



NXmap 2.9.5

User Manual

Table of Content

TABLE OF CONTENT	2
1 COPYRIGHT	5
2 INTRODUCTION.....	6
3 PACKAGE DESCRIPTION.....	7
3.1 INSTALLATION.....	7
3.2 RUN NXMAP	7
3.3 RUN NXPYTHON.....	7
4 NANOXPLORE DESIGN FLOW	8
4.1 CREATE PROJECT.....	9
4.2 SYNTHESIZE.....	9
4.3 PLACE & ROUTE.....	9
4.4 BITSTREAM GENERATION	9
4.5 DESIGN NETLIST GENERATION AND SIMULATION	9
4.6 STATIC TIMING ANALYSIS.....	9
4.6.1 <i>Timing path</i>	10
4.6.2 <i>Timing domain</i>	10
4.6.3 <i>Timing reports</i>	14
4.7 LOGGING	17
4.8 REPORT GENERATIONS.....	17
4.8.1 <i>Instances report</i>	18
4.8.2 <i>Ports report</i>	18
4.8.3 <i>Power consumption report</i>	18
4.8.4 <i>Timing report</i>	18
5 NXMAP SPECIFICATION	19
5.1 GRAPHICAL USER INTERFACE OVERVIEW (NXMAP)	19
5.1.1 <i>Load project</i>	20
5.1.2 <i>Recent projects</i>	21
5.1.3 <i>View</i>	22
5.1.4 <i>About</i>	23
5.1.5 <i>Settings</i>	24
5.1.6 <i>Command lines</i>	25
5.1.7 <i>Keyboard shortcuts</i>	25
5.2 DESIGN FLOW.....	26
5.2.1 <i>Synthesize</i>	26
5.2.2 <i>Place & Route</i>	26
5.2.3 <i>Static Timing Analysis</i>	26
5.3 GRAPHICAL INSPECTION.....	26
5.3.1 <i>Select command</i>	27
5.3.2 <i>Inspect command</i>	29
6 NXPYTHON SPECIFICATION	31
6.1 GENERAL COMMANDS.....	31
6.1.1 <i>createProject([workingDirectory])</i>	31
6.1.2 <i>getErrorCount()</i>	32
6.1.3 <i>getWarningCount()</i>	33
6.1.4 <i>info([object])</i>	34
6.1.5 <i>printError(message)</i>	35
6.1.6 <i>printText([message])</i>	36
6.1.7 <i>printWarning(message)</i>	37
6.2 PROJECT RELATED METHODS.....	38
6.2.1 <i>addBlackBox(name, type, mapping, interface)</i>	38
6.2.2 <i>addFalsePath(from_list, to_list)</i>	40

