The Fastest Verification

ZeBu-Server Release Note

Version 6_2_1 with patch B_00

August 2010

Table of Contents

About this document

This Release Note describes the major features available for Version 6_2_1 release of the ZeBu-Server software with cumulative patch B_00.

This document is based on the V6_2_1 Release Note: Chapter 1 is dedicated to specific information for the B_00 cumulative patch. Other sections of the V6_2_1 Release Note have been updated only in case of modification due to the cumulative patch (removal of out-of-date limitations, in particular).

Note that the present software version is intended for ZeBu-Server only and is not intended for use with ZeBu-XL, ZeBu-XXL, ZeBu-UF or ZeBu-Personal.

This document is intended for users who are familiar with the ZeBu product range.

You can find press releases, useful white papers and technology documents on our website: http://www.eve-team.com.

It is HIGHLY recommended to read the "Modified Default Settings" sections (§1.5 and §4.1) before launching a new compilation with this software release.

1 Specific Information for V6_2_1 B_00

This document is based on the V6_2_1 Release Note: this chapter is dedicated to specific information for B_00 cumulative patch; the other sections have been updated only in case of modification with the latest cumulative patch (removal of out-of-date limitations, in particular).

Information on Version Compatibility has changed. It is highly recommended to read Chapter 6 and Sections 1.2/1.3 before you use this release.

1.1 Contents of the patch

This qualified cumulative patch obsoletes any previous V6_2_1 patch. It includes all the files for installation. Software installation of the ZeBu V6_2_1 base release is NOT a prerequisite to install the B_00 cumulative patch.

This patch includes version 11.4i_patched of the Xilinx ISE software subset for FPGA Place & Route for ZeBu. Note that Xilinx ISE 11 requires specific Xilinx licenses which should be requested from EVE (license@eve-team.com).

After installation, a text-formatted patch note is available in the `$ZEBU_ROOT/version/patch.txt` file. It lists the most recent corrections and enhancements (not already delivered in patches) and gathers relevant information of intermediate patches since the V6_2_1 software release.

The documentation package of the release has been updated and should be downloaded for installation at the same time as the release.

1.2 Installation Procedure / Mandatory Hardware Update

The following steps are mandatory when installing the present cumulative patch:

- The diagnostics packages have changed (they are now version 4.0): they MUST be downloaded at the same time as the software package and installed as for a new software version (see ***Zebu-Server Installation Manual*** for details about the available diagnostics packages, their naming rules corresponding and their installation).
- Install the software as described in Chapter 4 of the ***Zebu-Server Installation Manual***.
- When upgrading from a software release older than V6_2_1 or V6_2_0B_12, proceed with update of PCIe boards by launching **`zUpdate`** and reboot the host PC. Note that any older software version will operate correctly after this update.
- Because the hardware configuration files are not compatible with previous software versions, it is necessary to create a specific `$ZEBU_SYSTEM_DIR`, directory when installing the present cumulative patch. For that purpose, modify the `setup.zini` file with the path to this new directory.

- Launch **zSetupSystem** to generate new calibration files in the new $ZEBU_SYSTEM_DIR directory.
- In the **zSetupSystem** log, some messages are displayed for update of the front-panel of the ZeBu-Server unit(s):

```
-- ZeBu : zUtils : WARNING : U0 Front-Panel software is not up-to-date
-- ZeBu : zUtils : WARNING : Please execute the following command :
-- ZeBu : zUtils : WARNING : zUtils -loadFw U0_FP_FX
     $ZEBU_ROOT/etc/firmwares/ZSE/default/zse_fpcb_soft.bin

-- ZeBu : zUtils : WARNING : U1 Front-Panel software is not up-to-date
-- ZeBu : zUtils : WARNING : Please execute the following command :
-- ZeBu : zUtils : WARNING : zUtils -loadFw U1_FP_FX
     $ZEBU_ROOT/etc/firmwares/ZSE/default/zse_fpcb_soft.bin

-- ZeBu : zUtils : WARNING : When you have executed ALL this commands
-- ZeBu : zUtils : WARNING : Power-OFF every updated units, then
power-ON them and execute
-- ZeBu : zUtils : WARNING : zUtils -initSystem
```

  - Set the $ZEBU_SYSTEM_DIR variable to the appropriate directory
  - Launch the commands given in the messages:
```
$ zUtils -loadFw U0_FP_FX $ZEBU_ROOT/etc/firmwares/ZSE/default/zse_fpcb_soft.bin
$ zUtils -loadFw U1_FP_FX $ZEBU_ROOT/etc/firmwares/ZSE/default/zse_fpcb_soft.bin
```
  - Power OFF and ON the ZeBu-Server units to finalize the updating process.
  - Initialize the ZeBu-Server system by launching:
```
$ zUtils -initSystem
```
    The resulting configuration files for compilation take the new diagnostics results into account.

1.3 Compatibility with Previous Versions

Because of modifications of the system FPGAs in the most recent versions, the following rules apply:

- Once your system is initialized with **zUtils** -initSystem for V6_2_1 B_00, you can:
  - Run any design compiled with V6_2_1 B_00 or V6_2_1 with V6_2_1 B_00 runtime software.
  - Run any design compiled with V6_2_1 B_00 or V6_2_1 with V6_2_1 A_03 runtime software.
    Installation of patch A_03 is mandatory; any attempt to run a design with V6_2_1 without patch A_03 will cause the following error message:
```
-- ZeBu : zServer : ERROR : LUI1878E : Software release you're using is not
compatible with the system, launch "zUtils -initSystem" with a compatible
release (0x00010001 - 0x00010000).
```
- A design compiled with V6_2_0B or earlier software version has to be re-compiled before proceeding with emulation runtime with V6_2_1 B_00.
  You have to initialize your system with **zUtils** -initSystem for V6_2_0B if you want to proceed with V6_2_0B emulation runtime for regression testing purposes.

1.4 New Features in V6_2_1 B_00

In addition to the V6_2_1 features described in Chapter 2, the following features have been added, either in an intermediate patch (A_xx) or more recently in the B_00 cumulative patch.

1.4.1 Higher System Clock Frequency

By default, the inter-FPGA communication system clock for emulation runtime is now set to 400 MHz without any specific declaration at compilation and runtime.

It is mandatory to launch the diagnostics and generate an updated configuration file to support the 400 MHz xclock frequency.

Note that a design previously compiled for V6_2_0 or for V6_2_1 can be used with V6_2_1 B_00 software but it runs at 300 MHz if no additional constraint was declared in the DVE file.

1.4.2 Support for 5-unit systems

This software release provides support for multi-unit ZeBu-Server systems with 5 units. As described in Section 2.1, it is mandatory to boot the host PCs of a 5-unit system with `memmap` option and use the `-bootAllocAddr` option launching **`zInstall`**.

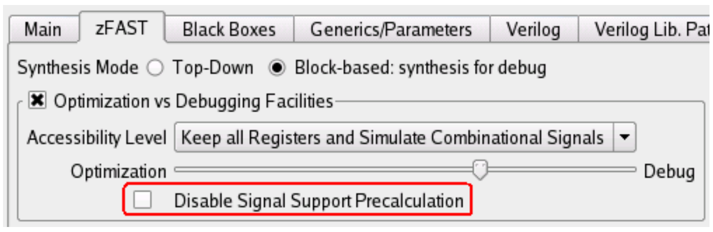
Since the interconnection in a 5-unit system is specific, it is not possible to software to use an older software version even for a design smaller than 5 units.

To optimize the initialization of a 5-unit system, the FPGAs are loaded in parallel (two at a time).

1.4.3 zCui NewFeatures

- In the **Preferences** → **Job Scheduler** panel of **`zCui`**, an optional subcategory has been created in order to support a specific set of commands and options for the system-level compilation (**zTopBuild Process**).
  If this option is not selected, the system-level compilation will use the **ZeBu Heavy Weight Processes** commands (if set) or the **ZeBu Compiler** commands.
- An icon has been added in the **`zCui`** toolbar to launch the **`zFAST`** Stat Browser (this comes in addition to the contextual menu access described in Section 2.11).
  Note that when the top name of the RTL group is declared in **`zCui`**, the opening window where the top name is selected is now skipped when launching **`zFAST`** Stat Browser.
- From the **View** menu, it is now possible to hide the **`zCui`** internal tasks (inputs checking, creation of scripts, result analysis, etc.) in the **Compilation** workspace.

- The compilation for CSA (Combinational Signals Availability) has been improved in terms of duration and quality of results. However, for multi-unit designs, it may be of interest to disable the longest part of this compilation (Signal Support Calculation) and do it after the design has been successfully compiled. For that purpose, a new check box has been added in the **zFAST** tab:



Synthesis Mode ○ Top-Down [✓ ● Block-based: synthesis for debug]
☒ Optimization vs Debugging Facilities
Accessibility Level: Keep all Registers and Simulate Combinational Signals
Optimization — Debug
☐ Disable Signal Support Precalculation

- Some issues with log updates have been fixed.

1.4.4 Improvements for Detection of Oscillating Combinational Loops

The present cumulative patch provides a C++ and C runtime API to control the detection of oscillating combinational loops in addition to the compilation and runtime capabilities which were present in V6_2_1 software (see Section 2.8).

Note that the default setting during compilation has changed, as described in Section 1.5.3.

The methods and functions to control the Detection of Oscillating Combinational Loops are described in `LoopDetector.hh` (C++ API) and `ZEBU_LoopDetector.h` (C API). These files are automatically included with `libZebu.hh`/ `libZeBu.h`.
Note that some methods and functions are described in these files for future enhancements of the feature; only the methods and functions listed below are intended for use with V6_2_1 B_00 software.

User can test if a combinational loop has oscillated during the last run or since the last detector reset:

- C++ method: `ZEBU::APosterioriLoopDetector::checkDetectors`
- C function: `ZEBU_APosterioriLoopDetector_checkDetectors`

Once the detectors have triggered, they have to be manually reset from the testbench for a future detection:

- C++ method: `ZEBU::APosterioriLoopDetector::resetDetectors`
- C function: `ZEBU_APosterioriLoopDetector_resetDetectors`

If an oscillating loop has been detected, a set of methods can be used to iterate through the list of loops:

- C functions: `ZEBU_APosterioriIterator_Create` / `ZEBU_APosterioriIterator_Destroy`

- C++ method: `ZEBU::APosterioriLoopDetector::Iterator::goToFirst`
- C function: `ZEBU_APosterioriLoopDetectorIterator_goToFirst`

- C++ method: `ZEBU::APosterioriLoopDetector::Iterator::goToNext`
- C function: `ZEBU_APosterioriIterator_goToNext`

- C++ method: `ZEBU::APosterioriLoopDetector::Iterator::isAtEnd`
- C function: `ZEBU_APosterioriIterator_isAtEnd`

- C++ method: `ZEBU::APosterioriLoopDetector::Iterator::getName`
- C function: `ZEBU_APosterioriIterator_getName`

Each loop has the same identifier as in the compilation reports (see Section 2.8).

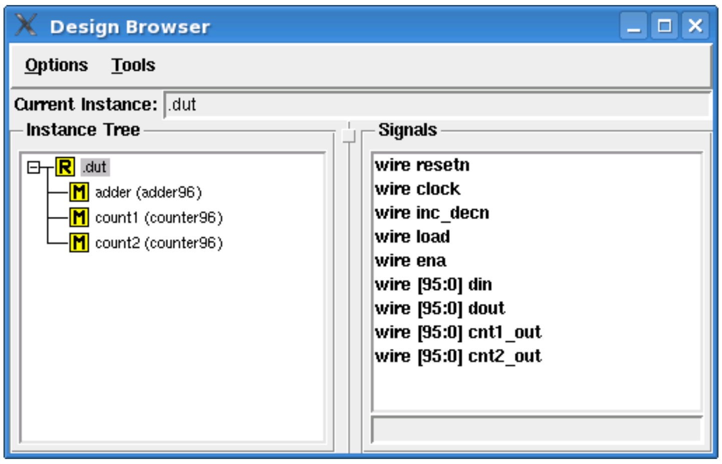
1.4.5 RTL Browser

When synthesizing with **zFAST**, it is now possible to browse the RTL hierarchy of a design after synthesis and easily get the equivalent path in the EDIF netlist.

The following command should be used:

```
$ zFastRtlBrowse <zcui.work> [-root <topname>]
```

The interface of this browser is very similar interface to the **zFAST** Stat Browser, with two panes. The left window displays the hierarchy tree of the design, and the right window displays the list of terminal elements in a given level of hierarchy:



```
wire resetn
wire clock
wire inc_decn
wire load
wire ena
wire [95:0] din
wire [95:0] dout
wire [95:0] cnt1_out
wire [95:0] cnt2_out
```

The EDIF equivalent path of an RTL terminal element can be obtained by right clicking on an RTL element in the right window and then select **Resolve.**

Note: when the RTL group name has been renamed (not “Default RTL Group”) or if there are several RTL groups, **`zFastRtlBrowse`** may not find automatically the requested file in `<zcui.work>`.
In such a case, the exact path to the synthesis directory has to be given in the command line: `<zcui.work>/synth/design_<RTL_group>`.

1.4.6 Performance Oriented Partitioning

It is now possible to activate the Performance Oriented Partitioning feature which, at zCore-level, automatically maps the design onto the design FPGAs. The main criterion for this partitioning is the achievable runtime performance and the FPGA filling rate may be slightly lower than with the automatic clustering.

This partitioning is activated by adding the following command in the additional command file declared in **zCui** through the **Advanced → Build System Parameters** panel:

```
zcorebuild_partitioning [ -zcore <my_zcore>] auto
```

Where `<my_zcore>` can be set to use performance-oriented partitioning for a given zCore only. If the `-zcore` option is not present, the automatic zCore-level partitioning is used for all the zCores of the design (user-declared zCores or automatically generated ones, as described in Section 2.12.4).

As for automatic zCore generation (described in Section 2.12.4), the ZeBu compiler proceeds with zCore-level partitioning instead of proceeding with the former automatic clustering. The settings of the **Clustering Control** frame are applicable for zCore-level partitioning:

- When **Full Automatic** is selected, the partitioning of each zCore is done with pre-defined settings.
- When **Automatic with Parameters** is selected, the filling rate sliders in the **Algorithm** frame are taken into account for zcore-level partitioning and they can be modified in **zCui**.
- When **Pre-existing Mapping** is selected, the compiler uses a mapping file generated in a former compilation.

Warning: In some cases, performance-oriented partitioning will not find a better solution than density-oriented clustering (i.e. automatic clustering), since the algorithms used can end up in a local minimum.

1.4.7 Aliases for Dynamic Forced Signals

When `force_dyn` commands are declared with `-global` option for the same signal at different levels of hierarchy or with different names, the force will be applied to the full net. The runtime control is possible with all the names corresponding to the signals declared in the `force_dyn` commands.

Note that in such cases, the following warnings are displayed in the log files:

```
### warning in DYN_FORCE [ZTB0164W] : A global dynamic force was previoulsy set on
hierarchical wire 'top.OUT_WIRE', an alias will be available at runtime for wire
'top.subWire.OUT'. Command 'force_dyn -wire "top.subWire.OUT" -type global' is conflicting
with command 'force_dyn -wire "top.OUT_WIRE" -type global'.
### warning in DYN_FORCE [ZTB0165W] : Discarding useless command 'force_dyn -wire
"top.subWire.OUT" -type global'
```

1.4.8 Improved Reporting for Static Timing Analysis

The Static Timing Analysis reports (**Create Timing DB** step in **zCui**) can now be accessed from a single index file which provides links to all other html reports. This index report is available in the contextual menu for **Create Timing DB** step.

All the reports have also been improved for multi-unit feature and to integrate information about IF FPGA and primary clocks.

1.4.9 Smart Z-ICE Remapping

When a design compiled for a given set of Smart Z-ICE connectors is actually run with a different set of connectors, user can now skip the recompilation of the design. In such a case, the remapping to the actual connectors should be declared in the `designFeatures` file.

The following line should be added when the design was compiled for connector `i` but the connector `j` is intended for runtime:

```
$smartZICE.connectorRemap_i = j;
```

Where:

- `i` is the compilation index of the Smart Z-ICE connector
- `j` is the index of the actual Smart Z-ICE connector for runtime

However, only the connectors of the unit connected to the host PC can be used for the Smart Z-ICE interface.

