

---

## WORKING PAPER

---

---

<b>Title:</b>	FishTail Toolset Evaluation
<b>Project:</b>	SoC
<b>Document Ref:</b>	RP001039
<b>Date:</b>	Wednesday, 28 July 2004
<b>Author :</b>	Danny Traynor
<b>Distribution:</b>	

---

## Contents

<b>1. Revision History .....</b>	<b>2</b>
<b>2. Executive Summary .....</b>	<b>3</b>
<b>3. Introduction .....</b>	<b>4</b>
<b>4. General Usability Issues .....</b>	<b>5</b>
<b>5. Reading Verilog RTL .....</b>	<b>6</b>
<b>6. Identification of Design Modes.....</b>	<b>7</b>
<b>7. False Paths and Multi Cycle Paths.....</b>	<b>10</b>
<b>8. Conclusions.....</b>	<b>11</b>
<b>9. APPENDIX.....</b>	<b>12</b>
9.1. Unix Run Script .....	12
9.2. Focus Run Script.....	13
9.3. Focus Setup Script.....	14
9.4. Focus Input Constraints File.....	15
9.5. Focus Pragma File .....	15

## 1. REVISION HISTORY

Version	Date	Notes
0.1	12 <sup>th</sup> July 2004	First preliminary release
0.2	28 <sup>th</sup> July 2004	Updated for first release

## 2. EXECUTIVE SUMMARY

The Focus product from FishTail Design Automation is intended to allow designers to quickly and easily generate golden timing constraints for their designs, at a very early stage in the design flow.

The tool primarily targets the exceptions to single-cycle Static Timing Analysis (STA). These are false paths and multi-cycle paths, and can present problems in reaching timing closure on a design: these timing issues often appear late in the design cycle, and require manual intervention to trace paths, identify and check the paths are false, and modify the STA constraint files to reflect the false paths. This adds extra effort and uncertainty late in the design cycle, and can impact tape-out. In addition, undetected timing exceptions can result in chips that are over optimized, and therefore consume more area and power than is absolutely necessary.

The evaluation showed that the tools could quickly yield a surprisingly large number of false and multi-cycle paths for a design. The tool also generates a list of assertions to allow the exceptions to be verified using a formal property checking tool, but this was not done as Jennic do not have a property checker. In our test case, meeting setup timing did not present any problems and so these exceptions would have little or no impact on timing or area. However, in large, fast designs, these exceptions could result in a smaller implementation area, or even make the difference between a block being implementable or not.

The tool could identify modes of operation where only one clock was propagated to the flops in the design. However, I was concerned that using the mode constraints directly could be risky, and would recommend that they be inspected carefully to ensure that they do not cause unchecked paths in STA.

Some issues were found with reading RTL (due to a known bug), which made it impossible to use the tool on one block.

A few minor usability issues were identified, and suggestions made to ease the use of the tool.

### 3. INTRODUCTION

Within Jennic, we have significant experience of STA and timing constraints creation, and have in the past seen issues both with constraint omissions and with STA tool performance on poorly constrained designs. Therefore, a tool that could quickly yield high quality constraints is of interest. Of particular interest is the identification of design modes and creation of constraints to configure the design in the various modes: our current STA tooling is unable to cope with multiple clocks driving a single endpoint, and hence a significant amount of effort is expended in identifying the configuration of the design to ensure no multiple driven clock endpoints.

FishTail make the following claims for their Focus tool:

- Push-button generation of false and multicycle paths.
- Accepts RTL descriptions in standard Verilog format.
- Outputs golden-timing constraints in standard SDC format.
- Analyzes multi-million gate designs in a matter of hours.
- Applicable for both flat and hierarchical design flows.
- Automatically identifies multiple design modes and generates separate constraint files for each mode.
- Provides verification of false-path definitions by generating assertions for import into formal property verification tools.
- Ability to filter SDC output so that only exceptions for timing critical-endpoints are imported into downstream tools.

