

ZeBu Basic Labs – 2018.09

Basics for Compilation – Runtime with zRun
SVAs & zDPI – Waveform Capture

Synopsys, Inc.

January 2019



CONFIDENTIAL INFORMATION

The following material is confidential information of Synopsys and is being disclosed to you pursuant to a non-disclosure agreement between you or your employer and Synopsys. The material being disclosed may only be used as permitted under such non-disclosure agreement.

IMPORTANT NOTICE

In the event information in this presentation reflects Synopsys' future plans, such plans are as of the date of this presentation and are subject to change. Synopsys is not obligated to develop the products with the features and functionality discussed in these materials. In any event, Synopsys' products may be offered and purchased only pursuant to an authorized quote and purchase order or a mutually agreed upon written contract.

ZeBu Basic Labs Overview

ZeBu Basic Labs

- After this lab, you will be able to:
 - Set your environment to use ZeBu
 - Launch compilation
 - Run emulation
 - Use SVA and zDPI features
 - Capture waveforms

Overview

Lab Overview

- Compilation
 - Prepare the design for emulation
 - Write the UTF file for compilation
- Runtime, controlled through a Tcl script
 - Basic controls for runtime
 - Force design signals
 - Runtime settings from designFeatures
 - SVAs usage
 - zDPI usage
 - Waveform Capture through Dynamic Probes and FWC
- Copy and untar the package in your work area: `ZeBu_BasicLabs_2018.09-<datecode>.tgz`

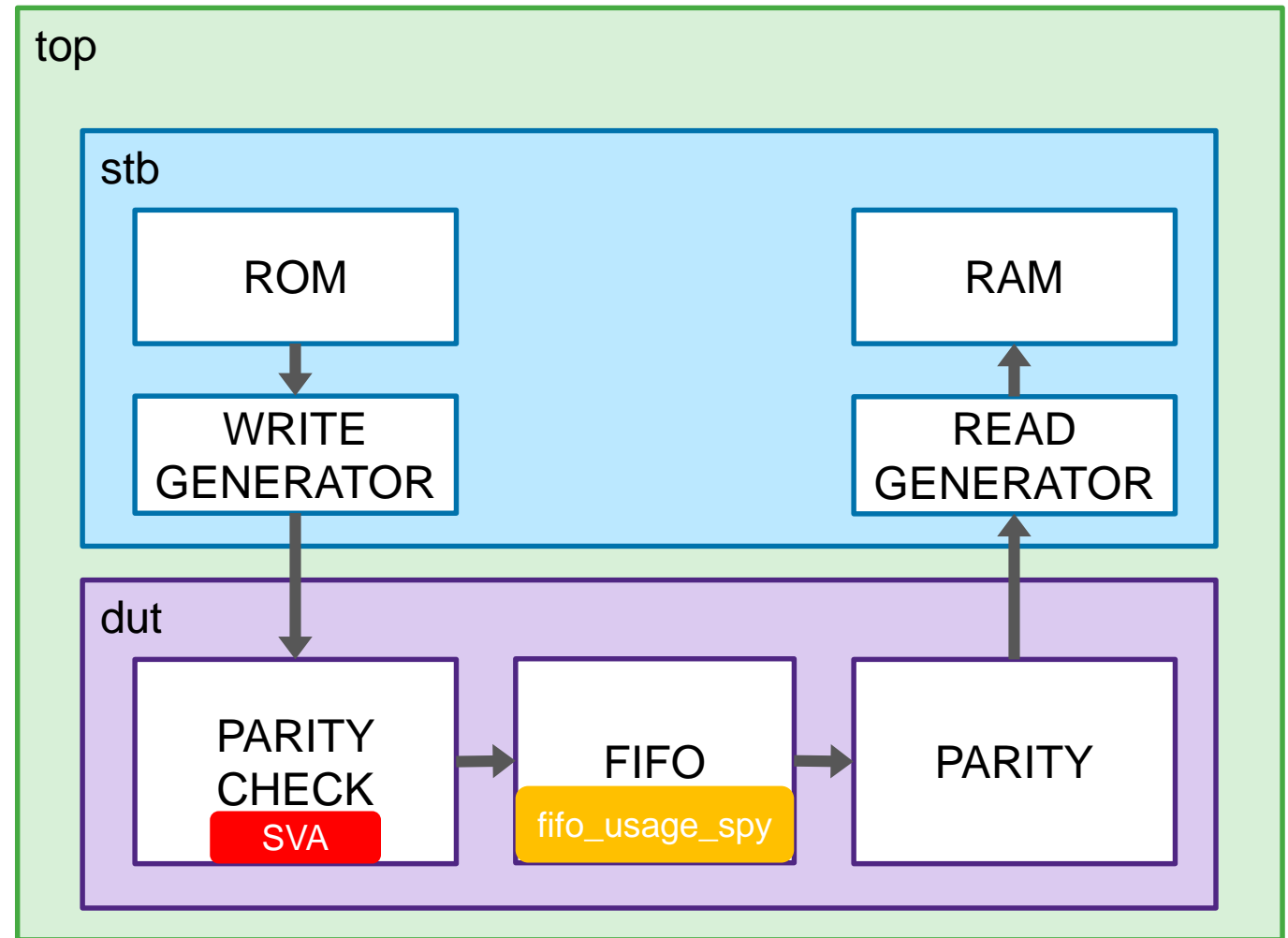
Overview

Design overview

Design is a simple FIFO with parity check on input and parity calculation on output

The synthesizable test (`stb`) is made of

- A write generator to the FIFO
 - Data comes from a ROM
 - Write frequency is controlled with the signal `proba_wr`
- A read generator for the FIFO
 - Read data are stored in a RAM
 - Read frequency is controlled with the signal `proba_rd`
- SVA in parity check to report errors
- A FIFO spy is implemented with zDPI feature to measure the maximum fifo depth used



