APPLICATION NOTE

Mentor Synthesis Group

LeonardoSpectrum to Precision Transition Guide

May 2003

Mentor Graphics®

Table of Contents

Introduction

Users migrating from LeonardoSpectrum to Precision should find the transition easy and straightforward. The basic command set in Precision is not backwards compatible with LeonardoSpectrum however this command set is simple and intuitive. Advanced netlist attributes used to control optimization have been carried forward into Precision. Users should be able to convert even the most complex script file to Precision's format in a few minutes. This document will help guide users through the process.

Installing Precision

Precision 2003a uses a newer version of licensing which isn't compatible the 6.1* versions of licensing that Mentor Graphics tools have been released with for years. You must stop the old mgcld vendor daemon before starting the new mgcld vendor daemon. If you have multiple vendor daemons for the same license manager daemon (lmgrd) and you don't want to stop and restart lmgrd, you can stop and restart just the mgcld vendor daemon by using the -vendor switch on the lmdown and lmreread commands (e.g., -vendor mgcld). For more information about the -vendor switch, refer to the FLEXlm End Users Guide available at

<http://www.macrovision.com/solutions/esd/support/enduser/TOC.htm> (sections 7.5 and 7.11).

Precisions User Interface

Design Center

Precisions user interface was re-architected to be more intuitive yet still provide the functionality available in LeonardoSpectrum. The main interface is called the "Design Center" and provides complete access to the tools functionality. The "Design Bar" guides users through the steps of the synthesis process similar to LeonardoSpectrum's Flow Tabs.

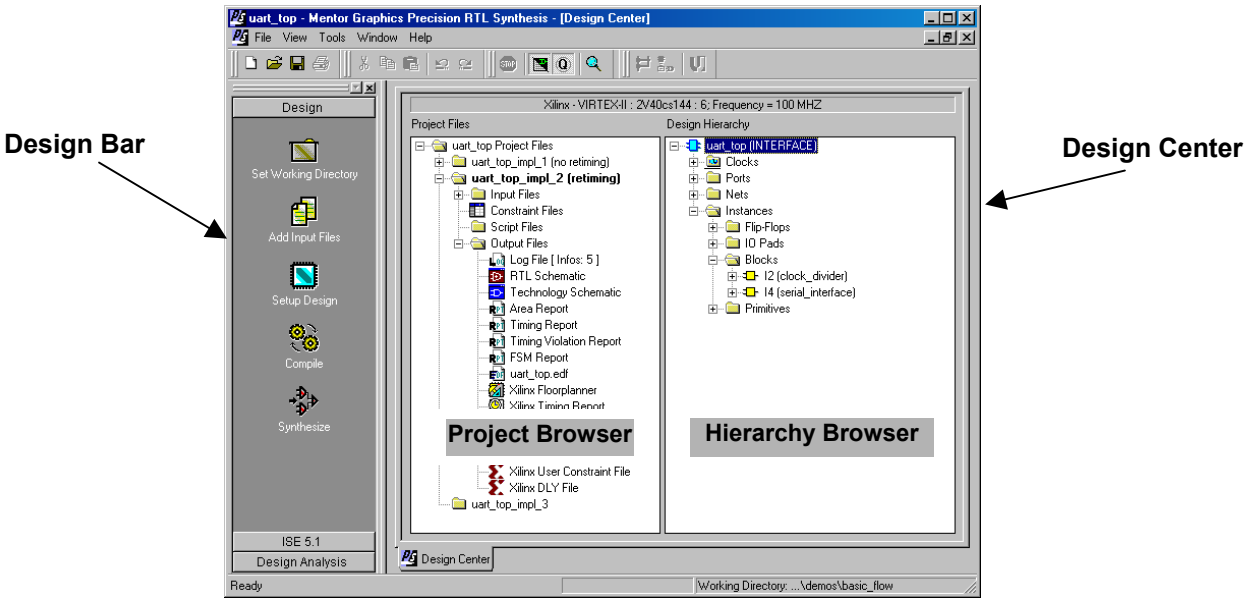

Figure 1 – Precision Design Center

Synthesis Options

Many of the options found in LeonardoSpectrum's FlowTabs are now available in the "Tools -> Set Options" form shown below. Unnecessary or obsolete options have been removed.

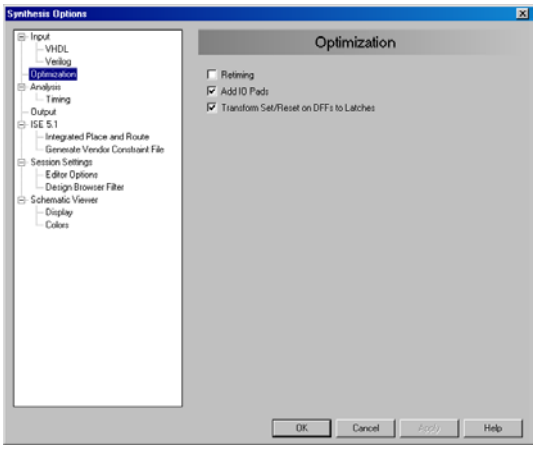

Figure 2 – Precision Options Setup Interface

Setting Constraints

The Precision user interface allows constraint setting on a netlist object (like a port) anywhere it can be selected. This includes the design hierarchy browser view, schematic viewer, graphical find window, as well as the HDL editor and report views.