The tool was run on a small (120k instances) Jennic chip. This design was built on a 0.18um process and had a low clock frequency (20MHz maximum). The RTL code used for the trial had previously been used for the physical implementation and subsequent tape-out of the design, and there were almost no maximum (setup) timing issues: therefore the multi-cycle and false paths were slightly less critical in this instance. However, the design was heavily optimised for low power operation, and so had complex power down modes, clock multiplexing and clock gating, which made the mode identification of significant interest.

This document overviews the tasks performed using the FishTail tools, and provides feedback on what went well and what could be improved.

Version 1.7.4 of the Focus tool was used, running on a Sun Unix (Solaris) platform.

## 4. GENERAL USABILITY ISSUES

Starting the tool is easy – type focus at the Unix Command prompt. A quirk of the tool is that all information appears to be output on the standard error, rather than on the standard output. Therefore, when running the tool in batch mode and creating a log file, it is necessary to use “>&” or “|&” to redirect the output to the log file rather than “>” or “|”, for example:

```
UNIX%> focus focus.tcl |& tee focus.log
```

The tool documentation is reasonably good, although I found one aspect particularly confusing. The “Focus User’s Guide” guides the user through the stages of reading a design, defining clocks and boundary timing, analysing modes of operation, and generating timing exceptions, and it appears to be relatively straightforward. However, the documentation for the “write\_gtc” command to generate the timing exceptions is very poorly explained, especially regarding the “design\_script” parameter. This is perhaps the least intuitive aspect of using the tool, as the flow appears to be to read in the design and constraints in order to identify the design modes, and then the design script provided to write\_gtc causes everything to be read in again in order to generate the exceptions. In the end, I only found out the contents of the write\_gtc design script by looking in the examples area in the FishTail installation area. Once the contents of the design script are understood, driving the tool becomes reasonably straightforward.

One possible enhancement, which I believe would greatly improve the ease with which Focus could be integrated into a design flow, is to add a “read\_sdc” command. Currently, the engineer must define the clocks and boundary constraints for the tool, but these are usually available as synthesis constraints at the stage when Focus would be run in a design flow. If it were possible to simply read the synthesis constraints directly into Focus, then the flow for running Focus could be made seamless: create synthesis constraints as usual; read constraints into Focus to create exceptions; append exceptions to constraints; run synthesis as usual. I was able to use the synthesis constraints for one block, but in order to do so I had to comment out several commands, including set\_max\_area, set\_clock\_latency, set\_clock\_uncertainty, etc. If the tool simply read and ignored these constraints the usability would be much improved.

I would also like the manual pages to be supplied in PDF format, along with the tool documentation: an indepth discussion of the commands in a browsable format would be invaluable for any engineer trying to drive the tool.

## 5. READING VERILOG RTL

To read the block level verilog RTL, the following code was used:

```
read_design \  
  verilog_files.txt \  
  -design_name ${BLOCK_NAME} \  
  -incdir /soc/FISHTAIL_TRIAL/${BLOCK_NAME}/20040405/RTL \  
  -v {/soc/FISHTAIL_TRIAL/MEMORIES/RAM/RAM8192W32B.v \  
    /soc/FISHTAIL_TRIAL/MEMORIES/ROM/ROM12288W32B.v}
```

The file “verilog\_files.txt” contained a list of all verilog RTL files for the entire chip, and the top-level module was picked out using the \${BLOCK\_NAME} variable. Verilog models were included for the RAM and ROM macro blocks. This worked without issue for four of the five blocks in the trial device.

However, two aspects of the RTL for one of the trial blocks caused the Focus tool to fail. In the block in question, the tool objected to the following items:

```
module ram2p (wr_data,wr_addr,wr_clk,wr_en,rd_data,rd_addr,rd_en,rd_clk);  
  // override as required in parent module  
  parameter DATA_W = 0,  
            ADDR_W = 0,  
            DEPTH = 0;  
  
  input [DATA_W-1:0] wr_data;
```

In this instance, the definition of the parameters (which are overridden in the instantiation of the module) caused the tool to define the first input as “DATA\_W[-1:0]”, and the tool gave the warnings “Negative range bound found” and “The module ‘ram2p’ will be ignored”.