Environment Setup

- Set the targeted ZeBu software version (`$ZEBU_ROOT`)
 - This material has been developed & tested **with ZeBu O-2018.09-1**
- Set the corresponding VCS-MX and Verdi software versions (`$VCS_HOME` and `$VERDI_HOME`)
- Check the settings for your emulation system:
 - Path to the ZeBu System Directory for emulation runtime (`$ZEBU_SYSTEM_DIR`)
 - Path to ZeBu Configuration File for compilation (typ. `zse_configuration.tcl` in `$ZEBU_SYSTEM_DIR`)
- Ensure that you have information about the Remote Commands expected on your network

Basic Compilation

Compilation for Emulation

Overview

- Goal

Create the setup for a small design and compile it

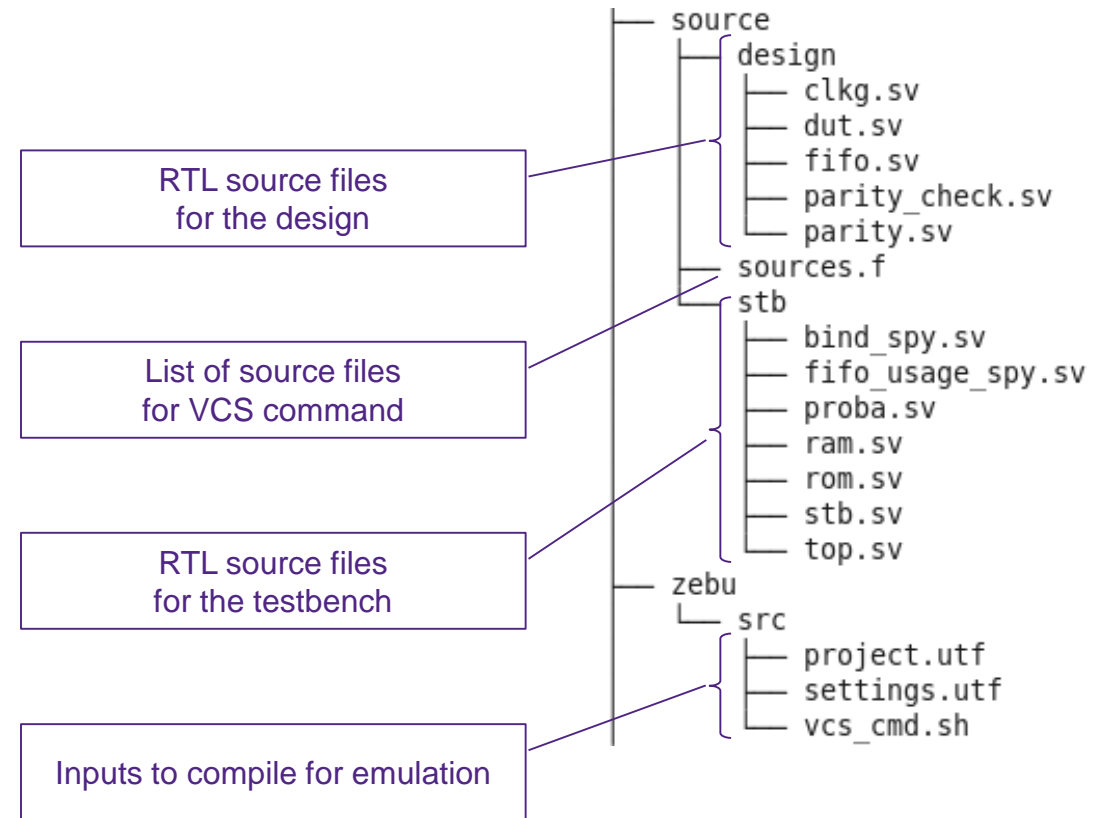
1. Basic Compilation for Emulation

1. Modifications in the design & testbench
2. Creation of minimum UTF file
3. Go through Front-End compilation and check logs

2. Full-featured compilation

1. Add all commands to UTF file for the coming runtime
2. Go through compilation and check logs

- If needed, the `help/` directory shows the solutions (files with modifications to get a working project).



Basic Compilation for Emulation

Prepare the design for Emulation

All RTL inputs for the design and testbench are available in `ZeBu_BasicLabs/source`

1. In `stb/stb.sv`, modify the generation of design clocks

- Using `always #1 clk != clk` is not synthesizable – won't work in emulation
- Instantiate ZeBu clock generator (`zceiClockPort`) for each design clock: the instance name is open for user's choice and `.cclock` output is connected to the clock signal (`clk0` and `clk1`)

```
zceiClockPort u_clk0 (  
    .cclock  (clk0)  
);
```

```
zceiClockPort u_clk1 (  
    .cclock  (clk1)  
);
```

- For an adaptable environment, the clock generators for ZeBu are added in an `'ifndef/'else` directive so that the original code is still applicable when using the design in a simulation environment

2. You can check syntax of your changes in Verilog by launching only VCS

- Source files are listed for VCS in `source/sources.f`
- Go to `ZeBu_BasicLabs/zebu/` to launch the following command

```
vcs -full64 -sverilog $VCS_HOME/etc/unizfe/verilog/zceiClockPort.v -f ../source/sources.f  
-top top +define+ASSERTIONS +define+ZEBU
```

Basic Compilation for Emulation

Creating a VCS script and UTF project file

3. Add VCS command to zebu/src/vcs_cmd.sh

```
vcs -full64 -sverilog -f ../source/sources.f -top top +define+ASSERTIONS +define+ZEBU
```

