HDL Co-simulation

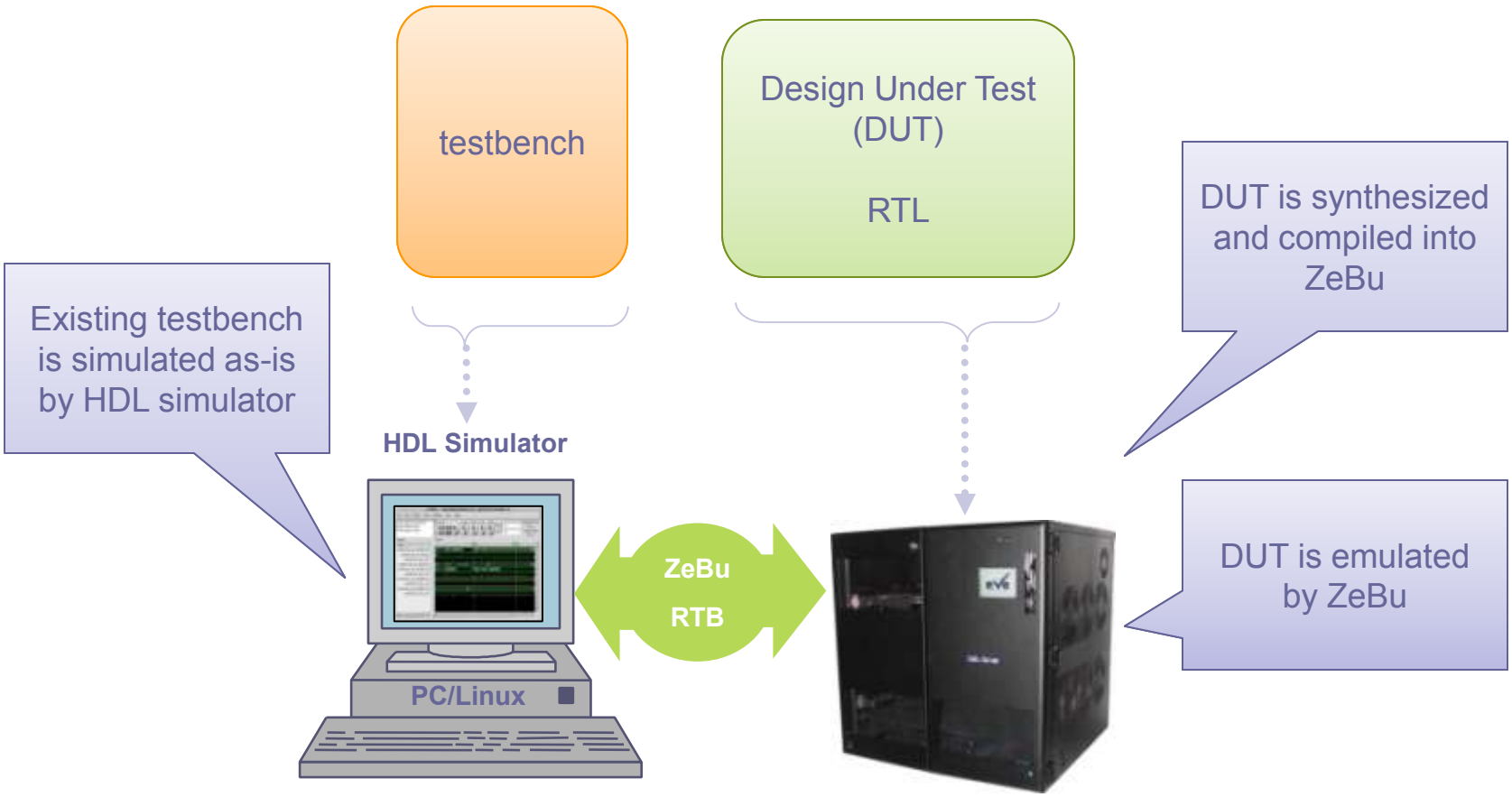
ZeBu-Server Training

*THE **FASTEST** VERIFICATION*

Agenda

- Overview
- Compilation
- Connecting the simulator
- Debug
- Saving and Replaying Patterns

