ZeBu-Server
Release Note

Version 6_3_0

January 2011

Table of Contents

About this document

This Release Note describes the major features available for Version 6_3_0 release of the ZeBu-Server software.

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" section (§3.1) before launching a new compilation with this software release.

1 New Features

1.1 Installation new features

1.1.1 Modifying the memory capacity for SRAM Trace

By default, 256 MBytes are reserved in each FPGA module for SRAM trace purposes. It is possible to modify the reserved memory capacity during the initialization of the ZeBu-Server system. Any modification applies to the entire ZeBu-Server system, whatever the configuration (in particular when different types of FPGA modules are connected in the system), and applies to all users.

To modify the reserved memory capacity, the `setup_template.zini` file has to be edited before launching **`zSetupSystem`**. The following line should be modified:

```
$TRACE_SIZE = <trace_size>;
```

Where `<trace_size>` is the size in 32-bit words, given as a hexadecimal value starting with `0x`. The maximum authorized values are the following ones:

- System with 8C/ICE modules: `0x20000000` (eq. to 2 GBytes)
- System with only 4C or 16C modules: `0x40000000` (eq. to 4 GBytes)
- Default: `0x4000000` (eq. to 256 MBytes)

In case of a ZeBu-Server system containing heterogeneous modules with different physical memory sizes (e.g. 8C/ICE and 16C modules with 2 GBytes and 4 GBytes), if user declares a 3 GBytes SRAM trace size then the complete memory will be reserved for SRAM trace in the 8C/ICE module.

The memory reserved for SRAM trace does not impact the available resource to map the design memory or the testbench memory, since it is connected to a control FPGA in each module.

1.1.2 Balancing SRAM Trace capacity and Bitstream Cache

The physical memory which supports SRAM trace, if not completely reserved for trace purposes, is also used to store the bitstream files of design FPGAs and RTB FPGAs. Thanks to this cache system, the ZeBu-Server system can be loaded faster when the same design is used repeatedly.

If the complete memory is reserved for SRAM trace, there is no available space for the cache and all the bitstream files are downloaded from the host PC to the ZeBu-Server system each time the emulation starts.

This cache works as a circular buffer to keep only the most recent bitstream files. If the bitstream files of a design are not present in the cache, they are downloaded from the host PC to the cache before the ZeBu-Server system is loaded.

The content of the cache is erased at system initialization with **`zSetupSystem`** or **`zUtils`** (when used with `-Initsystem` or `-dt` options). In case of error when loading FPGAs from the bitstream cache, the bitstream files are automatically downloaded from the host PC without using the cache mechanism.

1.1.3 Improvement of diagnostics duration

By default, **`zSetupSystem`** tests all the memories at the end of setup, after the calibration of the ZeBu-Server system. This is equivalent to:

```
$ zUtils -mem all_mem
```

If the duration of the diagnostics is an issue in your environment, these tests can be skipped by uncommenting the following line in the `setup_template.zini` file before launching **`zSetupSystem`**.

```
#$noMemTest = 1;
```

1.1.4 Getting board labels

For maintenance purposes, your EVE representative may ask you to provide information about the ZeBu system. You can obtain the board labels of a ZeBu-Server system (PCIe, hub, backplane, front panel, module) using the following command:

```
zUtils -getLabel <labelBoardName> <labelFileName>
```

Where:

- `<labelBoardName>` is any of the following boards:

PCIe board	Hub	Unit U0	Unit U1	Unit U2	Unit U3	Unit U4
S0_PCIe	HUB	U0_BP	U1_BP	U2_BP	U3_BP	U4_BP
S1_PCIe		U0_FP	U1_FP	U2_FP	U3_FP	U4_FP
S2_PCIe		U0_M0	U1_M0	U2_M0	U3_M0	U4_M0
S3_PCIe		U0_M1	U1_M1	U2_M1	U3_M1	U4_M1
S4_PCIe		U0_M2	U1_M2	U2_M2	U3_M2	U4_M2
		U0_M3	U1_M3	U2_M3	U3_M3	U4_M3
		U0_M4	U1_M4	U2_M4	U3_M4	U4_M4

  Where BP stands for backplane and FP for front panel.
  If `<labelBoardName>` is not specified, all labels will be dumped on screen.
- `<labelFileName>` is a file where the E2PROM content of the specified board will be stored. If `<labelFileName>` is not specified, then the board label will be dumped on screen.

For more details, please refer to Section 3.4 of the ***ZeBu-Server Installation Manual*** (Rev. E).

1.2 zFAST new features

1.2.1 Quality of Result (QoR)

This release introduces the following improvements in **zFAST** for QoR, in particular:

- Better LUT usage.
- Reduction of the port number when possible for memories inferred with ZeBu memory generator.
- Better ability to map memories with a large number of ports.
- Better support for SystemVerilog memory structures.

The following attributes can also be added in the **zFAST Additional Attributes File** to improve QoR:

- Enabling inference of LUT6_2.
  ```
  Compile:lutNodes=true
  ```
- Enabling small LUTs pairing.
  ```
  Compile:PLY:PairLuts=true
  ```
- Usage of ROM128x1, ROM256x1 for LUT7 and LUT8.
  ```
  Compile:CellSize=8
  ```
- Optimization of dead logic.
  ```
  Compile:DropDeadLogic=true
  ```
  This will remove dead logic but it will also decrease accessibility.

1.2.2 User attribute settings copied in RTL Front-End log file

When synthesizing with **zFAST**, the attributes declared in the **zFAST Additional Attributes File** and the ones corresponding to the settings in the **zCui** interface are now reported in log file of the **Make RTL Front-End** step (via the **Show Full Log** contextual menu item), as in the following example:

```
# step Before Analyze : Start of copy of file ../../../hcsrc, which has user attributes.
Compile:ForceFlops=mem.memory
# step Before Analyze : End of copy of file ../../../hcsrc

[…]

# step Before Analyze : Start of copy of file ../rtlfe_Default_RTL_Group.hcsrc, which has
which has user attributes.
# FE GUI options
Compile:MemorySizeThreshold=1
# step Before Analyze : End of copy of file ../rtlfe_Default_RTL_Group.hcsrc
```

1.2.3 New Attributes for Memory Modeling

This release introduces new attributes for memory modeling purpose:

- In order to have No-Read-On-Write ports for memories generated by the ZeBu memory generator:
  `MemoryType:NoReadOnWrite=<memory_paths_list>`
- In order to use the optimization feature of the ZeBu memory generator, as described in the *ZeBu-Server Compilation Manual* (Rev. C):
  `MemoryType:OptimizeCapacity=<memory_paths_list>`

In both cases, `<memory_paths_list>` can be a list of full paths from the top to the arrays or under the form `<module>.<array_name>`.

1.3 Support of version 2010 of Synplify synthesizers

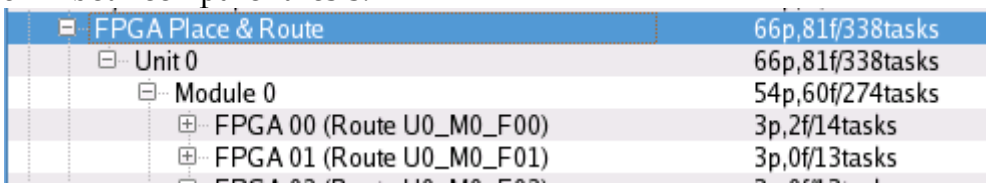
This release introduces support of version 2010 of Synplify synthesizers.