4. Open the zebu/src/settings.utf file with the description of environment

Declaring the Hardware Configuration file	architecture_file -filename \$env(ZEBU_SYSTEM_DIR)/config/zse_configuration.tcl
Set the remote commands	grid_cmd -queue {Zebu} -submit {<remote_command>} -delete {} -njobs {10} grid_cmd -queue {ZebuSynthesis} -submit {<remote_command>} -delete {} -njobs {10} grid_cmd -queue {ZebuIse} -submit {<remote_command>} -delete {} -njobs {10}

5. Open the zebu/src/project.utf file and add the following commands

Source the settings.utf script to describe the environment	source {./src/settings.utf}
Declaration of the VCS script	vcs_exec_command {src/vcs_cmd.sh}
Declaration of the top module	set_hwtop -module top
Set automatic number of modules to support multi-user for emulation runtime	design_size -mode AUTO

Basic Compilation for Emulation

Launching Compilation & Analyzing Results

From ZeBu_BasicLabs/zebu/ directory

6. Launch zCui with `project.utf` (-u), in Batch mode (-n), and stop after synthesis (-d)

```
zCui -u src/project.utf -n -d
```

7. Check your compilation results

- Generated directories

- Output directory for compilation: `./zcui.work`

- With the following subdirectories:

- | - design
 - | - vcs_splitter
 - | - utf_generatefiles
 - | - zCui

- Interesting logs

- `zCui/log/zCui.log`

- `zCui/log/vcs_splitter_VCS_Task_Builder.log`

- `zCui/log/design_Default_RTL_GroupBundle_0_Synthesis.log`

Full Compilation

Full Compilation – Overview

- Modifications of the source files to capture waveforms during emulation runtime
- Modifications of the VCS script to dump waveforms during emulation runtime
- Modifications of the UTF file to activate targeted features
 - Declaration of signals to be read or forced during emulation runtime
 - Prevention of inappropriate optimization during compilation (write-only memory)
 - SystemVerilog Assertions (SVAs)
 - DPI Function Calls (zDPI)
 - Waveform capture

Additional Source File for Waveform Capture

- Signals that are expected to be dumped with FWC or QiWC need declaration in a dedicated top-level module (`wave`)
- Create a new SystemVerilog source file (`source/stb/wave.sv`)

```
module wave;  
  initial begin: fwc_essential_signals  
    (* fwc *) $dumpvars (1,top.clk0);  
    (* fwc *) $dumpvars (1,top.clk1);  
    (* fwc *) $dumpport (top.u_dut.u_fifo);  
  end  
endmodule
```

- Modify the list of files for VCS (`source/sources.f`) to add the additional `wave.sv` file

```
../source/stb/wave.sv
```

- Modify the VCS script (`zebu/src/vcs_cmd.sh`) with the additional wave top-level module

```
vcs -full64 -sverilog -f ../source/sources.f -top top -top wave +define+ASSERTIONS +define+ZEBU
```

UTF Commands

The following commands are added in the previous `project.utf` file

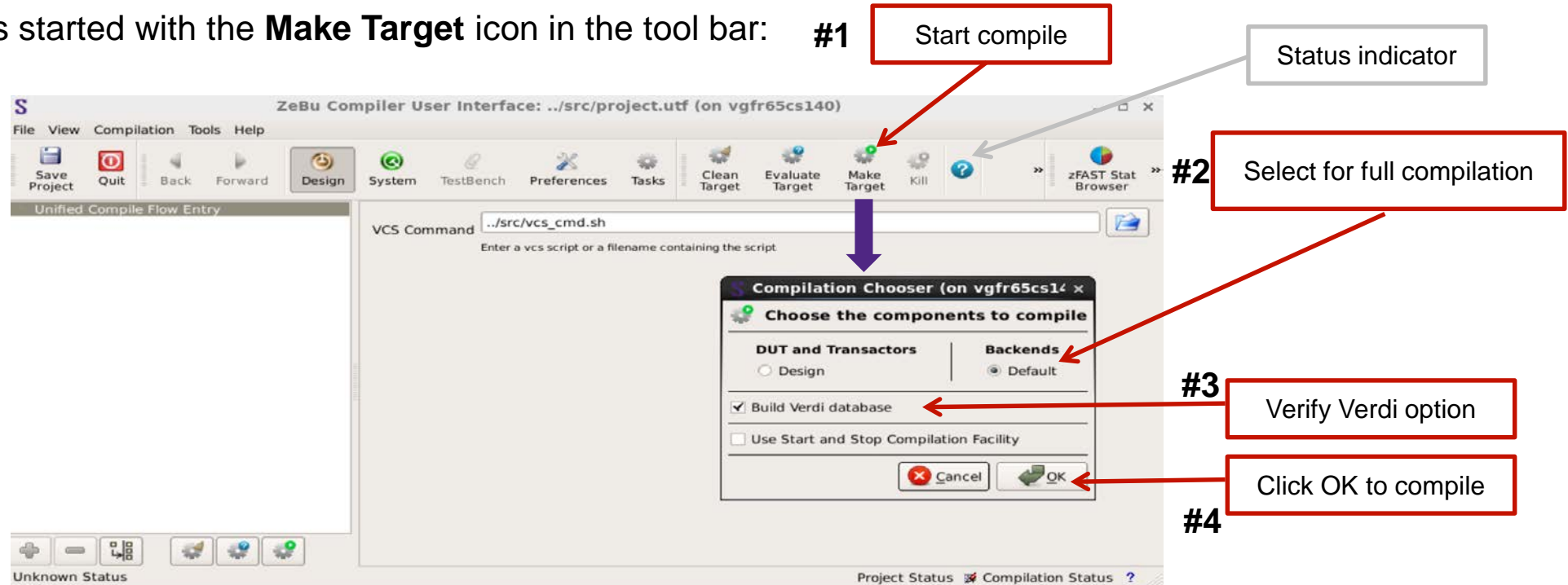
Activating debug features to capture waveforms	<code>debug -all true</code>
Preventing optimization of write-only memories by synthesis	<code>memories -drop_write_only false</code>
Enabling synthesis for zDPI	<code>dpi_synthesis -enable all</code>
Enabling synthesis for <code>\$display</code> system task	<code>system_tasks -enable -task {\$display}</code>
Enabling System Verilog Assertions	<code>assertion_synthesis -enable ALL -never_fatal</code>
Specifying signals that need to be forced during emulation runtime	<code>zforce -rtlname top.u_stb.proba_wr</code> <code>zforce -rtlname top.u_stb.proba_rd</code> <code>zforce -rtlname top.u_stb.rstn</code>
Specifying signals that need to be read during emulation runtime (via dynamic probes)	<code>probe_signals -rtlname top.u_stb.cnt_error_in</code> <code>probe_signals -rtlname top.u_stb.cnt_error_out</code>

