Static Timing Analysis for ZeBu

AN017

Revision c

ZeBu Application Note

June 2011

Purpose: This document explains how to perform timing analysis for a design mapped on ZeBu.

Applicability: This document is applicable for ZeBu-Server V6_3_1.

History: This table gives information about the content of each revision of this document, with indication of specific applicable version:

Doc Rev.	Product Version	Date	Evolution
c	V6_3_1	June 11	Updated sections: - **Static Timing Analysis** option in **`zCui`** set by default to **Full** (§2.1). - New default **`zTime`** script (§2.2). - `zFilterTime` recommended value added in the `des_zTime.xref` file generated via **Save Timing Parameters for Runtime** (§2.3). - `report_out_paths` description replaces `report_paths` (§3.3.1.1). - `report_out_clock_paths` description replaces `report_clock_paths` (§3.3.1.2). - `report_out_routing_paths` description replaces `report_routing_paths` (§3.3.1.3). Removed section: - Commands for input-oriented analysis (was §3.3.3 in rev B).
b	V4_3_3	Dec 08	First edition.
a	V4_3_1	Feb 08	Preliminary.

Table of Contents

1 Introduction

This document describes the Static Timing Analysis feature which is supported by the ZeBu-Server compilation.

Static Timing Analysis is performed after system-level Place and Route in order to estimate speed limitations (achievable `driverClock` frequency), a clock skew value (`zClockSkewTime`), and a glitch filtering value (`zFilterTime`) for runtime.

In order to quickly get the estimated runtime parameters, **`zCui`** launches an output-oriented analysis which gives quickly the longest path to the design outputs by analyzing the paths in the design from the outputs.

Once your design is functional, you can optimize the clustering solution by identifying ALL the longest paths which go through from the inputs of the design. This input-oriented analysis may result in erroneous recommendations for the runtime parameters because too many paths are explored. This is an advanced **`zTime`** usage, which is possible in **`zCui`** with additional commands (these input-oriented commands are mentioned in this Application Note wherever the equivalent output-oriented command is described).

In both cases, the Static Timing Analysis process takes into account the data rate factor for inter-FPGA communications and the achievable frequency of the memory models generated by **`zMem`** (BRAM-based and zrm-based memory models).

The values estimated by the Static Timing Analysis feature for `driverClock`, `zClockSkewTime`, and `zFilterTime` are always pessimistic and they can be optimized if the user has a good knowledge of the design. Identifying the paths that are erroneously considered as clock paths, also known as false paths, improves the accuracy of the estimated runtime parameters.

Some reports are issued to investigate the performance bottleneck with statistical information on the longest paths in the design.

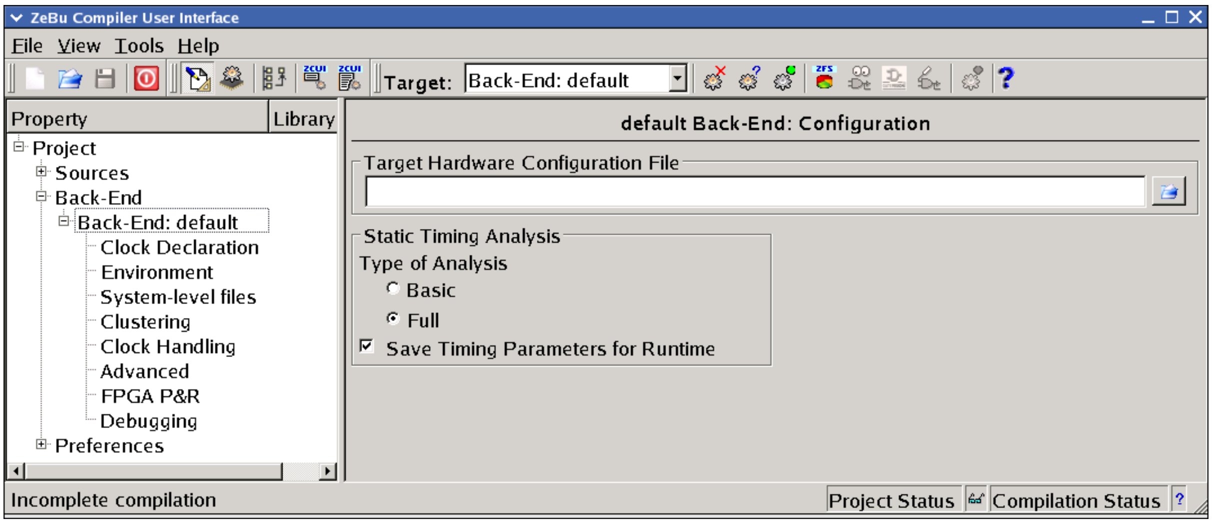
2 Using Static Timing Analysis in zCui

2.1 Configuring the Static Timing Analysis feature in zCui

The Static Timing Analysis feature is activated by default in **zCui** and can be configured in the back-end compilation.

In the **Back-end Configuration** panel of the **Project** workspace, **Static Timing Analysis** is set by default to **Full.** This takes into account combinational clock paths and asynchronous paths in the design, and may require false path declarations to obtain relevant results.

The **Save Timing Parameters for Runtime** box is selected by default in order to generate a `des_zTime.xref` file which includes the recommended values for the `driverClock` frequency, `zClockSkewTime`, and `zFilterTime`; these values will be used automatically at runtime.