1.4 Compilation new features

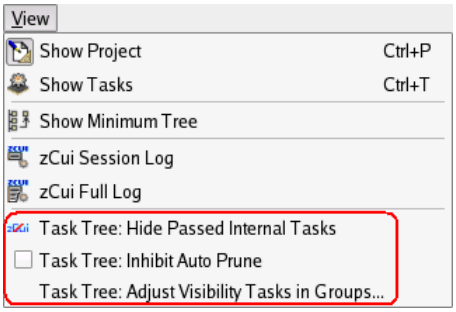
1.4.1 `zCui` new features

The features below are fully described in the ***ZeBu-Server Compilation Manual*** (Rev. C):

- Selection in a Log for Copy-Paste.
- Support of Transactor Mapping feature, as described in Section 1.4.3.
- Complete revamping of the **Task Tree** in **Compilation** view, mostly for support of the PARFF feature, described in Section 1.4.2.
- In the new **Task Tree**, the FPGA Place and Route section now matches the logical name of FPGAs and the logical name set by the system-level compiler is shown between parenthesis:



- The Performance Oriented Partitioning (POP) can now be activated and controlled from the **`zCui`** graphical interface.
- In the **View** menu, new items are available to improve the filtering capability in the task tree:



Note that these new filters always apply to the entire task tree and cannot be applied for a group of compilation tasks. Moreover, when a filter is activated by user, it applies to the already displayed tasks and to the future tasks as well.

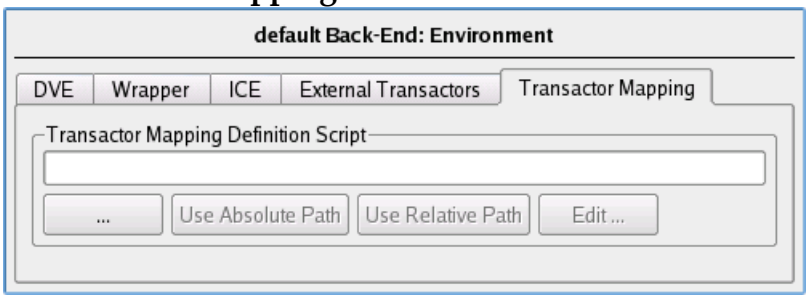
1.4.2 Parallel Automatic Recompilation of Failing FPGAs (PARFF)

For designs with a high number of FPGAs and when the FPGA filling rates are higher than the clustering default settings, the FPGA Place & Route of the design sometimes failed for very few FPGAs which would need different settings. In order to take such FPGAs into account, the ZeBu compiler can automatically launch the FPGA Place & Route with different settings if it fails with the default settings.

This feature is known as Parallel Automatic Recompilation of Failing FPGAs (PARFF) and can be activated from the corresponding frame in the **Advanced** panel, as described in the ***ZeBu-Server Compilation Manual*** (Rev. C).

1.4.3 Transactor mapping on different RTB FPGAs

This feature should be used in case of compilation problems due to a full IF FPGA. For that purpose, a dedicated file should be declared in the **Back-end → Environment → Transactor Mapping** tab:



See details in the *Transactor Mapping* section of the ***ZeBu-Server Compilation Manual*** (Rev. C).

1.4.4 zCore declaration with RTL paths

When synthesizing with **zFAST** in block-based mode, the manual declaration of zCores in the **zCore Definition File** with `defcore` commands now supports RTL-paths, using the `-rtlname` option, as described in the ***ZeBu-Server Compilation Manual*** (Rev. C).

Note that zCore declaration with RTL paths is not supported when synthesizing with **zFAST** in top-down mode or with third-party synthesizers.

1.4.5 Improvements for Density Oriented Clustering

New options have been introduced to better match the properties of the design and to improve the clustering performance.

1.4.5.1 User-defined estimation of the weight of an instance

In some cases, the weight of an instance of the design is not estimated correctly by automatic clustering and impacts the quality of result of the clustering. For example, the blackboxes are considered as empty instances by the automatic clustering but their size may impact the accuracy of the estimations for automatic clustering.

The following command modifies the estimation of an instance for each type of resource:

```
cluster set -weight -instance <inst_path> -<type> <weight_by_type>
```

Where:

- `-<type>`: `–reg`, `-lut`, `-ramlut`, `-bram`, or `–dsp`
- `<weight_by_type>`: user-defined estimation of instance `<inst_path>` for the corresponding `<type>`. This user-defined weight is used only for estimation by automatic clustering, and has no other effect on the compilation.

Note that this command can be used once for several types of resources and with a different value for each type.

In case of multiple instantiations of the same module, this command has to be declared once for each instantiation.

Example:

```
cluster set –weight –instance top.a.b.c –lut 40000 -reg 25000
```

1.4.5.2 Parameters to compute the initial mapping solution

The density oriented clustering first processes an initial mapping solution that will be optimized before completing the mapping solution. Several initial mapping solutions are computed by default and the zCore-level compiler selects the best one.

The log of the zCore-level compiler provides information about the initial mapping solutions and which method has been selected:

```
#   step AC : METH(coarse_blocks) - average_cut(401.00) , max_cut(401) 1.0170e+03(acceptable)
#   step AC : METH(fpga_frag) - average_cut(401.00) max_cut (401)      1.0170e+03(acceptable)
#   step AC : METH(nop_frag) - average_cut(401.00) max_cut (401)       1.0170e+03(acceptable)
#   step AC : Optimizing with coarse_blocks
```

In order to test if one of the not selected methods provides better results, user can force it with this command:

```
cluster set -method <fpga_frag|coarse_blocks|nop_frag>
```

Note: `cluster set -method nop_frag` reproduces the behavior of V6_2_1B.

1.4.5.3 Modifying the constraints for the initial mapping solution

Because it has been observed that ISE can sometimes map larger blocks if there is no other logic mapped on the FPGA, the method to establish the initial solution can map a large atomic block even if it overflows mapping constraints. By default blocks with 20% overflow on registers and LUTs with a cut lower than 3000 nets are accepted. These limits can be changed using one of the following commands:

```
cluster merge_blocks -atomic -maxfill -<type> <max_resource>
  [-max_cut <cut>] [-exclude_path_list {path_pattern1 path_pattern2 ... }]

cluster merge_blocks -atomic –overflow -<type> <max_overflow>
  [-max_cut <cut>] [-exclude_path_list {path_pattern1 path_pattern2 ... }]
```

Where:

- `–<type>`: `–reg`, `–lut`, `–ramlut`, `–bram`, or `–dsp`
- `<max_resource>`: relative amount of resource (in %)
- `<max_overflow>`: maximum authorized overflow of each resource (in %)
- `[–max_cut <cut>]`: blocks with a cut lower than the one specified will be selected (default is 3000).
- `[–exclude_path_list {path_pattern1 path_pattern2 ... }]`: blocks that match the pattern(s) will not be selected.

Note that this command can be used once for several types of resources and with a different value for each type.

This large block selection can be totally disabled with this command:

```
cluster disable -auto_merge_block_atomic
```

1.4.5.4 Disabling automatic modification of user-defined filling constraints

When user-defined filling constraints could not be met, the zCore-level compiler automatically modifies the constraints so that the clustering could be processed to map the design. This can now be disabled by adding the following command for the zCore-level compiler:

```
cluster disable [-]accept_changes_on_user_max_fill_constraints
```

1.4.5.5 FPGA filling constraints automatic adjustment

The mapping of the design on FPGAs relies on cost estimates, the accuracy of which evolves during the compilation flow. The goal of automatic adjustment is to make corrections to the filling constraints in order to reach the best compromise between resource usage and performance while relying on the most accurate cost estimates.

- `max_fill_limit`
  This parameter has been introduced to fix the limit that constraint adjustments cannot break. When the tool determines that the current filling constraint setting cannot be fulfilled, the filling constraints will be increased to satisfactory values. If the computed constraints are higher than `max_fill_limit`, a fatal error is thrown.

  Default `max_fill_limit` values:
  REG(85%), LUT(90%), RAMLUT(90%), BRAM(100%), DSP(80%).

If user declares a filling constraint higher than `max_fill_limit`, then `max_fill_limit` will be modified accordingly. For instance, if user sets: `cluster set max_fill_reg=90`, then the `max_fill_limit` REG value will be set to 90.