Overview



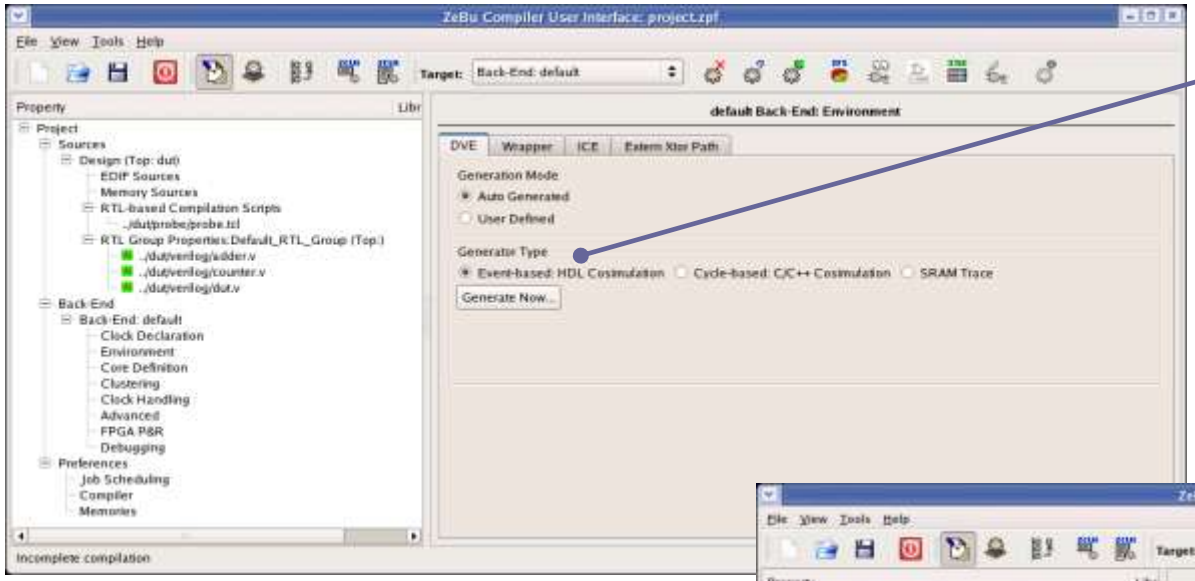
Overview

- Major HDL simulators supported
  - ModelSim, Mentor Graphics
  - NC-Sim, Cadence
  - VCS, Synopsys
- Support is done through a standard PLI library
  `libZebuPLI.so`

