDL is VERY rudimentary, essentially just enough to read or write a file

In Declarative Section of Architecture

```
file cmdfile : text is in "flashfile.txt";
```

file name   type   mode   filename on disk

- Note: no explicit opening / closing of file, it is done automatically

Package TextIO Commands

- procedure readline(f : in text; l : out line);
  →reads a line of text from the file and puts it in l.
- procedure read (l: inoutline;….);
  →used to get elements off of a line of text
- function endfile(f : in text) return boolean;
  →returns true if the end of the file was reached
- Basic flow:
  - open a file (with the declaration)
  - read a line of text
  - use **read, or its derivatives to extract vhdl elements** until the end of the file

Reading with TextIO

- read(l:inoutline; value: out bit; good : out bit)
- read(l:inoutline; value: out bit)

- Variants include this same pair for value types of:
  - bit_vector
  - boolean
  - character
  - integer
  - real
  - string
  - time

- Note : no std_logic_vector type

Writing with TextIO

- procedure write (l : inoutline; value : in bit; justified: in side := right; field : in width := 0);

- Variants include this same pair for value types of:
  - bit_vector
  - boolean
  - character
  - integer
  - real
  - string
  - Time

- All read and writes operate on a "line", and readline and writeline are used to read from and write that line to a file.

Package std_logic_textio

- Same with TextIO with the extension of types
  - std_ulogic
  - std_ulogic_vector
  - std_logic_vector
- Adding the appropriate read and write functions
- Use of HEX numeral system

Octal and Hex commands

- procedure HREAD(L:inoutLINE; VALUE:out STD_ULOGIC_VECTOR);
- procedure HREAD(L:inoutLINE; VALUE:out STD_ULOGIC_VECTOR; GOOD: out BOOLEAN);
- procedure HWRITE(L:inoutLINE; VALUE:in STD_ULOGIC_VECTOR; JUSTIFIED:in SIDE := RIGHT; FIELD:in WIDTH := 0);
- procedure OREAD(L:inoutLINE; VALUE:out STD_ULOGIC_VECTOR);
- procedure OREAD(L:inoutLINE; VALUE:out STD_ULOGIC_VECTOR; GOOD: out BOOLEAN);
- procedure OWRITE(L:inoutLINE; VALUE:in STD_ULOGIC_VECTOR; JUSTIFIED:in SIDE := RIGHT; FIELD:in WIDTH := 0);
- This set exists for value type : std_ulogic_vector, std_logic_vector

Example Application: Programming a ROM

- Assume an adapted listing file romcontents.txt:

```
001
002
003
004
005
006
007
008
009
00a
00b
00c
00d
00e
00f
200
210
220
230
240
250
260
270
```

Program Memory

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use std.textio.all;
use ieee.std_logic_textio.all;

entity progrom is
Port ( adr: in std_logic_vector(7 downto0);
data : out std_logic_vector(11 downto0);
clk: in std_logic);
end progrom;

architecture Behavioral of progromis
file datafile: text is "romcontents.txt";
type memorytypeis array(0 to 255) of std_logic_vector(11 downto 0);
signal rom: memorytype;
signal integer_address: integer range 0 to 255;
```

Program Memory

```
begin
  clkedge: process(clk)
  begin
    if rising_edge(clk) then
      integer_address <= to_integer(adr);
    endif;
  end process clkedge;
……
data <= rom(integer_address);
……
```

- Reading from memory as expected

Program Memory

```
fillrom: process
  variable line_in : line;
  variable ctr: integer;
  variable readfromfile: std_logic_vector(11 downto 0);
begin
  ctr:= 0;
  while not endfile(datafile) loop
    readline(datafile,line_in);
    hread(line_in,readfromfile);
    rom(ctr) <= readfromfile;
    ctr:= ctr+ 1;
  end loop;
  wait;
end process fillrom;
end Behavioral;
```

Generating Output Files

- Output files can be generated to be used in other applications:
  - They can contain the results in order to be compared with expected results produced by another algorithmic toolchain
  - They can contain control signals in order to check if the behavior of the design was as expected

Generating Output Files

```
process(Data,Adr,Fast_Clock,rd,wr,cs,reset)
  File outfile: TEXT is out "setramfile.txt";
  Variable L : LINE;
  Variable databitvector: Bit_Vector(0 to 15);
Begin
  If (Reset'Event) then
    if(Reset = '1') then
      WRITE(L, string'("h reset"));
    Else
      WRITE(L, string'("l reset"));
    endif;
    WRITELINE(outfile,L);
  endif;
  if(Adr'Event) then
    WRITE(L,string'("set address "));
    WRITE(L,adr);
    WRITELINE(outfile,L);
  if(cs'Event) and (cs= '1') then
    WRITE(L,string'("h chip_Select"));
```

References

- "Writing Testbenches: Functional verification of HDL models", Janick Bergeron, Kluwer Academic Publishers
- "Writing Efficient Testbenches", Mujtaba Hamid, Xilinx Application Note
- "Xilinx VHDL Test Bench Tutorial", Billy Hnath, Department of Electrical and Computer Engineering , Worcester Polytechnic Institute, eBook
- "Xilinx ISE Simulator (ISIM) VHDL Test Bench Tutorial", online tutorial, Digilent Corp.