User can change the default with the following command:

```
cluster set -max_fill_limit -reg <percentage> -lut <percentage>
            -bram <percentage> -dsp <percentage>
```

If user changes this parameter, the following table is displayed in the log file:

```
#   step AC : FPGA Filling constraints
#   step AC : ------------------------
#   step AC :
#   step AC : Constraint limits
#   step AC : ----------------
#   step AC :
# +---------------------------------+----+----+------+----+----+
# | MODE: customized                | REG| LUT|RAMLUT|BRAM| DSP|
# +---------------------------------+----+----+------+----+----+
# |Max fill (upper limit)           | 85%| 90%|   90%|100%| 80%|
# |Max fill (upper limit), tcl input| 90%|   -|     -|   -|100%|
# |actual values                    | 90%| 90%|   90%|100%|100%|
# +---------------------------------+----+----+------+----+----+
# |Max usage                        | 96%| 96%|   90%|100%|100%|
# |Max usage, tcl input             | 89%|  -%|     -|   -|   -|
# |actual values                    | 89%| 96%|   90%|100%|100%|
# +---------------------------------+----+----+------+----+----+
# |Min usage                        | 60%| 60%|    0%|  0%|  0%|
# |Min usage, tcl input             |   -|   -|     -|   -|   -|
# |actual values                    | 60%| 60%|    0%|  0%|  0%|
# +---------------------------------+----+----+------+----+----+
#   step AC : #
```

- `max_usage`

  This parameter defines the amount of resources available after the filling constraints are applied. Assuming the register filling rate is 65%, if the design requires 90% of the registers after applying the 65% filling rate, the REG resource usage is 90%.

  When the resource usage becomes high, the capacity increases but mapping becomes more and more difficult. The `max_usage` parameter allows to define the maximum usage that will not trigger an overflow error. When `max_usage` is low, more room is given for improvements at the expense of capacity.

  Default `max_usage` values:
  REG(96%), LUT(96%), RAMLUT(90%), BRAM(100%), DSP(100%).

  The commands that allow manual changes to this parameter is:

  ```
  cluster set -max_usage -reg <percentage> -lut <percentage>
              -bram <percentage> -dsp <percentage>
  ```

  If no filling rate is found which also satisfies the `max_usage` constraint, a fatal error is thrown.

- When the logic cost is over-estimated in the early stages of system-level compilation, user may have to declare high filling constraints in order to pass without error. During the zCore-level compilation stage, the accuracy of the estimations may show that these filling constraints are inappropriate and may lead to FPGA compilation failure.

  High filling rates would translate into lower resource usage. If the usage of resources becomes too low, then the filling constraints will be automatically lowered to a medium value between the default value and current value.

  For instance, if resource usage is below 60% and REG filling rate had to be set to 80%, the REG filling rate is then adjusted to 67 (default is 55%).

  This adjustment can be disabled with the following command:
  ```
  cluster disable max_fill_constraint_adjustment_by_usage
  ```

- Additional command:
  This command allows user to disable any modification on filling constraints:
  ```
  cluster disable accept_changes_on_user_max_fill_constraints
  ```

1.4.5.6 Specific estimations for muxf7 and muxf8 primitives

If the EDIF netlist of the design instantiated a high number of muxf7 and/or muxf8, the FPGA Place and Route was sometimes difficult because of the resulting registers filling rates, reported in RT#24912.

In order to avoid such issues, the zCore-level clustering algorithm now weighs by default each muxf7 or muxf8 as 1 register which results in mapping less muxf7 or muxf8 in FPGAs.

In the report of the zCore-level compiler (in the contextual menu of the `Build Core` step, select **Show zCoreBuild Report**), a column has been added in the `Design Mapping` table to show the number of muxf7 and muxf8 in each FPGA.

This new feature can be disabled by adding the following command in the **Build System** advanced command file:
```
cluster disable -muxf78_is_one_reg
```

This command is available in both system-level and zCore-level compilers (the system-level compiler forwards the configuration to the zCore-level compilers).

Associated to this feature is an extension which modifies the default cost for a Xilinx primitive:
```
cluster model -primitive_cost=<prim_name_pattern>  [-lut=<int>]
   [-ramlut=<int>] [-reg=<int>] [-bram=<int>] [-dsp=<int>]
```

Example:
```
cluster model -primitive_cost=muxf7* -reg=1
```
Associates the cost of 1 REG to any cell with a name starting by `muxf7`, (default is `0`).

If the redefined primitive is a LUT, the redefinition overrides any previous LUT weighting setup.

1.4.6 Managing Two `force assign` commands

When two `force assign` commands are declared for signals separated by a synthesis buffer, the 2 commands are applied separately by default and it is possible to remove the buffers by adding the following command in the **Netlist Edition File** declared in the z**Core Definition** panel, reported in RT#24752:

```
remove_syn_buffers
```

No parameters or options should be added; this command should be added before the `fix_multi_driver` command (the commands in the **Netlist Edition File** are processed in the order they appear).

1.4.7 Optimization for BRAM/LUTRAM-based memory models

The ZeBu memory generator favors performance over area optimization. It is now possible to optimize the amount of logic used for LUTRAM-based and BRAM-based memory modules in the FPGAs by adding the following command in the memory source file:

```
memory optimize_capacity true
```

Former optimization commands, `optimize_be` and `optimize_bie`, which already existed for bit-enabled and byte-enabled memory models will be obsoleted in a future release. In the meantime, these commands will be processed like the new ones and the following warnings will be issued if and when they are used:

```
### warning in DEFINE : Command 'optimize_bie' should be replaced by command
    'optimize_capacity', and will no longer be supported in next releases.
### warning in DEFINE : Command 'optimize_be' should be replaced by command
    'optimize_capacity', and will no longer be supported in next releases.
```

1.4.8 `keeper` resolution for tristate signals

Top-level tristate ports can now be defined with a `keeper` resolution. Recommendations for tristate signal handling are described with details in the ***Zebu-Server Compilation Manual*** (Rev. C).

1.4.9 Flattening the netlist

When a translation library has been merged in the design, it can optionally be flattened to accelerate the ZeBu compilation and make the runtime database easier to use (the additional hierarchical level added for the library is removed).

For that purpose, the following command is added for the system-level compiler in the **Netlist Edition File** (**zCore Definition** panel):

```
flatten -cell <cell_name> [-fnmatch]
```

Where `<cell_name>` is a string which is either the exact name of a cell or a pattern-based string when `–fnmatch` option is added (in such a case, `<cell_name>` includes `*` for several characters and/or `?` for a single character).

1.4.10 Reduction of Inter-FPGA Clocks

In order to optimize the achievable runtime frequency, it is possible to reduce the number of inter-FPGA clocks in order to reduce the `zClockSkewOffset` value and thus increase the maximum achievable runtime frequency. For that purpose, the appropriate clock trees are replicated in order to remove all inter-FPGA clocks when the **Skew Offset Options → Single FPGA clock path localization** checkbox is selected in the **Clock Handling** panel, as described in Section 5.3.2 of the ***ZeBu-Server Compilation Manual*** (Rev. C).

1.4.11 Configuring the allocation of clock resources for user clocks

1.4.11.1 Allocation of Clock Buffers (BUFGs)

By default, the ZeBu compiler allocates 6 clock buffers (BUFGs) for user clocks so that they are mapped automatically to the low-skew routes in the FPGAs. The clocks with the highest fanout values are mapped to these resources in priority.

The available number of such buffers for user clocks can be increased (maximum is 12) if more user clocks have to be mapped with BUFGs or decreased in case of FPGA compilation failures.

For that purpose, the following command should be added in the additional command file in the **Advanced → Build System Parameters** frame:

```
set_config -bufg <n>
```

Where `<n>` is an integer (minimum is 3; maximum is 12).

1.4.11.2 Settings for Clock Buffers Allocation

In order to optimize the allocation of clock buffers, the fanout criteria becomes stronger when the number of BUFGs increases (the minimum fanout to map a clock with a clock buffer increases).