ZeBu Compiler User Interface

File View Tools Help

Target: Back-End: default

Property | Library

- Project
  - Sources
  - Back-End
    - Back-End: default
      - Clock Declaration
      - Environment
      - System-level files
      - Clustering
      - Clock Handling
      - Advanced
      - FPGA P&R
      - Debugging
  - Preferences

default Back-End: Configuration

Target Hardware Configuration File

Static Timing Analysis
Type of Analysis
Basic [✓ Full]
[✓] Save Timing Parameters for Runtime

Incomplete compilation | Project Status | Compilation Status

Figure 1: ***Back-End Configuration*** **panel in zCui**

Choosing **Basic** for the **Type of Analysis** option provides user with an estimation of `driverClock`, `zClockSkewTime`, and `zFilterTime` for a functional emulation.

If you use several back-ends in your project, the Static Timing Analysis settings can be modified separately for each back-end.

Further settings can be declared in the **Timing Analysis (zTime)** frame of the **Advanced** panel (one for each back-end):

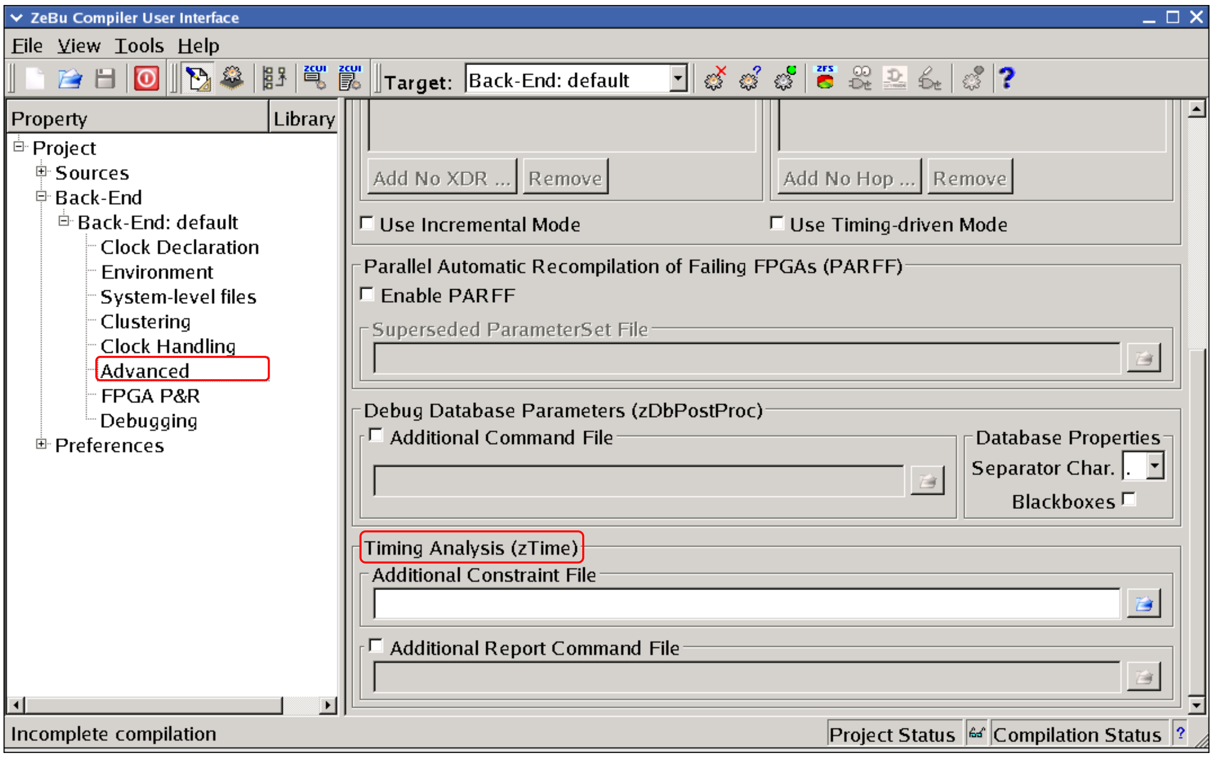
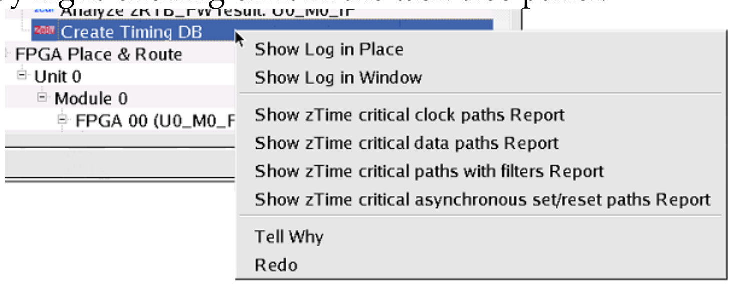


Figure 2: ***Advanced*** **panel in `zCui`**

You can declare an **Additional Constraint File** to add specific commands for the Static Timing Analysis process, in particular to declare the false paths of your design, as described in Section 3.2.

You can also declare an **Additional Report Command File** in which you add specific commands to get appropriate reports, as described in Section 3.3.1.

In the **Compilation** workspace, the task tree item which corresponds to static timing analysis is called **Create Timing DB**. Once this step is finished, you can check the log and report files by right-clicking on it in the task tree panel:


Analyze zRTB_FW result: U0_M0_IF
Create Timing DB
FPGA Place & Route
Unit 0
Module 0
FPGA 00 (U0_M0_F
Show Log in Place
Show Log in Window
Show zTime critical clock paths Report
Show zTime critical data paths Report
Show zTime critical paths with filters Report
Show zTime critical asynchronous set/reset paths Report
Tell Why
Redo

By default, **`zCui`** automatically saves estimated values of `driverClock`, `zClockSkewTime`, and `zFilterTime` in the `des_zTime.xref` file which will be automatically used at runtime by **`zServer`**. If the estimated values are not appropriate for ZeBu, **`zTime`** displays an error message and does not save the parameters for runtime.