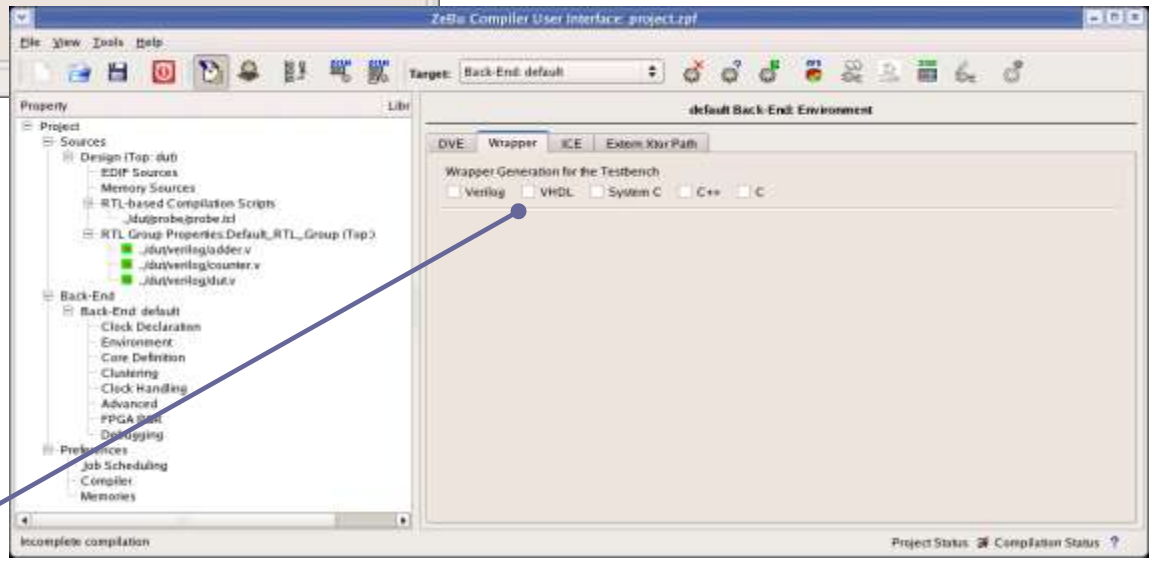
Agenda

- Overview
- Compilation
- Connecting the simulator
- Debug
- Saving and Replaying Patterns

Compilation
zCui



Select operation mode:
Event-based, HDL Cosimulation



Select testbench language, either Verilog or VHDL

Compilation
Wrapper file

- The compilation flow will generate a Verilog wrapper file named *<top>*.v
- The file will be in Verilog language even if VHDL is selected but the actual file content will be slightly different in order to operate correctly with a VHDL testbench

Agenda

- Overview
- Compilation
- Connecting the simulator
- Debug
- Saving and Replaying Patterns

Connecting the simulator

- To connect the testbench (running in the simulator) to the DUT (running in the ZeBu system) you need to:
  - Use the ZeBu PLI library
  - Replace the DUT original model by the wrapper generated during compilation
- Principle is the same for all 3 HDL simulators but the actual command line syntaxes vary

Connecting the simulator ModelSim

- Type the following commands:

```
export MTI_VCO_MODE=32|64
vlib work
vlog <testbench>.v zcui.work/zebu.work/<topModule>.v
vsim <testbench> <usualOptions> -pli libZebuPLI.so
      +zebu.work=zcui.work/zebu.work
```

Connecting the simulator NC-Sim

- Type the following commands:

```
ncverilog <testbench>.v zcui.work/zebu.work/<topModule>.v
      +nc64bit +loadpli1=libZebuPLI.so:bootstrap.bootstrap +access+rw
      +zebu.work=zcui.work/zebu.work
```

Connecting the simulator VCS

- **Type the following commands:**

```
vcs <testbench>.v -debug -gui -R zcui.work/zebu.work/<topModule>.v
      -P zcui.work/zebu.work/pli.tab $ZEBU_ROOT/lib/libZebuPLI.so
      +plusarg_save +zebu.work=zcui.work/zebu.work -full64
```

Agenda

- Overview
- Compilation
- Connecting the simulator
- Debug
- Saving and Replaying Patterns

Debug

- When debugging a design in HDL co-simulation mode you mainly use the simulator debug features:
  - Waveforms
  - Step-by-step in the testbench source code
  - …
- In addition on the ZeBu side, to monitor DUT internal signal you can use:
  - Static probes
  - Dynamic probes
  - Flexible probes
  - Simulated combinational signals
- You can also access ZeBu memories from the testbench

Debug
Static probes

- The wrapper file mimics the original design hierarchy, so signals probed using static probes are visible at their original location
  - E.g.: `top.u1.u2.signal`

Debug
Dynamic probes

- The wrapper file mimics the original design hierarchy, so signals probed using dynamic probes are visible at their original location
  - E.g.: `top.u1.u2.signal`
- Accessibility can be turned on and off to speed up emulation. When accessibility is turned off, dynamic probes display an 'X'
- Accessibility can be turned on and off using either of:
  - The `$ZEBU_readback` ZeBu system task

```
$ZEBU_readback(1);      // Switch on
$ZEBU_readback(0);      // Switch off
```

  - The `zebu_readback` signal

```
<testbench>.<dut_top>.zebu_readback = 1'b1;    // Switch on
<testbench>.<dut_top>.zebu_readback = 1'b0;    // Switch off
```

Debug
Simulated combinational signals

- Runtime behavior of simulated combinational signals is similar to dynamic probes and therefore the use model is the same