By default, the minimum fanout for the first BUFG allocation is 0 (which means that the highest fanout clock in the design will use a BUFG in any case). After each BUFG allocation the minimum fanout for allocation is increased by 2000.

The settings for this allocation can be modified by adding the following commands in the additional command file declared in the **Advanced → Build System Parameters** frame:

```
synchro fanout_min -value=<min_value>
synchro fanout_step -value=<step_value>
```

Where `<min_value>` and `<step_value>` are integers.

1.4.12 New options and commands for Static Timing Analysis

The following options and commands have been added for Static Timing Analysis in order to provide more accurate estimations:

- If a memory appears to be a bottleneck because it is a known slow memory which does not impact the driverClock frequency, this memory can be ignored in the static timing analysis by declaring it in the same way as the false paths. The declaration can be done either with the explicit path to the memory or on a pattern-matching basis:

```
set_false_memory -name=<Memory_name>
set_false_memory -match=<Pattern>
```

- When declaring false paths (with the `set_false_path_from` command, for example), it is now possible to use `–match_wire` or `–match_alias` options in order to use the wires and aliases names which are available in the html reports. It is much easier than using only FPGA port names for that purpose.

1.5 Runtime new features

1.5.1 ZeBu-Server system loads faster with same design

The physical memory which supports SRAM trace can also used to store the bitstream files of design FPGAs and RTB FPGAs. Thanks to this cache system, the ZeBu-Server system can be loaded faster when the same design is used repeatedly, as described in Section 1.1.2.

1.5.2 New clock generator

This release introduces a new optional feature for the ZeBu clock generator which provides the primary clocks to the design, in order to improve the runtime performance when several primary clocks are declared for runtime in a single clock group.

The performance improvement is more visible when the number of primary clocks increases and when the waveforms and ratios declared in the `designFeatures` file are heterogeneous.

In order to optimize the achievable runtime performance, it may be of interest to process simultaneously the clock edges which are very close to one another. This behavior can be applied on a group-by-group basis or on a clock-by-clock basis. To activate this new feature, the `designFeatures` file must include the following line for each group/clock which needs to be modified:

```
$U0.M0.<group>.Tolerance=<value>
```

or

```
$U0.M0.<clock>.Tolerance=<value>
```

Where <value> is a string with double quotes and with the following values:

- `<value>="no"`: Same behavior as before V6_3_0. This is the default value.
- `<value>="yes"`: Edges are merged.

When the tolerance set for a given clock group is different from the tolerance set for a single clock in this group, all the clocks belonging to the group will have the group tolerance except that clock, as shown in the example below:

```
#Group setting for group1
$U0.M0.group1.Tolerance = "yes";
#CLK0 uses the group setting for tolerance
$U0.M0.CLK0.VirtualFrequency = "8;
$U0.M0.CLK0.GroupName = "group1";
# CLK1 overwrites the group setting to disable tolerance
$U0.M0.CLK1.VirtualFrequency = "5;
$U0.M0.CLK1.GroupName = "group1";
$U0.M0.CLK1.Tolerance = "no";
# CLK2 uses the group setting for tolerance.
$U0.M0.CLK2.VirtualFrequency = "2;
$U0.M0.CLK2.GroupName = "group1";
```

If a clock has the same name as a clock group in `designFeatures` and causes ambiguity in the declaration of the tolerance, then the runtime will exit in error with the following message:

```
"LUI1990E" : "Ambiguous designFeatures declaration : $U0.M0.clk.Tolerance = "yes"
"A Clock Name and a Group Name are the same (clk) : "Tolerance" can not be attributed"
```

Notes:

- Tolerance can only be attributed to controlled clocks.
- At least 2 clocks are needed in the same group for an effective calculation.
- Only `yes/no` attributes are accepted and they are case insensitive.

1.5.3 New runtime database

A new runtime database has been introduced with this release to support large designs (up to 1B gates). This new unified database replaces two previous databases: one database was used for dynamic probes and another one for simulated signals. Therefore **`zDbPostProc`** and **`zSelectProbes`** continue to be supplied with the ZeBu software, but **`zSelectSignals`** and **`zVisiPostProc`** are no longer supplied.

- **`zDbPostProc`** is now the only script tool available for the selection of dynamic probes and/or simulated signals in the database.
  See Section 1.5.3.1 for details on changes in **`zDbPostProc`**.
- **`zSelectProbes`** is the graphical tool used to select signals for monitoring at runtime. See Section 1.5.3.2 for details on changes in **`zSelectProbes`**.

1.5.3.1 Changes in `zDbPostProc`

The following table summarizes the changes made to **`zDbPostProc`** Tcl commands. The command names are color-coded as follows:

- Black Command is kept as is.
- Blue Command renamed (as shown on same line).
- Green Command is new.
- Red Deprecated command (will be obsolete in one or more releases).

	Previous `zDbPostProc` commands	Current `zDbPostProc` commands
Database operations	close_db open_db save_db save_probes set_work	close_database open_database save_database save_selection
Navigation	scope upscope where	scope upscope where
Information retrieval	print print_banks print_chip print_memory	print print_banks print_chip print_memory print_equi
List operations	drivers instances memories probes signals static_probes	drivers instances memories probes signals static_probes

	Previous zDbPostProc commands	Current zDbPostProc commands
Selection	clear_probes load_probes remove select print_probes_number select -all	clear_probes load_probes remove select load_selection select_all select_all_csa select_all_design select_all_design_reg select_all_reg select_all_rtl select_all_rtl_reg
Vector operations	revectorize combine expand	revectorize
Miscellaneous	change_mem delete_static_probes_hierarchy help import_mem separator show static_probe wrapper delete dump_renamed merge_case name set_fpga_writability	change_mem delete_static_probes_hierarchy help import_mem separator show static_probe wrapper check_signals save_ascii save_check_signals_file

Soon to be obsolete **zDbPostProc** Tcl commands

Until they become definitely obsolete, a warning will be issued when one of the following commands is used, indicating that the command is now deprecated: `save_db`, `set_work`, `print_probes_number`, `expand`, `combine`, `delete`, `set_fpga_writability`, `name`, `merge_case`, `dump_renamed`.

New/Modified **zDbPostProc** Tcl commands

- `open_database`: opens the database.
- `close_database`: closes the database.
- `save_selection <file_name>`: records a selection of signals.
- `save_database`: saves the database.
- `print_equi`: displays all alias groups in the database.
  `print_equi <signal_name>`: displays all aliases of the specified signal.
  Note: the alias feature requires that the database be fully loaded into memory. According to the size of the database, the waiting time may be long when the first `print_equi` command is issued. Subsequent calls to `print_equi` are instantaneous.
- `select`: selects a single signal/vector or the signals and vectors of an instance down to a depth specified with the depth parameter.
- `select_all`: selects all signals.
- `select_all_csa`: selects all simulation signals.
- `select_all_rtl`: selects all signals coming from RTL sources in DUT. `select_all_design` may be used instead.