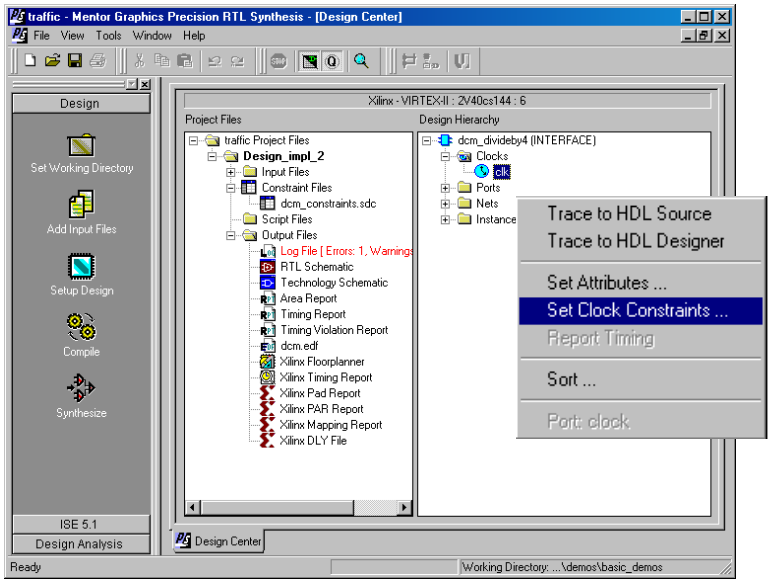

Figure 3 – Setting Constraint in Precision

Precision utilizes a context sensitive constraint editor, which tailors itself to the selected netlist object type. A different constraint editor form will appear depending on whether the object type is a clock, register, module or net. Refer to Appendix-A for a complete list of netlist object constraints that are available with Precision

Schematic Viewing

Schematic viewing in Precision is now included in the base product and available to all users. Users should experience a performance improvement of 5x to 10x on large designs. Symbols have been added to improve readability and users can now view the contents of LUTs.

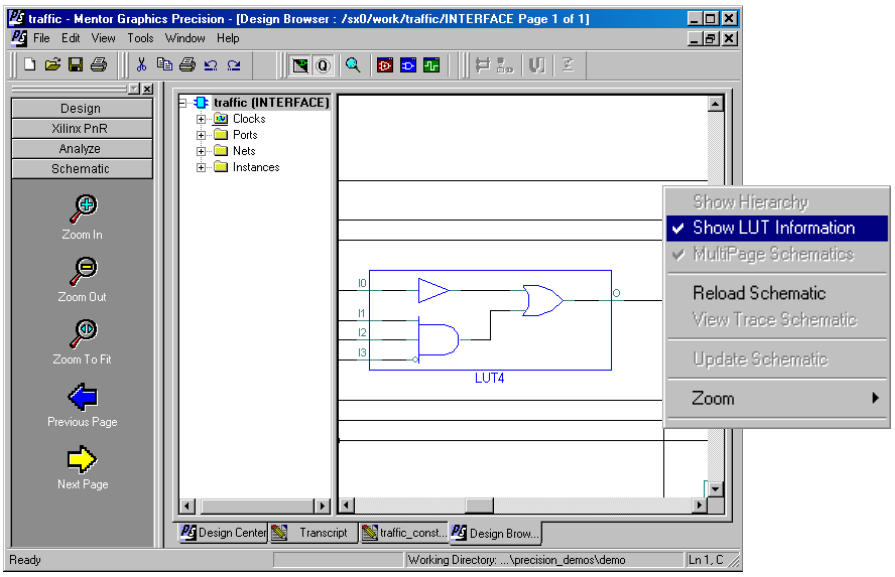

Figure 4 – Precision Schematic Viewer

Note: *The Precision schematic viewer will, by default, display bussed nets and instances as bundled objects. This "bundling" makes for an overall more readable schematic, however can also make debug challenging when trying to trace to an individual object within a bundle. The bundling feature can easily be disabled using the "Tools -> Options -> Schematic Viewer" form*

Script Conversion

Converting a LeonardoSpectrum script file to Precision is very straightforward. Shown below is a quick comparison of a basic synthesis script in LeonardoSpectrum and Precision. This section will cover the basics of converting a script by hand. An automatic script converter utility is available through customer support.

LeonardoSpectrum Synthesis Script

```
# Setup Technology Environment
set part 2V80cs144
set process 6
set wire_table xcv2-40-6_wc
load_library xcv2

#Read the Design
read {traffic.v}

# Setup Timing Constraints
set register2register 16.66667
set input2register 16.66667
set register2output 16.66667
set input2output 16.66667

# Optimize the Design
optimize
optimize_timing

# Generate Reports
report_area
report_delay

# Save Results
auto_write traffic.edf
```

Precision Synthesis Script

```
# Setup Technology Environment
setup_design -manufacturer "Xilinx" -
family "VIRTEX-II" -part "2V40cs144"
-speed "6"

# Add input files
add_input_file {traffic.v}

# Setup timing constraints
setup_design -frequency 62.5

# Complete the optimization process
compile
synthesize
```

Projects vs Scripts

LeonardoSpectrum was designed to work with script files while the Precision user interface is designed to work with projects. The difference is subtle. A project file is simply a script file minus the implementation commands, which include compile, synthesize and place_and_route. The Precision user interface can load, restore and manage designs when synthesized using projects. Refer to the documentation for a complete description of projects.