6.2.3	<i>addFile([library], file)</i>	41
6.2.4	<i>addFiles([library], fileList)</i>	42
6.2.5	<i>addInterface(name, location)</i>	43
6.2.6	<i>addInterfaces(interfaces)</i>	44
6.2.7	<i>addIP(name)</i>	45
6.2.8	<i>addMappingDirective(name, type, mapping)</i>	46
6.2.9	<i>addMaxDelayPath(from_list, to_list, delay)</i>	47
6.2.10	<i>addMemoryInitialization(name, type, file)</i>	48
6.2.11	<i>addMinDelayPath(from_list, to_list, delay)</i>	49
6.2.12	<i>addMulticyclePath(from_list, to_list, cycle_count)</i>	50
6.2.13	<i>addPad(name, parameters)</i>	51
6.2.14	<i>addPads(pads)</i>	53
6.2.15	<i>addPin(name, location)</i>	54
6.2.16	<i>addPins(pins)</i>	55
6.2.17	<i>addParameter(name, value)</i>	56
6.2.18	<i>addParameters(parameters)</i>	57
6.2.19	<i>addPLLLocation(name, location)</i>	58
6.2.20	<i>addVerilogIncludeDirectories(directoryList)</i>	59
6.2.21	<i>addVerilogIncludeDirectory(directory)</i>	60
6.2.22	<i>addWFGLocation(name, location)</i>	61
6.2.23	<i>createAnalyzer()</i>	62
6.2.24	<i>createClock(target, name, period, [rising, falling])</i>	63
6.2.25	<i>createGeneratedClock(source, target, name, relationship)</i>	64
6.2.26	<i>createSimulator()</i>	66
6.2.27	<i>developCKGs()</i>	67
6.2.28	<i>destroy()</i>	68
6.2.29	<i>display()</i>	69
6.2.30	<i>exportAsIPCore(file, options)</i>	70
6.2.31	<i>generateBitstream(file, [frames1], [frames2])</i>	71
6.2.32	<i>getDirectory()</i>	72
6.2.33	<i>getTopCellName()</i>	73
6.2.34	<i>getVariantName()</i>	74
6.2.35	<i>load(file)</i>	75
6.2.36	<i>place()</i>	76
6.2.37	<i>reportInstances()</i>	77
6.2.38	<i>reportPorts()</i>	79
6.2.39	<i>reportPowerConsumption()</i>	80
6.2.40	<i>resetTimingConstraints()</i>	81
6.2.41	<i>route()</i>	82
6.2.42	<i>save(file)</i>	83
6.2.43	<i>savePorts(filename)</i>	84
6.2.44	<i>setCaseAnalysis(value, net_list)</i>	85
6.2.45	<i>setClockGroup(group1_list, group2_list, option)</i>	86
6.2.46	<i>setInputDelay(clock, clock_mode, minimum_delay, maximum_delay, port_list)</i>	87
6.2.47	<i>setOption(name, value)</i>	88
6.2.48	<i>setOptions(options)</i>	91
6.2.49	<i>setOutputDelay(clock, clock_mode, minimum_delay, maximum_delay, port_list)</i>	92
6.2.50	<i>setTopCellName([library], name)</i>	93
6.2.51	<i>setVariantName(variant)</i>	94
6.2.52	<i>synthesize()</i>	95
6.3	STATIC TIMING ANALYSIS RELATED METHODS	96
6.3.1	<i>destroy()</i>	96
6.3.2	<i>launch(parameters)</i>	97
6.3.3	<i>Launching timing analyzer steps</i>	98
6.4	SIMULATION RELATED METHODS	99
6.4.1	<i>addTestBench([library], testBench)</i>	99
6.4.2	<i>addTestBenches([library], fileList)</i>	100
6.4.3	<i>addWave(wave)</i>	101
6.4.4	<i>addWaves(waveList)</i>	102
6.4.5	<i>destroy()</i>	103

6.4.6	<i>launch(graphical)</i>	104
6.4.7	<i>setTestBenchTop(testBenchTop)</i>	105
6.4.8	<i>setWorkingDirectory(workingDirectory)</i>	106
6.5	ARGUMENT RELATED METHODS	107
6.5.1	<i>getClock(clock_name)</i>	107
6.5.2	<i>getClockNet(clock_net_name)</i>	108
6.5.3	<i>getClocks(clock_name_expression)</i>	109
6.5.4	<i>getInstances(instance_name)</i>	110
6.5.5	<i>getModels(model_name)</i>	111
6.5.6	<i>getPort(port_name)</i>	112
6.5.7	<i>getPorts(port_name_expression)</i>	113
6.5.8	<i>getRegister(register_name)</i>	114
6.5.9	<i>getRegisterClock(register_name)</i>	115
6.5.10	<i>getRegisters(register_name_expression)</i>	116
6.5.11	<i>getWFGOutput(wfg_name)</i>	117
6.6	FULL SCRIPT EXAMPLE	118
Figure 1:	NanoXplore design flow	8
Figure 2:	Data and clock delay between a source and destination register	10
Figure 3:	Examples of timing domains D1, D2, D3 in a simple design	11
Figure 4:	Hold/Setup time relationship for active high launch and active high latch clocks	12
Figure 5:	Hold/Setup time relationship for active high launch and active low latch clocks	12
Figure 6:	Hold/Setup time relationship for active low launch and active high latch clocks	13
Figure 7:	Hold/Setup time relationship for active low launch and active low latch clocks	13
Figure 8:	Summary.timing example	14
Figure 9:	Example 1 of critical paths table	16
Figure 10:	Example 2 of critical paths table	16
Figure 11:	Example of path detail table	17
Figure 12:	<i>nxmap</i> main window	19
Figure 13:	<i>nxmap</i> menu	20
Figure 14:	Load an existing project	21
Figure 15:	Load a recent project	22
Figure 16:	Graphical representation of a design on <i>nxmap</i>	23
Figure 17:	About <i>nxmap</i>	24
Figure 18:	Settings tab	25
Figure 19:	Static Timing Analysis option in <i>nxmap</i>	26
Figure 20:	Command bar options	27
Figure 21:	Element options in Command bar	27
Figure 22:	List of matching elements	27
Figure 23:	A critical path in a timing domain	28
Figure 24:	Filtering the Matching elements	29
Figure 25:	Inspect command	30
Figure 26:	Inspect WFG	30
Figure 27:	<i>nxmap</i> GUI called by <i>display()</i> method	69
Figure 28:	Output tables from <i>reportInstances()</i> method	78