Debug
Flexible probes

- Flexibles probes are not visible from the testbench or even from the simulator
- They can only be dumped to a waveform file
- In order to use flexible probes in HDL co-simulation mode proceed in the following order:
  - Type `zRun -tesbench <launch simulator command>` to connect `zRun` to the emulation
  - Use zRun to configure flexible probes and dump waveforms
  - Run time steps in the simulator
  - Close the waveform file in `zRun` and generate standard format waveform file

Debug
Accessing ZeBu memories

- **To access a ZeBu memory during HDL co-simulation use the following ZeBu system tasks:**

```
$ZEBU_readmem("memFile","memoryName");      // load
$ZEBU_writemem("memFile","memoryName");     // dump
```

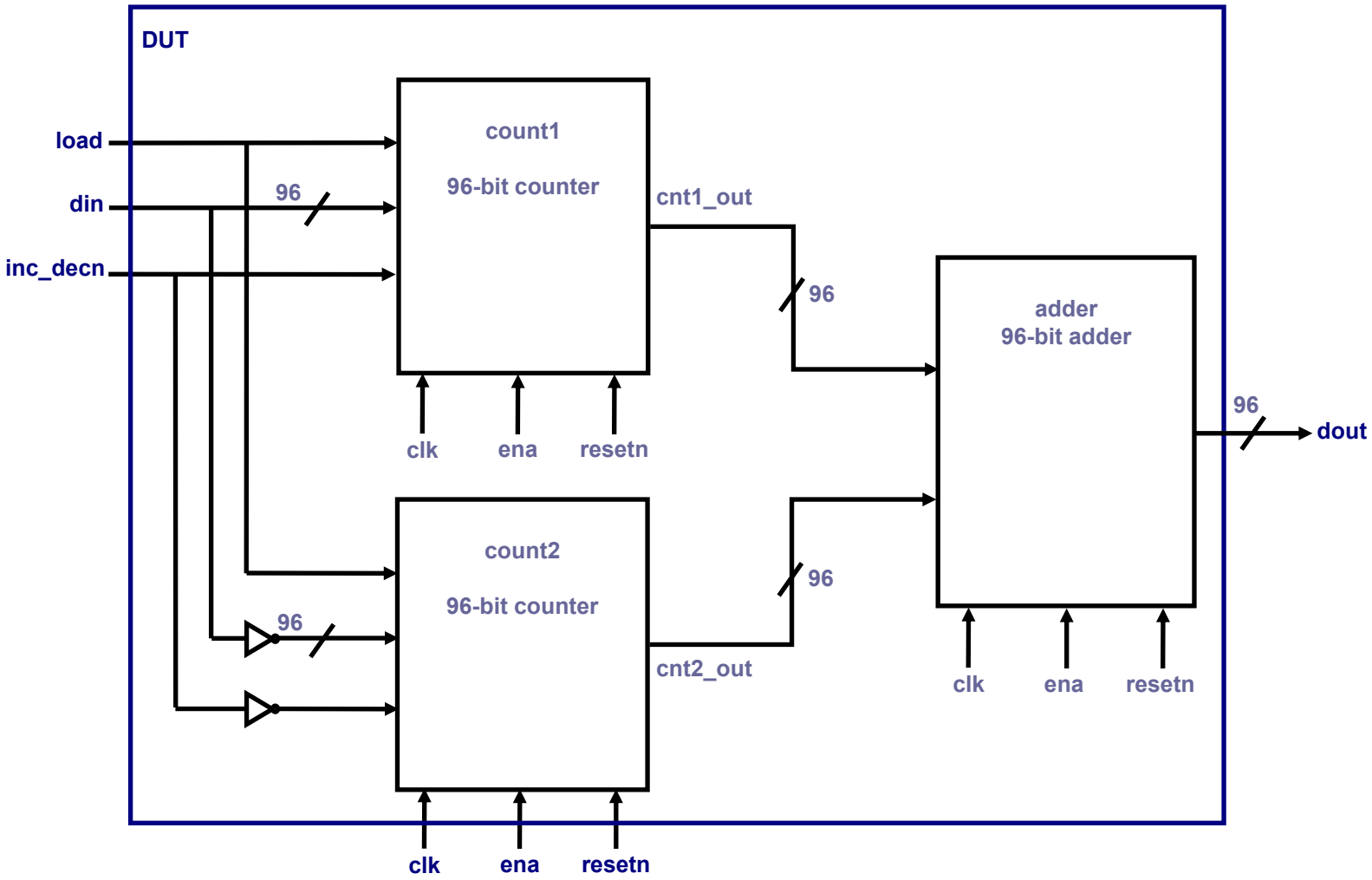
HDL co-simulation
Lab 1

- In this lab you will learn how to compile and emulate a design for HDL co-simulation
- The first step is to run a simulation, to verify that we get all files correctly and that we have a reference for behavior
- Then we will compile the design
- Finally we will emulate the design

HDL co-simulation
Lab 1

```
Trainee/lab1
|-- HDL_cosim
|   |-- simu.run
|   |-- zebu.run
|   `-- zebu.src
|-- dut
|   |-- probe
|   `-- verilog
|       |-- adder.v
|       |-- counter.v
|       `-- dut.v
`-- testbench
    `-- verilog
        |-- testbench1.v
        `-- testbench2.v
```

HDL co-simulation
Lab 1
The DUT



HDL co-simulation
Lab 1

- Run the simulation
  - Go into the `simu.run` directory
  - Type:
    `make SIM=<simulator> testbench1`
    Where *<simulator>* is one of
    - `ncsim`
    - `vcs`
    - `vsim`
  - See that the counters are counting and that the final check is ok. Output value should be `ffffffff`

HDL co-simulation
Lab 1

- Prepare the ZeBu compilation
  - Inspect the testbench for hierarchical references, you will have to create static probes for if you find some
  - Go into the `HDL_cosim` directory
  - Open `zCui`
  - Add `adder.v`, `counter.v` and `dut.v` to the RTL group, define the top module name of the group and select the synthesizer