Setup Technology Environment

LeonardoSpectrum

To set the target technology in LeonardoSpectrum users were required to set 3 variables then issue the load_library command as shown below:

```
set part v50ecs144
set process 6
set wire_table xcve50-6_avg
load_library xcv2
```

Precision

In Precision, technology setup has been consolidated into a single command. Users no longer need to specify the wire_table, which will be automatically derived from the part information.

```
setup_design -manuf "Xilinx" -family "VIRTEX-II" -part "2V40cs144" -speed "6"
```

Note: *Precision will load the library during the design synthesis. The setup_design command is (as you would expect) simply a setup command.*

Reading the Design

LeonardoSpectrum

LeonardoSpectrum provided 2 ways to read designs, the "read" command or the "analyze / elaborate" command pair. When the read command was issued the actual read was performed. Users were not required to define the complete file order, but the top-level module or entity had to be placed at the bottom of the file list and VHDL packages at the top.

```
#Read the Design
read { my_pkg.vhd a.vhd b.vhd c.vhd top.vhd }
```

Precision

Precision has 2 new commands for reading designs called "add_input_file" and "compile". This command defines the file location and properties such as format or work library. The compile command has no options. The equivalent Precision command set is as follows:

```
# Read the Design
add_input_file { my_pkg.vhd a.vhd b.vhd c.vhd top.vhd}
compile
```

Command Switch	Description
-format	Specifies the file type for file names that don't have the proper extension. Valid values are vhdl | verilog | edif | syn | lib | tcl | xnf | xdb | sdf. If this option is not used and a valid extension exists, then the file type will be automatically detected.
-work	Specifies the name of the work library for compiling the content of the file. If not specified, then the work library name **`work`** is assumed.
-search_path	Specifies additional directories that are pre-pended to the global search path that is specified in the **`setup_design`** command.
-exclude	Allows non RTL files to be added to the project. This switch will tell Precision to ignore the file for synthesis but to copy it into the place and route directory. Great for vendor constraint files and vendor pre-compiled IP blocks

Table 1 – add_input_file command switch summary

Notes:

- *A separate "add_input_file" command must be used for files with different formats.*
- *RTL files do not need to appear in any particular order in the input file list. Precision will automatically detect the top-level entity / module. There are 2 exceptions: first, all VHDL packages must appear at the top of the file list and second, designs with multiple-tops must have the desired top-level file appear at the bottom*

Setting Timing Constraints

This section will only cover the global frequency method of constraining designs. Port based timing constraints will be covered in a later section.

LeonardoSpectrum

Four separate variables are used to define global frequency in LeonardoSpectrum. Port-based timing constraints will override the global frequency values. These variables are:

```
set register2register 16
set input2register 16
set register2output 16
set input2output 16
```

Precision

Precision no longer supports the concept of a global variable to define timing requirements. Only port constraints will be considered during timing analysis, which eliminates the ambiguity of LeonardoSpectrum's dual constraining paradigm. Users can still define a global frequency value using the "setup_design" command. This command will instruct Precision to automatically generate port constraints on the design.

```
Setup_design –frequency 62.5
```

Once synthesis completes a file called "Constraints Report" will appear in the output file list. This is an actual constraint file that can be added to the input file list and used for subsequent runs.

Optimizing the Design

Precision's optimization process is controlled by constraints, not command switches as in LeonardoSpectrum. For example multi-pass optimization in LeonardoSpectrum was enabled through the "-effort standard" switch while additional optimization in Precision is automatically performed on blocks not meeting timing constraints.

LeonardoSpectrum

LeonardoSpectrum provides two commands for performing optimization, "optimize" and "optimize_timing". Each of these commands includes a set of switches to control how optimization is performed. Typical usage is as follows

```
Optimize –ta xcv2 –hier auto –effort standard –chip
Optimize_timing
```

Precision

Precision's optimizations are constraint driven rather that switch driven. Timing constraints determine when and where multi-pass optimization is performed on the design. Constraints applied directly to modules determine which hierarchy modifications will be performed, and the setup_design command must be used to control IO insertion.

To optimize a design with Precision, use "synthesze". This command has no switches and must be executed after the "compile" command.

```
Synthesize
```

To optimize a design without inserting IO buffers, the following command should be issued prior to the synthesize command.

```
Setup_design –addio=false
```

Precision includes new technology to automatically move logic across hierarchy boundaries when performance improvements can be realized thus reducing, if not eliminating, the need to flatten a netlist. Users can override this optimization by assigning a "hierarchy" attribute to an instance with a value of "preserve" or "flatten".

```
set_attribute -name hierarchy -value flatten -instance I1
```

Notes:

- *To optimize a design for area, leave the design unconstrained.*
- *To minimize run times apply a relaxed constraint to the design such as 1 Mhz.*

Generating Reports

When using LeonardoSpectrum, separate commands must be issued to generate area and timing reports. Precision has consolidated these report generation commands into the "synthesize" command. Upon completion of "synthesize" an area report file and a delay report file are generated automatically and placed into the implementation folder.

Users who wish to run these commands interactively will find they are still available with some modifications. The report_area command is identical in both tools. Report_delay has been obsoleted and replaced by report_timing.