Example:

When the design was compiled for connectors 0 and 1 but the connectors actually used are connectors 2 and 3, the following lines should be added in the `designFeatures` file:

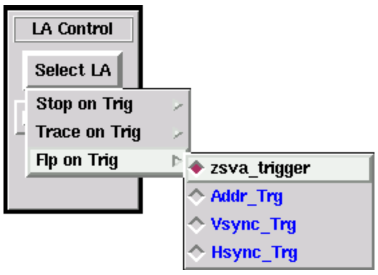
```
$smartZICE.connectorRemap_0 = 2;
$smartZICE.connectorRemap_1 = 3;
```

1.4.10 Improvement for Flexible Probes in zRun Logic Analyzer

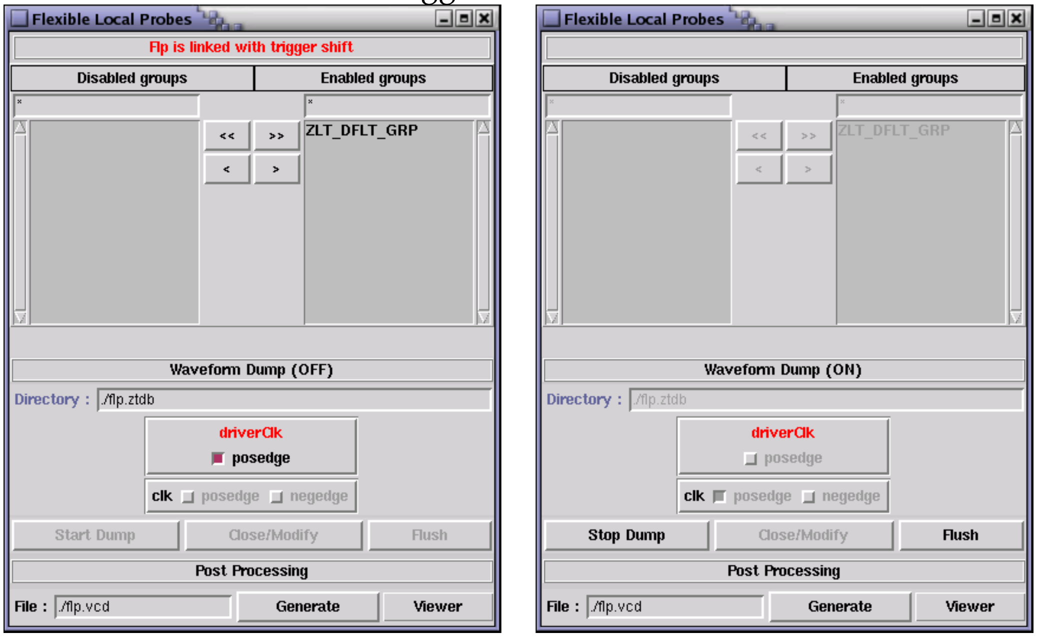
In the Logic Analyzer of **`zRun`**, it is now possible to automatically activate the dump from flexible probes on a trigger event. This improvement combines 3 existing features: programming the Stop On Trigger, activating the flexible probes after the trigger and run the design afterward.

The **`zRun`** graphical interface has been modified to support this new feature:

- The **Logical Analyzer Control** panel now provides the **Flp on Trig** value and any trigger declared in the DVE File can be selected in the combo box:



- The **Flexible Local Probes** panel displays a message which says that “Flp is linked with trigger shift” before the trigger event and the Waveform Dump frame is activated after the trigger event:


Flp is linked with trigger shift

There is no **zRun** Tcl command which integrates this feature but a script can be created with several commands for an equivalent scenario.

1.4.11 Temperature Alerts

The messages and behavior in case of FPGA overheat in the system have been improved:

- If one of the FPGAs on which the design is mapped reaches 90 °C (194 °F), the emulation stops and the temperature of all the FPGAs of the module is displayed in Celsius as in the following example:

```
Error : ========= Overheat detected =========
Error : M0 (physical M0) FC   : 71 C
Error : M0 (physical M0) FS   : 63 C
Error : M0 (physical M0) IF   : 90 C
Error : M0 (physical M0) F00   : 72 C
Error : M0 (physical M0) F01   : 71 C
Error : M0 (physical M0) F02   : 75 C
Error : M0 (physical M0) F03   : 45 C
Error : M0 (physical M0) F04   : 43 C
```

- Any connection to the corresponding module is impossible for any user as long as the overheated FPGA is over 85 °C (185 °F).

1.4.12 Using a SDP system with a design compiled with SRAM Trace Feature

Though SRAM Trace feature is not supported on SDP (Software Development Platform) systems, a runtime database compiled with SRAM Trace feature can now be used on a SDP system to avoid a dedicated re-compilation.

In such a case, the following warning message is displayed when the testbench connects to the system:

```
WARNING : The design instantiates some SRAM_TRACE drivers but the connected board is a SDP
(Software Development Platform): no access to embedded trace will be authorized. Use the
C/C++ functions ZEBU_Board_isHDP/ZEBU::Board::isHDP or the zRun tcl ZEBU_isHDP command to
disabled any embedded trace access
WARNING : Dismiss driver "<driver_name>", SRAM_TRACE is not authorized on SDP (Software
Development Platform)
```

1.4.13 Detecting SDP/HDP state of a system at runtime

It is now possible to detect at runtime if the system is HDP (Hardware Development Platform) or SDP (Software Development Platform) in order to activate or not HDP-only features (SRAM Trace in particular):

- C++ testbench: `ZEBU::Board::isHDP` method
- C testbench: `ZEBU_Board_isHDP` function.
- **`zRun`**: `ZEBU_isHDP` Tcl command.

1.5 Modified Default Settings Compared to V6_2_1

This section lists the modifications which may cause that a functional design and emulation environment may fail during compilation or emulation runtime.

1.5.1 Multi-unit systems with SDP and HDP Units

In the former software versions, it was not possible to map a multi-unit design when some units were SDPs (Software Development Platform) and some where HDPs (Hardware Development Platform).

With the present cumulative patch, if one of the units where the design is mapped is an SDP, the entire Zebu-Server system will be used with SDP capability and the following message is displayed:

```
-- ZeBu : zUtils : WARNING : All the units don't have the same HDP / SDP capabilities.
-- ZeBu : zUtils : WARNING :     Unit 0 is detected as HDP.
-- ZeBu : zUtils : WARNING :     Unit 1 is detected as HDP.
-- ZeBu : zUtils : WARNING :     Unit 2 is detected as SDP.
-- ZeBu : zUtils : WARNING :     Unit 3 is detected as HDP.
-- ZeBu : zUtils : WARNING :     Unit 4 is detected as HDP.
-- ZeBu : zUtils : WARNING :     The whole system will be forced to be SDP.
```

1.5.2 Modification for driverClock frequency estimation

The estimation of the driverClock frequency during Timing Analysis and System Place and Route has been modified to take the delays into account with more accuracy.
This modification may impact the estimated driverClock frequency which may be slightly lower (a few %) but the actual achievable runtime frequency should be the same.

1.5.3 Detection of Combinational Loops

By default, the ZeBu compiler automatically reported the combinational loops detected in the design in V6_2_1 version.

With the present cumulative patch, the detection of combinational loops is disabled by default because it sometimes increased drastically the duration of the compilation. It can be activated upon user request by adding the following command in an additional command file declared in **`zCui`** through the **Advanced** → **Build System Parameters** panel:

- To have the combinational loops reported after compilation (this was the default setting for V6_2_1):
  ```
  loop report
  ```
- To add the appropriate resources for runtime debugging (unchanged since V6_2_1):
  ```
  loop runtime_detect
  ```

1.5.4 Runtime Control for Dynamic Force

To improve runtime emulation frequency, the ZeBu runtime libraries have been modified so that there is no automatic writing into the hardware when a dynamic force is controlled from the user application:

- C++ testbench: `ZEBU::Signal::Force` / `ZEBU::Signal::Release` methods
- C testbench: `ZEBU_Signal_Force` / `ZEBU_Signal_Release` functions.
- **`zRun`**: `ZEBU_Signal_force` / `ZEBU_Signal_release` Tcl command.

The actual writing to the hardware must now be initiated manually by user:

- C++ testbench: `ZEBU::Board::writeRegisters` method
- C testbench: `ZEBU_Board_writeRegisters` function.
- **`zRun`**: `ZEBU_Monitor_flush` Tcl command.

The performance improvement is mostly sensible when several dynamic forces are modified simultaneously because the actual writing can be done only once.

Note that in C++/C co-simulation, the following methods/functions implicitly activate the writing to registers:

- C++ testbench: `ZEBU::Driver::run` and `ZEBU::Driver::wait` methods.
- C testbench: `ZEBU_Driver_run` and `ZEBU_Driver_wait` functions.

Therefore no call to `ZEBU::Board::writeRegisters` / `ZEBU_Board_writeRegisters` is necessary before these co-simulation run/wait commands.

1.6 Known Limitations for V6_2_1 B_00

In addition to the V6_2_1 limitations described in Chapter 3, the following additional limitations are applicable in the B_00 cumulative patch.

1.6.1 Deprecated Environment Variable

It is no longer possible to accelerate the dumping of dynamic probes to a FSDB file by setting the `ZEBU_WAVEFORM_DUMP_USE_THREAD` environment variable. This variable should not be set with the present cumulative patch.

If this environment variable is set, a warning will be displayed in the runtime log and the dumping process is not impacted.

Note that the `ZEBU_FSDB_DUMP_USE_THREAD` environment variable can be set to `YES` parallelize the writing of the FSDB file if necessary, as described in Section 2.15.3.

1.6.2 Disk Resource Conflicts for .ztdb Files

The `.ztdb` file format is used for both Flexible Probe (dumping file declared by user) and Off-line Debugging (sniffer directory declared by user). When several instances of these features are used simultaneously, several files with this extension coexist and the ZeBu runtime library may create several hardlink for them. If these files are located on different disks/partitions (in particular if some are on a local partition and some are remote resources), the following error may be displayed:

```
ERROR : ZPRIV0062E : Cannot create link "[<second path>/]<second
name>.ztdb/<space file>" on "/[<first path>/]<first name>.ztdb/<space
file>", Invalid cross-device link
```

To avoid this error, it is recommended to declare paths on the same disk/partition for Flexible Probes dumping files and for sniffer directory.

1.6.3 Some Initialization Phases not Carried out in Specific C/C++ Testbenches

When a C/C++ testbench controls the design using neither transactors nor cosimulation drivers (`HDL_COSIM`, `C_COSIM`, `MCKC_COSIM`), some of the initialization phases are not carried out, namely on memory. Under these circumstances, the design will not work even if no error is generated.

To avoid this, user must identify the testbench in the `designFeatures` file in the same way as is done for multi-process.

```
$nbProcess = 1;
$process_0 = "#process_name";
```

Note that the `#process_name` field must match the 3rd parameter of the open call (`zebu.work`, `designFeatures`, `process_name`).

This parameter not being mandatory, the default value can be used:

```
$nbProcess = 1;
$process_0 = "default_process";
```

At the beginning of the connection, an unidentified process will generate the following warning messages:

```
-- ZeBu : cpp_test_bench : WARNING : A process name has been specified "default_process" at
          open time, but it is not specified in the "./designFeatures" file.
-- ZeBu : cpp_test_bench : WARNING :     The list of specified process names is :
-- ZeBu : cpp_test_bench : Looking for a connection (pid 28241 at Tue 3 8 2010 - 17:46:57)
-- ZeBu : cpp_test_bench : "default_process" is a control-only process working on
          "../zebu.work".
```

An identified process will issue the following:

```
-- ZeBu : cpp_test_bench : Looking for a connection (pid 28200 at Tue 3 8 2010 - 17:46:11)
-- ZeBu : cpp_test_bench : "default_process" is a full-capability process working on
          "../zebu.work".
```

Unidentified processes can be used to investigate or control clocks, memories, logic analyzers, triggers, signals, etc. They cannot be used to control top-level IOs.

1.6.4 Using a Compatible `zInstall` from a Different Release

When a compatible **`zInstall`** from a different release than the one in use is running AND **`zInstall`** of the current release was never launched, one of the following error messages is generated (reported in RT#23927):

```
-- ZeBu : tb : ERROR : LUI0972E : Cannot detect the server which registered the design
   (97599494).
-- zServer : ERROR : LIPFC0012E : Cannot create fifo
   "/zebu/V6_2_1_B_100805/fifo_811309_S_2_CC0" : No such file or directory.
```

1.6.5 Using `zSelectProbe` and `zSelectSignal` in Same `zebu.work`

Using **`zSelectProbe`** and **`zSelectSignal`** in the same `zebu.work` may result in inconsistent *Simulated Combinational Probes* and *Dynamic Probes* databases which will be manifested, for example, by missing signals in waveforms.

1.7 Fixed Issues in V6_2_1 B_00

The following issues are now fixed, either in an intermediate patch (A_XX) or more recently in the B_00 cumulative patch.

In order to improve legibility of this Release Note, some minor corrections introduced in V6_2_1 software or in the present cumulative patch are not listed.

1.7.1 Using SUSE10 with `zKernel` memory allocation

The specific memory allocation process of the boot procedure of SUSE 10 operating system caused some communication failure when **`zInstall`** was launched with `-bootAllocAddr` option (described in Section 2.1 of the present document).

1.7.2 Compilation Fixed Issues

- When a `force assign` command is declared for signals connected to the same physical wire, the command is now ignored and the following warning message is displayed:

  ```
  ### warning in Force Assign Processing [BDC0078W] : Wire 'dut.xyz'
  and port 'dut.a.b.c.ck' belong to the same hierarchical wire,
  discarding command.
  ```

2 New Features in V6_2_1

This document is based on the V6_2_1 Release Note: Chapter 1 is dedicated to specific information for the B_00 cumulative patch; other sections of the V6_2_1 Release Note have been updated only in case of modification due to the cumulative patch (removal of out-of-date limitations, in particular).

2.1 New supported ZeBu-Server System Configurations

Most of the content of this section is outdated. The following sections provide updated information:

- Section 1.2 describes the installation procedure for V6_2_1 with cumulative patch B_00.
- Section 1.4.2 provides information for the support of 5-unit systems.

The following information is still applicable for any multi-unit system with 3 units or more:

- It is mandatory to boot the host PC with `memmap` and use the `-bootAllocAddr` option for **`zInstall`**, in order to allocate the 64 Mbits of the system memory for ZeBu-PC communication:
  1. Modify the boot command line to set the `memmap` option (with `size$address` parameter), for example `memmap=64M$64M`.
  2. Restart the PC.
  3. Launch **`zInstall`** with the `-bootAllocAddr` option and force the base address. When `memmap` is set as in the above example, the address to map the syntax should be one of the following (since `64M = 0x4000000`):

```
$ zInstall -bootAllocAddr 64M
$ zInstall -bootAllocAddr 0x04000000
```

2.2 zFAST Script Mode

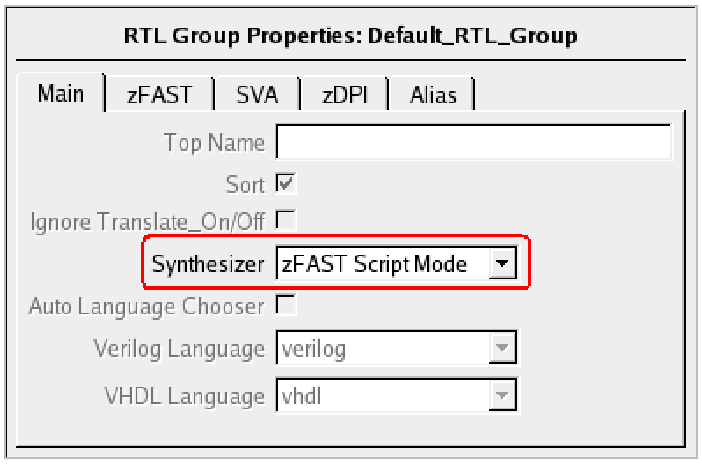
In addition to the standard mode described in the *ZeBu-Server Compilation Manual* (Rev b), **zFAST** provides a script mode which is based on input command files very similar to the simulation scripts. Using the script mode provides more flexible options for the analysis and elaboration steps before synthesis. Synthesis itself is not launched by the script: **zCui** launches the synthesis tasks, taking into account user settings in the graphical interface for chunks and bundles.

The script mode provides the same basic features as the standard mode since it is based on the same synthesizer and the synthesis output files are the same. As for standard mode, only one RTL group is recommended.

The script mode provides better flexibility than the standard mode for the source file-based options for synthesis:

- With the standard mode, you can define which library is to be used for analysis of a given RTL source file but there is no other file-based option.
- With the script mode, different options can be declared for each RTL source file in the Tcl script written by user, named **zFAST** script in the later sections.

In **zCui**, the script mode is selected as a specific synthesizer in the **Properties** panel of the RTL group:


RTL Group Properties: Default_RTL_Group

Main | zFAST | SVA | zDPI | Alias

Top Name:
Sort: ☑
Ignore Translate_On/Off: ☐
Synthesizer: zFAST Script Mode
Auto Language Chooser: ☐
Verilog Language: verilog
VHDL Language: vhdl

The tabs that are replaced by commands in the **zFAST** script are not available for the script mode.

All the other **zCui** features are unchanged: **zCui** manages the whole compilation process, including incremental compilation and job scheduling.

The **zFAST** script mode is fully described in the ***zFAST* *Synthesizer Manual***, Rev a, which is part of the V6_2_1 documentation package.