The second issue was a problem with multi-dimensional arrays of register wires:

```
reg [3:0] sym_err [15:0];
```

In this case, the tool appeared to be quite happy with various operations on the “sym\_err” array, for example, the following:

```
assign new_sum_0 = {2'h0, sym_diff[0]} + sym_err[0];
```

However, when it was fed into a sub-block with parameterised data widths, the tool gave an error message and then crashed:

```
mod_min_val #(4,4) mod_min_val0 (  
  .num0(4'd0),  
  .num1(4'd1),  
  .val0(sym_err[0]),  
  .val1(sym_err[1]),  
  ...
```

In this instance, the default parameter widths were set to 8, and were then overridden with 4 in the instantiation, but the RTL was nevertheless valid and had successfully been used for synthesis followed by physical implementation and then tape-out. Applications support confirmed that this is a known bug, which had not yet been fixed in the latest (version 1.8) release of the tool.

## 6. IDENTIFICATION OF DESIGN MODES

One major point of interest for the Focus tool was the identification of the modes of operation for a design, and creation of SDC files to configure the design into those modes.

The Focus User's Guide document contains the following discussion on design modes:

"During clock propagation Focus identifies if there are any clock sinks on a design that multiple clock sources propagate to. When this situation arises it is reflective of a design having different modes of operation, where depending on the mode a design is in, different clocks propagate to the same clock sinks.

Focus automatically identifies the different modes on a chip such that in any given mode only one clock source reaches a clock sink. The modes for a design and their associated conditions (the values on nets that are required to put a design into a given mode) can be queried using the report\_modes command."

In order to check this, one small block in the trial design was used. In BLOCK\_2, there are approximately 100 flops, all driven by a net called "muxed\_clk", which is created as follows:

```
always @(clk or backup_clk_32k or use_backup_clocks or test_rc_bp or test_en)
begin
  if (test_en & test_rc_bp)
    muxed_clk = backup_clk_32k;
  else if (use_backup_clocks)
    muxed_clk = backup_clk_32k;
  else
    muxed_clk = clk;
end
```

The use of pragmas is required (in addition to the clock and boundary constraints) in order to allow Focus to identify modes. For the example above, the pragmas were as follows:

```
set_pragma ft_clock_source clk
set_pragma ft_clock_source backup_clk_32k

set_pragma ft_mode_source test_en
set_pragma ft_mode_source use_backup_clocks
```

With this in place, Focus then identified modes of operation of the circuit, and produced set\_case\_analysis commands in the constraints files to configure the circuit in one of two modes, where either "clk" or "backup\_clk\_32k" were propagated to the flops. Focus was able to configure two modes of operation either with or without the "ft\_mode\_source" pragmas, so long as the "ft\_clock\_source" pragmas were set. The resulting constraints files for the two modes were as follows:

### Mode 1 (propagates BACKUP\_CLK\_32K to flops):

```
set case_analysis_sequential_propagation always

# Case statements for design mode 1
set_case_analysis 1 [get_pins -leaf -of [get_nets use_backup_clocks]]

# iteration 1

# Clock-to-Clock exceptions
set_false_path -from [get_clocks CLK_VIRT] \
               -to [get_clocks BACKUP_CLK_32K]
set_false_path -from [get_clocks BACKUP_CLK_32K] \
               -to [get_clocks CLK_VIRT]
```