Launching Compilation in zCui GUI

From ZeBu_BasicLabs/zebu/ directory

- Launch zCui in GUI mode
`$> zCui -u src/project.utf`
- Syntax errors in the UTF file (if any) are displayed when zCui opens.
- The compilation is started with the **Make Target** icon in the tool bar:

For syntax details, try
`vcs -help utf+<part of command name>`



Compilation Logs in zCui GUI

- zCui Global Log
 - Automatically displays when compilation finishes successfully
 - **Performance**: Shows the theoretical frequency computed by zTime ; final resource utilization
 - **Target**: Shows info on the ZeBu system size and resources, and the number of modules needed to compile the design
 - **Memory**: shows the memory resources actually used for the design
- When compiling in batch mode, use zBatchExplorer tool to reach the zCui Global Log

Runtime

Runtime Overview

- Goal

Create the runtime script and explore different debug capabilities

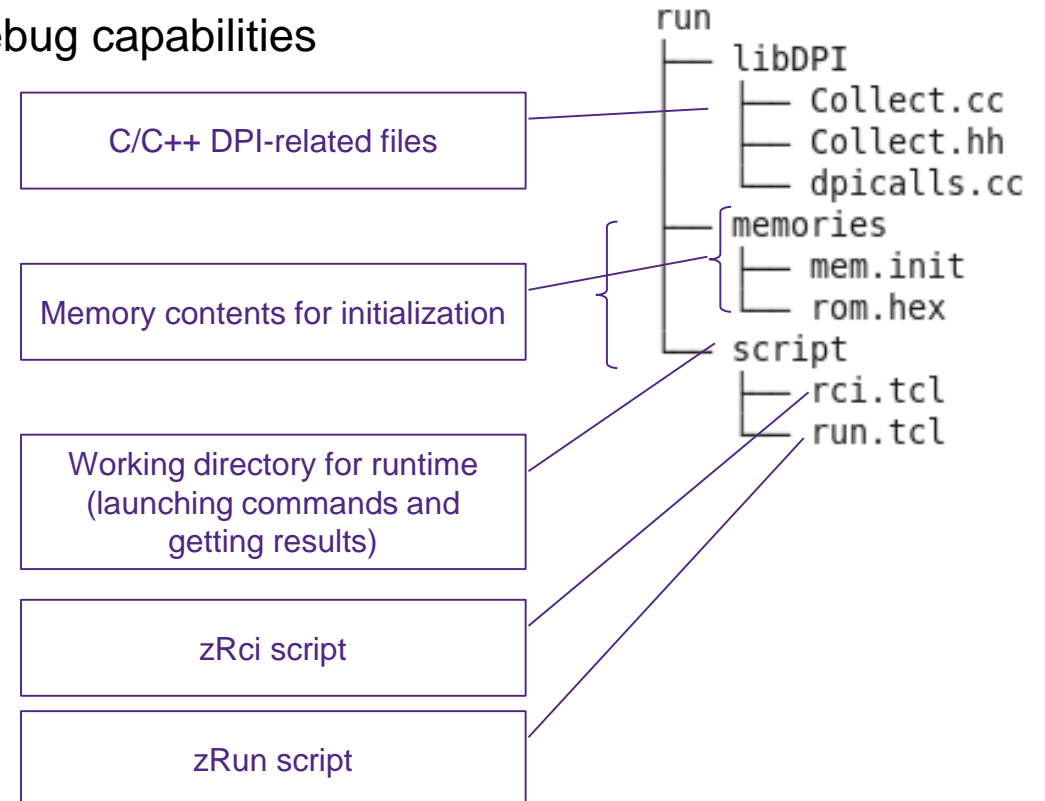
- With zRun

1. Basic Runtime
2. Runtime settings with designFeatures
3. Runtime with SVAs
4. Runtime with zDPI
5. Waveform Capture

- Viewing Waveforms in Verdi

- The following slides are applicable for zRun control interface

– Runtime control through zRci is a separate set



Setting Environment for Runtime

- Check that your runtime environment is correct
 1. Check that you have compiled your design for the right ZeBu Server system (hardware configuration file)
 2. Check that you are connected on the host PC connected to the right ZeBu Server system

All the commands for emulation runtime are launched from `ZeBu_BasicLabs/zebu/run/run_directory`

Basic Run with zRun

Basic Runtime Control with zRun

In `ZeBu_BasicLabs/zebu/run/script/zRun_script.tcl` file, add the following zRun commands