2.2 Static Timing Analysis settings in `zCui`

This section includes the default **`zTime`** script used by **`zCui`**. It is based on commands which are detailed in Chapter 3 of this manual. The Static Timing Analysis process can be modified by declaring an **Additional Constraint File** and the generated reports can be modified with an **Additional Report Command File.**

The default **`zTime`** script in **`zCui`** starts with a clock path analysis in order to compute an estimation of the `zClockSkewTime` and `zFilterTime` values that will be used for delay insertion. This value is used by the next Static Timing Analysis steps to estimate the achievable `driverClock` frequency.

```
# Timing analysis of the Clock network to estimate zClockSkewTime
# parameter (maximum clock skew)
set_false_path_context clock
analyze

# If Glitch filtering mode has been selected in zCui
# We should compute the worst Filter paths (zFilterTime parameter)
if { [get_fgs_mode -check] } {

# Backward exploration from filter outputs to inputs in the clock network
# to identify longuest filter paths
  build_out_filter_paths -async=true

# Display a report with all longuest filter paths found during backward
# exploration
  report_out_filter_paths -all
}

# Dump resulting filter paths in HTML format
drive_out_filter_paths

# Backward exploration from outputs to inputs in the clock network
# to identify longuest clock paths given previously computed zFilterTime
build_out_clock_paths -async=true -zfilter=true

# Dump resulting clock paths in HTML format
drive_out_clock_paths

# Dump all wire names part of the clock network
drive_clock_nets

# Display a report with all longuest clock paths found during backward
# exploration
report_out_clock_paths –all

# Timing analysis of the routing network to estimate maximum Driver Clock
# frequency
set_false_path_context routing
analyze

# Estimation of the memories (zRM) frequency
build_memory_paths
report_memory -all
```

```
# Backward exploration from outputs to inputs in the routing network
# to identify longuest data paths taken into account zdelay set to
# previously estimated zClockSkewTime
build_out_routing_paths -async=true -zdelay=true

# Dump resulting paths in HTML format
drive_out_paths

# Display a report with longuest data paths found for each output
report_out_routing_paths –all

# Save estimated zClockSkewTime and Driver Clock frequency parameters for
# runtime
save_run_param B0/des_ztime.xref
```

2.3 Analysis of the Reports

During the **`zTime`** process, the log window of **`zCui`** displays information on each analysis phase. The end of the log shows a summary of the relevant information processed by **`zTime`**:

- Longest inter-FPGA filter path delay, corresponding to `zFilterTime` parameter when **Glitch Filtering** is selected (**Clock Handling** panel of **`zCui`**).
- Longest inter-FPGA clock path delay, corresponding to `zClockSkewTime` parameter (maximum clock skew).
- Critical routing path delay.
- Longest memory period.
- Theoretical frequency using default settings and ignoring clock skew.

Following is a **`zTime`** log example as displayed in **`zCui`** (the commands displayed in italic are stored in the **`zTime`** log file but they are not displayed in **`zCui`**):

```
set_false_path_context clock
#   step ANALYSIS : Set false path context to clock
analyze
#   step ANALYSIS : Starting
#   step ANALYSIS : Initialize timing analyzer
#   step ANALYSIS : Identify primary inputs and outputs
#   step ANALYSIS : Identify false paths
#   step ANALYSIS : Found 7 false timing arc(s) in clock context
#   step ANALYSIS : Found 39 inputs and 39 outputs
#   step ANALYSIS : Search combinatory loops
#   step ANALYSIS : Found 0 combinatory loop(s)
#   step ANALYSIS : Build topological order
#   step ANALYSIS : The maximum topological depth is 9
#   step ANALYSIS : Done in user(0ms), sys(0ms), vm(81m)
build_out_clock_paths -async=true
#   step Build output paths : Starting
#   step ANALYSIS : Find worst path for each output clock ports of the board
#   step Build output paths : Done in user(0ms), sys(0ms), vm(81m)
#   step REPORT : #-------------------------------------------------------------
#   step REPORT :   Longest inter-fpga clock path delay (clock skew) is : 66 ns
#   step REPORT : #-------------------------------------------------------------
drive_out_clock_paths
#   step DRIVE PATHS : Drive all paths in file 'clock_out_paths.dump.gz' (DUMP format)
drive_clock_nets
#   step ANALYSIS : Find all inter-fpga clock nets of the board
#   step ANALYSIS : 2 clock port(s) found
#   step DRIVE NETS : Drive all clock nets in file 'clock_nets.dump'

[...]
```

```
set_false_path_context routing
#   step ANALYSIS : Set false path context to routing
analyze
#   step ANALYSIS : Starting
#   step ANALYSIS : Initialize timing analyzer
#   step ANALYSIS : Identify primary inputs and outputs
#   step ANALYSIS : Identify false paths
#   step ANALYSIS : Found 0 false timing arc(s) in routing context
#   step ANALYSIS : Found 36 inputs and 37 outputs
#   step ANALYSIS : Search combinatory loops
#   step ANALYSIS : Found 0 combinatory loop(s)
#   step ANALYSIS : Build topological order
#   step ANALYSIS : The maximum topological depth is 9
#   step ANALYSIS : Done in user(0ms), sys(0ms), vm(81m)
build_memory_paths
#   step ANALYSIS : Propagate delays through the timing graph
#   step ANALYSIS : Find all memories of the board
#   step ANALYSIS : Final number of memories explored: 2
#   step ANALYSIS : Sort those 2 memories by arrival time
#   step REPORT : #-------------------------------------------------------------
#   step REPORT :   Longest memory period is : 59 ns
#   step REPORT : #-------------------------------------------------------------
report_memory -all
#   step REPORT : Report memory statistics
#   step REPORT : Memory report
#   step REPORT : #-----------------+---------+--------+--------------------------
#   step REPORT : # Physical Memory | Delay   |  Type  | Logical Memory Name(s)
#   step REPORT : #-----------------+---------+--------+--------------------------
#   step REPORT : | F0_0_0_bank_0   |   59 ns |  ssram | zrm_sram_1d_4x32_1
#   step REPORT : | F0_0_1_bank_1   |   59 ns |  ssram | zrm_sram_1d_4x32
#   step REPORT : #-----------------+---------+--------+--------------------------
#   step REPORT : A total of 2 memory displayed
build_out_routing_paths -async=true -zdelay=true
#   step Build output paths : Starting
#   step ANALYSIS : Find worst path for each output ports of the board
#   step ANALYSIS : Given a zdelay (clock skew) of 66 ns
#   step Build output paths : Done in user(0ms), sys(0ms), vm(81m)
#   step REPORT : #-------------------------------------------------------------
#   step REPORT :   Critical routing path delay : 123 ns
#   step REPORT :   . Constant part    : 68 ns
#   step REPORT :   . Multiplexed part : 55 ns
#   step REPORT :   The theoretical frequency using default settings is 8106 Khz
#   step REPORT : #-------------------------------------------------------------

[...]

save_run_param B0/des_ztime.xref
#   step SAVE RUN PARAM : Save max frequency and clock skew time in file 'B0/des_ztime.xref'
```