2.3 Improved Support of RTL Paths

When synthesizing with **zFAST**, the RTL paths to signals and instances can be used for more compilation commands than before and can be used in the DVE file.

When synthesizing with a third-party synthesizer with **zRtlFrontEnd**, only the `probe_signals` command is available in RTL-based compilation scripts and only EDIF paths can be used for other commands and in the DVE file.

The same limitation is applicable for runtime support of RTL paths.

2.3.1 RTL-based Compilation Scripts

The **Sources → Design → Probe Files** tab in the **Project Tree** pane of **zCui** has been replaced by a new tab, **RTL-based Compilation Scripts**. Unlike the previous tab which could only be used to declare probe files (`*.prb` and `*.tcl` extensions), the **RTL-based Compilation Scripts** tab can be used to declare Tcl scripts (*.tcl extension only) with the following commands: `force`, `force_dyn`, `probe_signals` and `tristate`.

The declaration based on RTL paths is applicable for some elements in the following system-level compiler commands:

Command/Option	EDIF	RTL

`force`	`-pin <port>`
	`-net <net>`
	`-pin_input <port>`
	`-pin_output <port>`
	`-source_pin <port>`
	`-source_net <net>`
		`-rtlname <rtl_port>`
		`-rtlname <rtl_net>`
		`-rtlname_input <rtl_port>`
		`-rtlname_output <rtl_port>`
		`-rtlname_source <rtl_port>`
		`-rtlname_source <rtl_net>`

`force_dyn`	`-wire <signal>`
	`-wire_file <file>`
	`-pin <port>`
	`-pin_file <file>`
		`-rtlname <rtl_signal>`
		`-rtlname_file <rtl_file>`
		`-rtlname <rtl_port>`
		`-rtlname_file <rtl_file>`

`probe_signals` (applicable for all types: Dynamic, Flex, C-calls and Value Change)	`-wire <signal>`
	`-wire_file <file>`
	`-port <port>`
	`-port_file <file>`
	`-instance <instance>`
	`-instance <inst> -wire <s>`
	`-instance_file <file>`
		`-rtlname <rtl_signal>`
		`-rtlname_file <rtl_file>`
		`-rtlname <rtl_signal>`
		`-rtlname_file <rtl_file>`
		`-rtlname <rtl_instance>`
		`-instance <rtl_i> -rtlname <rtl_s>`
		`-rtlname_file <rtl_file>`

`probe_signals` (Flex type only)	`-enable_wire <signal>`
	`-enable_port <port>`
		`-enable_rtl <rtl_signal>`
		`-enable_rtl <rtl_port>`

`tristate`	`<signal>`	`-rtlname <rtl_signal>`
Note that options not listed in this table are not impacted by the declaration of RTL paths.		

RTL paths are also supported for pattern-matching declarations (`–fnmatch` option), except in case of conflict between special characters in the RTL paths and pattern matching characters.

Examples:

Declaring flexible probes with RTL paths for the interface of `top.instance`:

```
probe_signals -type flexible -rtlname top.instance
```

Declaring flexible probes with RTL paths for a `foo` signal in `top.instance`:

```
probe_signals -type flexible -instance top.instance -rtlname {foo}
```

Declaring a flexible probe with a pattern-matching RTL path:

```
probe_signals -type dynamic \
   -rtlname {top.instance.in* top.instance.out*} –fnmatch
```

Declaring a flexible probe RTL path for the signal and the enable:

```
probe_signals -type flexible -rtlname {top.instance} \
  -enable_rtl top.instance.probe_en
```

Notes:

- An RTL-based script supports both RTL paths declared with `-rtlname` options and EDIF paths with the former syntax but mixing both RTL and EDIF paths may be difficult for legibility.
- Existing EDIF-based declarations for the back-end compiler are still supported but should not be used simultaneously in RTL-based scripts.
- All other compilation commands for zCore definition, mapping constraints, and Direct ICE interface only support signals declared with EDIF paths.
- The ***ZeBu-Server Compilation Manual*** has not been updated for this feature. The syntax of these commands in Revision b is still applicable, but the files in which these commands are declared are no longer correct for use with RTL paths (table in Section 3.7 of the manual is only applicable for EDIF paths).

2.3.2 DVE file with RTL paths

It is possible to declare RTL hierarchical paths in the DVE file by adding the following line in it. When this parameter is present, the paths in the DVE file are automatically interpreted as RTL paths. Else, the paths in the DVE file are regarded as EDIF paths.

```
defparam rtlname = yes;
```

Where `yes` can be replaced by `true`.

Notes:

- RTL paths are not supported for elements such as `zceiClockPort`, `zClockPort` and `zIceClockPort`.
- RTL path expressions are Verilog compliant.
- All special characters are escaped.

Example:

The SRAM trace driver is instantiated with RTL paths which include some special characters:

```
defparam rtlname = yes;
SRAM_TRACE trace (
    .output_bin({
           top.mult1.r0,
           top.mult1.r1,
           top.mult1.\x[0] ,
           top.mult1.\z.1_
      })
);
```

2.3.3 Accessing RTL instances at runtime

It is now possible to access signals of the design with their RTL path from a C++ or a C testbench, in co-simulation or transaction-based environments. A specific header file (`zRtl_rt.h`) has to be included in order to support the corresponding data structures.

For any RTL object (that may end as a register, or as the content of a **`zMem`**-based memory, etc.), the access is possible via a `ZeBu::Signal` with its RTL path through the usual methods. The following example shows how to get a `ZeBu::Signal` with its RTL hierarchical path (adding a "`.`" before the top name of the design is mandatory):

```
#include <zRtl_rt.h>
[...]
// Load and initialize the data structures:
 esx::rtManager rtm (z);
 rtm.load ("zcui.work");
[...]
 // Get a Signal based on the RTL path
 Signal *addr = rtm.getSignalFromPath (".top.addr");
 // Get a Signal based on the RTL path of a memory word at address 10
 Signal *mem_word_10 = rtm.getSignalFromPath (".top.mem[10]");
[...]
```

In order to get a `ZEBU::Memory` object from an RTL path, the method `getEdifPathsOfIns` of `esx::Manager` has to be used to get the EDIF path of the memory, as shown below:

```
#include <zRtl_rt.h>
#include <iostream>
using namespace std;
int main()
{
 esx::Manager *m = esx::Manager;
 m->load("zcui.work/zebu.work/RTlDB", true);
 string rtl_path= "top.modgen.\\loopi[0] [1].loopj[0].\\s[0] ";
 string edf_path;
 m->getEdifPathOfIns(rtl_path, edf_path);
 cout << edf_path << endl;
}
```

In both cases above, the testbench has to be linked with the `zRtlRt.so` and `zRtl.so` libraries (both are mandatory) in addition to the ZeBu runtime library:

```
$ g++ -o tb tb.cc -I$ZEBU_ROOT/include -L$ZEBU_ROOT/lib \
   -lzRtl -lzRtlRt -lZebu –I
```

2.4 Improved Reliability for FPGA Compilation

This software release provides the following enhancements for FPGA compilation:

- The input netlists for ISE are now optimized to reduce the duration of the first tool of FPGA Place & Route process without impacting runtime accessibility. It is possible to disable this new feature by setting the `ZEBU_NO_MULTIQUIFY` environment variable before launching **`zCui`**.
- To reduce the duration of FPGA compilation, an additional placement constraint is now automatically added by the ZeBu compiler. It is possible to disable this new feature by setting the `ZEBU_NO_USET_ON_SOCKET` environment variable before launching **`zCui`**.
- To reduce the duration of FPGA compilation when **`zMem`** memory models are instantiated, an additional timing constraint is now automatically added by the ZeBu compiler.

In addition to these enhancements, this software release includes version 11.4i_patched of the Xilinx ISE software subset for Virtex-5 FPGA Place and Route for ZeBu. The makefiles for FPGA Place & Route generated by the ZeBu compiler now include the appropriate commands to support the ISE 11.4i_patched software. Note that Xilinx ISE 11 requires specific Xilinx licenses which should be requested from EVE (license@eve-team.com). After installation of this software release, it is not recommended to use an older version of the Xilinx ISE subset for ZeBu.

2.5 Improvements for Offline Debug Feature

This Offline Debug feature introduced in V6_2_0B has been improved in order to make the compilation easier: it is no longer necessary to modify the DVE file to instantiate the injector transactor.

This feature can be activated in the **zFAST** panel of **`zCui`**. When selecting **Activate ZeBu Offline Debugger**, the compilation options are set in order to provide the appropriate runtime accessibility to signals, memories and ports of the design.

- At runtime, the sniffer should be activated either from the testbench or from **`zRun`** in order to store a frame with the information for offline debugging.
- In a later session, it is possible to use the sniffed frame to stimulate the design in replacement of the actual verification environment (JTAG debugger or additional transactor).

Offline Debug requires specific license features for compilation and runtime (contact license@eve-team.com).

A dedicated Application Note will be available soon.

2.6 Improved Clock Generator Accuracy

To provide for a wider range of frequencies for primary clocks from the ZeBu clock generator, it is now possible to declare a 32-bit accuracy in the DVE file instead of the default 24-bit one:

```
defparam fk_clockgen_accuracy = 32 ;
```

This declaration is automatically propagated for runtime.

If the 32-bit accuracy is not declared in the DVE file, the default 24-bit accuracy prevails.

Note that the 32-bit accurate clock generator may impact the duration of the FPGA P&R for IF FPGA, or event the capability to compile it.

2.7 Value Change Feature

The present software release introduces the runtime support of the Value Change feature which detects that a signal has changed and activates a dedicated trigger. This feature is integrated in the C++/C API for use in C co-simulation.

The set of signals which can be connected to this trigger is declared by user in an RTL-based compilation script with a specific type (`-type vc`) for the `probe_signals` command:

```
probe_signals -type vc [-options]
```

Where the `[-options]` are similar to options for other types of probes, as described in the *ZeBu Compilation Manual* (Rev b).

At runtime, the signals declared as Value Change probes have to be enabled so that the Value Change trigger automatically fires when one of these signals changes. Note that some of the signals may not be enabled in order to detect a change in a subset of the probes.

The ZeBu runtime API provides the control capability for the Value Change feature from a C++ or C testbench:

- Enable/disable signals declared for Value Change during compilation.
- Wait for the Value Change trigger.
- Iterate over the signals which were detected as changed.

For a C++ testbench, the methods which provide control of the Value Change feature from the testbench are available in the `ZEBU::ValueChange` class.
The header file to include in the testbench is `libZebu.hh`. The detailed interfaces of the methods are described in the `ValueChange.hh` file.

For a C testbench, the header file to include in the testbench is `libZebu.h`. The detailed interfaces of the functions are described in the `ZEBU_ValueChange.h` file.

Note that the ZeBu compiler has been fixed for minor issues when signals were declared for Value Change:

- When declaring a signal for Value Change, it is now mandatory to declare the `clock_name` parameter.
- The Value Change trigger was sometimes raised at the very beginning of emulation (cycle 1), which was not correct and made it impossible to debug with this feature (reported in RT#22951).
- Several Value Change detectors were added for the same wire and caused erroneous activations of the Value Change triggers.

2.8 Detection of Combinational Loops

~~Starting with V6_2_1, the ZeBu compiler automatically detects the combinational loops in the design, when they are between different zCores or inside a zCore.~~

The default setting for the detection of combinational loops in the design has changed for V6_2_1 B_00. Section 1.5.3 provides updated information.

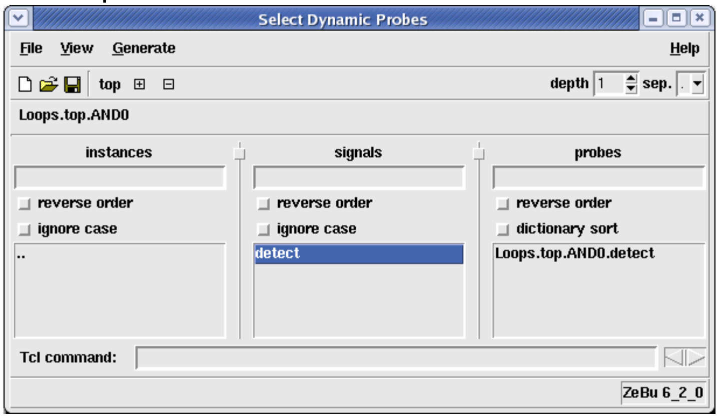
When some combinational loops are detected they are reported in a dedicated section of the compilation reports:

- Inter-zCores loops are reported by the system-level compiler and reported in `zTopBuild_report` file (section “14. Inter-partition combinational loops checks”).
- Combinational loops which are inside a zCore are reported by the corresponding zCore-level compiler (section “8.5. Combinational loop reporting” and section “8.6. Combinational loop instrumenting”).

The Zebu compiler can insert in the runtime database the appropriate resources to support runtime debugging of combinational loops. The following command should be added in an additional command file declared in **`zCui`** through the **Advanced → Build System Parameters** panel:

```
loop runtime_detect
```

In such a case, a dedicated register is available for each combinational loop in the `Loops.<path_to_loop>` hierarchical level in **`zSelectProbes`**:



Loops.top.AND0

..

detect

Loops.top.AND0.detect

In the `Loops` hierarchical level, each loop is identified with the same name as in the compilation reports (`top.AND0` here) and the `detect` register is available: it switches to 1 when the corresponding combinational loop oscillates at runtime. In the testbench, it is possible to iterate through the detection registers in order to check that a combinational loop is not the source of design instability or runtime misbehavior.

Once the detection registers have toggled to 1, they are not automatically reset to 0. This can be done manually for all the detection registers at once, by modifying from the testbench the value of the `zdve_zloops_reset_dve_reg` register which is created automatically and is similar to a DVE register:

```
Signal* rst = z->getSignal("zdve_zloops_reset_dve_reg");
```

With this example, `rst` has to be set to 1 to reset the detectors and then set to 0 to go on with detection of oscillations.

2.9 Installation New Features

This software release provides the following enhancements for installation:

- The list of sample configuration files delivered with this software release has been updated, in particular to support loopback modules (identified as `LB`). These are available after installation in the `$ZEBU_ROOT/etc/configurations` directory:

ZeBu-Server System	FPGA Modules	Name of Sample Configuration File
5-slot single-unit	4x16C + 1xLB	`sample_ZSE_5S_16C_16C_16C_16C_LB_LX330.tcl`
"	3x16C + 2xLB	`sample_ZSE_5S_16C_16C_16C_LB_LB_LX330.tcl`
"	2x16C + 3xLB	`sample_ZSE_5S_16C_16C_LB_LB_LB_LX330.tcl`
"	4x8C + 1xLB	`sample_ZSE_5S_8C_8C_8C_8C_LB_LX330.tcl`
"	3x8C + 2xLB	`sample_ZSE_5S_8C_8C_8C_LB_LB_LX330.tcl`
"	2x8C + 3xLB	`sample_ZSE_5S_8C_8C_LB_LB_LB_LX330.tcl`
"	4x4C + 1xLB	`sample_ZSE_5S_4C_4C_4C_4C_LB_LX330.tcl`
"	3x4C + 2xLB	`sample_ZSE_5S_4C_4C_4C_LB_LB_LX330.tcl`
"	2x4C + 3xLB	`sample_ZSE_5S_4C_4C_LB_LB_LB_LX330.tcl`
5-slot multi-unit	4x16C + 1xLB	`sample_ZSE_5SM_16C_16C_16C_16C_LB_LX330.tcl`
"	3x16C + 2xLB	`sample_ZSE_5SM_16C_16C_16C_LB_LB_LX330.tcl`
"	2x16C + 3xLB	`sample_ZSE_5SM_16C_16C_LB_LB_LB_LX330.tcl`
"	4x8C + 1xLB	`sample_ZSE_5SM_8C_8C_8C_8C_LB_LX330.tcl`
"	3x8C + 2xLB	`sample_ZSE_5SM_8C_8C_8C_LB_LB_LX330.tcl`
"	2x8C + 3xLB	`sample_ZSE_5SM_8C_8C_LB_LB_LB_LX330.tcl`
"	4x4C + 1xLB	`sample_ZSE_5SM_4C_4C_4C_4C_LB_LX330.tcl`
"	3x4C + 2xLB	`sample_ZSE_5SM_4C_4C_4C_LB_LB_LX330.tcl`
"	2x4C + 3xLB	`sample_ZSE_5SM_4C_4C_LB_LB_LB_LX330.tcl`

- `zSetupSystem` has been optimized in such a way that the ZeBu-Server system is now loaded as few times as possible for the calibration process.