Initializing memory	<code>ZEBU_Memory_loadFromFile top.u_stb.u_rom.mem ../memories/rom.hex</code>
Forcing signals to set parameters (testbench initialization)	<code>ZEBU_Signal_force top.u_stb.proba_rd\[9:0\] [proba2hex 30]</code> <code>ZEBU_Signal_force top.u_stb.proba_wr\[9:0\] [proba2hex 30]</code> <code>ZEBU_Monitor_Flush</code>
Forcing <code>rstn</code> signal to enable reset	<code>ZEBU_Signal_force top.u_stb.rstn 0 %h</code> <code>ZEBU_Monitor_Flush</code>
Running 10 cycles	<code>ZEBU_Clock_enable top.u_stb.clk0 10</code> <code>while { [ZEBU_Clock_getStatus top.u_stb.clk0]=="running" } { after 100 }</code>
Disabling reset	<code>ZEBU_Signal_force top.u_stb.rstn 1 %h</code> <code>ZEBU_Monitor_Flush</code>
Running 1000 cycles	<code>ZEBU_Clock_enable top.u_stb.clk0 1000</code> <code>while { [ZEBU_Clock_getStatus top.u_stb.clk0]=="running" } { after 100 }</code>
Displaying signals value in decimal format	<code>puts "Number of errors detected:"</code> <code>puts " on input : [ZEBU_Signal_read top.u_stb.cnt_error_in %d]"</code> <code>puts " on output: [ZEBU_Signal_read top.u_stb.cnt_error_out %d]"</code>
Dumping memory content to a <code>dump.hex</code> text file	<code>ZEBU_Memory_storeToFile top.u_stb.u_ram.mem memdump.hex</code>

Launching zRun with a Tcl script

Basic Runtime Controls

- Launch emulation in ZeBu_BasicLabs/zebu/run/run_directory

```
zRun -zebu.work ../../zcui.work/zebu.work/ -do ../script/zRun_script.tcl -nogui
```

- Observe the output of the test

```
#####  
# Force signals ( wr : 30% , rd : 30% )  
# Resetting design during 10 clock cycles  
# Run 1000 cycles  
# Number of errors on input : 16  
# Number of errors on output: 0  
# Store ram memory content to dump.hex file  
#####
```

Some additional log lines may interleave on the console with the runtime results.

- File created from memory dump (dump.hex file)

```
// memory: top.u_stb.u_ram.mem , depth=256, width=9  
// verilog-H memory format  
048  
145  
14C  
14C  
145
```

Consistency check can be done with the file for memory initialization:
run/memories/rom.hex
→Differences are expected with parity check & correction in the testbench

Checking Runtime Log Files

Basic Runtime Controls

- `zServer.<hostPC>.0.log`
 - Logs the details of the communication between Host PC and ZeBu system
 - Logs the settings of the ZeBu system for runtime, typ. clock settings
 - At the end of the log, you can see the number of clock cycles for each design clock (zceiClockPort)

```
-- ZeBu : zServer : Unit 0 myGroup::top.u_stb.clk1 cycle counter : 1,010
-- ZeBu : zServer : Unit 0 myGroup::top.u_stb.clk0 cycle counter : 1,010
```

- `zRun.<hostPC>.0.log`
 - Logs the detailed activity of zRun

- `designFeatures.<hostPC>.help`
 - Runtime settings used for your emulation
 - In particular for design clock settings
 - Useful for future re-use in a later emulation session

```
# Note : Your design clocks are (possible labels for "my_clock") :
#   -0- top.u_stb.clk0
#   -1- top.u_stb.clk1
#
# Your current clocks parameters are :
#   -0- top.u_stb.clk0
$U0.M0.top.u_stb.clk0.Waveform = "-";
$U0.M0.top.u_stb.clk0.Mode = "controlled";
# No Frequency defined
# No TimePeriod defined
# No DelayUnit defined
# No DutyLo defined
# No DutyHi defined
# No Phase defined
$U0.M0.top.u_stb.clk0.VirtualFrequency = 1;
$U0.M0.top.u_stb.clk0.GroupName = "myGroup";
$U0.M0.top.u_stb.clk0.Tolerance = "no";
#   -1- top.u_stb.clk1
$U0.M0.top.u_stb.clk1.Waveform = "-";
$U0.M0.top.u_stb.clk1.Mode = "controlled";
# No Frequency defined
# No TimePeriod defined
# No DelayUnit defined
# No DutyLo defined
# No DutyHi defined
# No Phase defined
$U0.M0.top.u_stb.clk1.VirtualFrequency = 1;
$U0.M0.top.u_stb.clk1.GroupName = "myGroup";
$U0.M0.top.u_stb.clk1.Tolerance = "no";
```

Runtime with designFeatures

Emulation Runtime with designFeatures

Overview

- Instead of the default settings for design clocks, you will specify settings through the designFeatures file
 - Re-use the `designFeatures.*.help` created by the previous runtime exercise
 - `driverClock`, `zclockFilterTime` and `zClockskewTime` are unchanged for this exercise
 - The characteristics of the design clocks will be modified to have `clk0` twice faster than `clk1`

Writing a custom designFeatures file

Emulation Runtime with designFeatures

1. Go to ZeBu_BasicLabs/zebu/run/run_directory
2. Copy the generated designFeatures.*.help to designFeatures
3. Change \$U0.M0.top.u_stb.clk0.VirtualFrequency from 1 to 2

Default settings

(previous designFeatures.help)

```
# Your current clocks parameters are :  
# -0- top.u_stb.clk0  
$U0.M0.top.u_stb.clk0.Waveform = "_-";  
$U0.M0.top.u_stb.clk0.Mode = "controlled";  
$U0.M0.top.u_stb.clk0.VirtualFrequency = 1;  
$U0.M0.top.u_stb.clk0.GroupName = "myGroup";  
$U0.M0.top.u_stb.clk0.Tolerance = "no";  
# -1- top.u_stb.clk1  
$U0.M0.top.u_stb.clk1.Waveform = "_-";  
$U0.M0.top.u_stb.clk1.Mode = "controlled";  
$U0.M0.top.u_stb.clk1.VirtualFrequency = 1;  
$U0.M0.top.u_stb.clk1.GroupName = "myGroup";  
$U0.M0.top.u_stb.clk1.Tolerance = "no";
```