LeonardoSpectrum

```
Report_delay –to | -from | -through | -num_paths | -show_schematic
```

Precision

```
Report_timing –to | -from | -through | -num_paths | -show_schematic
```

Notes: *Precision's "report_timing" command includes significant functionality enhancements over LeonardoSpectrum's "report_delay". Users might want to read the reference manual or type "report_timing –help" from the command line to see the new options.*

Saving Results

Precision's "synthesize" command also generates the final netlist and constraint file for place and route. This eliminates the need for the "write" and "auto_write" commands found in LeonardoSpectrum. Precision saves an .xdb and .edf netlist file after each synthesis run. The command "auto_write" is still supported to allow the interactive generation of additional netlist formats

Precision provides users complete control over the naming and location of output files. This is accomplished through the *"setup_design"* and *"set_working_dir"* commands. To specify the location and name of an output file use the following commands:

```
Set_working_dir c:/designs/uart
Setup_design –impl uart_imp_1
Setup_design –basename my_design
```

This will place the output EDIF file into c:/designs/uart/uart_imp_1/my_design.edf

Note: *Precision uses TCL for a scripting language. TCL requires the use of forward slashes "/" to specify directory structures, even on Windows OS.*

Converting Timing Constraints

Precision supports the Synopsys Design Constraints (SDC) format for defining timing constraints. This is an industry standard constraint format that is not backwards compatible with LeonardoSpectrum. Precision includes a new, highly advanced timing analysis engine that requires the following additional information to be specified in the constraints:

1. Clocks must be defined as synchronous or asynchronous to other clocks
2. IO constraints require a reference clock to be specified

LeonardoSpectrum Constraint File

```
# Clock Constraints
Set_clock –port –name clk –clock_cycle 10

# Input Constraints
arrival_time 2 data_en
arrival_time 3 data_in*

#Output Constraints
required time 7 data_stb
required_time 2 data_out*
```

Precision Constraint File

```
# Clock Constraints
create_clock clk –period 10 –domain main

# Input Constraints
set_input_delay 2 data_en –clock clk
set_input_delay 3 data_in* -clock clk

# Output Constraints
set_output_delay 3 data_stb –clock clk
set_output_delay 8 data_out* -clock clk
```

Clock Constraints

LeonardoSpectrum

LeonardoSpectrum uses 3 commands to fully specify a clock. These commands are:

```
set_clock -port -name .work.traffic.INTERFACE.clock -clock_cycle "10.000000"
set_clock -port -name .work.traffic.INTERFACE.clock -pulse_width "5.000000"
set_clock -port -name .work.traffic.INTERFACE.clock -clock_offset "2.00"
```

Precision

In Precision a single command, "create_clock" can be used to fully specify the clock as shown below:

```
create_clock { clk } -domain Domain0 -period 10 -waveform { 2 7 }
```

Notes:

a. The "-waveform" switch sets the rise and fall edges of the clock signal over an entire clock period. There must be a non-zero even number of edges and they are assumed to be alternating rise and fall. The first value in the list is a rising transition, typically the first rising transition after time zero.

b. Precision now requires users to specify a "clock domain". Clocks assigned to the same domain are considered synchronous and clock assigned to different domains are considered asynchronous.

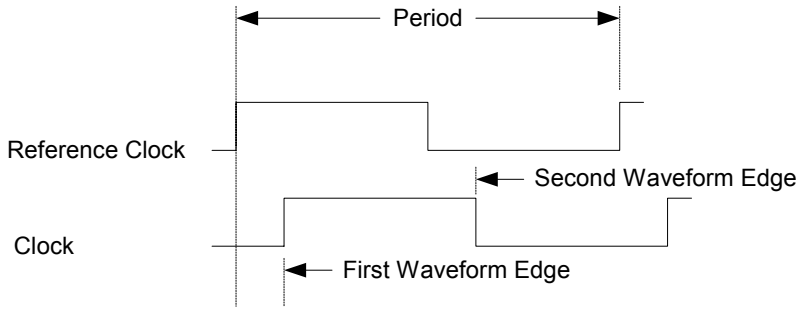

Figure 5 – Create_clock command Waveform

Input Constraints

LeonardoSpectrum

LeonardoSpectrum references input constraints to "time zero" which means that constraints and clocks all reference the same point in time. For this reason timing constraints only require a port name and delay value.

```
Arrival_time 4 read_en
```

Precision

The main difference in Precision is that timing constraints are now relative to a clock edge, not "time zero". This allows users to specify unique constraints on ports in designs with multiple clocks.

```
set_input_delay 4 read_en -clock clk
```

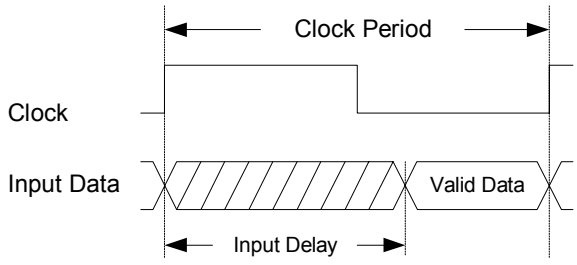

Figure 6 – set_input_delay waveform

Output Constraints

LeonardoSpectrum

Similar to the "arrival_time" constraint, LeonardoSpectrums output constraints are relative to "time zero" not a specific clock. The output constraint called "required_time" is an absolute time constraint on an output block of logic.

```
Required_time 7 data_out_strobe
```

Precision

There are two distinct differences between the output constraints of Precision and LeonardoSpectrum.