  - Define the top module name of the design
  - Prepare a probe file in `../dut/probe/probe.tcl`
    - Make sure we declare the hierarchical reference as a static probe
    - We also want to see signals `dut.adder.inA` and `dut.adder.inB` as flexible probes
  - Add the probe file to the project

HDL co-simulation
Lab 1

- Set the target hardware configuration file according to your ZeBu system
- Add the clock in “Clock Declaration” panel
- Save the project
- Generate and modify a DVE file. Static probes need to be manually added to the DVE file but we can generate a template:
  - Left-click on the “Environment” property
  - Click on “Auto Generated”
  - Set “Event-based: HDL Cosimulation”
  - Click on “Generate Now…”
  - If prompted to launch synthesis, click on “Proceed”
  - When the DVE file browser opens, create the DVE file in `zebu.src` directory, name it `lab.dve`
  - (synthesis and DVE generator run)
  - Click on “Edit…”. This opens a text editor on the DVE file
  - Add the static probe in the `output_bin` section of the DVE file

HDL co-simulation
Lab 1

  - In the “Environment” panel, click on the “Wrapper” tab and set “Wrapper Generation for the Testbench” to “Verilog”
  - Set the job scheduling preferences according to your network settings
- Save the project
- Launch the compilation

HDL co-simulation
Lab 1

- Run the emulation
  - Go into the `zebu.run` directory
  - Compile the testbench
    `../../testbench/verilog/testbench1.v`
  - Compile the wrapper
    `../zcui.work/zebu.work/dut.v`
  - Run the emulation using the ZeBu PLI
  - Check that the behavior is the same and is correct

HDL co-simulation
Lab 1

- Now add a dynamic probe:
  - Type:
    `zSelectProbes -p ../zcui.work/zebu.work`
  - Select the `dut.count1.cnt[95:0]` signal
  - Save the database
  - Click on the “Generate”>”Verilog” menu item
  - Click on the “Generate”>”Generate” menu item
    - This will generate a new wrapper, have a look to it
  - Exit `zSelectProbes`