### Mode 2 (propagates CLK to flops):

```
set case_analysis_sequential_propagation always

# Case statements for design mode 2
set_case_analysis 0 [get_pins -leaf -of [get_nets use_backup_clocks]]
set_case_analysis 1 [get_pins -leaf -of [get_nets test_rc_bp]]
set_case_analysis 0 [get_pins -leaf -of [get_nets test_en]]

# iteration 1
# Exceptions for endpoint sys_ctrl_16m_rst
set_false_path -from [get_clocks CLK_VIRT] \
               -to [get_ports { sys_ctrl_16m_rst }]
# Exceptions for endpoint anper_v_reg_pd
set_false_path -from [get_ports { test_rdio_pd }] \
               -to [get_ports { anper_v_reg_pd }]
# Exceptions for endpoint rf_v_reg_pd
set_false_path -from [get_ports { test_rdio_pd }] \
               -to [get_ports { rf_v_reg_pd }]
# Exceptions for endpoint syn_v_reg_pd
set_false_path -from [get_ports { test_rdio_pd }] \
               -to [get_ports { syn_v_reg_pd }]
# Exceptions for endpoint vco_v_reg_pd
set_false_path -from [get_ports { test_rdio_pd }] \
               -to [get_ports { vco_v_reg_pd }]
# Exceptions for endpoint en_xtal_33
set_false_path -from [get_ports { test_xtal_pd }] \
               -to [get_ports { en_xtal_33 }]

# Clock-to-Clock exceptions
set_false_path -from [get_clocks CLK_VIRT] \
               -to [get_clocks CLK]
set_false_path -from [get_clocks CLK] \
               -to [get_clocks CLK_VIRT]
```

Several issues arise from a closer inspection of these constraints:

1. I am nervous about using “set case\_analysis\_sequential\_propagation always” in STA, as it causes constant values (set using “set\_case\_analysis”) to be propagated through sequential cells when set. I believe it can be difficult to convince oneself that a design will not have unchecked paths when using this switch, especially in glue logic which could fall between STA modes, and so in general I recommend we avoid using it within Jennic. This is, of course, purely a personal opinion.
2. In mode 2, the second set\_case\_analysis (1 on “test\_rc\_bp”) is not actually required in order to propagate CLK to the flops when “test\_en” is low. It is included either with or without “test\_rc\_bp” being defined using an “ft\_mode\_source” pragma. Furthermore,



mode 2 could also be enabled by “test\_rc\_bp” low, or both signals low.

For STA, I need to be sure that all logic paths are checked. From the above two modes, any logic which is enabled by (!test\_en & !test\_rc\_bp & !use\_backup\_clocks) or (test\_en & !use\_backup\_clocks) will be disabled in both modes and therefore unchecked in STA using these constraints files. It is not clear whether Focus, when defining modes, ensures that all logic is enabled in at least one mode – this needs to be clarified.

3. In both modes, Focus has created false paths between the propagated clock and CLK\_VIRT. This means that an STA run using these modes will leave all boundary constraints unchecked. This block used virtual clocks to define boundary constraints (this is a reliable way to overcome inconsistencies in the interpretation of clock latencies between Magma and PrimeTime). In creating these false paths, Focus has caused all the virtual clock boundary timings to be disabled for this block, and this is incorrect.

When I used the pragmas, I found that running the “write\_assertions” command caused Focus to crash. As we do not have the property checker required to verify the assertions, I did not investigate this further.

## 7. FALSE PATHS AND MULTI CYCLE PATHS

The main command to identify false and multi-cycle paths in Focus is “write\_gtc”. When this was run, false and multi-cycle paths were identified. The command was run as follows:

```
set sequential_control 0
write_gtc \
  -design_script setup.tcl \
  -output_file ${BLOCK_NAME}_fp.gtc
```

The sequential\_control variable controls whether false paths (0) or multi-cycle paths (1) are identified: two runs of write\_gtc command are required to create both sets of exceptions.

The statistics from the test runs carried out are as follows:

Block	Complexity	# Clock Sources	# Modes	# False Paths	# Multi-Cycle Paths	Run Time
BLOCK_1	45k insts 5.5k flops	5	2	974 / 1042	180 / 182	7 Hrs
BLOCK_2	2.2k insts 110 flops	2	2	2 / 8	8 / 8	3 Mins
BLOCK_3	1.6k insts 0 flops 3 RAMs	1	None	0	0	1 Min

In the most complex block, an astonishingly large number of false and multi-cycle paths were identified. There were no setup timing issues in the design in question, and so these results make little or no difference to timing. However, in high frequency designs, and in larger designs where interconnects are much longer, these false and multi-cycle paths could make the difference between a design being implementable or not.