- When the diagnostics patch declared in the `setup.zini` file does not match the actual configuration detected by `zSetupSystem`, a message is now displayed to give the name of the appropriate diagnostics patch, in addition to the error message (requested in RT#21826):

```
-- ZeBu : zUtils : Computed compatible patch name : C00-00-512_43_xx_xx_xx_xx.diag
-- ZeBu : zUtils : Info : "xx" means that all module types are acceptable.
-- ZeBu : zUtils : Info : "Er" means that detected board is unknown.
```

- In a multi-unit system, `zSetupSystem` now powers up the Smart Z-ICE interfaces of all units. Prior to this software release, `zSetupSystem` powered up the Smart Z-ICE interface of unit U0 only.
  After a `zUtils -initSystem` in a multi-unit system, the following message is displayed (on the PC from which `zUtils -initSystem` is launched) for each and every unit (`Ux`) present in the system:

```
-- ZeBu : zUtils : Power ON the Ux Smart Z-ICE to 1.5 V.
```

2.10 `zCui` New Features

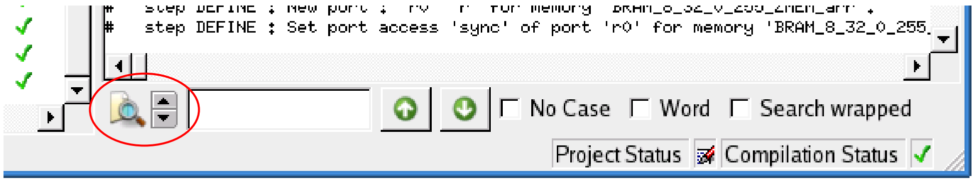
2.10.1 New Search Tools for Source files and Logs

`zCui` now provides search capability in the source files and in the logs with different criteria. The toolbar at the bottom of the source/log pane shows different icons for these criteria:

- Go to Line (): this is the default mode.
- Search for text ()
- Filtering ()
- Error Navigation ()

A specific toolbar () is available to modify some viewing properties such as the font size and automatic scrolling for the current source/log file.

To change the active search toolbar, use the up-down arrows placed next to the search criteria icon:



```
#    step DEFINE : Set port access 'sync' of port 'r0' for memory 'BRAM_8_32_0_255
```

Note that the source/log files can also be edited with a separate editor by selecting **Show in a separate window** in the contextual menu of the corresponding task.
The editor which displays the source/log files can be set by user in the **Preferences → Compiler** table (**External facilities** frame) in the **Project** view.

2.10.2 Improvements when modifying FPGA P&R Options

In the **Backend → FPGA P&R** panel, the left-hand area of the **FPGA ISE Parameters** frame is now displayed as a spreadsheet. This new format makes horizontal scrolling much easier because the FPGA identifier is always visible.

2.10.3 Archiver Feature in zCui

The **zCui** Archiver feature is now available in 2 different modes:

- Batch mode: for runtime files only.
- Graphical mode: for runtime files as well as project sources and debug files.

2.10.3.1 Archival in Batch Mode

In batch mode, only back-end files destined for emulation runtime can be archived. To archive other files such as project sources and debug files, see Section 2.10.3.2.

To archive the default back-end only:

```
zCui -p <project>.zpf -n -t <def_backend>  (with no file extension)
zCui -p <project>.zpf -n -t <def_backend>.tgz
zCui -p <project>.zpf -n -t <def_backend>.tar.gz
zCui -p <project>.zpf -n -t <def_backend>.tbz2
zCui -p <project>.zpf -n -t <def_backend>.tar.bz2
```

Where:

- `-n` option stands for "no GUI" and is mandatory.
- `<def_backend>`: any name to identify the default back-end archive.
- `tbz2` and `tar.bz2` allow for more compression than `tgz` and `tar.gz`

As a result, 2 tar files are generated:

- `<def_backend>.tgz` (or `tar.gz`, `tbz2`, `tar.bz2`):
  Archived files (with dereferenced links)
- `<def_backend>_links.tgz` (with `tgz` extension only):
  No dereferenced links (without archived files)

If you specify...	...then the following files will be generated:
`<def_backend>` (with no file extension)	`<def_backend>.tgz` `<def_backend>_links.tgz`
`<def_backend>.tgz`	`<def_backend>.tgz` `<def_backend>_links.tgz`
`<def_backend>.tar.gz`	`<def_backend>.tar.gz` `<def_backend>_links.tgz`
`<def_backend>.tbz2`	`<def_backend>.tbz2` `<def_backend>_links.tgz`
`<def_backend>.tar.bz2`	`<def_backend>. tar.bz2` `<def_backend>_links.tgz`

To archive the default back-end along with other back-ends:

```
zCui -p <project>.zpf -n -t <def_backend>.tgz
  -b <my_backend_X> -t <my_backend_X>.tgz
```

Where `<my_backend_X>` is any back-end other than the default back-end
In the example above, `tar.gz`, `tbz2`, and `tar.bz2` extensions can also be used and the same remarks as above apply.

Note: If `–t <my_backend_X>` is missing in the command, then only the default back-end will be archived. In other words, `<my_backend_X>` will be compiled but not archived, and the following files will not be generated: `<my_backend_X>.tgz` and `<my_backend_X>_links.tgz`

The following examples show that user can issue a single command to compile and/or archive several back-ends. In all examples, the default backend is always compiled AND archived as `default_backend.tgz`.

Example 1:

```
zCui -p project.zpf -n -t def_backend.tgz
  -b my_backend_1 -t archive_1.tgz
  -b my_backend_2 -t archive_2.tgz
  -b my_backend_3 -t archive_3.tgz
```

`backend_1`, `backend_2`, and `backend_3` will be archived respectively as `archive_1.tgz`, `archive_2.tgz`, and `archive_3.tgz`.

Example 2:

```
zCui -p project.zpf -n -t def_backend.tgz
  -b my_backend_1
  -b my_backend_2 -t archive_1.tgz
```

`backend_1` will be compiled only (it will not be archived).
`backend_2` will be archived as `archive_1.tgz`.

Example 3:

```
zCui -p project.zpf -n -t def_backend.tgz
  -b my_backend_2 -t archive_1.tgz -t archive_2.tgz
```

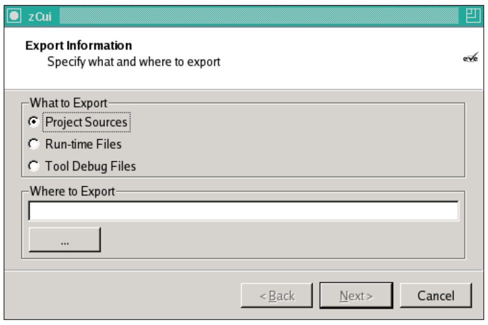
`backend_2` will be archived as `archive_1.tgz` and `archive_2.tgz` (`archive_1.tgz` and `archive_2.tgz` are identical).

2.10.3.2 Archival in Graphical Mode

In graphical mode, user can archive back-end files destined for emulation runtime as well as project sources and debug files (runtime, debug, log). The list of archived files depends on the user-specified compilation options and is displayed in the last **Export Information** window.

Whatever the type of files you want to archive, the following first steps are identical:

1. In the **`zCui`** menu bar, select **File** → **Archive**.
2. In the **Export Information** window:
   a. Select **Project Sources**, **Runtime Files**, or **Tool Debug Files**.
   b. Type the name (and path) of the archive file in the **Where to Export** field or click the **[...]** button to select an existing archive file.


zCui
Export Information
Specify what and where to export
What to Export
(●) Project Sources
( ) Run-time Files
( ) Tool Debug Files
Where to Export
...
< Back | Next > | Cancel

The table below shows what file extensions are generated with respect to the file extension declared in the command:

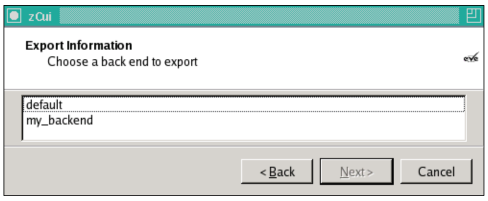
If you declare…	…then the following files will be generated:
`<my_backend>` (with no file extension)	`<my_archive>.tgz` `<my_archive>_links.tgz`
`<my_archive>.tgz`	`<my_archive>.tgz` `<my_archive>_links.tgz`
`<my_archive>.tar.gz`	`<my_archive>.tar.gz` `<my_archive>_links.tgz`
`<my_archive>.tbz2`	`<my_archive>.tbz2` `<my_archive>_links.tgz`
`<my_archive>.tar.bz2`	`<my_archive>.tar.bz2` `<my_archive>_links.tgz`

Archival of Project Sources:

1. Select **Project Sources** in the **Export Information** window, specify the archive file name and click **Next**.
2. In the next **Export Information** window, click the **Archive** button.
3. Wait until the progress bar has reached **100%** and click the **Finish** button.

Archival of Runtime Files

1. Select **Runtime Files** in the **Export Information** window, specify the archive file name and click the **Next** button.
2. In the next **Export Information** window, select the back-end you want to archive (**default** for default back-end, or any other back-end in the list) and click the **Next** button. Note that only one back-end can be selected in an archive.


zGui

Export Information
Choose a back end to export

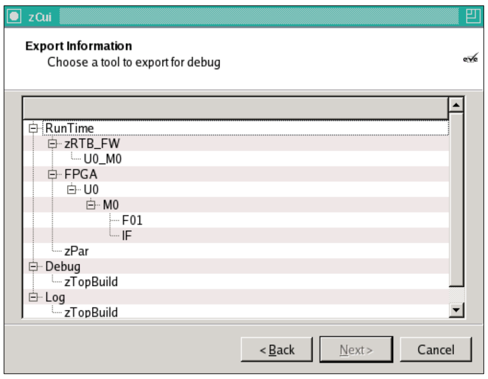
default
my_backend

3. In the next **Export Information** window, click the **Archive** button.
4. Wait until the progress bar has reached **100%** and click **Finish**.

Archival of Tool Debug Files

1. Select **Tool Debug Files** in the **Export Information** window, specify the archive file name and click the **Next** button.
2. In the next **Export Information** window, select the back-end you want to archive (**default** for default back-end, or any other back-end in the list) and click the **Next** button. Note that only one back-end can be selected in an archive.

3. In the last **Export Information** window, zoom in on the type of tool debug files you want to archive and click the **Next** button. Please note that multiple selections are not allowed.



4. In the next **Export Information** window, click the **Archive** button.
5. Wait until the progress bar has reached **100%** and click **Finish**.

2.11 zFAST Synthesis New Features

In addition to the script mode (described in Section 2.2 of the present Release Note), this software release introduces the following enhancements for `zFAST` synthesis:

- **`zFAST`** can now synthesize tristate wires declared with `tri0/tri1/trireg` in the Verilog source files (requested in RT#21785). For that purpose, the following attribute has to be declared in the **Additional zFAST Attribute file** in **`zCui`**:
  ```
  Compile:ConvertTri=true
  ```
  This attribute is `false` by default (such tristate wires cause a fatal error during synthesis).
- **`zFAST`** can now synthesize `pmos/nmos` Verilog primitives (requested in RT#22624). For that purpose, the following attribute has to be declared in the **Additional zFAST Attribute file** in **`zCui`**:
  ```
  Compile:ConvertMOS=true
  ```
  This attribute is `false` by default (such primitives cause a fatal error during synthesis). Note that the `pmos/nmos` Verilog primitives are converted to `bufif0/bufif1` and the strength value is ignored.

- It is now possible to use the optimized bit-enable mode of **zMem** for a particular memory (`set optimize_bie`). For that purpose, the following attribute has to be declared in the **Additional zFAST Attribute file** in **zCui**:
```
MemoryType:OptimizeBie=(<mod1.mem1> <mod2.mem2> <mod_i.mem_j>)
```
- VHDL files can now be automatically sorted-out:
  - In standard mode, this is the new default (the **Sort** checkbox is automatically selected when creating a project).
  - In script mode, the `-sort` option has to be added in the analysis command for the VHDL files. The order of the analysis commands for Verilog/SystemVerilog and VHDL files impacts the sorting capability.

- Verilog hierarchical references to VHDL were not supported (reported in RT#22209). In order to support such hierarchical references, the following line should be added in the **zFAST Additional Attribute File** declared in **zCui**:
```
Compile:HrefToVHDL=true
```

- The following VHDL pragma to tell the synthesizer to resolve a wire as WAND was not supported (requested in RT#21987):
```
-- synopsys resolution_method wired_and
```
Now, the behavior is the following (where wire is always resolved as WAND):
```
-- eve resolution_method
```
and the following will only work if `compile:obeysynopsystranslate` is set to `true` in the **zFAST Additional Attribute File** declared in **zCui**:
```
-- synopsys resolution_method
```

- VHDL and Verilog parsing for synthesis pragma are now similar (requested in RT#21926):
  - Always processed:
    - VHDL: `-- eve translate_off/on`
    - Verilog: `// eve translate_off/on`
  - Processed only if `compile:obeysynopsystranslate=true`:
    - VHDL: `-- synopsys translate_off/on`
    - Verilog: `// synopsys translate_off/on`
  - Processed only if `compile:obeysynopsystranslate=true`:
    - VHDL: `-- pragma translate_off/on`
    - Verilog: `// pragma translate_off/on`

- In order to investigate the synthesis results for the Xilinx primitives actually inferred by **zFAST**, a dedicated tool can now be accessed from **zCui**. Once the RTL group is successfully synthesized, right click on the RTL Group name in the project tree and select **Explore the Group with zFASTStatBrowse** in the contextual menu. A separate window opens, which shows the instances of the group.
  This tool is fully described in the ***zFAST Synthesizer Manual***, Rev a, which is part of the V6_2_1 documentation package.

2.12 Back-End Compilation New Features

2.12.1 Improvements for FPGA Allocation

The system-level compiler supports the following syntax for FPGA allocation:

```
use_fpga –zcore <my_zcore>
  [-fpga <unit>.<module>.<fpga> | -module <unit>.<module>]
  [-number <int-nb>] [-constraint <constraint-list>]
```

FPGA allocation can now be used with the `–constraint` option. This command is added in the **Core Definition File** in **`zCui`**.

- Directive mode: user chooses and specifies the FPGAs which will be allocated, using the `–fpga` option (to allocate a single FPGA):
  ```
  use_fpga –zcore <my_zcore> –fpga <module>.<unit>.<fpga>
  ```
  or the `–module` option (to allocate all the FPGAs of a given module):
  ```
  use_fpga -zcore <my_zcore> –module <unit>.<module>
  ```
- Constraint mode: user declares some constraints for the number of FPGAs (`-number` options) or for authorized/forbidden FPGAs (`-constraint` option):

  For example, the compiler can be configured to use 15 FPGAs
  ```
  use_fpga –zcore <core-name> -number 15
  ```
  It is also possible to configure the compiler to choose all the FPGAs on modules M0 and M1 in unit U0 and exclude FPGA16 from the selected FPGAs on U0.M0:
  ```
  use_fpga –zcore <core-name>
    -constraint { U0.M0 U0.M1 ~U0.M0.F16 }
  ```

2.12.2 Timing-driven Optimization

The support of timing-driven optimization is now available in **`zCui`** by selecting the **Use Timing-driven Mode** check-box in the **Back-End** → **Advanced** panel (**System Place & Route Parameters** frame).

When selected, the system place and route optimizes the most critical nets of the design for a higher runtime frequency.

This option should be selected for optimization purposes only, once the design is already up-and-running.

2.12.3 Automatic Clustering

This release introduces the following enhancements for Automatic Clustering:

- In order to avoid register over-mapping during FPGA compilation, the automatic clustering process can now consider the number of registers for XDR implementation.
  The following command should be added in an additional command file declared in **zCui** through the **Advanced** → **Build System Parameters** panel:
  ```
  cluster enable -include_xdr_cost
  ```
  When this command is active, the clustering report shows the estimated resources for XDR in the last column of the `Auto Clustering: Logic Allocation` table.

- In order to avoid unnecessary constraints for zCore-level clustering and for FPGA P&R, the filling rate constraints are now automatically adjusted by the zCore-level compiler. This is useful in particular when the filling rate constraints were set to higher values than the default to pass the system-level compilation.
  However, it is possible to disable this automatic adjustment by adding the following command in an additional command file declared in **zCui** through the **Advanced** → **Build System Parameters** panel:
  ```
  cluster disable -max_fill_constraint_adjustment_by_usage
  ```

- It is now possible to set the maximum filling rate for Flexible Probes in the FPGAs with the following command:
  ```
  cluster set -max_fill_fp_bit=<fp_rate>
  ```
  Where `<fp_rate>` is a percent value of the total number of bits available for flexible probes in one FPGA (30,720).
  This command is added in an additional command file declared in **zCui** through the **Advanced** → **Build System Parameters** panel. If the command is not present in the script, Automatic Clustering uses `<fp_rate>=100`.
  If the command is in the script, it is listed with its impact in the log of the zCore-level compilation.