1 Copyright

All the contents of this document are protected by the copyright law. They may not be disclosed to third parties or copied or duplicated in any form without the written consent of NanoXplore.

2 Introduction

This document is intended to guide users of NanoXplore *nxmap* software through all the steps involved in the design flow and the options available in NanoXplore design suite.

- *nxmap* is a graphical interface that allows user to view and compile an existing project. To create a new project, the user has to use *nxpython* tool.
- *nxpython* is a wrapper around Python executable that allows user to control *nxmap* software. As a wrapper, it fully supports Python syntax, structures and external modules.

To better explain the operations executed in this document for *nxmap* graphical interface, an example of *dec_viterbi* is considered which is essentially a VHDL implementation of the Viterbi decoder.

To use the *nanoxmap* module in python, user must first call the following command in *nxpython*:

```
from nanoxmap import *
```

All operations executed in this document for *nxpython* can be applied to the provided VHDL "simple" example.

Currently, *nxmap* is provided to be run on a 64 bits Linux workstation running one of the following distributions:

- RedHat Enterprise Linux alike version 6 or 7,
- Ubuntu 14.04 LTS, 16.04 LTS or 18.04 LTS
- Debian 9

nxmap needs Python but does not provide its own version to prevent conflicts. Thus, Python is required to be installed on user's workstation. Compatible Python versions include:

- 2.6 for RedHat Enterprise Linux alike version 6,
- 2.7 for others.

For any other distribution or platform, please contact NanoXplore team at support@nanoxplore.com.

Note

When a command's argument is surrounded by brackets ([argument]), it means that this argument is optional.

3 Package description

The provided package NXmap-2.9.5.tar.gz contains the following directories:

- NXmap root directory
 - o 2.9.5 version
 - bin binary files for each supported Linux distribution
 - doc documentation files in pdf format
 - example project examples with design sources in VHDL
 - lib64 dynamic libraries and Python modules for each supported Linux distribution and associated Python version
 - share additional files (vhdl libraries, simulation libraries, etc...)

3.1 Installation

To install *nxmap*, the user needs to unpack NXmap-2.9.5.tar.gz file into the installation directory (e.g. /opt/NanoXplore) using the following command:

```
$> tar xzf NXmap-2.9.5.tar.gz -C /opt/NanoXplore
```

To set the license, it is required to export the following shell variable:

```
$> export LM_LICENSE_FILE=27000@servername
```

where *servername* is the hostname of the server running the license daemon and 27000 is the port the daemon listens to.

3.2 Run nxmap

To run *nxmap*, use the following command:

```
$> /opt/NanoXplore/NXmap/2.9.5/bin/nxmap
```

3.3 Run nxpython

To run *nxpython*, use the following command:

```
$> /opt/NanoXplore/NXmap/2.9.5/bin/nxpython
```

Note

To use *nxpython* command lines, do not forget to load nanoxmap libraries:
>>> from nanoxmap import *

4 NanoXplore design flow

As mentioned earlier, user can perform the design flow by graphical user interface (*nxmap*) or Python command-line interface (*nxpython*). Figure 1 shows the overview of a multiplatform design flow by NanoXplore.

Note

NanoXplore design environment does NOT include any simulation tool. However, user can decide to use a third-party tool to simulate or verify the design by using HDL files generated by NanoXplore design flow.