3 Commands for Advanced Usage

`zTime` Tcl commands are intended for use in an **Additional Constraint File** or an **Additional Report File** declared in **`zCui`**.

Some commands listed in this chapter must be used with care in an additional script in **`zCui`** because they may cause malfunctions in the Static Timing Analysis process and generate inaccurate parameters for runtime.

`zTime` commands are case sensitive but their options are not. **`zTime`** supports Unix-like wildcards as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13.1.

3.1 Getting Help on `zTime` Commands

You can list all the **`zTime`** commands as follows:

```
$ zTime -help
```

Note that the commands which can be used in an additional script in **`zCui`** are marked with a `(z)`.

For each command, you can also display the possible options:

```
$ zTime –help <command-name>
```

3.2 Commands for the Additional Constraints File

The **Additional Constraints File** which can be declared in **`zCui`** **Advanced** tab is intended for declaration of the false paths.

A False Path is a path in the design that should not be considered for timing analysis because it is not functional or it is not used during emulation (typically a path starting from a reset or from a scan test port).

`zTime` offers two different ways to describe false paths:

- All paths coming from a port (or from a set of ports) are to be considered as false paths by **`zTime`**: `set_false_path_from` command.
- All paths ending at a port (or set of ports) are to be considered as false paths by **`zTime`**: `set_false_path_to` command.

3.2.1 Definitions

- Input: output port of any design FPGA or IF FPGA.
- Primary Input: input port of the design, physically connected to the timing graph as an output port of IF FPGA.
- Output: input port of any design FPGA or IF FPGA.
- Primary Output: output port of the design, physically connected to the timing graph as an input port of IF FPGA.

- Primary Port: can be primary input or primary output.
- Timing arc: a direct connection between two FPGA ports.
- Path: goes from one input or primary input to one output or primary output.

3.2.2 Declaring the Context of False Paths

Before declaring the false paths in your script, you have to define for which step of the timing analysis the consecutive constraints will be taken into account:

- When the context is set to `clock`, the consecutive false paths can be used only when analyzing clock paths (`build_out_clock_paths` and `build_clock_paths`). They are not used for routing path analysis.
- When the context is set to `routing`, the consecutive false paths can be used only when analyzing routing path (`build_out_routing_paths` and `build_routing_paths`). They are not used for clock path analysis.
- When the context is set to `all`, the consecutive false paths are used for both clock path analysis and routing path analysis (any `build*` command).

```
set_false_path_context [-clock] [-routing] [-all]
```

Note that the context is applicable for all the listed false paths until the next `set_false_path_context` command is issued. All the constraints will be used by the next appropriate `build_*` command for the considered context. In script mode, you need to check that the false path declarations are positioned for the appropriate context and before the corresponding `build*` command.

3.2.3 Declaring False Paths

3.2.3.1 `set_false_path_from`

This command specifies false path constraints from a given node context: all paths coming from a port (or from a set of ports) should be considered as false paths.

```
set_false_path_from [-ins=<Ins_name>] [-port=<Port_name>]   \
[-match_ins=<Pattern_ins>] [-match_port=<Pattern_port>]     \
[-match_wire=<Pattern_wire>] [-match_alias=<Pattern_alias>] \
[-match=<Pattern>] [-verbose=true|false] [-case_sens= true|false]
```

If it is not a primary port then the `-ins` option should be used to specify the FPGA to which the port belongs.

Ports and instances can be declared explicitly (with `-port` or `-ins` option) or using wildcard characters (with `-match_ins`, `-match_port`, `-match`, `-match_wire`, and `-match_alias`). Unix-like wildcards are supported.