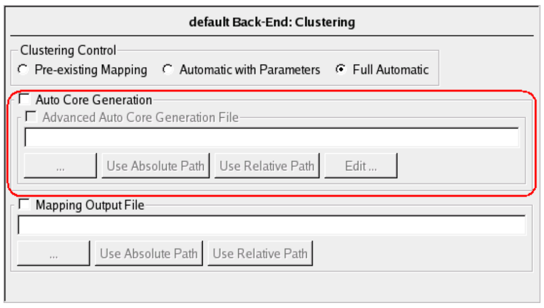
- The command to set the maximum filling rate for Flexible Probe Tracers has been modified to have a similar syntax to the above command:
  ```
  cluster set -max_fill_fp_tracer=<fp_tracer_rate>
  ```
  Where `<fp_tracer_rate>` is a percent value of the total number of available tracers for flexible probes in one FPGA (64).
  This command is added in an additional command file declared in **zCui** through the **Advanced** → **Build System Parameters** panel.
  If the command is not present in the script, Automatic Clustering uses `<fp_tracer_rate>=100`.
  The `cluster set -max_fill_flexible_probe_tracer` deprecated command is still supported for backward compatibility purposes.

- Automatic Clustering now estimates a better size for the control logic for zrm memories based on DDR memory resources.
- To prevent from splitting a module into several FPGAs:
  ```
  cluster model -atomic_module=<module_name>
  ```
  The `-ztopbuild` or `-zcorebuild` options can be added to force this command to be applicable only for system-level or zCore-level compilation.
- To use a new estimation algorithm in the zCore-level compiler when delay insertion is used for clock modeling:
  ```
  cluster enable -zdelay_dynamic
  ```
- To disable a new estimation algorithm in the system-level compiler when delay insertion is used for clock modeling:
  ```
  cluster disable -tb_zdelay_simple
  ```
- To declare filling rate constraints for LUTRAM because of the shortage of such resources in Virtex-5 FPGAs (only 25% of LUTs can be used as LUTRAMs):
  ```
  cluster set -max_fill_ramlut=<n>
  ```
- For user declaration of specific blocks to be removed from the Automatic Clustering process:
  ```
  cluster model -extract_path_list {<path or path_pattern>} [-all]
  ```
  Where `–all` option also extracts the primitives.
  The `-ztopbuild` or `-zcorebuild` options can be added to force this command to be applicable only for system-level or zCore-level compilation.
- Note that the commands should be added in an additional command file declared in **`zCui`** through the **Advanced** → **Build System Parameters** panel.

2.12.4 Automatic zCore Generation

For very large designs which are mapped on several modules, the ZeBu compiler supports a new feature which automatically partitions the design into several zCores which are each mapped on a module in the ZeBu-Server system. This automatic zCore generation is processed by the system-level compiler, and is supported for designs which do not use the Direct ICE interface.

In `zCui`, this feature can be activated from the **Clustering** panel in which a dedicated frame has been added:


default Back-End: Clustering

Clustering Control: ( ) Pre-existing Mapping ( ) Automatic with Parameters (●) Full Automatic

☐ Auto Core Generation
☐ Advanced Auto Core Generation File

☐ Mapping Output File

When the **Auto Core Generation** checkbox is selected, the compiler proceeds with the creation of the zCores instead of proceeding with the former automatic clustering. The settings of the **Clustering Control** frame are applicable for automatic zCore generation:

- When **Full Automatic** is selected, the zCores are created to match the size of modules; taking into account the default filling rate settings for FPGA resources (see Section 4.1.3 for updated values).
- When **Automatic with Parameters** is selected, the filling rate sliders in the **Algorithm** frame can be modified for the generation of the zCores.
- When **Pre-existing Mapping** is selected, the compiler uses a mapping file generated in a former compilation, as described below.

As for automatic clustering, it is recommended to declare a **Mapping Output File**. This file is intended for use in future compilations: the automatic core generation can be skipped:

- Clear the **Auto Core Generation** checkbox
- Select **Pre-existing Mapping** in the **Clustering Control**
- Declare the path to the mapping file in the **Mapping Input File**

An **Advanced Auto Core Generation File** can be declared to configure the generation process. You should contact EVE Support (support@eve-team.com) for further details.

2.12.5 Other Back-End Compilation New Features

This software release introduces the following compilation enhancements:

- In order to make the system routing and the FPGA Place & Route process easier, it is now possible to remove the zViews in the design regardless of whether it is connected or not, by adding the following command in the `Netlist Edition File`:
  ```
  blackbox remove zview
  ```
  Note that these instances are not removed when declaring blackboxes with pattern-matching commands, in particular `-fnmatch *`.

- When some zViews were isolated in a zCore (no reader(s) nor driver(s) in the zCore), they caused inter-FPGA paths. They are now automatically moved in a zCore where they have reader(s) or driver(s) to make FGPA Place & Route easier and thus improve the achievable performance of the design.

- A new section has been added in the system-level report (**Build System** step in **`zCui`** task tree) with the list of the zCores declared by the user. This section is part of Section 8 of the report (`Netlist split`):
  ```
  8.1.Initial core definition (defcore commands)
  ```

- When proceeding with static timing analysis, the inter-FPGA paths with asynchronous set/reset are now taken into account. An additional report is also generated for these paths (`ztime_asyncsr_out_paths.html`).

2.13 New Features for Combinational Signals Accessibility (CSA)

This patch provides the following enhancement for the CSA feature (Combinational Signals Accessibility):

- The achievable runtime frequency with CSA signals has been increased.

- The compilation tasks for CSA have been separated into different processes in **`zCui`** in order to minimize the risk of failure due to memory consumption. The following items now appear in the **`zCui`** Task Tree pane (previously the first item included both compilation tasks):
  - **Build Accessibility Graph**
  - **CSA Support Calculation**

- The memory consumption has been optimized when compiling for CSA signals and when processing the CSA database. The compilation for CSA signals has been improved in terms of duration.

- The selection of CSA signals for the first time during emulation runtime is faster than in previous software releases.

- Some types of CSA signals are now easier to access at runtime, in particular multi-dimensional arrays and some undriven/unloaded wires connected to input/output ports in the design.
- The processing of a selection of CSA signals has been optimized to avoid loading the entire content of the CSA database, namely when all selected probes are added in a waveform file.
- The dumping of CSA signals in a waveform file has been optimized.

2.14 New Features for Database Post-Processing Tools

This software release includes the following new commands for database post-processing tools (**`zDbPostProc`** and **`zVisiPostProc`**).

Command	Description
`print_selection_stat`	This command has been added in **`zVisiPostProc`** only in order to provide the number of signals selected and the actual number of bits for these signals.
`print_banks` `print_memory`	These commands have been added in **`zDbPostProc`** and **`zVisiPostProc`** in order to provide information about the memory banks and the user-defined memory instances.
`change_mem`	This command has been added in **`zDbPostProc`** and **`zVisiPostProc`** in order to modify the depth and width parameters of a memory instance, provided that the total size of the memory in bits (`width*depth`) is not changed.

2.15 Runtime New Features

2.15.1 C/C++ API New Features

This software release introduces the following runtime enhancements:

- When opening the connection to the ZeBu system (`Board::open`), some parallelization is now enabled by default. It is possible to disable this feature by setting the `ZEBU_CONNECTION_USE_THREAD` environment variable to `NO`.
- The log of the testbench now includes information about the memory usage and the execution time for each initialization step when opening the connection to the ZeBu system (`Board::open`).
- The ZeBu API has been improved with the support for loading a memory of the design from a text file with binary-formatted data (`readmemb`) and storing the content of a memory into a text file with binary-formatted data (`writememb`):
    - C++ API: `Memory::readmemb` and `Memory::writememb`
    - C API: `ZEBU_Memory_readmemb` and `ZEBU_Memory_writememb`

  Equivalent methods with explicit names are also available:
    - C++ API: `Memory::loadFromTxt` and `Memory::storeToTxt`
    - C API: `ZEBU_Memory_loadFromTxt` and `ZEBU_Memory_storeToTxt`

Note that the capability to load and dump memories in human-readable hexadecimal files is now possible with a similar prototype as for binary files (`readmemh` and `writememh`):

- C++ API: `Memory::readmemh and Memory::writememh`
- C API: `ZEBU_Memory_readmemh and ZEBU_Memory_writememh`

The old-type methods are still supported for compatibility purposes.

- `ZEBU_CCall.h` has been reviewed with respect to function prefixes and function mangling for C ANSI.
- The `ZEBU_Board_dumpflush` function and `ZEBU::Board::dumpflush` method have been introduced to synchronize to disk the waveform file opened with the `ZEBU_Board_dumpfile / ZEBU::Board::dumpfile` function/method so that the file being generated can be read before its closing (requested in RT#22210).

2.15.2 New Features for Threadsafe Environment Integration

This software release introduces the following enhancement when integrating a threadsafe environment:

- A user-defined threshold is now available to optimize the synchronization of the `ZEBU_Driver_run` and `ZEBU_Driver_wait` functions. The default threshold to swap between threads is a waiting delay of 80µs. If the runtime frequency is very slow (`driverClock` frequency <200 kHz), the threshold should be set to a higher value so that the threads are swapped only after longer waiting delays.
  For that purpose, the threshold to swap the thread can be declared by setting the `ZEBU_DRIVER_WAITING_THRESHOLD` environment variable (the delay is in micro-seconds, µs).

2.15.3 New Features for FSDB Waveform Dump

This software release introduces the following enhancements when dumping FSDB waveforms:

- The **zRun** `ZEBU_Dump_file` and `ZEBU_Flp_setOutputDir` Tcl commands have been updated. They now support additional parameters: `compressionLevel`, `dumpVirtualTime` and `timescale`:

```
ZEBU_Dump_file fileName clockName [signalListFilename=""
[compressionLevel=0 [dumpVirtualTime=0 [timeScale=-9 (for
1ns)]]]]
ZEBU_Flp_setOutputDir directory [compressionLevel=0
[dumpVirtualTime=0 [timeScale=-9 (for 1ns)]]]
```

  Where:
  - `compressionLevel` modifies the actual compression for VCD and binary waveform files. The value is an integer `[0..9]`, where `9` is the strongest compression. By default, `compressionLevel=0`.

- o `dumpVirtualTime` is intended to change the clock on which the sampling data are dated in the waveform file, using a virtual clock declared in the `designFeatures` file. It makes it possible to get waveform files with a timescale independent of the sampling method, thus allowing merge and comparison of waveforms created from different features (dynamic probes, flexible probes, HDL simulation, etc.). When set to `1`, virtual time dumping is active. The default value is `0`, so that the waveforms are dated with a number of cycles of the sampling clock.
  `timeScale` is a power of ten (for example, `2` is for a 100s-timescale, `0` is for a 1s-timescale. By default `timescale=-9` for 1 ns.

- When dumping into an FSDB waveform file, the file writing process can be accelerated by setting the `ZEBU_FSDB_DUMP_USE_THREAD` environment variable to `YES`.
  Note that this parallelization is concurrent to the parallelization of dynamic probe dumping.
  The parallelization when writing the FSDB file should be enabled only if there are many events per sampling cycle in the waveform file; else you should use the parallelization for dynamic probe dumping.

- Note that activating simultaneously parallelization for dumping and parallelization for file writing does not provide improvement of the performance but the memory and CPU footprints are higher.

2.16 New Features for Direct ICE

This software release introduces the following enhancements for the Direct ICE feature:

- It is now possible to declare the `delay_step` option for `zice default`, `zice_io` and `zice_clock` commands in the Direct ICE compilation script.
  The delay step that is used for runtime programming can now be declared as an option of the `zice default` and `zice_io` commands in the Direct ICE Compilation script. This option (`-delay_step`) is fully described in the *ZeBu-Server Direct ICE Manual* (Rev b).

- It is now possible to declare delay insertion for local output clocks in the Direct ICE compilation script: the `delay_fclkout` option has been added for the `zice default` command.

- The system-level compiler now checks that a signal connected as an input is not declared more than once in the Direct ICE compilation script.

- The error messages have been improved for an easier identification of the failing command in the Direct ICE compilation script.

- The appropriate files to compile for a 2-module Direct ICE configuration have been added in this software release. The *ZeBu-Server Direct ICE Manual* has not been updated for this. Contact EVE Support if you need further information (support@eve-team.com).
- `fclkout` clocks can now be synchronized (no skew). For example, to declare `fclkout` with a specific delay in the Direct ICE compilation script:
  ```
  zice_clock fclkout -pin {dut.cko} -con_loc J1100.31 -delay 50ns
  ```
  All `fclkout` clocks declared with the same delay value will be aligned in skew (they must be in the same Direct ICE FPGA to ensure a very small skew) and thus belong to the same group. Therefore, user can modify one delay value at runtime for an entire `fclkout` group.
- It is now possible to configure the trace feature with new options of the `zice` and `zice_io` commands in the Direct ICE Compilation script. These options are fully described in the *ZeBu-Server Direct ICE Manual* (Rev b).
- In **`zCui`**, the Direct ICE module is identified with its physical position. The compiler now automatically manages logical re-numbering in case the `use_module` command is used.

2.17 Smart Z-ICE HE10 Adapter

To make integration easier with a software debugger, an adapter is now available to connect to the Smart Z-ICE interface using standard HE10 connectors. One adapter can be plugged on each Smart Z-ICE port on the rear of the Zebu-Server unit.

There is no functional difference when connecting to the Smart Z-ICE with the HE10 adapter (all the signals of the interface are available and the same operating modes are supported).

When connecting to the Smart Z-ICE interface with the HE10 adapter, the DVE file includes the following specific declarations:

- Declaration of the type of connection:
  ```
  defparam <my_smart_zice_driver>.connector = HE10;
  ```
- In the instantiation of the `SMART_ZICE_ZSE` driver, the connections of signals are declared explicitly with the pin numbering of the HE10 connectors.
- The HE10 pin numbering is also used for the connection of clock signals.

Detailed information to integrate this HE10 adapter (physical description, pin-out of the connectors, instantiation example, etc) will be integrated in the *ZeBu-Server Smart Z-ICE Manual* in a near future (the HE10 adapter for ZeBu-Server is the same as for ZeBu-XXL). Contact EVE Support if you need further information (support@eve-team.com).

2.18 Improved Performance with ZEMI-3

The performance of a ZEMI-3 transactor in any environment (with **`zEmirun`** or any other environment) can be improved by modifying the inter-thread synchronization for export functions.

For that purpose, the `ZEMI3_EXPORTS_USE_WAITGROUP` environment variable should be set to any value before launching the test (`bash` shell example):

```
$ export ZEMI3_EXPORTS_USE_WAITGROUP=1
```

To optimize the synchronization, the timeout which controls the waiting period can be modified by setting the `ZEMI3_EXPORTS_WAITGROUP_TIMEOUT` environment variable to the appropriate value:

```
$ export ZEMI3_EXPORTS_WAITGROUP_TIMEOUT=<N>
```

Where `<N>` is the duration of the timeout in microseconds (default is 1000).
A very good knowledge of the environment is necessary to adjust this timeout value since it can drastically decrease the performance if it is modified with a wrong value.

3 Documentation Package

This chapter has been updated for V6_2_1 B_00 cumulative patch.

3.1 Master Document in the Documentation Package

The ZeBu documentation package includes a master document which is a PDF file with bookmarks for direct access to all documents of the package (in the bookmark panel of Acrobat).
Each document of the package includes a specific bookmark which returns to the master document (available at the top of the bookmark tab of Acrobat for this Release Note).

For ergonomics purposes, the master document includes a bookmark which opens the **Acrobat Full Search** window.

3.2 ZeBu-Server Documentation Package

The following table lists the complete documentation package, showing the manuals which are available for ZeBu-Server and mentioning which manuals can be used when ZeBu-Server dedicated manuals or up-to-date ZeBu common manuals are not available. Note that new or updated documents are in *italics*.

Ind.	Manual Name	Rev	Comments
01	**Installation Manual**	d	• Added missing 8C/ICE module in list of modules. • Updated design capacity values. • Added missing U_1-U_4 connection in multi-unit figures. • Added list of available diagnostics patches. • Added example of diagnostics patch with loopback modules. • Updated list of sample configuration files with loopback modules. • Added `-mu` option to **`zConfig`** to generate a configuration file for a single 5-slot unit in multi-unit configuration.
02	**Compilation Manual**	b	Major update for V6_2_0. See detailed History table in the manual.
03	**HDL Co-Simulation Manual**	b **ZeBu**	Last update for V 3_1_0.
04	**C++ Co-Simulation Manual**	b **ZeBu**	Last update for V4_3_0.
08	**zRun Manual**	c **ZeBu**	Last update for V4_3_3.
10	**Smart Z-ICE Manual**	a **ZSE**	First edition.
11	**Direct ICE Manual**	b	Major update for V6_2_0 B_00. See detailed History table in the manual.

Ind.	Manual Name	Rev	Comments
13	**C API Reference Manual** **C++ API Reference Manual**	V6_2_1	
14	**ZEMI-3 Manual**	b **ZeBu**	Last update for V4.3_3.
15	**zFAST Manual**	a	First edition.

3.3 Additional Documents

In order to keep track of recent new features not yet described in the ZeBu Manuals, Release Notes for previous releases are available in the documentation package.

The following Application Notes are included in the documentation package and can be used with the present software release. Note that new or updated documents are in *italics*.