1. The output constraints, "set_output_delay", must be set relative to a clock edge.
2. The set_output_delay value is the opposite of "required_time". What this means is that if the user wanted to limit the clock to out delay of an output path to 7ns and the clock period were 10 ns, the "required_time" value would be 7; the "set_output_delay" value would be [clock period – output delay limit] or 3.

```
Set_output_delay 3 data_out_strobe –clock clk
```

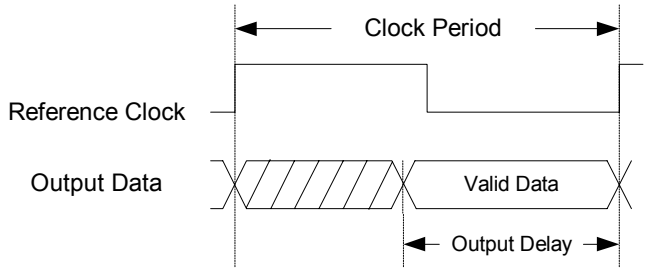

Figure 7 – set_output_delay waveform

Notes: *Set_input_delay and set_output_delay commands require that a constraint be relative to a clock edge. The default is "-rise". Users wishing to constrain a port to a falling clock edge must explicitly set the "-fall" switch on these commands.*

Path Constraints

LeonardoSpectrum

LeonardoSpectrum supports one path constraint called "set_multcycle_path" which is used to set both multi-cycle paths and false paths (by assigning a large number of cycles to the path).

```
set_multicycle_path -to {.work.traffic.INTERFACE.reg_state(0) } -value 2
set_multicycle_path  -from {.work.traffic.INTERFACE.sensor1 } -value 10000
```

Precision

There are 3 distinct differences that must be considered when applying path-based constraints using Precision.

1. Two path constraints exist, set_multicycle_path and set_false_path. Users no longer have to work around false paths with the set_multicycle_path command.
2. Path constraints are applied to a pin of an instance. LeonardoSpectrum would allow users to specify only the register name but not the pin.
3. The way that internal objects are referenced has changed. LeonardoSpectrum used an EDIF *lib.cell.view.instance* nomenclature where Precision allows a simple "/" to separate objects.

```
set_multicycle_path 2 -to { reg_state(0)/in }
set_false_path -from { sensor1 }
```

Applying Timing Constraints to a Design

After completing the conversion of the constraint file, there are 2 different ways to apply the constraints to a design.

Applying constraints from a separate file

To apply timing constraints, contained in a separate file, use the "add_input_file" command to add the constraint file as an input file. This command needs to appear prior to the "compile" and "synthesize" commands.

```
Add_input_file {traffic.v}
Add_input_file {traffic.sdc}
compile
synthesize
```

Note: *Precision will assume that a file with an .sdc suffix is a constraint file*

Applying constraints from the main script file

Port constraint commands can also be placed directly into the main synthesis script. When using this method constraint commands must be placed in the file after the compile command

```
Add_input_file {traffic.v}
Compile
Create_clock clk –period 10 –domain main
synthesize
```

Controlling IO Mapping

One significant difference between Precision and LeonardoSpectrum that users need to be aware of involves the implementation of IO's. Precision does the following IO operations automatically based on a timing driven algorithm.

- IO Register Mapping
- IO Driver Resizing
- IO constraint relaxation

IO Register Mapping

Precision will automatically assign registers to the IO pads when the internal reg-to-reg slack is positive. When no timing constraints are applied, then all possible registers will be assigned to the IO pad. Users can force registers into the IO pad by assigning an attribute to a port as follows

```
Set_attribute –name OUTFF –value TRUE –port done_stb   # Force out flops into IO
Set_attribute –name INFF –value TRUE –port read_mode    # Force in flops into IO
Set_attribute –name TRIFF –value TRUE –port load_en     # Force tri en flops into IO
```

IO Driver Sizing

Precisions will automatically "upsize" an output driver if the clock-to-out timing is not being met. Doing so can improve timing significantly. Users can force the use of a particular buffer using the "IOSTANDARD" attribute as shown below.

```
set_attribute -name IOSTANDARD -value SSTL3_II_DCI -port sensor1
```

IO Constraint Relaxation

If the clock to out timing is not being met then Precision will automatically "relax" the IO timing constraint. This is done to allow a design to go through place and route without errors. When Precision performs this operation the actual constraints on the database will change and be recorded in the "constraints report" file. The users original constraint file will not be modified. The functionality is only supported for Xilinx and can be disabled using the following command:

```
setup_place_and_route -flow {ISE 5.2} -command {Generate Vendor Constraint File} -
enable_auto_offset_relaxation 0
```

False Timing Regressions using Precision

Undoubtedly many LeonardoSpectrum customers may run existing designs through Precision and compare results. When doing this it is important to note that there are several reasons why results may appear worse when in fact they are the same or better. Most of these are a result of Precision's more accurate timing analysis and vendor constraint file generation system. This section will discuss the most common reasons for "false regressions".

Vendor Constraint File Generation

The most common reason for getting a false regression involves automatically generated vendor constraint files – especially the Xilinx UCF constraint file. This problem is exasperated when constraining a design using global frequencies. Precision's constraint file generation is comprehensive and accurate while LeonardoSpectrum generates a set of oversimplified global constraints. The following timing conditions go undetected when using an automatically generated LeonardoSpectrum constraint file but are detected using Precision generated constraint file.