As shown in Figure 1, user can employ *nxpython* for the entire design flow at once or user can specify the targeted steps to be performed. It can be useful when debugging a design or customizing a run of the flow (for more information about *nxpython*, please refer to section 6).

nxmap provides graphical interface from synthesis to routing phase of the design that helps user to investigate and inspect the design at various stages (for more information about *nxmap*, please refer to section 5).

nxpython is a Python wrapper that allows user to control *nxmap* software. As a wrapper, it fully supports Python syntax, structures and external modules.

Both *nxpython* and *nxmap* are fully compatible with each other such that the files generated by one can be used by the other at any given stage of the design flow.

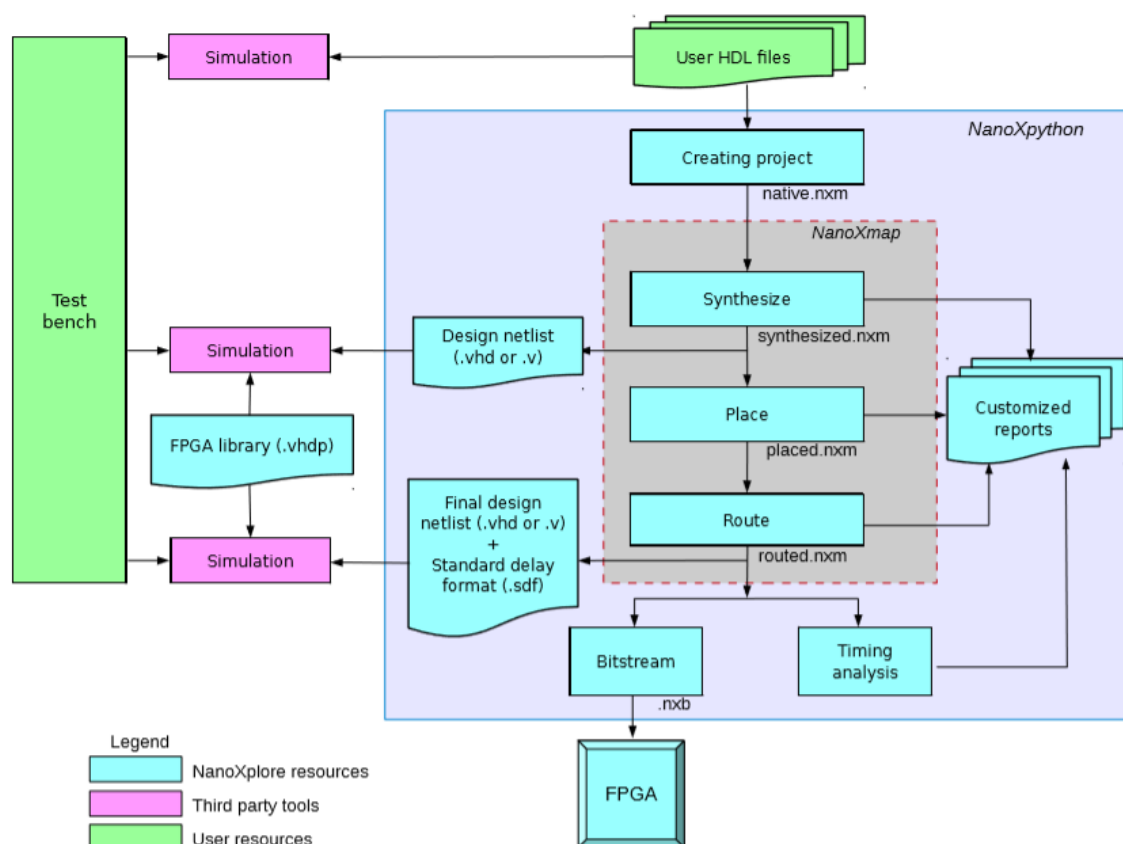


Figure 1: NanoXplore design flow

4.1 Create project

To create a new project, user needs to run a few Python commands from the *npython* interface (please refer to section 6.1.1).

Creating a new project starts with creating an empty project, configuring it and setting several optional settings.