Ind.	Application Note	Rev	Comments
AN017	**Timing Analysis with `zTime`**	**b**	Not up-to-date with the modified default setting for **Timing Analysis** in **`zCui`** (which changed in V6_2_1 from **Basic** to **Full**).
AN021	**Importing Memories for Easy Runtime Access**	**a**	
AN022	**ZW-FPGA Usage**	**c**	
AN025	**Integrating with ZeBu thread-safe library**	**b**	
AN027	**Accessing Combinational Signals with ZeBu**	**c**	
AN028	**SystemVerilog Assertion (SVA) for ZeBu**	**a**	
AN029	**zDPI Feature for ZeBu**	**a**	New edition.
AN032	***Migrating from ZeBu-XXL to ZeBu-Server***	***a***	*New edition.*

For your reference, the license agreement for the ZeBu software and for usage of the third-party software packages delivered with the ZeBu software is available as a PDF file in the documentation package.

- The Xilinx license agreement is included.
- The Concept Engineering license agreement is also available for information about RTLvision PRO and GateVision PRO usage.

4 Known Limitations for V6_2_1

This chapter lists the known limitations in V6_2_1 software release.

Note that limitations which are no longer applicable for V6_2_1 B_00 cumulative patch have been removed.

4.1 Modified Default Settings

4.1.1 *Full* is the New Default Type for Static Timing Analysis in `zCui`

In the **Back-End Properties** panel, the default setting for **Type of Analysis** has changed from **Basic** to **Full** in the **Timing Analysis Options** frame.

Full takes into account inter-FPGA combinational paths which, in case of false paths, can lead to more pessimistic frequency estimations than what was obtained with the former default setting, which you can set back by selecting the **Basic** mode.

4.1.2 Mandatory Update of PCIe Interface Board

The content of this section is outdated. Section 1.2 describes the installation procedure for V6_2_1 with cumulative patch B_00.

4.1.3 Modified Settings for Automatic Clustering in `zCui`

When **Automatic with Parameters** mode is selected, the default filling rates have been modified (compared to V6_2_0B default settings):

- Registers: 55% (5% overflow activated by default)
- LUTs: 70% (10% overflow activated by default)
- DSP: 80%
- BRAM: 80%

4.1.4 Tcl Commands Executed before the first `ZEBU_xxx` Call

When **`zRun`** was used with the `-do` option and a Tcl script, a `ZEBU_open` occurred systematically before the Tcl script was interpreted. If user wanted to issue a `ZEBU_restore` in the script, he ended up with a double connection: `ZEBU_open` (systematically), then `ZEBU_close`, then `ZEBU_restore`.

With this release, if the `ZEBU_open` call is not executed from the Tcl script it will be executed during the first call to any `ZEBU_xxx` function. This means that Tcl commands executed before the first call to a `ZEBU_xxx` function are now executed before a connection is made to the machine.

4.1.5 Multi-process Feature

The multi-process feature is different than what it was in releases prior to V6_2_0. The main difference is that the number of processes and the process name must be declared in the `designFeatures` file instead of the DVE file. For example:

```
$nbProcess = 2;
$process_0 = "default_process";
$process_1 = "jtag_process";
```

4.2 Limitations for Operating System

4.2.1 Specific SUSE Linux Enterprise Server 10 Requirement

The present version of the ZeBu software requires the availability of the `compat-readline-4.3` package which is not installed by default on SUSE Linux Enterprise Server 10 operating system.

4.2.2 Red Hat Enterprise Linux 5

The present version of the ZeBu software can be used with Red Hat Enterprise Linux 5 with the following limitations:

- Compiling with **`zCui`** graphical interface is not possible because of a compatibility issue for the graphical libraries. However a design compiled with Red Hat Enterprise Linux 4 runs correctly on a ZeBu-Server system connected to a PC with Red Hat Enterprise Linux 5 operating system.
  It is possible to compile with **`zCui`** in batch mode (with `–nogui` option).
- The `autofs` Linux functionality, which allows automatic mounting of a remote disk volume, does not work correctly (note that this Red Hat Enterprise Linux 5 limitation is unrelated to ZeBu software).
  It is necessary to install the following Red Hat Enterprise Linux 5 RPMs:
  - `kernel-2.6.18-53.1.6.el5.x86_64.rpm`
  - `kernel-devel-2.6.18-53.1.6.el5.x86_64.rpm`
  - `kernel-headers-2.6.18-53.1.6.el5.x86_64.rpm`

Some additional recommendations must be taken into account:

- You have to compile any software which is linked to a ZeBu API (binary library) such as a C/C++ testbench or the software part of a transactor using `gcc/g++` version 4.1. This is the default version for this operating system.
- The C++ standard library version 6 (`libstdc++.so.6`) is the default version for this operating system.
- The Linux environment variable `LD_ASSUME_KERNEL` must be unset.

4.2.3 GTKWave Limitation

The package for GTKWave waveform viewer can be installed only with Red Hat Enterprise Linux 4 operating systems, but not with Red Hat Enterprise Linux 5 or SUSE Linux Enterprise Server 10.

Once installed from a RHEL4 workstation, the program runs correctly on all supported platforms.

4.3 `zCui` Limitations

4.3.1 Project File Compatibility

Once you have saved an existing **`zCui`** project file (`.zpf`) with Version 6_2_1, you can no longer use this file with previous releases. If you intend to use your previous version project file for non-regression testing, you should save it with a different name before migrating to Version 6_2_1.

When opening an existing **`zCui`** project file, the **Report** window comes to front and shows the problems with the project file, in particular the **`zCui`** items that have been modified and that will not be kept in the V6_2_1 project file.

4.3.2 `zCui` uses `/bin/sh` and ignores `.cshrc` aliases/functions

`zCui` uses `/bin/sh` and ignores aliases and functions defined elsewhere (for example in `.cshrc`). Make sure you use scripts in your PATH instead of aliases.

4.3.3 Limitations with Multiple RTL Groups

In **`zCui`**, the RTL source files are gathered in one or several RTL Groups. By default, **`zCui`** creates one RTL group (**Default_RTL_Group**).

It is recommended to have only one RTL group for the project, since most of the ZeBu debugging features are supported for one group only, in particular SystemVerilog Assertions, zDPI feature, RTL-based compilation scripts and CSA signals. Note that hierarchical references in the RTL sources are supported only inside an RTL group (no hierarchical references between different RTL groups can be synthesized).

Additional RTL groups can be of interest when integrating RTL sub-projects or IPs, when investigating synthesis issues or when a sub-module needs some specific synthesis features for performance purposes in particular.

Each RTL group has its own synthesis options and synthesis output files are automatically merged by the ZeBu compiler.

The declaration of RTL source files is described with details in Section 3.2.3 of the *Zebu-Server Compilation Manual* (Rev b).

4.3.4 Other Limitations

- In a panel displaying a log or a source file, user can only select entire lines.
- User must have Write access rights to evaluate a project.
- Debug options are not gathered in a single panel.
- When the **`zCui`** Archiver feature is used in graphical mode, only one back-end can be selected at a time.

4.4 zFAST Limitations

4.4.1 Functional Limitations

- Top-Down Synthesis:
  - The generated netlist is flattened or partially flattened. That can introduce clustering issues since clustering is based on hierarchy.
  - The runtime accessibility is reduced: only registers can be accessed in the runtime database.
  - Hierarchical references are not supported.
- When there are several RTL groups in the design, the declaration of a top name for each group is mandatory. If there is only one RTL group, the name of the top of the design will be used for the group.
  For other limitations with multiple RTL groups, see Section 4.3.3.
- The top of the design cannot have a Verilog escaped identifier.

4.4.2 Interface Limitations with `zCui`

When selecting **Verilog95** in the **Main** → **Verilog Language** list, **`zFAST`** actually synthesizes the design with Verilog2001 constraints, in particular the Verilog2001 keyword usage in Verilog95 code will cause synthesis errors.

4.4.3 Memory Synthesis Limitations with `zFAST`

Memories inferred as **`zMem`** memory models by **`zFAST`** are sometimes not optimized, creating backend compilation issues or decreasing the runtime performance. The following cases have been identified.

4.4.3.1 Bit- and Byte-enabled Memories

In this case **`zFAST`** may create a **`zMem`** memory model with a large number of ports that cannot be compiled in the back-end or decrease the runtime performance.

Following is a list of possible workarounds:

- Write manually the **`zMem`** Tcl script of the memory
- Modify the memory inference threshold in **`zCui`** (but it impacts all the memories in the design).
- Force this specific memory as flops in the **Additional zFAST Attribute File** in **`zCui`**:
  ```
  Compile:ImplementAsFlops=(module1.mem1 module2.mem2 ...)
  ```
  Note that the command which forces usage of a **`zMem`** memory model is:
  ```
  Compile:ImplementAsMemory=(module1.mem1 module2.mem2 ...)
  ```

4.5 Limitations for RTL-based Compilation Scripts

4.5.1 Unsupported characters in the RTL paths

The following characters are not supported in the RTL paths of the RTL-based compilation scripts:

- Curly brackets (`{` and `}`) are usually not supported by synthesizers. If some are present in the RTL paths of the design, the possible work-around is to use the EDIF path in which support for special characters is better.
- When RTL-based commands are used with pattern matching (`-fnmatch` option), the pattern matching characters (`*`, `[` and `]`) cannot be present as in an RTL path.

4.5.2 Simultaneous use of SVAs and Flexible Probes

Flexible probes and SystemVerilog Assertions can be used in the same design but the sampling clock for Flexible Probes is the SVA clock. Therefore no specific sampling clock can be declared when declaring the flexible probes in the RTL-based compilation script: the `-clock_name` option must not be present (the ZeBu compiler fails if the option is present and SVAs were synthesized).

4.6 Back-end Compilation Limitations

4.6.1 Limitation for Tristate Signals

Top-level tristate ports cannot be defined with `keeper` resolution (other resolutions are correctly supported). Recommendations for tristate signal handling are described with details in Section 3.7.3 of the ***Zebu-Server Compilation Manual*** (Rev b).

4.6.2 Clock Connection to the `SRAM_TRACE` driver

Connecting a primary clock (`zceiClockPort` output) as a traced signal in the instantiation of the `SRAM_TRACE` driver may cause a routing error during FPGA Place & Route.

4.6.3 Limitations for Memory Modeling

4.6.3.1 Limitation on Clocks for DDR2 zrm Memories

Because of an inappropriate frequency range, it is not possible to drive a DDR2 zrm memory instance with a synthesized uncontrolled primary clock generated by the frequency synthesizers (declared by instantiating a `zClockPort` in the DVE file). The minimum frequency of the synthesizers is much too high to connect such a clock to DDR2 memories.

4.6.3.2 Limitation on Read/Write Priority for Multi-port zrm Memories

The read-before-write priority is set individually on one port. The read/write priority between two ports of the same memory instance is not deterministic for zrm memories, even if both ports use the same clock signal.

4.7 Runtime Limitations

4.7.1 Trace Feature

The content of this section is outdated for V6_2_1 with cumulative patch B_00. It is now possible to activate the trace feature (SRAM trace) for any `xclk` frequency.

4.7.2 Clock Declaration for HDL Co-simulation

For HDL co-simulation, the clocks controlled by the HDL testbench must be declared without being attached to a group in the `designFeatures` file.

Additional design clocks which are not controlled from the HDL testbench can be grouped.

4.7.3 Limitation for Threadsafe Environment

The `Z_REPEAT_CALLBACK` is not available when emulating in a threadsafe environment (when the testbench is linked with `libZebuThreadsafe.so` library).

4.7.4 Limitation on Fast Hardware State

A fast hardware state object must be initialized after board initialization.

4.7.5 Limitation for Selection/Deselection of Dynamic Probes

Selection and deselection of a dynamic probe are not supported when a `.bin` waveform file is opened. Any change in the list of selected dynamic probes causes the following fatal error:

```
"ZWAVE0050E: "Cannot continue to dump bin file while recomputing dynamic probes".
```

You can avoid these issues by closing the `.bin` waveform file before selecting or deselecting any new signal.

Note the following points:

- A selection is implicit if you read the value of a signal for the first time.
- A deselection is implicit if you destroy a signal handler.
- RACC checking or randomization features can also change implicitly the selected dynamic probes.

4.7.6 Limitation on Clock Specification for Restore

An error message is correctly issued when the clock parameters are declared in the C/C++ testbench **after initialization**.
However the same error message is erroneously issued when the clock parameters

are declared in the C/C++ testbench **before initialization**, which makes it impossible to declare the parameters when restoring a saved state.
As a workaround for restore, EVE recommends that the user declares clock parameters from the `designFeatures` file only (not from the C/C++ testbench).

4.7.7 `zInstall` Multi-release Limitations

The **`zInstall`** multi-release feature will not work if a job is running on the ZeBu hardware. To run the multi-release feature of **`zInstall`**, make sure that no job is running on the ZeBu hardware when you run **`zInstall`**.

4.8 Restrictions to Write Operations, in particular with RLDRAM Memory Models

4.8.1 Restrictions on Writing to Registers

For most FPGAs in the system, writing to registers at runtime is supported when the **Enable BRAM Read&Write / Write Register / Save&Restore** item in the **Debugging** tab is enabled in **`zCui`** (it is disabled by default).

However, it is never possible to write to registers in FPGAs connected directly on RLDRAM in 4C and 8C/ICE modules, when RLDRAM is used by the design.

This restriction impacts the following commands of the RTL-based Compilation Scripts when they apply to signals/instances mapped into an FPGA connected to RLDRAM (no message is displayed during compilation; the write access limitation only appears during emulation runtime):

- `force_dyn`
- `force –value REG`
- `blackbox –value REG`

This restriction also impacts the SVA feature, as described in Section 4.9.

4.8.2 Restrictions on Writing to BRAM Memories

Writing to BRAM memories at runtime is supported with the following restrictions.

If neither the **Enable BRAM Read&Write / Write Register / Save&Restore** nor the **BRAM Instrumentation for Read&Write** items in the **Debugging** tab have been set, then at runtime, all output registers of all BRAMs in a FPGA will be reset each time one BRAM is read or written on that FPGA.

Setting the **Enable BRAM Read&Write / Write Register / Save&Restore** item can avoid this behavior, except for FPGAs connected directly on RLDRAM in 4C or 8C/ICE modules, when RLDRAM is used by the design. To avoid the behavior described above for these FPGAs, set the **BRAM Instrumentation for Read&Write** item.

4.9 Limitation for SystemVerilog Assertions (SVAs)

It is not possible to proceed with emulation runtime if SVAs were synthesized and the design instantiates RLDRAM memory models.

The following message is displayed at runtime because the ZeBu software enables SVAs by writing to registers, which is not supported when RLDRAM memory models are instantiated in the corresponding FPGA:

```
ERROR : ZHW0193E : Signal ZSVA.<name>.sva_enable is not writable.
```

4.10 Advanced Triggers

Advanced triggers are not supported in the current version.

4.11 ZEMI-3 Limitation for Synthesis

The XST synthesizer is not supported for ZEMI-3 transactors.

5 Fixed Issues in V6_2_1

5.1 Installation

This software release fixes the following installation issues:

- When `zSetupSystem` or `zUtils -DT` failed before the download of the test configuration in the ZeBu-Server system (in particular when failing because of an inappropriate `ZEBU_SYSTEM_DIR` declaration), it took some time to reload the standard configuration even if it was not necessary.
- When `zSetupSystem` detected some limited frequency pairs at a frequency higher than 300 MHz, these pairs were reported as failing at 300 MHz if `zSetupSystem` was launched a second time with the same `ZEBU_SYSTEM_DIR`.
- The Smart Z-ICE P4 connector on 5-slot ZeBu-Server units can now be used to connect data to the Smart Z-ICE interface. The following message was displayed prior to this software release:

```
Unexpected parameter : if_con 0 connector 4 (max = 3-4)
```

- In a multi-unit system, `GCLK_IN` did not work on a Direct ICE module located in a unit other than U0.

5.2 zFAST Synthesis

This software release fixes the following **zFAST** issues:

- When an import `<package>` was called before the package was defined, **zFe** threw a syntax error on the import call (reported in RT#21631). The error message was not clear enough and has been changed to:

```
ERROR [97.1566]: <top.sv:2>: The package identifier ( my_pkg ) not found.
      [...]
ABORTX [114.18]: Analysis of Verilog design files failed
```

- Elaboration of a SystemVerilog design was failing with the following error message (reported in RT#22228):

```
# step Elab : sha1sum is bd9eb0088ad670ffd97629644dbaa0162570e2f1
# step Elab : execing failed: command did not correctly finish:
              child killed: SIGABRT
### fatal error in Elab [ZFE0099F] : Elaboration (hcs_mixed) of 'gsrpkg_te' failed
```

- The front-end post-process step was failing with the following error message (reported in RT#22240 and RT#22280):

```
### syntax error in fuse_cib_rsm_4_2_d3_2.edf:1316 [PLU0215S] :
    Module 'Xnor' not declared in library 'xcve'
### tcl error in zNetgen [CCT0104F] :
```

- When a SystemVerilog hierarchical reference contained a variable index, synthesis was failing with the following fatal error message (reported in RT#22271):

```
### fatal error in ZFAST [110.12904] :
    Forward mapping for non-static bit/part select is not supported...
### fatal error in ZFAST : <FILE:LINE>:
                            Href top.ins_array[index].tgt:
                            'ins_array' could not be mapped to its
                            post-synthesis name
```

  The hierarchical reference connection is now done properly.