- Combinatorial input to output paths
- Internal clock dividers using registers
- Timing violations resulting from input-to-reg or reg-to-output paths
- Output port timing relative to a phase shifted DCM internal clock
- Timing paths between multiple clocks that are synchronous
- Multiplied or divided clocks through DCMs

Solution

When comparing LeonardoSpectrum results to Precision results, the vendor constraint file for the LeonardoSpectrum netlist should be regenerated through Precision. You can read the EDIF netlist generated by LeonardoSpectrum into Precision, reapply constraints in the Precision environment, and regenerate the vendor constraint file. An example script is provided below:

```
# Setup the target technology
setup_design -manufacturer Xilinx -family VIRTEX-II -part 2V1500ff896 -speed 5

# Add the LeonardoSpectrum EDIF netlist
set_working_dir .
add_input_file Pscr_leo.edf

# Reconstrain the design using a global frequency (or use SDC constraints)
setup_design -frequency=150

# Disable "auto-relaxation" of IO Vendor constraints
setup_place_and_route -enable_auto_offset_relaxation 0

# Read the LeonardoSpectrum design into Precision
compile

# Reconstrain design using SDF constraints (Do this or use the global frequency)
create_clock clk -period 7.5 -domain ClockDomain1

Generate a Precision UCF file for the LeonardoSpectrum netlist
place_and_route {gen_vcf}

# Perform integrated place and route
place_and_route
```

Clock Domains

If your design has multiple clocks, then pay close attention to clock domains. LeonardoSpectrum does not support multiple asynchronous clocks and its support for synchronous clocks is limited. If clocks are specified as synchronous, then Precision will expand the clock waveforms out as many cycles as necessary to find the minimum delay between clock edges. If there is combinatorial logic that crosses these synchronous clock domains then Precision will detect this as a critical path and report on it. Also, Precision's UCF file generation will instruct Xilinx ISE to do the same. LeonardoSpectrum will not detect these paths.

IO Register Mapping

When a register is mapped into the IO the chip "on / off" timing is improved but the "max frequency", as reported by Xilinx, can worsen. The "max frequency" value reported in the Xilinx Trace timing report only reflects the

maximum reg-to-reg timing and does not take into account the chip IO timing. Pulling a register into the IO will increase the route delays on reg-to-reg paths and may lower this "max frequency" value.

LeonardoSpectrum by default will not map registers into either the Xilinx IOB or the Altera complex IO cells. Precision will do this automatically based on a timing driven algorithm. When LeonardoSpectrum is instructed to map registers into the IO, only top-level registers will be mapped but registers, buried in hierarchy, will not be mapped. To make matters even more complicated Xilinx ISE will map some registers into the IOB when the "-timing" switch is used with the "map" program.

Solution

There are two approaches to consider; first, observe the critical path and look for this condition. Placing mappable registers into the IO pad cell is considered the desired behavior, even at the expense of "max frequency", which is why Precision and Xilinx now do this automatically. The second approach would be to disable IO pad cell mapping in all tools to get a more accurate comparison. This can be done as follows:

```
# In Precision
set_attribute –name INFF –value FALSE –port [all_inputs]
set_attribute –name OUTFF –value FALSE –port [all_outputs]
set_attbibute –name TRIFF –value FALSE –port [all_inouts]

# In LeonardoSpectrum
set virtex_map_iob_registers FALSE    # For Xilinx
set altera_map_complex_ios FALSE      # For Altera

# In Xilinx
map <don't set –timing switch>
```

Tips and Tricks

1. To bring up the command line double-click on the "Log File" file in the Outputs folder of the project browser or issue the pull-down command "view ->Transcript Window"
2. The easiest way to migrate a design from LeonardoSpectrum to Precision is to use the GUI to create the synthesis script and constraint file automatically. When the synthesis process completes issue the pull-down menu command "file -> save project". To convert a project file into a script file simply add the command "synthesize" at the bottom of the file. When using a Project file as a script file remove the "go" and "setup_design –impl" commands from the project file first. These two commands are not necessary and may cause problems
3. When using Precision, if a global frequency is specified in the "Setup Design" dialog box a complete constraint file will be automatically generated. This appears as a file called "constraints report" in the output folder. This file can be added as a constraints file and modified to the exact constraints.
4. When setting constraints, a series of "get" commands are now supported for finding "typed" netlist objects (such as a pin or instance). These commands are an extremely powerful way to constrain a set of internal objects such as the CE pin on a bank of internal registers. Precisions reference manual includes several examples. Below is an example of setting a multi-cycle constraint on the CE pin of a 64 bit internal bus

   ```
   set_multicycle_path -to [get_pins I2/reg_clk_cnt*/CE] -value 2
   ```

5. To synthesize for area, leave the design unconstrained
6. To minimize runtime, enter a relaxed constraint that is easily met (like 1 MHZ)
7. You can double-click on any command, TCL or Precision, in the transcript window to bring up the manual page for that command
8. You can type the start of a command from the command line then hit the "tab" key to display all commands with that prefix. For example type "report_" <tab>" to display all our reporting commands
9. When using the schematic viewer 'trace forward" and "trace backwards" feature, select the pin of a cell rather than the entire cell. This will give better control over growing the schematic fragment
10. When using the schematic viewer query mode pop-up information box. If you want to record the information simply right-click on the object in the schematic then execute the pop-up command "Copy 'Query Info to Clipboard"
11. You can re-apply a constraint by modifying the SDC file, then right-clicking on the constraint file and execute the pop-up command "apply constraint file". You can regenerate the FPGA vendor constraint file by right-clicking the output folder from the project browser and execute the pop-up command "generate vendor constraint file".