- `select_all_rtl_reg`: selects all dynamic probes coming from DUT. `select_all_design_reg` may be used instead.
- `select_all_reg`: selects all dynamic probes, including ZeBu signals.
  Note: `select_all*` and `select -depth` display one or more top names in the list of selected probes, not individual signal names. To get around this inconvenience, instances will appear in the list of probes as shown below:

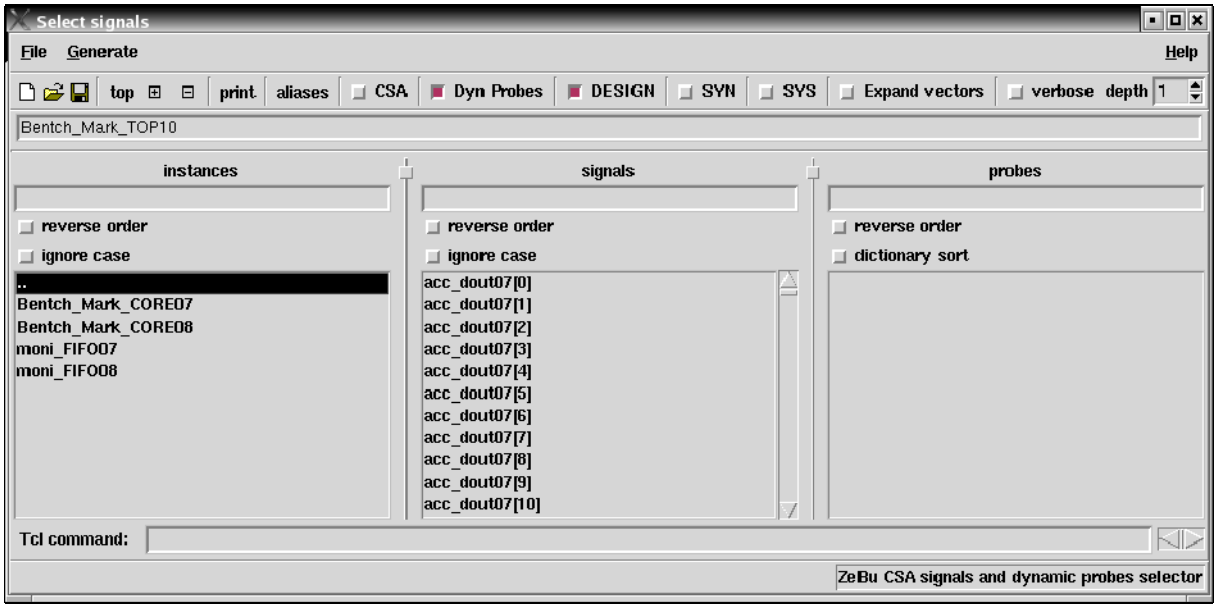
```
top.i1.i2 (ALL-CSA-DYN-RTL-EDF-SYN-SYS)
top.i1.i2 (3-CSA-DYN-RTL-EDF-SYN-SYS)
top.i1.i2 (3-DYN-RTL-EDF)
```

  Where:
  - `ALL` and `3`: depth of selection (ALL for entire sub-hierarchy)
  - `CSA`: simulated signals are not filtered.
  - `DYN`: dynamic probes are not filtered.
  - `RTL`: signals coming from RTL sources are not filtered.
  - `EDF`: signals coming from EDIF sources are not filtered.
  - `SYN`: signals added by zFAST are not filtered.
  - `SYS`: signals added by ZeBu are not filtered.
- `load_selection <file_name>`: loads a selection of signals.
- `check_signals`: returns a simple list of registers.
- `save_check_signals_file`: returns a simple list of registers.
- `save_ascii`: dumps the database in a text file.

1.5.3.2 Changes in **`zSelectProbes`**

The following buttons have been added in the **`zSelectProbes`** graphical interface:

- **print**: dumps in the console information linked to the selected object.
- **aliases**: displays the various aliases of the highlighted signal. Same as **`zDbPostProc`** command `print_equi` with a signal name as argument.
- **CSA**: displays/hides simulated signals. If set to be displayed, these signals will only be shown if the CSA feature is enabled.
- **Dyn probes**: displays/hides dynamic probes.
- **DESIGN**: displays/hides signals coming from RTL as well as EDIF sources in the DUT. Signals coming from RTL sources will only be shown if the CSA feature has been activated.
- **SYN**: displays/hides the signals added by **`zFAST`**.
- **SYS**: displays/hides the signals added by ZeBu.
- **Expand vectors**: displays/hides vectors.
- **Verbose depth:** specifies a verbosity level.
- **depth** selector: specifies the depth when an instance is selected.



1.5.4 New C/C++ API command to hide messages in screen outputs

This release introduces a new command in the C and C++ APIs to hide all messages (warnings, information/error messages) in the screen outputs. However, if they are set to be hidden in the screen outputs, these messages will still appear in the log files.

C API :

```
int ZEBU_setMsgVerboseMode(ZEBU_Board *, int verbose);
```

C++ API (in the `Board` class):

```
bool setMsgVerboseMode(bool verbose) throw(std::exception);
```

1.6 Support of system calls in ZEMI-3 transactors

`$display` and `$write` system calls are now supported in ZEMI-3 transactors. The following attribute has to be set in the **Additionnal Command File** of the ZEMI-3 transactor **Properties** tab in **`zCui`**:

```
Compile:Zxtor:SupportDollarDisplay=true
```

The default setting is:

```
Compile:Zxtor:SupportDollarDisplay=false
```

The only supported tasks are:

- `$display`, `$displayh`, `$displayb`, `$displayo`
- `$write`, `$writeh`, `$writeb`, `$writeo`

Note: All `$display` calls in a module are serialized with respect to each other (so that they come out in order and so that they share resources), but they remain independent (not serialized) with respect to DPI calls.

1.7 CSA Improvements

- Shortened computation time for CSA using multiple processes.
- Additional step to merge information gathered during automatic clustering.
- Offline CSA and iCSA improved in terms of memory and speed.

1.8 Improved interoperability of a design

This release introduces the following enhancements for interoperability of a design:

- A design compiled for a single module (with `use_module` command in the z**Core Definition** file) can now be used without recompiling on any type of Zebu-Server unit (2-slot unit, 5-slot unit either in a single unit or multi-unit configuration), assuming the modules populating the unit are the same ones.
  Note that if the single-module design uses 5 Smart Z-ICE ports, it cannot be used on a 2-slot unit.
- This enhancement requires to use dynamic relocation at runtime as described in V620/V620B and V621/621B Release Notes.
  To relocate the design on a module, set `ZEBU_M0_PHYSICAL_LOCATION` to `M<i>` (where `<i>` is the index of the module).
  To relocate the Smart Z-ICE connector, the following line is added in the `designFeatures` file:

```
$smartZICE.connectorRemap_#i = #j;
```

This release also offers dynamic relocation for the Smart Z-ICE connector through the following environment variable:

```
ZEBU_SMARTZICE_P#i_PHYSICAL_LOCATION=P#j
```

2 Documentation Package

2.1 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.

2.2 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***	e	New sections: • Getting the boards labels. • Using zInstall in a multi-unit configuration. • Memory tests. • Parameterizing the SRAM trace size. • SRAM trace Vs. bitstream cache. • Runtime errors caused by faulty memories. • PCIe interconnection board and cable. • Automatic temperature control.
02	***Compilation Manual***	c	Major update for V6_3_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***	d **ZeBu**	Major update for V6_3_0. See detailed History table in the manual.
10	***Smart Z-ICE Manual***	c **ZSE**	New sections: • Relocation. • Remapping for runtime.
11	***Direct ICE Manual***	c	Minor update for V6_3_0.
13	***C API Reference Manual*** ***C++ API Reference Manual***	V6_3_0	
14	**ZEMI-3 Manual**	b **ZeBu**	Last update for V4_3_3.
15	**zFAST Manual**	a	First Edition for V6_2_1.

2.3 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 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*	**d**	For ZW-FPGA V1.2. Added support of **`zFAST`** synthesizer and new ZW-FPGA components.
AN025	**Integrating with ZeBu thread-safe library**	**b**	
AN028	**SystemVerilog Assertion (SVA) for ZeBu**	**a**	
AN029	**zDPI Feature for ZeBu**	**a**	
AN032	**Migrating from ZeBu-XXL to ZeBu-Server**	**a**	

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.

3 Known Limitations

This chapter lists the known limitations in the V6_3_0 release.

3.1 Modified default settings

3.1.1 Database for the zFAST Stat Browser

The database for the **zFAST** stat browser is no longer generated by default, as it used to be in previous releases. To have a similar behavior as in the previous release, the **Build database for Stat Browser** checkbox has to be selected in the **zFAST** panel.