- When signed additions were used in V2K synthesis, they were 0-extended (reported in RT#22154). They are now sign-extended, as defined by V2K standard.

- The following code generated an internal error during synthesis (reported in RT#22156):

```
genvar l;
generate
for (l = 0; l < 12; l = l + 1) begin : dumb
modA i1 (~i_A[8*l+:8], o_A[4*l+:4], o_B[8*l+:8]);
end
endgenerate
```

- A problem was causing the synthesis to fail with the following fatal error (reported in RT#21994):

```
### fatal error in ZFAST [110.9622] :
    <path to file/filename.v:1> Cyclic dependancy found in Design Hierarchy
```

- In some cases, global SystemVerilog bind was generating an internal error message at elaboration (reported in RT#21953):

```
# step Elab : Will print sha1sum for the previous command, which had a bad error.
# step Elab : sha1sum is d7929a88f8d4473eadfaaa07013208e9d5dd95ae
              /usr/import/zebu/V6_2_0A_patched/bin/hcs_mixed
### fatal error in Elab [ZFE0099F] : Elaboration (hcs_mixed) of 'top' failed
### fatal error in Elab [ZFE0048F] : Problem in elaboration. Cannot
    open simh.simdir/elab.result: couldn't open
    simh.simdir/elab.result": no such file or directory
### fatal error in Elab [ZFE0100F] : Number of uspecs is 1.
    Number of SimTops is 0. They should be equal.
```

- Static operations with the index of a `for` loop were generating the following error in V2K synthesis (reported in RT#22029):

```
### fatal error in ZFAST [110.5255] :
    <FILE:LINE>: Illegal bit select. Index -1 out of range [15:0]. Exiting ..
```

- The following SystemVerilog code:

```
localparam type \t_.toto = type( my_struct.my_bit) ;
```

  generated the following error (reported in RT#21911):

```
### fatal error in ZFAST [39.670] : (INTERNAL) <../../../top.v:16>:
    Can't localize local href in multiple-instantiated module
    because of the usage of type in the localparam construct.
```

- When a design contained Verilog arrays used as fully combinational logic and the `Compile:MakeSyncWrites` attribute was used, the following error was generated (reported in RT#21612):

```
### fatal error in ZFAST : Compile:MakeSyncWrites does not make
    sense for continuously written memories
```

- The following SystemVerilog line:

```
localparam My_x = {3{1'bx}};
```

generated the following error (reported in RT#21975):

```
fatal error in ZFAST [0.0] : <nomTharas_bit.cc:2371>:
Assertion '0 && "trying to find res type " "of illegal net type"' failed.
```

- The `finalOpt` attribute which was set by default caused some erroneous logic for tristate busses, as `vcc` was connected to the tristate bus instead of a pullup (reported in RT#22727). The corresponding logic is now correctly generated with the `finalOpt` attribute.

- When synthesizing with **Accessibility Level** set to **Keep all registers and Simulate Combinational Signals**, the following code caused a gnd/vcc short circuit on `m` signal (reported in RT#22619):

```
module top ( c, a, o );
 input c;
 input [9:0] a;
 output [35:0] o;
 reg [2:0] r2;
 integer m;
 reg [4:0] r3;
 reg [9:0] r1;
 always @( a) begin
  for ( m = 0; m < 10; m = m + 1 ) begin
   r1[m] = a[m];
  end
 end
 always @( r2) begin
  for ( m = 0; m < 5; m = m + 1 ) begin
   r3[m] = ( r2 == m );
  end
 end
endmodule
```

- Combinational SystemVerilog arrays with a number of bits larger than the **`zMem`** inference threshold (**Resource Inference Threshold → zMem memory** in **`zCui`**) and used as multiplexers were not synthesized correctly and caused erroneous runtime results (reported in RT#22360 and RT#22277).

- SystemVerilog memories of N bits width with bit-enable were sometimes synthesized as **`zMem`** memories with N ports of 1 bit with bit-enable, which caused sub-optimal logic in the netlist (reported in RT#22403).
It is now possible to optimize the synthesis of such memories by adding the following line in the **zFAST Additional Attribute File** declared in **`zCui`**:

```
Compile:CombineBitWrites=true
```

- When a SystemVerilog array was used as an `always_comb` and as an output port, its outputs were partially disconnected (reported in RT#22276).
- Synthesis of a single module with a very large gate count failed after a long runtime elapsed and a lot of memory (>40GB) was used (reported in RT#22327).
- Using SystemVerilog hierarchical reference with a local index in a loop as in the following example:

```
for(int entry = 0;(entry < P6_RS_DEPTH); entry += 1)
begin
 ins[entry] = top.ins1.ins2[array1[entry]];
end
```

  generated the following fatal error (reported in RT#22322):

```
### fatal error in ZFAST [97.349] : <FILE:LINE>: Task or function
    name ( entry ) not defined.
```

- An empty module was created when the user module name was the same as a built-in primitive name (`mem_val_bus`) (reported in RT#22001).
- Synthesis of an SVA failed with the following error message even though the **Ignore All** option was selected in **`zCui`** (reported in RT#22325).

```
### fatal error in ZFAST [97.349] : <top.sv:15>: Task or function
    name ( verify ) not defined.
```

- The front-end step of **`zFAST`** synthesis failed with the following message when synthesis was re-launched in **`zCui`** after a first synthesis (reported in RT#22336):

```
do not know how to make <include file path>
```

5.3 Third-party Synthesis

This software release fixes the following synthesis issues for third-party synthesizers (using **`zRtlFrontEnd`**):

- VHDL synthesis failed when a memory contained more than one write port in non-optimized synthesis mode (reported in RT#18017). VHDL memories are now detected and multi-ports are checked on VHDL memories as well as Verilog memories.
- Synthesis failed when a VHDL function included in a package was not found because the package was renamed but not the call to the VHDL function (reported in RT#22251).
- Synthesis failed because the vector direction was not the same in the entity and in the component declaration (reported in RT#22312). The mismatch occurred when `depth - 1 downto 0` was replaced by `0 to 0` at range resolution when depth was equal to `1`.
  This software release renames `0 to 0` to `0 downto 0`.

5.4 Compilation

This software release fixes the following compilation issues:

- The compilation flow sometimes created several dynamic probes for a single RTL signal, which caused conflicts when accessing the signal at runtime (reported in RT#22677). For example `mysig` RTL signal may be available in the runtime database as `mysig_1`, `mysig_2` and `mysig` dynamic probes.
  There were sometimes up to 5 or more aliases for a single signal. With this software release, most RTL signals will have only 1 dynamic probe in the runtime database. However, it may happen that 1 or 2 aliases are still displayed and the conflict issue remains in such cases.

- When RTL wires were undriven and unloaded in the RTL source files, they were removed from the edif netlist synthesized by **`zFAST`** even if they were declared in a `probe`, `force assign` or `force_dyn` command (reported in RT#22621 and RT#22485).
  In addition, the zCore-level compiler did not process correctly such nets when the `force assign` and `force_dyn` commands were declared with runtime programmable values and the corresponding logic resources were removed from the netlists.
  All these signals are now available in the synthesized EDIF netlist and they are processed correctly by the backend compiler.

- The compilation sometimes failed in the `FPGA Compilation` step because the `design.xdl` file could not be read when creating an intermediate database (after FPGA Place & Route); the following message was displayed and the compiler stopped:

```
basic_string::substr
```

- The processing of tristate signals in the `ZNETGEN STEP` of FPGA Place & Route task was not correct and the accessibility was lost for some feedthrough tristate signals (reported in RT#22580).

- FPGA Place & Route sometimes failed because some `ZeBu`-prefixed names were not renamed correctly during the previous steps of the FPGA compilation (reported in RT#22537).

- FPGA compilation kept running infinitely in spite of the correct generation of the FPGA `design.bit` file. The following warning was displayed (reported in RT#22249):

```
### warning in DATABASE GENERATION [KRB0033W] : Inconsistency between
xilinx cross references and edif
 (wire 'U1_M0_F1/U1_M0_F1_core/A/B/C[3]/FDCP' not found in netlist)
```

- Some unnecessary timing constraints have been removed because they were causing trouble for FPGA Place & Route process, in particular in terms of duration (reported in RT#22509). In some cases, the FPGA Place & Route failed because these unnecessary timing constraints could not be met.

- The zCore-level compiler sometimes caused a fatal error in the constant propagation process:

```
### internal error [LST0070E] : Instance zebu_propag_const_gnd
already exist in module 'dut'.
```

- When an unsupported BRAM memory model was instantiated in the design, an internal error occurred when generating the runtime database (reported in RT#22604 and RT#20804):

```
### internal error [ZDB0049E] : Unknown bram model: 'ramb16'
### internal error [ZDB0049E] : zdb_Memory.cc, line 346 : bramDepthV4
```

  In spite of this message, the compilation flow ended correctly and the resulting runtime database was correct. This message has been changed into a warning message:

```
### warning [KRB0159W] : cannot deal with bram model ramb16
```

- An internal error sometimes occurred in the zCore-level compiler during clock processing (reported in RT#22575):

```
### internal error[clk0153E]: cannot find wire 'TOP.A.B.C' in cone.
```

- System Place and Route sometimes failed to route a system net dedicated to flexible probes and the following error message was displayed (reported in RT#22474 and RT#22699):

```
### internal error [KPR0552E] : Flexible probes :
    cannot reserve ressources for module U1_M0
### internal error [KPR0552E] : system_ltrace.cc, line 292 :
    reserveResourceForStopClocksWires
```

- In some cases, dynamic probes generated inappropriate timing constraints during FPGA compilation which led to compilation failure on some FPGAs (reported in RT#22329). These timing constraints have been removed.

- In some cases, when flexible probes were defined on hierarchical instance ports, the compiler issued the following warning (reported in RT#22315):

```
### warning in CMD-TRANSLATOR [ZTB0118W] : Could not find port
    'top.ins0.portname[7]' from command 'probe_signals
    -instance top.ins0 -port {*} -hier_sep . -fnmatch
    -type flexible -fifo low -group pins'
```

  This warning indicates that one flexible probe command was not processed correctly during system-level compilation.

- In some cases, the filling rate constraints adjusted by the system-level compiler to produce a mapping were too high and caused FPGA overmapping.
  As of this software release, the zCore-level compiler adjusts the filling rate instead of the system-level compiler. The original behavior (prior to this software release) can be reproduced with the following command:

```
cluster disable -keep_original_filling_rate
```

  which must be included in an additional command file declared in **zCui** through the **Advanced → Build System Parameters** panel.

- When the `ZEBU_FIX_FLEXPROBE_TRACER` environment variable was set as a workaround for a faster FPGA compilation process with flex probes, the compiler did not work properly (reported in RT#22398).

The compiler has been optimized for the detection of flexible probe modules.

- The zCore-level compiler generated log files with extremely large sizes. Data dump has been suppressed from `zCoreBuild_AC_report` to reduce the size.
- A segfault sometimes occurred during constant propagation in the system-level and zCore-level compilers.
- **`zMem`** generated X's on Read ports (in Verilog simulation models) when multi-port memories (>2) with an Asynchronous Read port were in `READBEFOREWRITE` mode.
- **`zDbPostProc`** sometimes created badly formed signal names (due to pointer corruption).
- The `zice_io inout` command can be applied on top inout ports of the DUT (reported in RT#22298 and RT#22302). The top ports of the DUT are changed in output ports which can only be read in the RTB (no support for tri-state drivers in RTB).
- When compiling for the CSA feature, bad processing of combinational loops caused a segmentation fault.
- The system-level compiler introduced some undriven registers in the netlist when executing the `force assign ... -value REG` command, which led to FPGA unroutability (error in ISE/`ngdbuild`).
- The DUT mapped on ZeBu had a wrong behavior when the user wrote to internal registers (reported in RT#22237). A similar issue may be encountered when the Save&Restore feature is used. The problem is due to glitches introduced on inter-FPGA signals of the DUT.
- When a bidirectional port at the top of the design was not connected, the `Place and Route System` step failed with the following error message (reported in RT#22899):

  ```
  ### fatal error in TRISTATE [KPR0264F] : Interface port 'P_ZVDDPCIE' isunconnected
  ```

  This message has been changed into a warning so that the design may be compiled in such a case:

  ```
  ### warning in CHECK [KPR0265F] : Interface port 'my_port' is unconnected
  ```

- If some **`zMem`**-based memory models were instantiated in the design and were simplified by ISE (e.g. if the clock was stuck to 0 for testing purposes), FPGA Place & Route failed because some user constraints were declared in the makefile for a group which was empty after ISE simplifications. The content of such a group is now modified by the ZeBu compiler so that it is no longer simplified and can be processed correctly by ISE.

- The clock processing commands and options were sometimes not correctly passed by the system-level compiler: some zCores were compiled with incorrect settings for clock modeling.
- The glitch filtering algorithm has been improved in order to reduce the size of the resulting netlists which were sometimes too big for a correct Automatic Clustering process.
- When the design included Verilog WAND/WOR directives, the `Place and Route System` step failed because of unconnected ports. The following error message was displayed:

```
### fatal error in INIT [KPR0303F] :
    1 error found during initialization of the database
```

- The generation of the runtime database sometimes hanged for FPGAs with a very high number of clock signals (>thousands of clocks), reported in RT#23204.
- In the zCore-level compilation, the processing of tristate signals was too long (reported in RT#22386). It has been improved in order to reduce its duration.
- The timing analysis did not report correct information about the clock paths because the zCore-level compiler did not provide the expected information for analysis (only the primary clocks were taken into account).

5.4.1 Automatic Clustering

This patch fixes the following issues for Automatic Clustering:

- The compiler failed with the following error message while computing the constraints for Automatic Clustering (reported in RT#22990 and RT#22946):

```
### warning in AC [ACC0049W] : The filling constraints will be adjusted
    to the closest values that match the target.
### internal error [ACC0113E] : Cannot resize contraints parameters
### internal error [ACC0113E] : clustering_mode.cc, line 625 :
    generate_constraints_from_mode1
```

- The compiler failed with the following error message while analyzing the design for Automatic Clustering (reported in RT#22850 and RT#22545):

```
### internal error [NPT0045E] : Error counting leafs:
    sblk_pa.pa.ucgtt_local_pa dom_leaf(0) hier_dom_leaf(392) all_leaf(391)
### internal error [NPT0045E] : npt_cell_util.h, line 250 : visit_up
```

- The zCore-level compiler failed with the following error message when the `cluster core` command was present in the script (reported in RT#23113):

```
### internal error [AOC0022E] : Cannot find territory resource for _ZUNCORE_
### internal error [AOC0022E] : territory.cc, line 135 :
    homogeneous_arch_territory2fpga_logic_resource
```

5.4.2 Static Timing Analysis

This patch fixes the following issue for static timing analysis:

- Static Timing Analysis generated extra backslashes which caused a `fastnet` command line generated by Static Timing Analysis to be rejected by System Place and Route (reported in RT#22233).