Appendix A – Precision Netlist Object Constraints

Netlist Object	Constraints	Vendor Specific
Clock	Period	
	Clock Domain	
	Buffer Type	
	Pin Number	
Input Port	Input delay	
	Buffer Type	
	Pin Number	
	False_path	
	Multi-Cycle Path	
	Max Delay	
	Force Input Flop into Input Pad	Xilinx, Altera
Output Port	Output Delay	
	Buffer Type	
	Pin Number	
	False Path	
	Multi-Cycle Path	
	Max Delay	
	Force Output Flop into Output Pad	Xilinx, Altera
Net	Fanout	
	Preserve Signal	
	Preserve Driver	
	Assign to LOWSKEW	Xilinx
Input Pin	Output Delay	
	False Path	
	Multi Cycle Path	
	Max Delay	
Register	Don't Touch	
	Disable Pipeline DFF	
	Use DFF Enable	
	Max Delay	
Module	Preserve Hierarchy	
	Flatten Hierarchy	
	Don't_touch	
	Max Delay	
RAM	Use Distributed RAM	Xilinx
	RAM Block Type	Altera
	Max Delay	
Multiplier	Use Dedicated Multiplier	Xilinx, Altera
	Max Delay	
Tri-State Buffer	Preserve Tri State	

Appendix B – Block Based Design Script Example

```
## Design Setups
set_working_dir C:/Precision/Demos/block_design
setup_design -manufacturer Xilinx -family VIRTEX-II -part 2V40cs144 -speed 6
setup_design -addio=false
setup_design -vendor_constraint_file=false
setup_design -frequency=100

setup_design -impl top_impl_1

# The following "file delete" command is using tcl to delete an internal project file,
# which is causing issues when running the block based design flow in this manner.
# The file must be deleted after the "setup_design -impl" command is issued. This is only
# necessary because we are using project implementation directories but not using projects.
# This is a temporary issue and should be fixed in the 2003b software release

file delete [glob top_impl_1/*.psi]

# Optimize block A
add_input_file hdl/block_a.vhdl
compile
synthesize

# Optimize block B
remove_input_file hdl/block_a.vhdl
add_input_file hdl/block_b.vhdl
compile
synthesize

# Optimize block C
remove_input_file hdl/block_b.vhdl
add_input_file hdl/block_c.vhdl
compile
synthesize

# Optimize Top
setup_design -addio=true
setup_design -vendor_constraint_file=false

remove_input_file hdl/block_g.vhdl
add_input_file {
top_impl_1/block_a.xdb
top_impl_1/block_b.xdb
top_impl_1/block_c.xdb}
add_input_file hdl/top.vhdl

compile

# Set dont touch properties on previously optimized blocks
set_attribute -name DONT_TOUCH -value "TRUE" -instance U1
set_attribute -name DONT_TOUCH -value "TRUE" -instance U2
set_attribute -name DONT_TOUCH -value "TRUE" -instance U3

# Perform final synthesis
synthesize

# Perform place and route
place_and_route
```

Appendix C – Team Design Script Example

The following script example shows how individual designers may use separate scripts for blocks. A final "top" script does design assembly and final optimizations

```
# Block A Synthesis Script
set_working_dir C:/Precision/Demos/team_design/block_a
setup_design -manufacturer Xilinx -family VIRTEX-II -part 2V40cs144 -speed 6
setup_design -addio=false
setup_design -vendor_constraint_file=false
setup_design -frequency=100
setup_design -impl block_a_impl_1
add_input_file hdl/block_a.vhdl
compile
synthesize
```

```
# Block B Synthesis Script
set_working_dir C:/Precision/Demos/team_design/block_b
setup_design -manufacturer Xilinx -family VIRTEX-II -part 2V40cs144 -speed 6
setup_design -addio=false
setup_design -vendor_constraint_file=false
setup_design -frequency=100
setup_design -impl block_b_impl_1
add_input_file hdl/block_b.vhdl
compile
synthesize
```

```
# Block C Synthesis Script
set_working_dir C:/Precision/Demos/team_design/block_c
setup_design -manufacturer Xilinx -family VIRTEX-II -part 2V40cs144 -speed 6
setup_design -addio=false
setup_design -vendor_constraint_file=false
setup_design -frequency=100
setup_design -impl block_c_impl_1
add_input_file hdl/block_c.vhdl
compile
synthesize
```

```
# Top Level Design Assembly and Optimize Script
set_working_dir C:/Precision/Demos/team_design/top
setup_design -manufacturer Xilinx -family VIRTEX-II -part 2V40cs144 -speed 6
setup_design -addio=true
setup_design -vendor_constraint_file=true
setup_design -frequency=100
setup_design -impl top_impl_1

add_input_file ../block_a/block_a_impl_1/block_a.xdb
add_input_file ../block_b/block_b_impl_1/block_b.xdb
add_input_file ../block_c/block_c_impl_1/block_c.xdb
add_input_file hdl/top.vhdl

compile

# Set dont touch properties on previously optimized blocks
set_attribute -name DONT_TOUCH -value "TRUE" -instance U1
set_attribute -name DONT_TOUCH -value "TRUE" -instance U2
set_attribute -name DONT_TOUCH -value "TRUE" -instance U3

# Perform final synthesis
synthesize

# Perform place and route
place_and_route
```