3.1.2 Changes in zDbPostProc/zSelectProbes

- User can no longer save a list of selected signals as a text file. However, user can still load files containing lists of selected signals.
- To select a vector via a command line or via a file containing a list of selected signals, user must not add any range to the name of a vector.
- Some **zDbPostProc/zSelectProbes** commands have been deprecated or replaced by new ones, as described in Section 1.5.3.1 and 1.5.3.2. Scripts written for a previous software version may need to be updated.

3.2 Operating Systems

3.2.1 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.

3.2.2 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.

3.2.3 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.

3.3 No communication with units when a PC is OFF

In a multi-user system configuration (either single- or multi-unit), if one of the host PCs connected to any unit is switched OFF, you may no longer be able to communicate with the units from any PC connected to the system. In such a case, the following error will be thrown:

```
-- ZeBu :  zUtils : Loading U0_BP_FC0 with "/usr/share/zebu/V6_3_0/etc/firmwares/ZSE/
           default/zse_sbp_2s_fc.bit" (using CPU).
.......... LAU0331E : Cannot check if U0_BP_FC0 is done (cannot read status).
-- ZeBu : zUtils : ERROR : FAILED.

-- ZeBu : zUtils : WARNING : Retrying to load U0_BP_FC0 (1/10).
-- ZeBu : zUtils : Loading U0_BP_FC0 with
          "/usr/share/zebu/V6_3_0/etc/firmwares/ZSE/default/zse_sbp_2s_fc.bit" (using CPU).
          LAU0330E : Cannot reset U0_BP_FC0 (cannot check cmd 0x00000002).
-- ZeBu : zUtils : ERROR : FAILED.
```

In most cases, communication resumes as soon as ALL the PCs are switched ON. If not, you must power OFF/ON all units then launch either `zSetupSystem` or `zUtils -initSystem`. For more details, please refer to Sections 7.2 and 8.2 of ***ZeBu-Server Installation Manual*** (Rev. E).

3.4 zCui limitations

3.4.1 Project file compatibility

Once you have saved an existing **zCui** project file (.zpf) with release V6_3_0, you can no longer use this file with previous releases. If you intend to keep your previous version project file for non-regression testing, you should save it with a different name before migrating to Version 6_3_0.

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_3_0 project file.

3.4.2 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.

3.4.3 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 the ***Zebu-Server Compilation Manual*** (Rev. C).

3.4.4 Limitations in the Compilation view

- The status icon is not displayed for the groups of tasks.
- The comment for the **FPGA Place & Route** branch may indicate a wrong value. For example, if a single task passes then the following value could be reported even if some tasks failed:

```
1p, 0f/2 tasks
```

- Some jobs may keep on running forever when the **`zCui`** task engine times out.
- The **`zCui`** task engine may block when an FPGA actually succeeded but **`zCui`** never received the `SIGCHLD` signal from Unix. In such a case, the compilation does not finish correctly and must be manually aborted by user. This FPGA will be displayed as failed.
- At the end of each compilation, the winner FPGA parameters are stored in a file located in the Back-End directory. However this file is re-written after each compilation, leading to the loss of previous winner FPGA parameters after each compilation.

3.4.5 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.
- The **Post-Compilation FPGA Summary** and **Interactive FPGA Summary** are not available. The corresponding icons and menu items have been removed.
- The information displayed in the **FPGA P&R** panel corresponds to the original Place & Route settings but it is not updated with the actual parameter set which was used to pass the Place & Route.

3.5 zFAST limitations

3.5.1 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 3.4.3.
- The top of the design cannot have a Verilog escaped identifier.

3.5.2 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.

3.5.3 Synthesizer inefficient when a variable is used in many places

The synthesizer is inefficient when a variable of a large array or structure type is used in many places (hundreds or thousands after loop unrolling).

3.6 Compilation limitations

3.6.1 Limitations for declarations with RTL paths

- For a VHDL design or a mixed-language design with VHDL, RTL names are not fully supported for the back-end compilation or runtime access, in particular for CSA (Combinational Signals Accessibility) feature.
- Curly brackets (`{` and `}`) are usually not supported by synthesizers. If some are present in the RTL paths in the RTL-based compilation scripts, 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`), pattern matching characters (`*`, `[` and `]`) cannot be present as in RTL paths.
- The RTL database does not support mixed RTL/EDIF paths, for example a mixed path including an RTL path down to the following EDIF macro:

```
probe_signals -wire top.path.rtl.suffix.in.edif.wirename
```

  A new error policy will cause **zFAST** to exit in error when a path ends in an EDIF macro

- When RTL names are used in the DVE file, it is not possible to use the `force assign` command with `–source_dve` option.
- zCore declaration with RTL paths is not supported when synthesizing with **`zFAST`** in top-down mode or with third-party synthesizers. It is only supported when synthesizing with zFAST in block-based mode, as described in Section 1.4.4.
- In the RTL-based compilation script, the `force` command no longer supports `rtlname_input/rtlname_output` options.
- It is not possible to use RTL-based declarations in the DVE file to declare the path to a `generate` block with an escaped ID, as in the following example:

```
top.genblk1.\BLOCK.ESCAPED[0] [0].ins.\OUT.result ,
```

Where `\BLOCK_ESCAPED` is declared as in the following RTL source code example:

```
module bb(CKOUT);
...
module sub (IN, CK, OUT);
...
module top(\IN.0 , \OUT.0 );
input [5:0] \IN.0 ;
output [5:0] \OUT.0 ;

genvar i;
generate begin
for (i = 0; i < 2; ++i) begin : \BLOCK.ESCAPED[0]
wire wckout;
bb bbck(.CKOUT(wckout));
sub ins(.IN(\IN.0 [i+2]), .OUT(\OUT.0 [i+2]), .CK(wckout));
end
end
endgenerate
endmodule
```

3.6.2 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.

3.6.3 Erroneous info in the zCore-level log with `cluster core` command

When the `cluster core` command is declared for the zCore-level compiler, some FPGA resources do not appear correctly in the table summarizing sizes, constraints and resource usage.

3.6.4 Limitations for memory modeling

3.6.4.1 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.

3.6.4.2 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.

3.7 Limitations with the new runtime database

3.7.1 Limitations for co-simulation wrappers

In the present software release, it is not possible to generate the following co-simulation wrappers with **`zDbPostProc`**/**`zSelectProbes`**:

- SystemC wrappers.
- HDL wrappers with `-bb` option (presence of a blackbox).

The limitation for the SystemC is also applicable when the corresponding checkbox is selected in the **Wrapper** tab in **`zCui`**.

3.7.2 Limitation for the hierarchical separator in `zSelectProbes`

If the hierarchical separator of the runtime database has been changed (to "|" or "/" for example) with the `separator` command of **`zDbPostProc`**/**`zSelectProbes`**, the default separator "." may still prevail in the **`zSelectProbes`** display.

3.8 Runtime limitations

3.8.1 C++ Cosim `run` method deprecated in non-blocking mode

Section 4.7.1 of the ***ZeBu C++ Co-Simulation Manual*** (*C_COSIM driver for Cycle-Based Testbench*) describes the execution of cycles using the `CDriver::run` method in non-blocking mode:

```
unsigned int CDriver::run (unsigned int numCycles, bool block = false)
```

The `block` parameter is now ignored by the ZeBu runtime software. Therefore the behavior of the `run` method is always in blocking mode. If for some reason (for ex. a trigger event) the specified `numCycles` cannot be executed, the C++ testbench will hang in the `run` method.

To get around this inconvenience, use the `CDriver::wait` method instead:

```
unsigned int CDriver::wait(unsigned int trigger_mask, unsigned int
numCycles=0xffffffff);
```

Where each of the 16 `trigger_mask` LSBs must be set to `1` to stop the run if the corresponding trigger value is `1`. If no trigger is fired, then the specified number of cycles will be executed.