4.2 Synthesize

Synthesis translates the design hardware description into a netlist defining the used FPGA resources. The “Synthesize” step of design flow can be executed in both *nxmap* and *npython* environments (please refer to section 5.2.1 and 6.2.52 respectively).

4.3 Place & Route

The steps “Place” and “Route” are performed on the synthesized design generated in the previous step. In these steps solutions are found to fit user's design to the FPGA physical architecture. The task of placement is to calculate the positions of the basic instance elements. Then the routing process assign signals to routing resources in order to successfully route all signals while achieving a given overall performance.

The “Place” and “Route” steps of design flow can be executed in both *nxmap* and *npython* environments (please refer to section 5.2.2 for *nxmap*, 6.2.36 and 6.2.41 for *npython*).

4.4 Bitstream generation

The bitstream is used to program the FPGA, this step can only be launched after successful routing step. Currently, the bitstream generation can only be performed by *npython* (please refer to section 6.2.30).

4.5 Design netlist generation and Simulation

In *npython*, it is possible to generate post-synthesis, post-place and/or post-route design netlists (Verilog or VHDL) which can be useful for simulation purposes (please refer to section 6.2.42).

For design verification after the synthesis, place and/or route, third-party simulation tools can be employed where generated design netlist is given along with the FPGA elements library. This library is provided to the user in VHDL protected format. The library file can be found at the following directory path:

```
$> /opt/NanoXplore/NXmap/2.9.5/share/modelsim/nxLibrary.vhdp
```

Note

For now, NanoXplore FPGA library files are compatible with Modelsim.

4.6 Static Timing Analysis

In *nxmap*, Static Timing Analysis (STA) can be performed by using its Timing Analyzer tool to analyze, debug, and validate the timing performance of the design. This feature produces a detailed report on the timing characteristics of the design, i.e. timing domains in the design, longest/shortest paths in the domain, path delay, data arrival time, clock skew, etc.

Static Timing Analysis can be performed by *nxmap* (see section 5.2.3) and *npython* (see section 6.3.3).

4.6.1 Timing path

Timing path is the connection between two certain nodes in the design i.e. connection between primary input and output ports, connection between primary input ports/clock pins and data input pins of the sequential elements, and connection between intermediate clock pins and primary output ports.

4.6.1.1 Data delay

In *nxmap*, data delay is the delay of a timing path. For example, the delay between the launch edge of the clock at the source register and data input pin of the destination register. In Figure 2, the path for the data delay between source register DFF1 and destination register DFF2 is shown in blue.

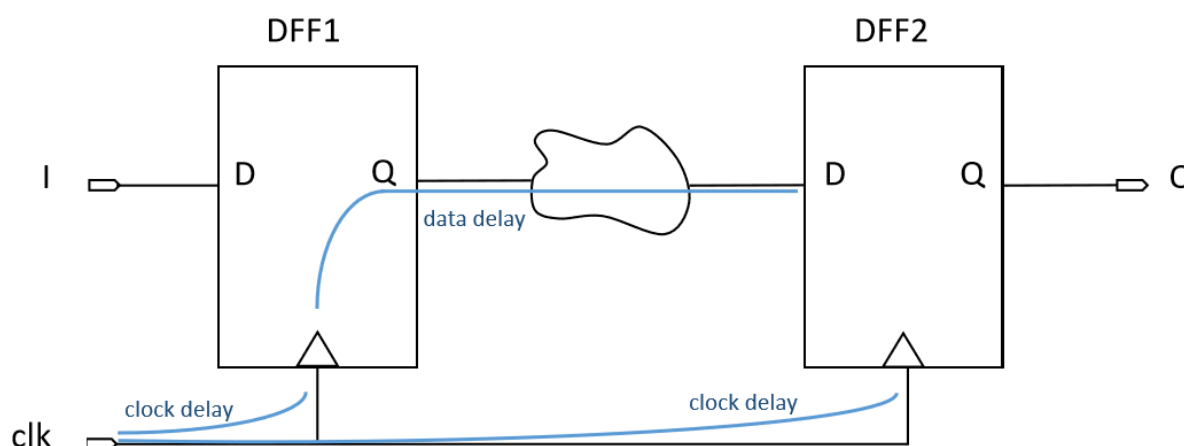


Figure 2: Data and clock delay between a source and destination register

4.6.1.2 Clock skew

Clock skew is the difference between the latch clock delay at the destination register and launch clock delay at the source register. Considering the example shown in Figure 2, clock skew is calculated as follows:

$$\text{clock skew} = \text{Clock delay at DFF2} - \text{Clock delay at DFF1}$$

4.6.1.3 Setup/Hold time

Setup time is the minimum amount of time required for a signal to be held stable before the clock transition. Hold time is the minimum amount of time required for a signal to retain its value after the clock transition. In *nxmap*, the analysis of setup/hold time is performed in order to calculate the data arrival time which will be explained shortly.