5.5 Runtime

This software release fixes the following runtime issues:

- The memory was not correctly released by the ZeBu library and a segmentation fault occurred when exiting the testbench, after the emulation actually stopped (reported in RT#22618).
- When a combinational dynamic probe was declared in a `PartSignal` command, the emulation runtime failed with an internal error (reported in RT#22300):

```
-- ZeBu : tb : ERROR : LUI1121E : 11 (SEGV : "Segmentation fault")
```

- It was not possible to proceed with emulation runtime (starting the clocks in free-running mode from **`zRun`**) when the design was compiled for a heterogeneous ZeBu-Server system which had at least one 8C/ICE module installed, for example 4C+8C (configuration of the reported issue) or 8C+16C+16C+16C (reported in RT#22939).
- The ZeBu-Server system failed when the testbench attempted to access to memory resources which were not initialized (reported in RT#22890). In such a case, it was necessary to launch `zUtils -initSystem` to be able to use the system. If the memory resources are not initialized, the testbench now fails with the following message when the `Memory::set()`, `Memory::clear()`, and `Memory::erase()` methods are used:

```
ERROR : LAU0298E : You're trying to use uninitialized memory : Unit 0
                   Module 0 (zse_8c_ice_lx330_v1) F08 DDR2.
```

- When waveforms were dumped from the **`zRun`** GUI, **`zRun`** sometimes froze because the testbench attempted to write to a register or to a BRAM or to proceed with `Save` (reported in RT#22958). This was due to conflicts when **`zRun`** accessed the clock counters that were displayed in the GUI.
  To avoid such conflicts, the **`zRun`** GUI now works correctly in such cases.
- The `designFeatures.help` did not include the `$nbProcess` and `$process_x` parameters which are necessary in a multi-process environment or in particular when integrating a CoWare environment (reported in RT#22311, RT#22496 and RT#22986).
- When no zrm memories were instantiated in the design and when no `designFeatures` file was used, the `designFeatures.help` file was created with commands to set the frequency of memories to `0` (reported in RT#22741). Such values were not actually supported and caused runtime errors when using the file as a template to create a proprietary

`designFeatures` file for later use. The `designFeatures.help` file now includes some commented lines with `#value` for better legibility:

```
# $ddr2.frequency = #value;
# $ssram.frequency = #value;
# $rldram.frequency = #value;
# $ddr2Trace.frequency = #value;
```

When a testbench did not terminate correctly, some lock files were sometimes not removed from the `/zebu` filetree and caused the following error message when launching a later emulation (reported in RT#22752):

```
-- ZeBu : testbench : ERROR : LDEF0171E : Cannot open lock file
   "/zebu/.zebu_2010_04_01.log.lock" : File exists.
```

Such lock files are now checked for consistency:

- If the corresponding process id is still present, the new emulation fails with the following error message:

  ```
  -- ZeBu : zUtils : ERROR : LDEF0171E : Cannot open the lock file
     "/zebu/.zebu_2010_05_03.log.lock", it's already locked by the pid 11358.
  ```

- If the corresponding process id no longer exists, a warning message is displayed:

  ```
  -- ZeBu : zUtils : WARNING : The lock file
     "/zebu/.zebu_2010_05_03.log.lock" has been destroyed, it was
     locked by the pid 11358.
  ```

- After having restored a hardware state, it was sometimes not possible to read from DDR2 zrm memories: the returned values were not correct.

- In **`zRun`**, the **Flexible Local Probes** panel did not open correctly if the clock groups were declared and selected in an inappropriate way (reported in RT#22688). The following pop-up message was displayed:

  ```
  invalid command name ".flp.dumpSelection.clock.grp_1.domain.pos"
  ```

- The `designFeatures.help` file generated at runtime was not correct for the maximum number of clocks available. It still showed "`only 8 clocks`" which was an obsolete value (reported in RT#22731).
  The `designFeatures.help` now shows the following line in the `New clocks declaration` section:

  ```
  # Note : only 16 clocks are available (including your design clocks)
  ```

- The `designfeatures.help` file was not generated correctly for the memory initialization command (it included `$memoriesInitDB` instead of `$memoryInitDB`) and caused the following error message when used for runtime (reported in RT#22561):

  ```
  -- ZeBu : zServer : ERROR : LGP0034E : line 106 " $memoriesInitDB =
                       "./memory.init";" has not been evaluated.
  ```

- When word-by-word Read/Write accesses were attempted on a `ZEBU::PartMemory` object, an FPGA where the memory was mapped could no longer be accessed.

- The cache was invalidated on Read/Write word accesses from `ZEBU::PartMemory` objects.

- In HDL co-simulation, an internal error occurred on readback probes selected and introduced in the ZeBu Verilog wrapper. This error sometimes causes a segmentation fault at the beginning of an HDL co-simulation when a database of CSA signals was available. The error could disappear when the `ZEBU_SIMULATION_SIGNAL` environment variable was set to `NO`.

5.5.1 Threadsafe Environment Integration

This software release fixes the following runtime issues when integrating a threadsafe environment:

- When an input port had been saved in the hardware state but it was not connected when restoring the design, the threadsafe library caused an error.

5.5.2 Save and Restore

This software release fixes the following issue with Save and Restore feature, reported in RT#22216:

- When the design instantiates zrm memories mapped onto RLDRAM resources (only on 4C and 8C/ICE modules in ZeBu-Server), it was sometimes not possible to read the content of the memories after a Restore.
  The following message was displayed:
  ```
  Error : LAU0318E : U0 M0 F00 RLDRAM 0 memory bank read : timeout detected.
  ```
  This software release provides a partial correction for this issue. The same error still occurs sometimes when restoring a fast hardware state.
  To know if RLDRAM memories are actually used in the design, user should check section 5.1.2 (`ZRM mapping in the design`) in the zCore-level compiler report (**System Compilation → Build Core <zCore>** item in the **`zCui`** task tree).

5.6 CSA Feature (Combinational Signals Accessibility)

This software release fixes the following issues on the CSA feature:

- When the CSA feature was active, the following error was thrown either during compilation or at runtime, due to a problem in the processing of unconnected input and inout ports (reported in RT#22011 and RT#22191):
  ```
  # info : signal <signal_name> has support of size 2
  ### internal error [SEG0079E] : Cannot find node <node_name> in the graph
  <graph_name>
  ### internal error [SEG0079E] : SupportAndConeFinder.cc, line 175 : _findArcInGraph
  ```
- Some RTL vectors with escaped names were not visible at runtime using on-line or off-line CSA and hence were not available in the CSA database. They should now appear in the waveforms and in the **`zRun`** monitor.

- Some `edif` attributes were not correctly processed and caused some wrong values in CSA signals.
- The top-level module of the DUT was not processed correctly during compilation (**Build Accessibility graph** step) if an instance in this top-level module had the same name as the top-level module itself.
- At runtime, some CSA signals had incorrect values because signals modified through dynamic probes were not correctly taken into account.
- In some cases, registers were not correctly processed, causing wrong values for CSA signals.
- A segmentation fault occurred at runtime when the CSA feature was used for signals previously selected with **`zSelectsignals`** because of a database corruption issue.
- An intermediate file was not correctly closed when processing the CSA signals which reduced the achievable runtime frequency.
- During compilation, the **Build Accessibility graph** step stored some incorrect initialization values for LUTs, causing some wrong values for CSA signals.
- Escaped signal names were not correctly processed and caused the following error message:

```
### fatal error [SEG0114E] : Cannot find instance in the database
```

- In some cases, the values of combinational signals were not correct at runtime.
- The CSA-dedicated database is now split in several files to avoid problems with files bigger than 4 GB.
- A segfault occurred in **`zRun`** when some CSA signals were selected (internal error in `SimuWireEvaluator.cc:238`).
- Some CSA signals had wrong values because all their inputs were not taken into account to generate these values.

5.7 Smart Z-ICE

This software release fixes the following Smart Z-ICE issue:

- The emulation runtime failed if more than one Smart Z-ICE connector was used in the design.

5.8 Direct ICE

This software release fixes the following Direct ICE issues:

- When different tristate nets had different resolution functions declared in the Direct ICE compilation script, the system-level compiler failed and the following message was displayed (reported in RT#22697):

```
### fatal error in TRISTATE [ZTB0167F] : Conflict on wire resolution
    for wire 'top.zice_O', found both zebu_WAND and zebu_WOR connected
    to the net.
```

- When dynamic probes were declared on bidirectional pins of the Direct ICE interface, they were not set correctly and caused some erroneous values at runtime (reported in RT#22693).
- When the `zice trace` command was used several times in the Direct ICE compilation script (for example with 1 option per line), the system-level compiler only took the last line into account (previous lines were ignored).
- It was not possible to connect bi-directional IOs with the `zice_io inout` command because the system-level compiler failed to process the corresponding IOBUF of the Direct ICE FPGA (reported in RT#22417). The following error message was displayed:

```
### warning in ICE [ZTB0251W] : Changing direction of port 'A.C_2[0]'
    from inout to output as it is referenced by a 'zice_io inout' command.
### fatal error in ICE [ZTB0219F] : Cannot connect an ICE inout to an
    unconditionnal xilinx signal 'A.C_2[0]' driven by 'A.B.C_iobuf[0].IO' (iobuf).
```

- It was not possible to proceed with emulation runtime with Direct ICE on a non-U0 unit if the Direct ICE programmable voltage was set to any other value than 1.5V (reported in RT#22558). The following message was displayed (note that the modules listed in this message are used with relocation: `U0 M0` is the compilation naming; `U1 M0` is the physical location):

```
- ZeBu: zServer: ERROR : LUI1855E : U0 M0 Direct ICE vccIo_1 mismatch
- ZeBu: zServer: ERROR : Your design is compiled for vccIo_1 1.8 V
                 (U0/M0/rtb.xref),
- ZeBu: zServer: ERROR : but U0 M0 vccIo_1 is set to 1.5 V (config.dff)
- ZeBu: zServer: Power OFF Unit 0 Module 0 (physical Unit 1 Module 0)
                 Direct ICE vccIo_1.
```

- The zCore-level compiler did not process the pullup and pulldown information correctly and information was not correct for FPGA Place & Route.

5.9 zDPI Feature

This software release fixes the following issues for zDPI and C-Calls probes:

- The `enable` signal of probes declared for zDPI or C-Calls was not processed correctly by the system-level compiler when this signal was shared between multiple zCores.
- The scope of DPI calls was sometimes incorrect because of multiple references generated by the zCore-level compiler. The scopes are now checked to make sure that only non-empty scopes are processed.

- Several DPI calls were sometimes merged into a single call during FPGA compilation and created incorrect runtime behavior for the corresponding tracers.

5.10 ZEMI-3

This software release fixes the following ZEMI-3 issue, reported in RT#21739:

- ZEMI-3 did not support DPI import function calls where the return value is ignored (either explicitly via a void cast or without). Such functions are now correctly handled.
- Compilation of the software part of a ZEMI-3 transactor was potentially generating the following error (reported in RT#22254):

```
$ZEBU_ROOT/include/XtorBase.hh:30:
error: 'uint' does not name a type
```

- The `$readmem` calls in ZEMI-3 transactors were clobbered with the following warning (reported in RT#22289):

```
WARNING [39.820]: <../../patch/clkgen.v:15>: This ($readmemb)
                  and all other system task enables inside the
                  hardware will be clobbered
```

  The `$readmem` file is now inlined into the synthesized logic if the file is available at compile time.

6 Version Compatibility

This chapter includes information which is applicable for Version 6_2_1.

6.1 Supported PC Configurations

The ZeBu compilation and runtime tools are 64-bit software which can be used only on a 64-bit PC configuration (this software release does not support 32-bit PCs).

This version of ZeBu has been successfully tested on the following PC configurations:

- Compilation:

o Dell PowerEdge 1950	(RHEL 4.5)
o Dell PowerEdge 2950	(RHEL 4.5)
o Dell PowerEdge R410	(RHEL 4.5)

- Emulation Runtime:

o Dell Precision T5500	(RHEL 5.4)
o Dell Precision T5500	(SUSE 10 SP3)
o Dell Optiplex 760	(RHEL 4.7)
o HP xw6400	(RHEL 4.5)
o HP ProLiant DL360 G6	(RHEL 4.7)

6.2 Operating System Compatibility

EVE recommends using the following operating system which has been successfully tested for both compilation and emulation runtime:

- Red Hat Enterprise Linux 4 operating system, kernel 2.6.

This version of ZeBu has also been successfully tested on the following operating systems (installed in 64 bits):

- Red Hat Enterprise Linux 5, kernel 2.6 (for emulation runtime and compilation with **`zCui`** in batch mode (with `-nogui` option). See specific limitations in Section 4.2.2.
- SUSE Linux Enterprise Server 10, kernel 2.6 (for emulation runtime only). See specific limitations in Section 4.2.1.

6.3 Hardware Compatibility

This software release is only compatible with ZeBu-Server. Section 1.2 describes the installation procedure for V6_2_1 with cumulative patch B_00.

This software release is not compatible with ZeBu-XL, ZeBu-XXL, ZeBu-UF and ZeBu-Personal.

6.4 Software Compatibility

The content of this section is outdated. Section 1.2 describes the installation procedure for V6_2_1 with cumulative patch B_00.

6.5 ZeBu Licenses

License features are the same as for ZeBu version 6_2_0. However, licenses which were used on ZeBu-XXL/UF/Personal cannot be used on ZeBu-Server.

The following new features require specific licenses:

- Offline Debugger

Specific licenses are also required for:

- Xilinx ISE 11 software (mandatory)
- Concept Engineering netlist analyzers (optional, EVE can provide evaluation licenses for this product)

You should contact your usual EVE representative for detailed information.

6.6 EVE Vertical Solutions Compatibility

6.6.1 ZeBu Memory IPs

ZeBu Memory IPs are all supported by V6_2_1 software (with specific licenses).

6.6.2 ZeBu Transactors

The V6_2_1 software is compliant only with 64-bit EVE transactors and utilities.

These transactors/utilities can be requested from your usual EVE representative. The following table shows the latest releases of packages/utilities in a 64-bit version.

Transactor/Utility	64-bit Version	Doc Id
UART	2.2	VSXTOR002
JTAG with TAP controller API	1.2	VSXTOR003
SRAM SW	2.0	VSXTOR004
PCI Express	5.0	VSXTOR005
Ethernet (GMII)	1.3.1	VSXTOR006
`etherPlug` utility	1.1	VSXTOR006
I2S	1.3	VSXTOR007
Digital Video	3.2	VSXTOR008
IEEE 1394 FireWire	1.2.1	VSXTOR009
MMC Device	1.2	VSXTOR010
AHB Master	2.0	VSXTOR011
Streaming Toolkit	1.6	VSXTOR012
USB	1.2	VSXTOR013
AXI Master	2.2.1	VSXTOR014
AXI Slave	2.2.1	VSXTOR014
I2C	1.0	VSXTOR015
MMC Host	1.1	VSXTOR016
HDMI Sink	1.0	VSXTOR017
TLM2 Adaptor	1.0	VSXTOR020
DRAM SW	1.0	-

6.7 Third-Party Tools Compatibility

The following table gives information about the third-party tools that have been successfully tested by EVE with the present version of the ZeBu software. This list is given for information purposes and does not necessarily imply that other versions will cause system malfunction.

Tool	Tested Versions	Tested OS compatibility *	Remarks
Synplify Pro (Synopsys)	2009-06.SP1	RHEL 4.0	FPGA synthesizer. http://www.synopsys.com
XST (Xilinx)	11.4i	RHEL 4.0	FPGA Synthesizer. http://www.xilinx.com
Precision (Mentor Graphics)	Not Tested		FPGA Synthesizer. http://www.mentor.com
VCS (Synopsys)	mx-2008.09	RHEL 4.0	HDL Simulator. http://www.synopsys.com
ModelSim (Mentor Graphics)	6.2f	RHEL 4.0	HDL Simulator. http://www.mentor.com
NCSIM (Cadence)	Not Tested		HDL Simulator. http://www.cadence.com
SystemC	2.2.0	RHEL 4.0	Required for SystemC co-simulation. http://www.systemc.org
GTKWave	3.1.11	RHEL 4.0	Waveform viewer. http://gtkwave.sourceforge.net/
Debussy (Novas)	2009.10	RHEL 4.0	Interface for design debugging and waveform viewer. http://www.springsoft.com/
RTLVision GateVision PRO (Concept Engineering)	4.6.5	RHEL 4.0	Graphical netlist analyzers. http://www.concept.de/
gcc	3.4	RHEL 4.0	C Compiler. Part of the Linux operating system distribution. Required for C/C++ and SystemC co-simulation. May be required during installation process in case of kernel compilation. http://gcc.gnu.org/
	4.1	RHEL 5	For runtime on a RHEL 5 operated PC, the testbench and the software part of the transactor should be compiled with this version of gcc C/C++ compiler.
ISE Place & Route (Xilinx)	11.4i_patched	RHEL 4.0	Xilinx tools for FPGA Place and Route during ZeBu compilation. http://www.xilinx.com Included in ZeBu software package.

* The Operating System Compatibility of these tools can change and you are recommended to visit the third-party websites for information.

7 EVE Contacts

For product support, contact: support@eve-team.com.

For general information, visit our company web-site: http://www.eve-team.com

Europe Headquarters	EVE SA 2-bis, Voie La Cardon Parc Gutenberg, Bâtiment B 91120 Palaiseau FRANCE Tel: +33-1-64 53 27 30
US Headquarters	EVE USA, Inc. 2290 N. First Street, Suite 304 San Jose, CA 95054 USA Tel: 1-888-7EveUSA (+1-888-738-3872)
Japan Headquarters	Nihon EVE KK KAKiYA Building 4F 2-7-17, Shin-Yokohama Kohoku-ku, Yokohama-shi, Kanagawa 222-0033 JAPAN Tel: +81-45-470-7811
Korea Headquarters	EVE Korea, Inc. 804 Kofomo Tower, 16-3, Sunae-Dong, Bundang-Gu, Sungnam City, Kyunggi-Do, 463-825, KOREA Tel: +82-31-719-8115
India Headquarters	EVE Design Automation Pvt. Ltd. #143, First Floor, Raheja Arcade, 80 Ft. Road, 5th Block, Koramangala Bangalore - 560 095 Karnataka INDIA Tel: +91-80-41460680/30202343
Taiwan Headquarters	EVE-Taiwan Branch 5F-2, No. 229, Fuxing 2nd Rd. Zhubei City, Hsinchu County 30271, TAIWAN (R.O.C.) Tel: +886-(3)-657-7626