Alternatively, you can use the `-match` option to specify a match pattern like for `<Pattern_ins>` and `<Pattern_port>`.

Option	Description
`-port`	FPGA port name.
`-ins`	Name of FPGA instance in EDIF netlist generated during zCore-level compilation. (Optional).
`-match_ins`	Matching pattern for FPGA instance name.
`-match_port`	Matching pattern for port name.
`-match`	Matching pattern using the form `<Pattern_ins>` or `<Pattern_port>`.
`-match_wire`	Matching wire name.
`-match_alias`	Matching alias name.
`-verbose`	Displays the name of the false paths actually identified when `-verbose=true` (same form as `view_false_paths` command). Default is `false`.
`-case_sens`	Matching pattern is case sensitive when `-case_sens=true`.

Example:

The following command adds all the ports and FPGAs which have `reset` in their name to the list of false paths:

```
set_false_path_from –match=*reset*
```

3.2.3.2 `set_false_path_to`

This command specifies false path constraints: all paths ending at a port (or a set of ports) should be considered as false paths.

```
set_false_path_to [-ins=<Ins_name>] [-port=<Port_name>]     \
[-match_ins=<Pattern_ins>] [-match_port=<Pattern_port>]     \
[-match_wire=<Pattern_wire>] [-match_alias=<Pattern_alias>] \
[-match=<Pattern>] [-verbose=true|false] [-case_sens= true|false]
```

If it is not a primary port then the `-ins` option should be used to specify the FPGA to which the port belongs.

Ports and instances can be declared explicitly (with `-port` or `-ins` option) or using wildcard characters (with `-match_ins`, `-match_port`, `-match`, `-match_wire`, and `-match_alias`). Unix-like wildcards are supported.

Alternatively, you can use the `-match` option to specify a match pattern like for `<Pattern_ins>` and `<Pattern_port>`.

Note: Same option descriptions as in Section 3.2.3.1.

Example:
The following command adds all the ports and FPGAs which have `reset` in their name to the list of false paths:

```
set_false_path_from –match=*reset*
```

3.2.3.3 `set_false_arc`

This command specifies false path constraints: all paths using a timing arc (direct connection between two FPGA ports) should be considered as false paths.

```
set_false_arc [-from_ins=<Ins_name>] [-from_port=<Port_name>]     \
[-from_match_ins=<Pattern_ins>] [-from_match_port=<Pattern_port>] \
[-from_match=<Pattern>]                                           \
[-to_ins=<Ins_name>] [-to_port=<Port_name>]                       \
[-to_match_ins=<Pattern_ins>] [-to_match_port=<Pattern_port>]     \
[-to_match=<Pattern>] \
[-verbose=true|false]
```

If the ports are not primary ports, then the `-from_ins` and/or `-to_ins` arguments should be used to specify the FPGA to which they belong.
Ports and instances can be declared explicitly (with `-port` or `-ins` options) or using wildcard characters (with `-from_match_*` and `-to_match_*` options).
Alternatively you can use the `-from_match` and/or `-to_match` options to specify a match pattern like `Pattern_ins` or `Pattern_port`.

Example:
The following command marks all arcs in `F0_0_0` starting from any port matching `*reset*` to any port matching `*data_out*` as false arcs:

```
set_false_arc -from_match=F0_0_0/*reset* -to_match=F0_0_0/*data_out
```

3.2.4 Displaying False Paths

In order to check the actual false paths considered by **`zTime`** when the declaration has been done with wildcards, you can dump the information in the **`zTime`** log file with the `view_false_paths` command:

```
view_false_paths
```

Example:
```
set_false_path_from -match_port=reset*
view_false_paths
```

```
#   step FALSE PATH : Port 'resetn' has been added to false_paths_from set
[...]
#   step ANALYSIS : Display false paths constraints
#   step ANALYSIS : False path from: resetn
#   step ANALYSIS : 1 false path(s) constraints displayed
```

3.3 Commands for the Additional Report Command File

The **Additional Report Command File** is intended to modify the types of reports which are generated by **`zTime`** and to configure the saving of Critical Paths and Clock Paths. It is processed after the **`zCui`** script for **`zTime`**.

This **Additional Report Command File** can include specific analysis commands which must not be overridden by the **`zCui`** script for **`zTime`**. Such commands are typically used for output-oriented analysis with specific parameters or for input-oriented analysis when working on performance optimization.

If you place specific analysis commands in the **Additional Constraints File**, they will be processed before the **`zCui`** script for **`zTime`** and their result may be overwritten by the output-oriented analysis which is launched by **`zCui`**.

3.3.1 Modifying the Settings for Reports

When the default timing analysis reports generated with **`zCui`** do not match your requirements (see Section 2.2), you can modify the settings by declaring some commands in a script added in the **Additional Report Command File** field.

3.3.1.1 `report_out_paths`

This command displays reports with statistical information on critical paths and clock paths previously identified with `build_out_paths`.

```
report_out_paths [-<ReportType>] [-num_path=<Value>] \
[-max_diff=<Value>] [-wire] [-zcore] [-num_classes]  \
[-routing] [-memory]
```

Option	Description
`-<ReportType>`	Possible values for `-<ReportType>`: • `-paths` List of the `num_path` critical paths. • `-diffs` List of `max_diff` first different critical delays. • `-histogram` Histogram of the `max_diff` first different critical delays. • `-all` All reports (`paths`, `diffs` and `histogram`).
`-num_path`	Maximum number of paths to be displayed (for `paths` report).
`-max_diff`	Maximum different delays displayed (for `diffs` and `histogram` reports).
`-wire`	Displays wire names instead of port names.
`-zcore`	Displays zCore names instead of FPGA names.
`-num_classes`	Specific option for `histogram` report showing number of classes (i.e. height).
`-routing`	Enables/Disables routing paths in the report.
`-memory`	Enables/Disables memories in the report.