Modified settings

(new designFeatures)

```
# Your current clocks parameters are :  
# -0- top.u_stb.clk0  
$U0.M0.top.u_stb.clk0.Waveform = "_-";  
$U0.M0.top.u_stb.clk0.Mode = "controlled";  
$U0.M0.top.u_stb.clk0.VirtualFrequency = 1;  
$U0.M0.top.u_stb.clk0.VirtualFrequency = 2;  
$U0.M0.top.u_stb.clk0.GroupName = "myGroup";  
$U0.M0.top.u_stb.clk0.Tolerance = "no";  
# -1- top.u_stb.clk1  
$U0.M0.top.u_stb.clk1.Waveform = "_-";  
$U0.M0.top.u_stb.clk1.Mode = "controlled";  
$U0.M0.top.u_stb.clk1.VirtualFrequency = 1;  
$U0.M0.top.u_stb.clk1.GroupName = "myGroup";  
$U0.M0.top.u_stb.clk1.Tolerance = "no";
```

Launching Emulation and Analyzing Results

Emulation Runtime with designFeatures

1. Launch emulation in ZeBu_BasicLabs/zebu/run/run_directory/

```
zRun -zebu.work ../../zcui.work/zebu.work/ -do ../script/zRun_script.tcl -nogui -design designFeatures
```


2. zServer log shows that the number of cycles for clk1 is 2x lower than for clk0

```
-- ZeBu : zServer : Unit 0 Module 0 myGroup::top.u_stb.clk0 cycle counter : 1,010.  
-- ZeBu : zServer : Unit 0 Module 0 myGroup::top.u_stb.clk1 cycle counter : 505.
```



3. The number of errors reported is slightly different from previous exercise

```
#####  
# Force signals ( wr : 30% , rd : 30% )  
# Resetting design during 10 clock cycles  
# Run 1000 cycles  
# Number of errors on input : 29  
# Number of errors on output: 0  
# Store ram memory content to dump.hex file  
#####
```



Runtime – zRun with SVAs

Runtime script for SVAs

1. Clean your run directory
2. In `ZeBu_BasicLabs/zebu/script/zRun_script.tcl` file, add the following commands for `zRun` in the “Start SVA Processing” section

	<code>if {[ZEBU_Sva_isDefined] == 1} {</code>
Initializing the SVAs	<code>ZEBU_Sva_init</code>
Defining the SVA severity	<code>ZEBU_Sva_setSeverityReport info</code>
Starting the SVAs	<code>ZEBU_Sva_start live top.u_stb.clk0</code>
Enabling all SVAs	<code>ZEBU_Sva_activate ZSVA.* 0</code>
	<code>}</code>

You can either enable some specific SVAs:

Enabling some specific SVAs	<code>ZEBU_Sva_activate ZSVA.top.u_dut.u_fifo.assert_0 0</code> <code>ZEBU_Sva_activate ZSVA.top.u_dut.u_check.immediate_assert_0 0</code> <code>ZEBU_Sva_activate ZSVA.top.u_stb.u_check.immediate_assert_0 0</code> <code>ZEBU_Sva_activate ZSVA.top.u_stb.u_check.sva_parity_error 0</code>
-----------------------------	---

Launching Emulation and Analyzing Results

Runtime with SVAs

3. From ZeBu_BasicLabs/zebu/run/run_directory, launch emulation

```
zRun -zebu.work ../../zcui.work/zebu.work/ -do ../script/zRun_script.tcl -nogui
```

4. The parity errors detected by SVAs are visible along the testbench execution

```
#####
# Force signals ( wr : 30% , rd : 30% )
# Resetting design during 10 clock cycles
# Run 1000 cycles
** Error: Parity error detected..
Time: 161 Scope: top,u_dut,u_check,unnamed$$_0 File: ../source/design/parity_check.sv Line: 26
** Error: Parity error detected..
Time: 295 Scope: top,u_dut,u_check,unnamed$$_0 File: ../source/design/parity_check.sv Line: 26
** Error: Parity error detected..
Time: 477 Scope: top,u_dut,u_check,unnamed$$_0 File: ../source/design/parity_check.sv Line: 26
** Error: Parity error detected..

** Error: Parity error detected..
Time: 1885 Scope: top,u_dut,u_check,unnamed$$_0 File: ../source/design/parity_check.sv Line: 26
# Number of errors on input : 16
# Number of errors on output: 0
# Store ram memory content to dump.hex file
#####
** SVA Report : SVA top,u_dut,u_check,unnamed$$_0 :
                    Number of times failed : 16
** SVA Report : SVA top,u_stb,u_check,unnamed$$_0 :
                    Number of times failed : 0
```


Runtime – zRun with zDPI

Runtime Script for zDPI

1. The DPI functions are implemented in `dpicalls.cc` (in `run/libDPI`)

– The DPI functions report the minimum of the FIFO free place

2. Compile C code of the DPI (in `run/libDPI`)

```
g++ -m64 -O2 -I. -I$ZEBU_ROOT/include -Wall -fexceptions -rdynamic -fPIC -c Collect.cc -o Collect.o
g++ -m64 -O2 -I. -I$ZEBU_ROOT/include -Wall -fexceptions -rdynamic -fPIC -c dpicalls.cc -o dpicalls.o
g++ -shared -o libDPI.so -m64 -lpthread -L$ZEBU_ROOT/lib Collect.o dpicalls.o
```

3. In `run/script/zRun_script.tcl` file, add the following zRun commands in the “Enable DPI” section

For easier reading of the results, you should also comment/remove the SVA-related lines in the script.