## 8. CONCLUSIONS

The FishTail Focus tool provides a means to easily create some of the more arduous and error prone parts of STA constraints. The creation of false and multi-cycle paths becomes a very quick process. I am still slightly concerned about the need to verify the constraints – having false exceptions could mean a design is under-constrained, which would be a disaster in STA – but the use of a formal property checker should give confidence in this.

The mode identification feature in Focus could be very useful indeed: it correctly identified those nets that need to be controlled in order to correctly propagate one clock to the flops in the design. However, based upon my findings I would be very concerned at using the mode constraints directly, due to the risk of having unchecked paths in STA. Instead I would use the information from the mode constraints, along with input from the circuit designers, to understand the modes of operation of the circuit and create a constraints suite, which correctly constrains the design.

The evaluation showed that the tools could quickly yield a surprisingly large number of false and multi-cycle paths for a design. In our test case, meeting setup timing did not present any problems and so these exceptions would have little or no impact on timing or area. However, in large, fast designs, these exceptions could result in a smaller implementation area, or even make the difference between a block being implementable or not.

## 9. APPENDIX

### 9.1. Unix Run Script

```
#!/bin/csh
#-----
#  **COMMERCIAL IN CONFIDENCE** Copyright Jennic Ltd @ 2004
#
#  Jennic Ltd. Sheffield, Furnival Street,
#  Sheffield S1 4QT.
#  Tel: +44 144 2812655 e-mail:info@jennic.co.uk
#-----
#  Project Name      :  FISHTAIL_TRIAL
#  Filename          :  go_focus.csh
#  Originator        :  Danny Traynor
#  Function           :  Top-Level FishTail Focus Invocation script
#-----

module load perl
module load fishtail

cd /soc/FISHTAIL_TRIAL/BLOCK_2/20040405/FISHTAIL

#####
# Environment Variables ...
#####
setenv BLOCK_NAME BLOCK_2

#####
# Run Focus ...
#####
focus \
    focus.tcl \
    |& tee focus.log

#####
# End.
#####
```

## 9.2. Focus Run Script

```
#-----
#  **COMMERCIAL IN CONFIDENCE** Copyright Jennic Ltd @ 2004
#
#  Jennic Ltd. Sheffield, Furnival Street,
#  Sheffield S1 4QT.
#  Tel: +44 144 2812655 e-mail:info@jennic.co.uk
#-----
# Project Name      : FISHTAIL_TRIAL
# Filename          : focus.tcl
# Originator       : Danny Traynor
# Function          : Top-Level FishTail Focus Invocation TCL script
#-----

set detailed_notes 2

#####
# Run variables ...
#####
set BLOCK_NAME $env(BLOCK_NAME)

#####
# Read verilog files ...
#####
read_design \
    verilog_files.txt \
    -design_name ${BLOCK_NAME} \
    -incdir /soc/FISHTAIL_TRIAL/${BLOCK_NAME}/20040405/RTL \
    -v {/soc/FISHTAIL_TRIAL/MEMORIES/RAM/RAM8192W32B.v \
        /soc/FISHTAIL_TRIAL/MEMORIES/ROM/ROM12288W32B.v}

#####
# Read synthesis constraints and pragmas ...
#####
source ${BLOCK_NAME}_input_constraints.tcl
source ${BLOCK_NAME}_pragmas.tcl

#####
# Identify modes of operation ...
#####
update
report_modes -all -output_file ${BLOCK_NAME}_modes.txt

#####
# Write golden timing constraints ...
#####
set sequential_control 0
write_gtc \
    -design_script setup.tcl \
    -output_file ${BLOCK_NAME}_fp.gtc
# write_assertions \
#     ${BLOCK_NAME}_fp.gtc \
#     -output_file ${BLOCK_NAME}_fp.gtc.assertions

set sequential_control 1
write_gtc \
    -design_script setup.tcl \
    -output_file ${BLOCK_NAME}_mcp.gtc
# write_assertions \
#     ${BLOCK_NAME}_fp.gtc \
#     -output_file ${BLOCK_NAME}_fp.gtc.assertions

quit

#####
# End.
#####
```