Note: The equivalent command for input-oriented analysis is `report_paths` (which must be called after `build_paths`).

Example:

```
report_out_paths -paths
```

```
#   step REPORT : Report FPGA paths statistics
#   step REPORT : Report a maximum of 50 first routing path(s) ignoring memory
#   step REPORT : #-----------+------+------+---------+-----------------------------------
#   step REPORT : # Port Name | Delay| Fpga | Port Name| Path    -> single   => lvds
#   step REPORT : # Source    |      |      | Target   |         -) async   --/== hop
#   step REPORT : #-----------+------+------+---------+----------------------------------
#   step REPORT : | dout_c[9] |126 ns|    4 | dout[9]  | M0/F03 => M2/F00 -- M2/F00 => M0/F01
#   step REPORT : |           |      |      |          | -- M0/F01 => IF
#   step REPORT : | dout_c[99]|126 ns|    4 | dout[99] | M0/F03 => M2/F00 -- M2/F00 => M0/F01
#   step REPORT : |           |      |      |          | -- M0/F01 => IF
#   step REPORT : | dout_c[98]|126 ns|    4 | dout[98] | M0/F03 => M2/F00 -- M2/F00 => M0/F01
#   step REPORT : |           |      |      |          | -- M0/F01 => IF
#   step REPORT : | dout_c[97]|126 ns|    4 | dout[97] | M0/F03 => M2/F00 -- M2/F00 => M0/F01
#   step REPORT : |           |      |      |          | -- M0/F01 => IF
#   step REPORT : | dout_c[96]|126 ns|    4 | dout[96] | M0/F03 => M2/F00 -- M2/F00 => M0/F01
#   step REPORT : |           |      |      |          | -- M0/F01 => IF
#   step REPORT : | dout_c[95]|126 ns|    4 | dout[95] | M0/F03 => M2/F00 -- M2/F00 => M0/F01
#   step REPORT : |           |      |      |          | -- M0/F01 => IF
#   step REPORT : | din[409]  |123 ns|    3 |din_c[409]| IF => M0/F01 -- M0/F01 => M0/F03
#   step REPORT : | din[364]  |123 ns|    3 |din_c[364]| IF => M0/F01 -- M0/F01 => M0/F03
#   step REPORT : | din[359]  |123 ns|    3 |din_c[359]| IF => M0/F01 -- M0/F01 => M0/F03
...
#   step REPORT : A total of 50 path(s) displayed
```

Example:

```
report_out_paths -diffs
```

```
#   step REPORT : Report a maximum of 50 first different delay(s) ignoring memory
#   step REPORT : #-----------+-----------+-----------+-----------+-----------+
#   step REPORT : # Delay     | Paths     | Max Fpga  | Max Hop   | Max Async |
#   step REPORT : #-----------+-----------+-----------+-----------+-----------+
#   step REPORT : |    126 ns |         6 |         4 |         2 |         0 |
#   step REPORT : |    123 ns |       444 |         3 |         1 |         0 |
#   step REPORT : |    120 ns |        24 |         4 |         2 |         0 |
#   step REPORT : |    118 ns |         2 |         4 |         2 |         0 |
#   step REPORT : |    116 ns |       190 |         3 |         1 |         0 |
#   step REPORT : |    113 ns |       506 |         3 |         1 |         0 |
#   step REPORT : |    110 ns |         2 |         4 |         2 |         0 |
#   step REPORT : |    108 ns |        76 |         4 |         2 |         0 |
#   step REPORT : |    106 ns |        28 |         4 |         2 |         0 |
#   step REPORT : |    105 ns |       768 |         2 |         0 |         0 |
#   step REPORT : |    100 ns |         6 |         4 |         2 |         0 |
#   step REPORT : #-----------+-----------+-----------+-----------+-----------+
#   step REPORT : A total of 11 different delay(s) displayed
```

Example:

```
report_out_paths -histogram
```

```
#   step REPORT : Histogram with a maximum of 50 first different delay(s) ignoring memory
#   step REPORT : #-----------------+-----------------------------------------------
#   step REPORT : # Delay           | Paths
#   step REPORT : #-----------------+-----------------------------------------------
#   step REPORT : [  100 ns:  106 ns ] ---------------------------------------> 802
#   step REPORT : ]  106 ns:  113 ns ] ---> 78
#   step REPORT : ]  113 ns:  119 ns ] ---------------------------------> 698
#   step REPORT : ]  119 ns:  126 ns [ -----------------------> 474
#   step REPORT : #-----------------+-----------------------------------------------
#   step REPORT : A total of 11 different delay(s) encountered
```

3.3.1.2 report_out_clock_paths

This command displays reports with statistical information on critical clock paths previously identified with `build_out_clock_paths`.

```
report_out_clock_paths [-<ReportType>] [-num_path=<Value>] \
[-max_diff=<Value>] [-wire] [-zcore] [-num_classes]        \
[-routing] [-memory]
```

The above options are described in Section 3.3.1.1.

Note: The equivalent command for input-oriented analysis is `report_clock_paths` (which must be called after `build_clock_paths`).