To load DPI dynamic library	<code>ZEBU_CCall_loadDynamicLibrary ../libDPI/libDPI.so</code>
To set the sampling clock	<code>ZEBU_CCall_selectSamplingClocks top.u_stb.clk0</code>
To start the DPI	<code>ZEBU_CCall_start</code>

Launching Emulation and Analyzing Results

Runtime with zDPI

4. Launch emulation in run/run_directory

```
zRun -zebu.work ../../zcui.work/zebu.work/ -do ../script/zRun_script.tcl -nogui
```

5. The parity errors are reported along the testbench execution

```
#####
# Force signals ( wr : 30% , rd : 30% )
# Resetting design during 10 clock cycles
# Run 1000 cycles
# Parity error detected on input at cycle      80 of clk0
# Parity error detected on input at cycle     147 of clk0
# Parity error detected on input at cycle     238 of clk0

# Parity error detected on input at cycle      907 of clk0
# Parity error detected on input at cycle      942 of clk0
# Number of errors on input : 16
# Number of errors on output: 0
# Store ram memory content to dump.hex file
#####
-- ZeBu : zRun : Test finished.
#### Collect : #####
#### Collect :   Statistics Summary

#### Collect :   top.u_dut.u_fifo.u_fifo_usage_spy : 1
#### Collect : #####
```

Waveform Dump with zRun

Waveform Dump with zRun

Writing the Tcl Script for zRun

In `ZeBu_BasicLabs/zebu/run/script/zRun_script.tcl` file, add the following zRun commands

- For Dynamic Probes

Start dump (Dynamic Probes)	<code>ZEBU_Dump_file "waveforms_ztdb/dynprobes_dump.ztdb" top.u_stb.clk0</code> <code>ZEBU_Dump_on</code>
Run cycles	<code>[...]</code>
Stop dump (Dynamic Probes)	<code>ZEBU_Dump_flush</code> <code>ZEBU_Dump_close</code>

- For FWC

Start dump (FWC)	<code>ZEBU_Fwc_setSamplingClock "top.u_stb.clk0".</code> <code>ZEBU_Fwc_addValueSet fwc_essential_signals</code> <code>ZEBU_Fwc_setOutputDir "waveforms_ztdb/fwc_dump.ztdb"</code> <code>ZEBU_Fwc_dumpOnOff "on"</code>
Run cycles	<code>[...]</code>
Stop dump (FWC)	<code>ZEBU_Fwc_flush</code> <code>ZEBU_Fwc_close</code>

Waveform Dump with zRun

Running Emulation

- Create a directory to store the waveforms

```
mkdir runtime_output
```

- Launch emulation from `run/run_directory` directory

```
zRun -zebu.work ../../zcui.work/zebu.work/ \  
-do ../script/zRun_script.tcl -nogui \  
-selectionDB ../../zcui.work/zebu.work/zrdb/csa_supports.zrdb
```

Viewing Waveforms with Verdi

Viewing Dynamic-Probes Waveform in Verdi

- From `run/run_directory` directory
- Waveforms captured through Dynamic-Probes need waveform reconstruction before viewing in Verdi

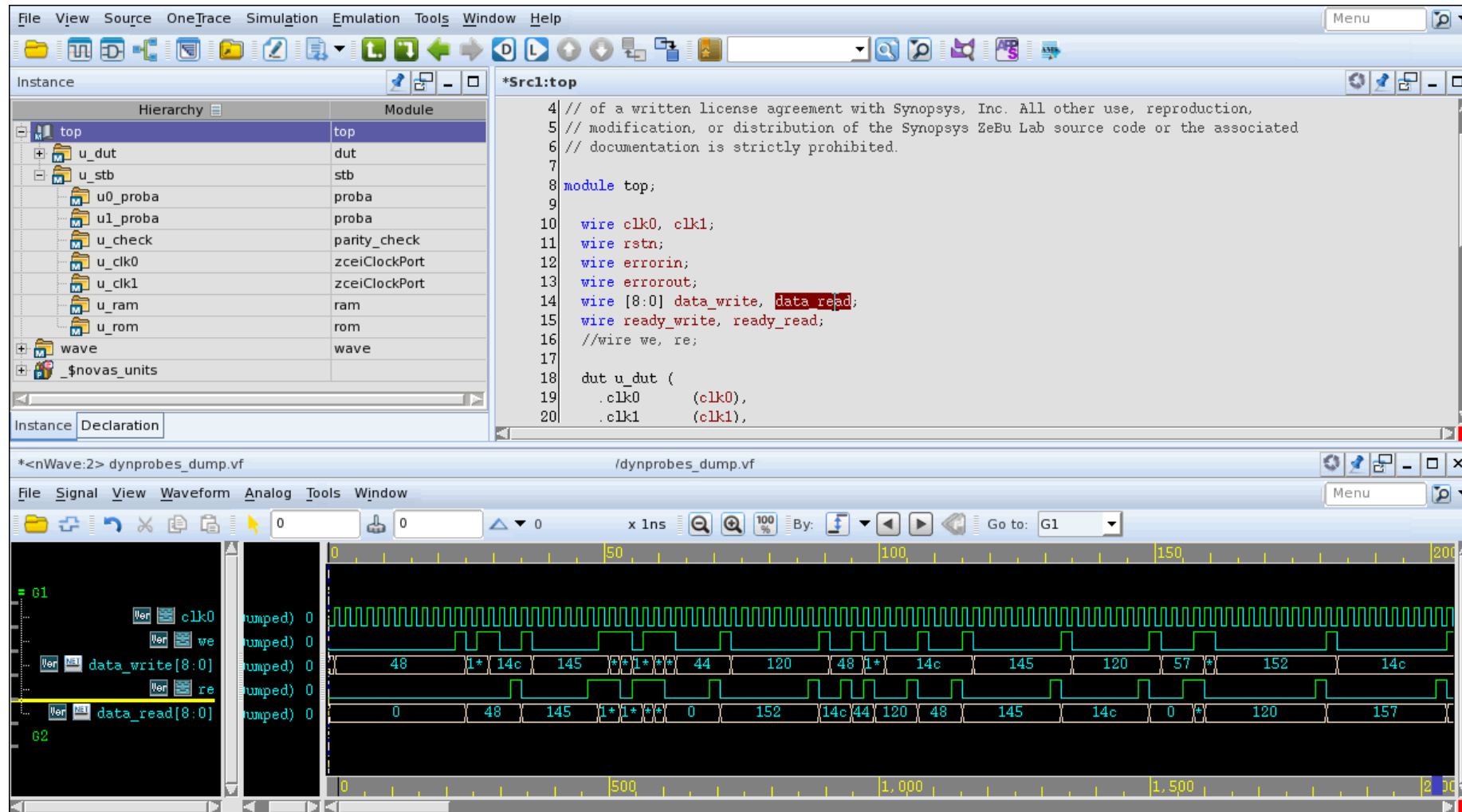
```
zWaveform --work ../../zcui.work/zebu.work/ \  
--ztdb runtime_output/dynprobes_dump.ztdb \  
--fsdb dynprobes_dump --timescale 1ns
```