3.8.2 Restrictions when dumping wavefiles using the `WafeFile` class

Wavefiles can be generated for internal signals synchronized with a controlled clock or with `driverClk` by creating an object in the `WaveFile` class.

Signals can be added or removed from the wavefile as required, without affecting the DUT and the hardware part of transactors. The resulting waveform file contains all dynamic probes selected via **`zSelectProbes`** and **`zDbPostProc`** (see *Accessing Dynamic Probes from the C++ Testbench* in the ***ZeBu C++ Co-Simulation Manual***) or a set of dynamic probes explicitly added into the wavefile at runtime. The drawback is the cost in speed. However, this feature can be enabled/disabled (`dumpon`/ `dumpoff` methods) to dump only during a simulation time-window where the data is critical for debugging.

However, using the `WaveFile` class imposes the following restrictions:

- This class is only supported with the threadsafe versions of `libZebu.so`: `libZebuThreadSafe.so` and `libZebuThreadSafeDebug.so`.
- **`zRun`** is not supported while a file is open.
- You can only open one file at a time.
- You cannot access signals, memories and clocks while a file is open.
- You cannot save the hardware state (with `saveHardwareState` method), if the `dumpoff` method was not called before. Else, a deadlock will occur in the `saveHardwareState` method.
- You must close or destroy the file before closing ZeBu, else a deadlock occurs.

`WaveFile.hh` header file describes the `WaveFile` class. Its interface is similar to the `Board` and `Driver` classes used to dump interface and internal signals in C++ co-simulation.

3.8.3 Limitation for selection/deselection of dynamic probes

Selection and deselection of a dynamic probe are not supported when a `.bin` dump 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` dump 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.

3.8.4 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.

3.8.5 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.

3.8.6 Clock declaration for HDL co-simulation

For HDL co-simulation, the clocks controlled by the HDL testbench must be declared in separate groups (one clock per group) in the `designFeatures` file. Additional design clocks which are not controlled from the HDL testbench can be grouped.

3.8.7 Limitations for threadsafe environment

- The `Z_REPEAT_CALLBACK` is only available when emulating in a threadsafe environment (when the testbench is linked with `libZebuThreadsafe.so` library).
- When running C/C++ co-simulation in a threadsafe environment with **`zRun`**, dumping a waveform file from dynamic probes with **`zRun`** becomes very slow.

3.8.8 Limitation on fast hardware state

A fast hardware state object cannot be initialized before board initialization.

3.8.9 Limitation on clock specification for restore

`false` is returned when the clock parameters are declared in the C/C++ testbench **after initialization**. However the same value is erroneously returned 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).

3.9 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.
```

3.10 Restrictions on Write operations, namely with RLDRAM memory models

3.10.1 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 enabled 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 3.9.

3.10.2 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, select **BRAM Instrumentation for Read&Write**.

3.11 Advanced triggers

Advanced triggers are not supported in the current version.

3.12 ZEMI-3 limitations

- The XST synthesizer is not supported for ZEMI-3 transactors.
- A ZEMI-3 transactor cannot be called with a name from the ZEMI-3 public API, such as `reset`, `terminate` or `disconnect`. This API is described in the `$ZEBU_ROOT/include/ZEMI3Handler.hh` header file.

4 Fixed Issues

The following issues are now fixed, either in an intermediate patch for V6_2_1B or more recently for the present software release.

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

4.1 zFAST Synthesis

4.1.1 Synthesis restarted when attribute changes

With V6_2_1B software, if the **Additional zFAST Attribute File** was modified, the synthesis was not relaunched automatically.
With the present software version, the modifications in the attributes are automatically checked in order to relaunch elaboration and synthesis only for the affected bundles.

4.1.2 Memory Modeling Attributes

When a **`zFAST`** attribute for memory modeling has a `<memory_paths_list>` argument, this can be a list with either hierarchical paths to the memories from the top of the design or `<module>.<array>`.
This is correctly described in the ***zFAST Manual*** (Rev. A) but the software did not support correctly the hierarchical paths arguments in previous software versions (reported in RT#22847).

4.2 Compilation

This release fixes in particular the following compilation issues:

- When a symbolic link to a source file was declared in the **Add Source File** instead of the source file, the **Initial Check** step in **`zCui`** failed because the actual path to the source file was not correctly resolved, reported in RT#24808. The declaration of source files with symbolic links is now correctly supported.
- Some signal names in the DUT caused errors when **Filter Glitch** was selected in **`zCui`** (default), reported in RT#24595. The zCore-level compiler (**Build Core xxx** task in **`zCui`**) failed with the following error message:

```
# step FILTER GLITCHES SYNCHRONOUS : adding filters in design
### internal error [LST0070E]: Instance zebu_mux_sel_mode_regM107L[0][7]
already exist in module 'ms_4'.
### internal error [LST0070E] : znl_Module.cc, line 196 : CreateInstance
```

- Some inter-zCores accessibility resources, added by synthesis or because of `probe_signals` commands, were not processed correctly by the system-level compiler and caused an error during System P&R, reported in RT#24147:

```
### fatal error in CHECK [KPR0418F] : In design module U0_M2, wire a.b.c is not driven
### fatal error in INIT [KPR0303F] : 1 error found during initialization of the
database
```

- In the zCore-level compiler, it is now possible to declare the options for the `cluster enable` and `cluster disable` commands with a dash '-' (similar syntax as in the system-level compiler).

4.3 Runtime

This release fixes in particular the following runtime issues:

- When several groups of flexible probes were dumped in the same `.fsdb` file, the lists of signals in the different groups were not correctly merged and no value was present in the generated `.fsdb` file for signals that were present in several groups, reported in RT#24489.
- The ZeBu clock generator is now more tolerant than it was in the previous releases, reported in RT#24724.
  As of this release, the ZeBu clock generator applies the values programmed by the user, provided that they fall within the following ranges:
  - `zClockFilterTime` + 10ns ≤ `zClockSkewTime`
  - `zClockSkewTime` + 10ns ≤ `driverClk` frequency

5 Version Compatibility

This chapter includes information which is applicable for Version 6_3_0.

5.1 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 release does not support 32-bit PCs).

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

- Compilation:
  - Dell PowerEdge 1950 (RHEL 4.5)
  - Dell PowerEdge 2950 (RHEL 4.5)
  - Dell PowerEdge R410 (RHEL 4.5)
- Emulation Runtime:
  - Dell Optiplex 760 (RHEL 4.7)
  - Dell Optiplex 760 (RHEL 4.8)
  - Dell Optiplex 760 (SUSE 10)
  - Dell Precision T5500 (RHEL 5.4)
  - HP xw6200 (RHEL 4.6)
  - HP z400 (RHEL 5.4)

5.2 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 3.2.2.
- SUSE Linux Enterprise Server 10, kernel 2.6 (for emulation runtime only). See specific limitations in Section 3.2.1.

5.3 Hardware compatibility

- This release is compatible with ZeBu-Server only.
- It is not compatible with ZeBu-XL, ZeBu-XXL, ZeBu-UF, and ZeBu-Personal.

5.4 Software compatibility

ZeBu software V6_3_0 is NOT compatible with previous ZeBu releases. If you need to upgrade from a previous release, then make sure you follow the steps below:

- Generate a new target hardware configuration file with the most recent diagnostics patches.
- Recompile your design from scratch with V6_3_0: in **`zCui`**, set the compilation **Target** to **Project**, launch a **Clean** operation () then a **Compile** operation ().
- Before proceeding with emulation runtime, make sure you do the following:
  - For each PC, update the PCIe interface board: launch **`zUpdate`** (no option, no specific settings).
  - Reboot the PCs with `memmap`.
  - Launch **`zInstall`** from the V6_3_0 environment with the `-memmap` option in order to allocate the required system memory for ZeBu ↔ PC communication, as described in Section 6.2.4 of the ***ZeBu-Server Installation Manual*** (Rev. E).

5.5 ZeBu licenses

License features are the same as for ZeBu-Server version 6_2_1B. Licenses which were used on ZeBu-XXL/UF/Personal cannot be used on ZeBu-Server.

Starting with the present software release, the following features require specific licenses:

- Performance Oriented Partitioning (POP), as described in the ***ZeBu-Server Compilation Manual*** (Rev. C).

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.

5.6 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	2010.09-SP1	RHEL 4.5/5.4 SUSE 10	FPGA synthesizer. http://www.synopsys.com
XST	11.4i	RHEL 4.5/5.4	FPGA Synthesizer. http://www.xilinx.com
VCS	mx-2009.06	RHEL 4.6/4.7/4.8/ 5.4 SUSE10	HDL Simulator. http://www.synopsys.com
ModelSim	6.6b	RHEL 4.6/4.7/4.8/ 5.4 SUSE10	HDL Simulator. http://www.mentor.com
NC-Sim	Not Tested		HDL Simulator. NC-Sim should work since ZeBu uses the standard PLI. http://www.cadence.com
SystemC	2.2.0	RHEL 4.6/4.7/4.8/ 5.4 SUSE10	Required for SystemC co-simulation. http://www.systemc.org
GTKWave	3.1.11	RHEL 4.5	Waveform viewer. http://gtkwave.sourceforge.net Included in ZeBu software package.
Debussy	2009.10	RHEL 4.6/4.7/4.8/ 5.4 SUSE10	Interface for design debug and waveform viewing. http://www.springsoft.com/
gcc	3.4.6	RHEL 4	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 SUSE 10	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	11.4i_patched	RHEL 4.5	Xilinx tools for FPGA Place & Route during ZeBu compilation. http://www.xilinx.com Included in ZeBu software package.
RTLvision PRO GateVision PRO	5.0.0	RHEL 4.5	RTL and gate-level netlist analyzers. http://www.concept.de Included in ZeBu software package.

OS compatibility may change. Please visit third-party websites for information.

5.1 Diagnostics patches

Diagnostics patches are specific to each release. They are used by the diagnostics tool, **`zUtils`**, to check your system once installed. To reduce the size of the ZeBu software package, diagnostics patches are not included and should be downloaded as shown in Section 4.5.3 of the ***ZeBu-Server Installation Manual*** (Rev. E).

ZeBu-Server System Type	Available Diagnostics Patches
2-slot Single-unit	zebu_V6_3_0.patch_0.diags_202_43_43.V40.sh.gz zebu_V6_3_0.patch_0.diags_202_91_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_202_91_43.V40.sh.gz zebu_V6_3_0.patch_0.diags_202_91_91.V40.sh.gz zebu_V6_3_0.patch_0.diags_202_93_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_202_93_43.V40.sh.gz zebu_V6_3_0.patch_0.diags_202_93_93.V40.sh.gz zebu_V6_3_0.patch_0.diags_202_161_161.V40.sh.gz
5-slot Single-unit	zebu_V6_3_0.patch_0.diags_501_43_161_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_43_43_43_02_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_43_43_43_43_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_43_43_43_43_43.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_91_161_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_91_43_43_43_43.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_91_91_91_02_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_91_91_91_91_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_91_91_91_91_91.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_161_02_02_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_161_161_02_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_161_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_43_43_43_43.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_93_161_02_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_93_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_93_93_02_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_93_93_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_93_93_93_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_93_93_93_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_93_93_93_93_93.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_161_161_161_02_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_161_161_161_161_02.V40.sh.gz zebu_V6_3_0.patch_0.diags_501_161_161_161_161_161.V40.sh.gz
5-slot Multi-unit (1-unit config)	zebu_V6_3_0.patch_0.diags_C00-00-512_43_43_43_43_43.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-00-512_91_161_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-00-512_91_91_91_91_91.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-00-512_93_161_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-00-512_93_93_93_93_93.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-00-512_161_161_161_161_161.V40.sh.gz
5-slot Multi-unit (2-unit config)	zebu_V6_3_0.patch_0.diags_C00-02 -512_91_161_161_161_161 -512_161_161_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-02 -512_93_93_93_93_93 -512_93_93_93_93_93.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-02 -512_161_161_161_161_161 -512_161_161_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-02 -512_161_161_161_161_161 -512_91_161_161_161_161.V40.sh.gz

ZeBu-Server System Type	Available Diagnostics Patches
5-slot Multi-unit (3-unit config)	zebu_V6_3_0.patch_0.diags_C00-02 -512_93_93_93_93_93 -512_93_93_93_93_93 -512_93_93_93_93_93.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-02 -512_161_161_161_161_161 -512_161_161_161_161_161 -512_161_161_161_161_161.V40.sh.gz
5-slot Multi-unit (4-unit config)	zebu_V6_3_0.patch_0.diags_C00-02 -512_93_93_93_93_93 -512_93_93_93_93_93 -512_93_93_93_93_93 -512_93_93_93_93_93.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-02 -512_161_161_161_161_161 -512_161_161_161_161_161 -512_161_161_161_161_161 -512_161_161_161_161_161.V40.sh.gz
5-slot Multi-unit (5-unit config)	zebu_V6_3_0.patch_0.diags_C00-02 -512_93_93_93_93_93 -512_93_93_93_93_93 -512_93_93_93_93_93 -512_93_93_93_93_93 -512_93_93_93_93_93.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-02 -512_93_161_161_161_161 -512_93_161_161_161_161 -512_93_161_161_161_161 -512_93_161_161_161_161 -512_93_161_161_161_161.V40.sh.gz zebu_V6_3_0.patch_0.diags_C00-02 -512_161_161_161_161_161 -512_161_161_161_161_161 -512_161_161_161_161_161 -512_161_161_161_161_161 -512_161_161_161_161_161.V40.sh.gz

5.2 EVE Vertical Solutions compatibility

The following sections list which versions of ZeBu transactors and ZeBu memory IPs are applicable with the present software version.

5.2.1 ZeBu transactors

The V6_3_0 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.

Doc Id	Transactor/Utility	Version (64-bit)
VSXTOR002	UART	2.3
VSXTOR003-2	JTAG XTENSA	1.7
VSXTOR003-3	JTAG + TAP controller API	1.3
VSXTOR004	SRAM SW	2.0
VSXTOR005	PCI Express	5.0
	PCIe Viewer	3.0
VSXTOR006	Ethernet (GMII)	2.1
	`etherPlug` utility	1.3
VSXTOR007	I2S	1.3
VSXTOR008	Digital Video	3.2
VSXTOR009	IEEE 1394 FireWire	1.2
VSXTOR010	MMC Device	1.2
VSXTOR011	AHB Master	2.0
VSXTOR012	Streaming Toolkit	1.7
VSXTOR013	USB	1.2
VSXTOR014	AXI Master + Slave	2.3
VSXTOR015	I2C	2.0
VSXTOR016	MMC Host	1.1
VSXTOR017	HDMI Sink	1.1
VSXTOR019	Video In	1.5
VSXTOR020	Ethernet Controller	1.1
VSXTOR021	ZDDRx (DDR2, DDR3)	1.0
VSXTOR022	TLM2 Adapter	1.1
VSXTOR023	KMI	2.0
VSXTOR024	SDIO Device	1.0

5.2.2 ZeBu memory IPs

ZeBu Memory IPs are all supported by V6_3_0 software (with specific licenses).

Doc Id	Memory IP	Version
VSIP001	ZDDR	3.3
VSIP002	ZDDR2	3.2
VSIP003	ZSDRAM	3.2
VSIP004	NAND Flash	1.4
VSIP005	ZMOBILEDDR	2.2
VSIP007	ZGDDR5	2.1
VSIP008	ZDDR_SP	1.2
VSIP009	ZDDR2_SP	1.4
VSIP010	ZDDR3_SP	1.2
VSIP011	ZMOBILEDDR_SP	1.4
VSIP012	ZLPDDR2_SP	1.3
VSIP013	ZLPDDR2_NVM	1.1

6 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