3.3.1.3 report_out_routing_paths

This command displays reports with statistical information on critical output paths previously identified with `build_out_routing_paths`.

```
report_out_routing_paths [-<ReportType>] [-num_path=<Value>] \
[-max_diff=<Value>] [-wire] [-zcore] [-num_classes]          \
[-routing] [-memory]
```

The above options are described in Section 3.3.1.1.

Note: The equivalent command for input-oriented analysis is `report_routing_paths` (which must be called after `build_routing_paths`).

3.3.1.4 report_memory

This command displays reports with statistical information on memories.

```
report_memory [-<ReportType>] [num_path=<Value>] \
[max_diff=<Value>] [-num_classes]
```

The above options are described in Section 3.3.1.1.

3.3.1.5 report_ports

This command displays reports with statistical information on the source ports of the longest critical paths.

```
report_ports [-max_port=<PNumber>]
```

Where `<PNumber>` is the maximum number of ports to be displayed (default: 100).

Example:

```
report_ports
```

```
#   step REPORT : Report source ports statistics
#   step REPORT : Report a maximum of 100 first different source port(s)
#   step REPORT : #--------------------+---------+-------+-----------+---------+-----------+
#   step REPORT : # Port Name          | Delay   | Paths | Max Fpga  | Max Hop | Max Async |
#   step REPORT : #--------------------+---------+-------+-----------+---------+-----------+
#   step REPORT : | F0_0_0/zext0_din[8] |   65 ns |     1 |         4 |       2 |         0 |
#   step REPORT : | F0_0_0/zext0_din[7] |   65 ns |     1 |         4 |       2 |         0 |
#   step REPORT : | F0_0_0/zext0_addr[4]|   55 ns |     1 |         2 |       0 |         0 |
#   step REPORT : | F0_0_0/zext0_din[19]|   55 ns |     1 |         3 |       1 |         0 |
...
#   step REPORT : #--------------------+---------+-------+-----------+---------+-----------+
#   step REPORT : A total of 100 different port(s) displayed
```

3.3.1.6 `report_clock_ports`

This command performs the same operation as `report_ports` but only takes into account the paths of the clock network (starting from clock ports).

```
report_clock_ports [max_port=<PNumber>]
```

Where `<PNumber>` is the maximum number of ports to be displayed (default: 100).

3.3.1.7 `view_port`

This command displays the name of FPGA ports, primary ports and clock ports.

```
view_port [-match_port=<Pattern>] [-match_ins=<Pattern>] \
[-match=<Pattern>] [-clock=<CBool>]
```

Option	Description
`-clock`	Display clock ports (default is `true`)
`-match`	Matching pattern using standard UNIX wildcards You can use the `–match_ins` option to specify an FPGA instance name pattern and the `–match_port` option to specify an FPGA port name pattern

A filter can be specified with the `-match` option, using standard UNIX wildcards. Alternatively you can use the `–match_ins` option to specify an FPGA instance name pattern and the `–match_port` option to specify an FPGA port name pattern.

3.3.1.8 `view_clock`

This command displays the name of clock ports for all the FPGAs.

```
view_clock [-match_port=<Pattern>] [-match_ins=<Pattern>] \
[-match=<Pattern>]
```

The above options are described in Section 3.3.1.7.

3.3.2 Commands for Saving Critical Paths and Clock Paths

The script generated by **`zCui`** already contains commands to save the critical paths and clock paths in HTML report files (`ztime_out_paths.html` and `ztime_clock_out_paths.html`). Nevertheless, the commands described in this section can be added in a script declared in the **Additional Report Command File** field to use options other than the default ones.

3.3.2.1 `drive_out_paths`

After the output oriented analysis, this command generates a file with all critical routing paths found either in html format (`.html`, default format) or in a proprietary text format (`.dump`).

```
drive_out_paths [filename] [-format=<format>] [-port_only]
```

The default file name is `ztime_out_paths` and `<format>` can be `html` or `dump`.

When using the `dump` format, the `-port_only` option can be used to exclusively save FPGA port names for each path.

Examples:

The following command generates the file in dump format using the default file name:

```
drive_out_paths -format=dump
```

```
#   step DRIVE PATHS : Drive all paths in file 'ztime_out_paths.dump.gz' (DUMP format)
```

Contents of `ztime_out_paths.dump.gz` file:

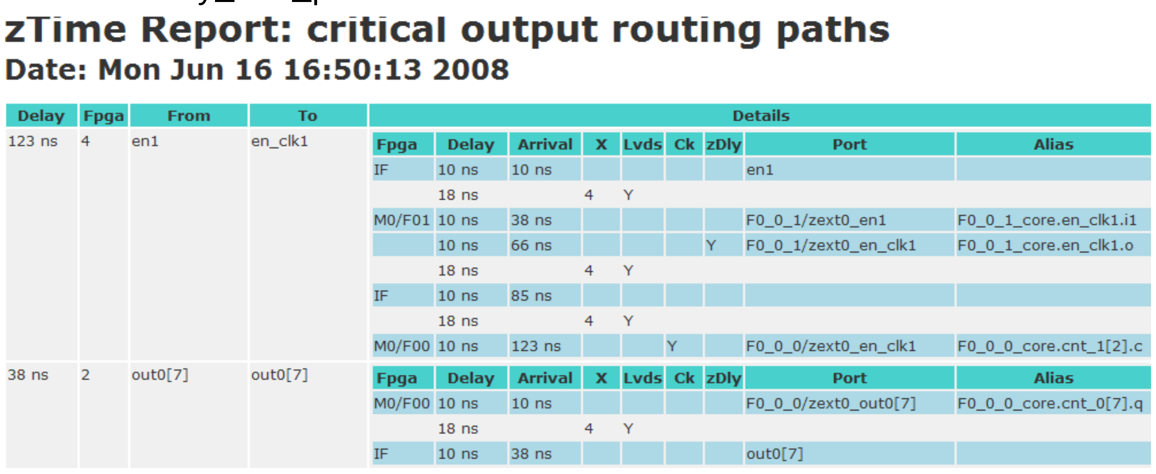
```
Delay,Fpga,Port_from,Port_to,Path: -> single => lvds -) async --/== hop
80 ns, 0, F8_0_1_bank_3, F8_0_1_bank_3, F8_0_1_bank_3
65 ns, 4, F0_0_0.din[8], F8_0_1.din[8]
....
```