HDL co-simulation
Lab 1

- Re-run the emulation:
  - Re-compile the wrapper with `testbench2.v` this time
  - Re-run the emulation, try to force the `/top/dut1/zebu_readback` signal to `0` and `1` to de-activate/activate the dynamic probe
    - See that the `dut.count1.cnt` signal is then set to unknown or to its real value

Agenda

- Overview
- Compilation
- Connecting the simulator
- Debug
- Saving and Replaying Patterns

Saving and Replaying Patterns
zPattern

- Patterns increase the speed of non-regression testing
- Patterns (I/O pin activity) are recorded during HDL co-simulation and played back without the HDL simulator
- At runtime the recorded input events are applied on the design inputs and the design outputs are checked against the recorded output values
- This mode is not interactive and does not allow testbench reaction at runtime
- Using pattern mode requires:
  - Generating a pattern database using a C/C++ or an HDL co-simulation
  - Replaying the patterns using `zPattern`

Saving and Replaying Patterns
Generating the Pattern Data Base

- In an HDL Co-simulation, the pattern database is created by adding an option on the simulator command line:

```
+zebu.dumpfile=pattern.bin
```

- In C/C++ Co-simulation, add a command to dump to a binary file in the testbench:

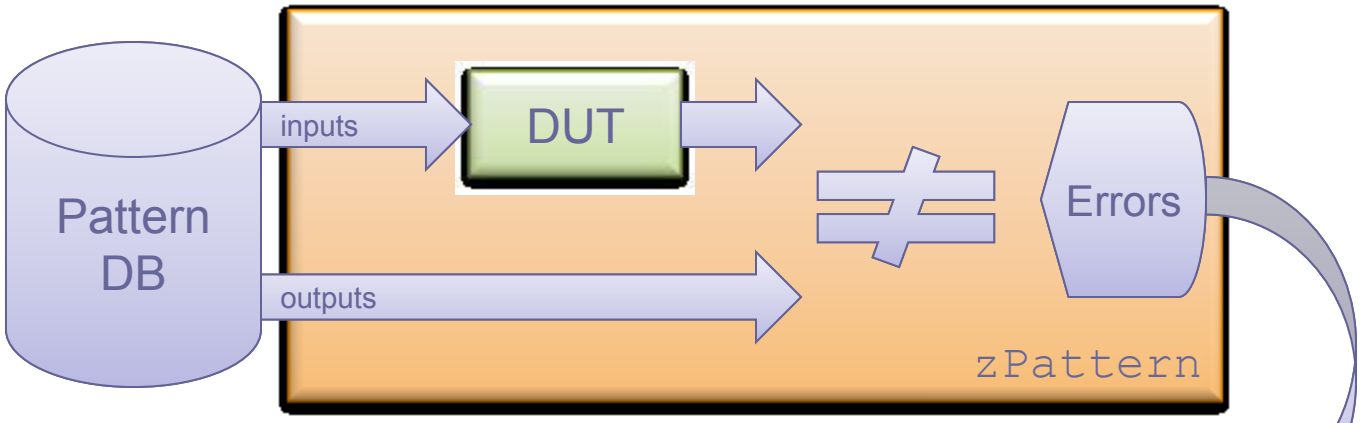
```
test_ccosim->dumpfile("pattern.bin",0);
```

  - Note that pattern mode is compatible with the MCKC driver but not with the C_COSIM driver

- Those commands result in a binary file containing the patterns
- Static probes, if defined, are used as comparison points

Saving and Replaying Patterns
Replaying Patterns with zPattern

```
zPattern
     -i <input bin file>       Pattern data base file
     [-z <zebu work path>]     zebu.work directory
     [-n <number of pattern>] Number of patterns to be executed (by default all
                               patterns)
     [-m <memory init file>]   memory initialization file
```




```
-- ZeBu : zPattern : WARNING: Error on signal "data_out", at pattern 5487.
-- ZeBu : zPattern : WARNING: Error on signal "data_out", at pattern 5489.
-- ZeBu : zPattern : WARNING: Error on signal "check_sum", at pattern 5489.
-- ZeBu : zPattern : WARNING: Error on signal "data_out", at pattern 19425.
-- ZeBu : zPattern : WARNING: Error on signal "mem_ctrl", at pattern 19425.
```