### 9.3. Focus Setup Script

```
#-----
#  **COMMERCIAL IN CONFIDENCE** Copyright Jennic Ltd @ 2004
#
#  Jennic Ltd. Sheffield, Furnival Street,
#  Sheffield S1 4QT.
#  Tel: +44 144 2812655 e-mail:info@jennic.co.uk
#-----
# Project Name      : FISHTAIL_TRIAL
# Filename          : setup.tcl
# Originator       : Danny Traynor
# Function          : FishTail Focus write_gtc design script
#-----

set BLOCK_NAME $env(BLOCK_NAME)

#####
# Read verilog files ...
#####
read_design \
  verilog_files.txt \
  -design_name ${BLOCK_NAME} \
  -incdir /soc/FISHTAIL_TRIAL/${BLOCK_NAME}/20040405/RTL \
  -v {/soc/FISHTAIL_TRIAL/MEMORIES/RAM/RAM8192W32B.v \
    /soc/FISHTAIL_TRIAL/MEMORIES/ROM/ROM12288W32B.v}

#####
# Read synthesis constraints and pragmas ...
#####
source ${BLOCK_NAME}_input_constraints.tcl
source ${BLOCK_NAME}_pragmas.tcl

#####
# End.
#####
```

## 9.4. Focus Input Constraints File

```
#####
# BLOCK_2 clock definitions and boundary constraints ...
#####

#####
#   Created by Design Compiler write_sdc on Thu Apr 15 14:33:58 2004
#####
set sdc_version 1.3

create_clock -name "CLK" -period 60 -waveform {0 30} [get_ports {clk}]
create_clock -name "CLK_VIRT" -period 60 -waveform {0 30}
create_clock -name "BACKUP_CLK_32K" -period 60 [get_ports {backup_clk_32k}]

set_input_delay 20 -max -clock CLK_VIRT [get_ports {test_rdio_pd}]
set_input_delay 0.25 -min -clock CLK_VIRT [get_ports {test_rdio_pd}]
set_input_delay 20 -max -clock CLK_VIRT [get_ports {test_rc_bp}]
set_input_delay 0.25 -min -clock CLK_VIRT [get_ports {test_rc_bp}]
...
set_input_delay 20 -max -clock CLK_VIRT [get_ports {async_rst_n}]
set_input_delay 0.25 -min -clock CLK_VIRT [get_ports {async_rst_n}]

set_output_delay 20 -max -clock CLK_VIRT [get_ports {disable_level_shifters}]
set_output_delay -0.25 -min -clock CLK_VIRT [get_ports {disable_level_shifters}]
set_output_delay 20 -max -clock CLK_VIRT [get_ports {en_bypass_xtal_33}]
set_output_delay -0.25 -min -clock CLK_VIRT [get_ports {en_bypass_xtal_33}]
...
set_output_delay 20 -max -clock CLK_VIRT [get_ports {en_xtal_33}]
set_output_delay -0.25 -min -clock CLK_VIRT [get_ports {en_xtal_33}]

#set_clock_uncertainty 1 -setup CLK
#set_clock_uncertainty 0.25 -hold CLK
#set_clock_latency 0.5 CLK
#set_clock_uncertainty 1 -setup CLK_VIRT
#set_clock_uncertainty 0.25 -hold CLK_VIRT
#set_clock_latency -source 0.5 CLK_VIRT
#set_max_area 0
#set_wire_load_mode "enclosed"

#####
# End.
#####
```

## 9.5. Focus Pragma File

```
#####
# BLOCK_2 pragmas ...
#####
set_pragma ft_clock_source clk
set_pragma ft_clock_source backup_clk_32k

set_pragma ft_mode_source test_en
set_pragma ft_mode_source use_backup_clocks

#####
# End.
#####
```