The following command generates the file in html format using a user-defined file name:

```
drive_out_paths my_out_paths.htm
```

```
#   step DRIVE PATHS : Drive all paths in file 'my_out_paths.htm' (HTML format)
```

Contents of `my_out_paths.htm` file:



zTime Report: critical output routing paths

Date: Mon Jun 16 16:50:13 2008

Delay	Fpga	From	To	Details: Fpga	Delay	Arrival	X	Lvds	Ck	zDly	Port	Alias
123 ns	4	en1	en_clk1	IF	10 ns	10 ns					en1	
					18 ns		4	Y				
				M0/F01	10 ns	38 ns					F0_0_1/zext0_en1	F0_0_1_core.en_clk1.i1
					10 ns	66 ns				Y	F0_0_1/zext0_en_clk1	F0_0_1_core.en_clk1.o
					18 ns		4	Y				
				IF	10 ns	85 ns						
					18 ns		4	Y				
				M0/F00	10 ns	123 ns			Y		F0_0_0/zext0_en_clk1	F0_0_0_core.cnt_1[2].c
38 ns	2	out0[7]	out0[7]	M0/F00	10 ns	10 ns					F0_0_0/zext0_out0[7]	F0_0_0_core.cnt_0[7].q
					18 ns		4	Y				
				IF	10 ns	38 ns					out0[7]	

3.3.2.2 `drive_out_clock_paths`

After the output oriented analysis, this command generates a file with all critical clock paths found either in html format (`.html`, default format) or in a proprietary text format (`.dump`).

```
drive_out_clock_paths [filename] [-port_only]
```

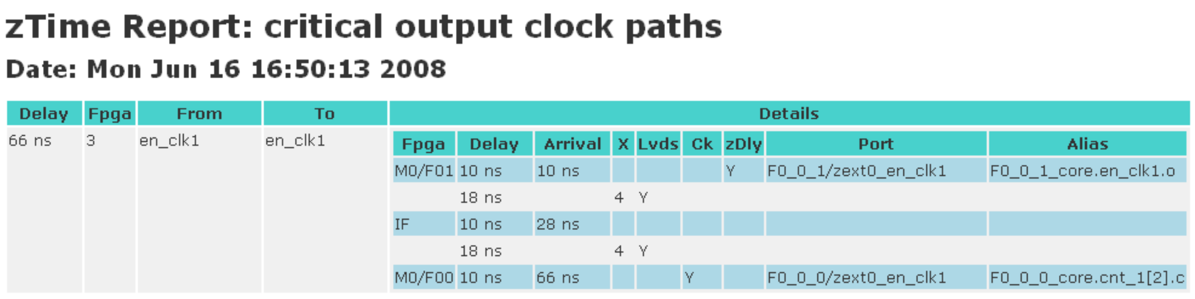
The default file name is `ztime_clock_paths` and `<format>` can be `html` or `dump`.

When using the `dump` format, the `-port_only` option can be used to exclusively save FPGA port names for each path.

Example:

```
drive_out_clock_paths
```

Contents of `ztime_clock_out_paths.html` file:



zTime Report: critical output clock paths

Date: Mon Jun 16 16:50:13 2008

Delay	Fpga	From	To	Details: Fpga	Delay	Arrival	X	Lvds	Ck	zDly	Port	Alias
66 ns	3	en_clk1	en_clk1	M0/F01	10 ns	10 ns				Y	F0_0_1/zext0_en_clk1	F0_0_1_core.en_clk1.o
					18 ns		4	Y				
				IF	10 ns	28 ns						
					18 ns		4	Y				
				M0/F00	10 ns	66 ns			Y		F0_0_0/zext0_en_clk1	F0_0_0_core.cnt_1[2].c

4 Using Static Timing Analysis Results at Runtime

The `des_zTime.xref` file generated by **`zTime`** is automatically used by **`zServer`**.

It contains the speed limitations due to routing network and memory characteristics (achievable `driverClock` frequency) and to the worst values for `zClockSkewTime` and `zFilterTime`.

There is no need to declare the runtime parameters manually in the `designFeatures` file.

The values of `driverClock`, `zClockSkewTime`, and `zFilterTime` which are actually used are listed in the `designFeatures.help` file (automatically generated by the ZeBu runtime software). The `zClockSkewTime` and `zFilterTime` values are also listed at the beginning of the **`zServer`** log file.

The `des_zTime.xref` file is automatically deleted when **`zCui`** starts a new back-end compilation so that you can be sure that your runtime environment will match your most recent timing analysis options.

Note that if the estimated values for `driverClock`, `zClockSkewTime`, and `zFilterTime` are not appropriate for ZeBu, **`zTime`** displays an error message and does not save the parameters for runtime.

5 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 Room 806 8F, No. 20 GuanChian Road Taipei, Taiwan 100 Tel: +886 2 2375 9275