– Results in `dynprobes_dump.vf` file + `dynprobes_dump/` subdirectory, for later use by Verdi

- Launch Verdi with resulting file

```
verdi -emulation -zebu-work ../../zcui.work/zebu.work -ssf dynprobes_dump.vf
```


Viewing Dynamic-Probes Waveform in Verdi



Viewing FWC Waveforms in Verdi

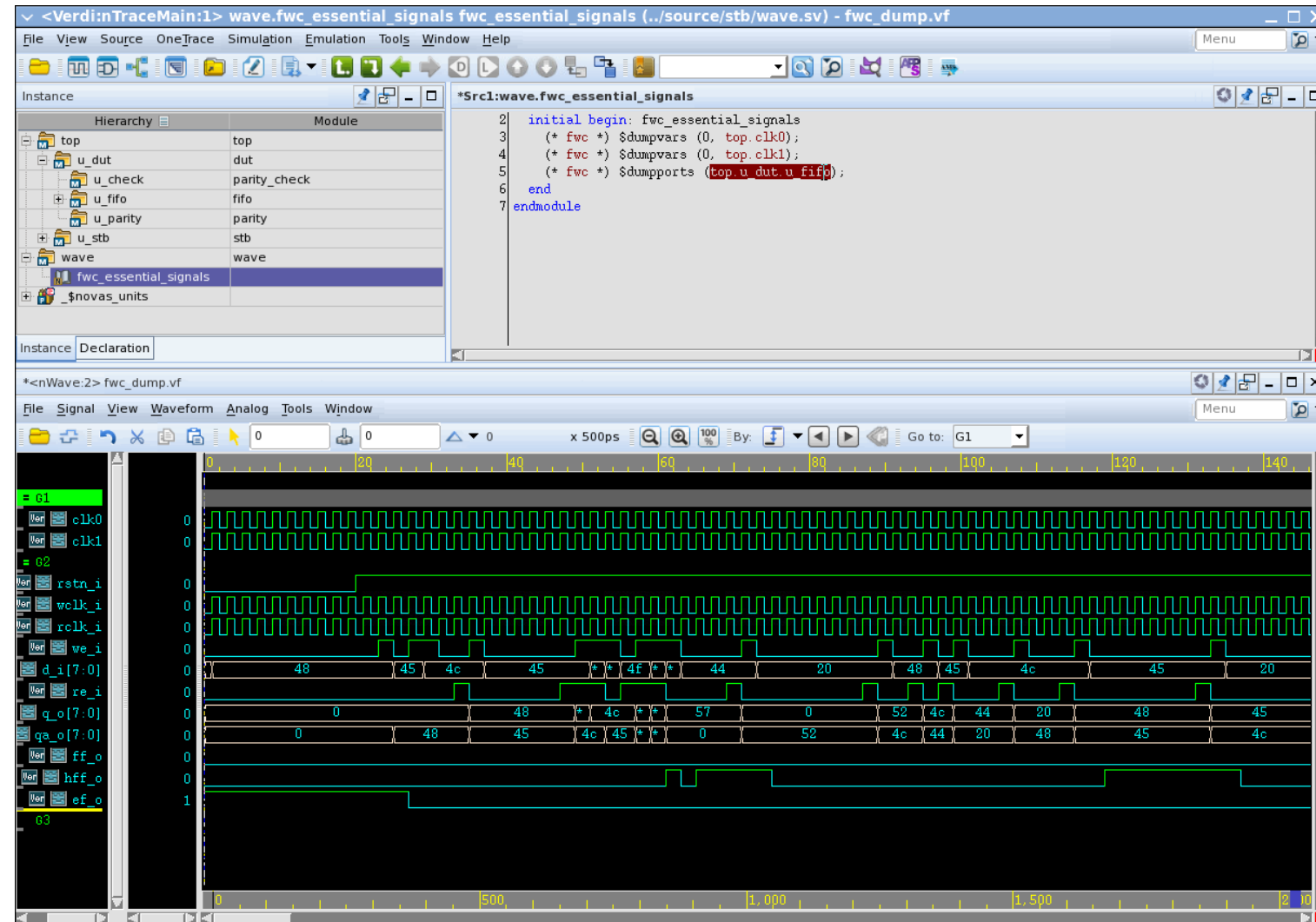
- From `run/run_directory` directory
- Waveforms captured through FWC need to be converted to FSDB format

```
zConvertToFsdB --zebu-work ../../zcui.work/zebu.work \  
--ztdb runtime_output/fwc_dump.ztdb \  
--fsdb fwc_dump --timescale 1ns
```

- Launch Verdi with resulting file

```
verdi -emulation --zebu-work ../../zcui.work/zebu.work/ -ssf fwc_dump.vf
```

Viewing FWC Waveforms in Verdi



Thank You