Appendix D – Incremental Design Script Example

The following script shows how the team design script example in Appendix C can be extended to provide an incremental design flow on a block basis

```
# Re-synthesis Block C with functional change – put into implementation #2 folder
set_working_dir C:/Precision/Demos/team_design/block_c
setup_design -manufacturer Xilinx -family VIRTEX-II -part 2V40cs144 -speed 6
setup_design -addio=false
setup_design -vendor_constraint_file=false
setup_design -frequency=100
setup_design -impl block_c_impl_2
add_input_file hdl/block_c.vhdl
compile
synthesize
```

```
# Assemble the top-level design using the new Block C and existing blocks A and B
set_working_dir C:/Precision/Demos/team_design/top
setup_design -manufacturer Xilinx -family VIRTEX-II -part 2V40cs144 -speed 6
setup_design -addio=true
setup_design -vendor_constraint_file=true
setup_design -frequency=100
setup_design -impl top_impl_1

add_input_file ../block_a/block_a_impl_1/block_a.xdb
add_input_file ../block_b/block_b_impl_1/block_b.xdb
add_input_file ../block_c/block_c_impl_2/block_c.xdb
add_input_file hdl/top.vhdl

compile

# Set dont touch properties optimized blocks
set_attribute -name DONT_TOUCH -value "TRUE" -instance U1
set_attribute -name DONT_TOUCH -value "TRUE" -instance U2
set_attribute -name DONT_TOUCH -value "TRUE" -instance U3

# Perform final synthesis
synthesize

# Perform place and route
place_and_route
```

Appendix E – Common Attributes used to Control Optimization

Preserving hierarchy in a design

```
Set_attribute –name HIERARCHY –value PRESERVE –instance /*          # Globally
Set_attribute –name HIERARCHY –value PRESERVE –instance U1          # Instance level only
Set_attribute –name HIERARCHY –value PRESERVE –instance U1/*        # Instance and child level
```

Specifying Fanout

```
Set_attribute –name MAX_FANOUT –value 50 –net /*                          # Globally
Set_attribute –name MAX_FANOUT –value 50 –net U1/*                        # On an instance
Set_attribute –name MAX_FANOUT –value 50 –net U1/load_en            # On a net
```

Controlling Retiming of Registers

```
Setup_design –retiming=true | false                                  # Globally
Set_attribute –name DONT_RETIME –value TRUE –instance U1/*           # On an instance
Set_attribute –name DONT_RETIME –value TRUE –instance U1/reg_a       # On a register
```

Controlling Xilinx Block RAM Inference

```
Set_attribute –name BLOCK_RAM –value FALSE –instance U1/*            # On an instance
Set_attribute –name BLOCK_RAM –value FALSE –instance U1/ram_a        # On a register
```

Controlling Xilinx Block MULT Inference

```
Set_attribute –name BlOCK_MULT –value FALSE –instance /*            # Globally
Set_attribute –name BLOCK_MULT –value FALSE –instance U1            # Instance level only
Set_attribute –name BLOCK_MULT –value FALSE –instance U1/mult_a     # On a mult
```

Controlling Xilinx and Altera IO Register Mapping

```
Set_attribute –name OUTFF –value TRUE –port [all_outputs]   # Force all out flops into IO
Set_attribute –name INFF –value TRUE –port [all_inputs]     # Force all in flops into IO
Set_attribute –name TRIFF –value TRUE –port [all_inouts]    # Force all tri en flops into IO
set_attribute -name OUTFF -value FALSE -port int            # Forces specific port OUT of the IO
```

Controlling Altera RAM Inference

```
set_attribute -name ram_block_type -value AUTO | M512 | M4K | MegaRam -instance I2.mem
set_attribute -name ram_block_type -value AUTO | M512 | M4K | MegaRam -instance I2/*
```

Controlling Altera DSP Block Inference

```
set_attribute -name DEDICATED_MULT -value ON -instance modgen_mult  # on an instance
set_attribute -name DEDICATED_MULT -value ON -instance I2/*         # All mults in block I2
set_attribute -name DEDICATED_MULT -value ON -instance /*           # Entire design
```

Controlling Tri-State to Mux Conversion

```
Set_attribute –name PRESERVE_Z –value FALSE –net /*         # Globally
Set_attribute –name PRESERVE_Z –value FALSE –net U1         # Instance level only
Set_attribute –name PRESERVE_Z –value FALSE –net U1/net_a   # On a net driving a tristate
```

Controlling Actel Rad Hard Optimization (act1, act2, act3, 54sx)

```
Setup_design –radhardmethod= cc | tmr | tmr_cc | none               # Globally
Set_attribute –name radhardmethod –value tmr –inst U1               # Instance level only
Set_attribute –name radhardmethod –value tmr –inst U1/reg_a         # On a register
```

Preserving nets and logic

```
Set_attribute –name preserve_signal –value TRUE –net U1             # Preserves net name
Set_attribute –name preserve_driver –value TRUE –net U1/net_a       # Preserve redundant logic
```