4.6.1.4 Recovery/Removal time

Recovery time is the minimum amount of time required for an asynchronous control signal to be held stable after its deassertion before the next active clock edge. The recovery time calculation corresponds to the setup time calculation but for the asynchronous control signals i.e. reset.

Similarly, removal time is the minimum amount of time for an asynchronous signal to be held stable before its deassertion after the previous active clock signals. The removal time calculation corresponds to the hold time calculation but for the asynchronous signals.

4.6.2 Timing domain

In *nxmap*, the term 'domain' is used to define a group of certain timing paths. As mentioned earlier, the timing paths include the connection between primary input and output ports, connection between primary input ports/clock pins and data input pins of the sequential elements, and connection between intermediate clock pins and primary output ports.

Depending on the initial and terminal nodes of the timing path, a number of timing domains can be defined in a design. Paths having a common clock signal at the source node as well as a common clock signal at

the destination node are considered to be in the same domain. Also, the primary input and output ports connected by combinatorial logic are considered to be in the same domain.

For example in Figure 3, three domains D1, D2 and D3 are shown between input and clock "clk" (D1: input to clk), between clock "clk" of source registers and clock "clk" of destination registers (D2: clk to clk), and between clock "clk" and output (D3: clk to output) respectively.

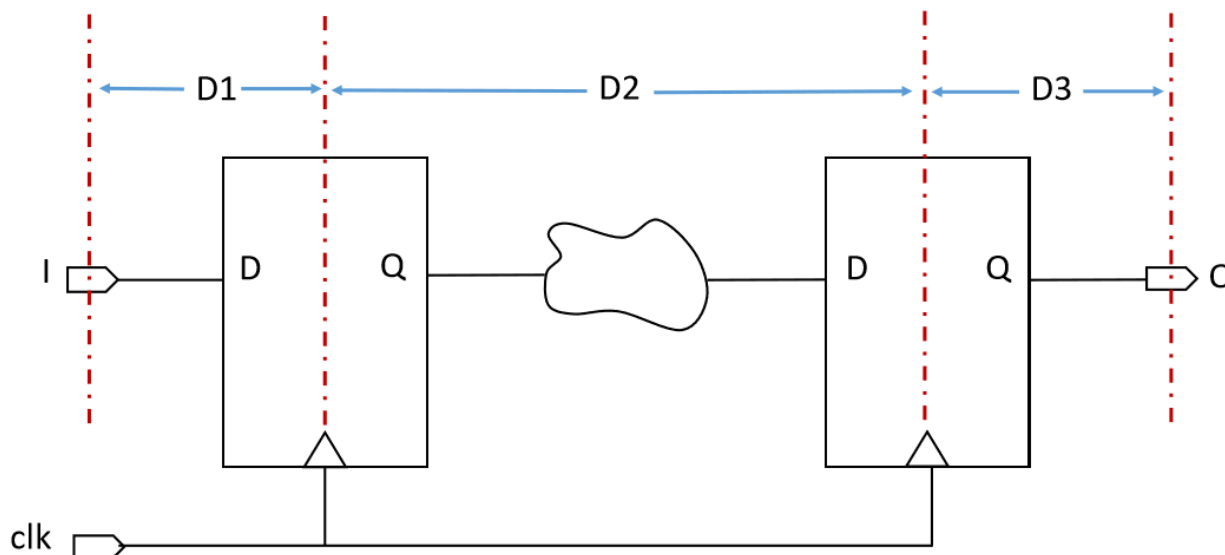


Figure 3: Examples of timing domains D1, D2, D3 in a simple design

4.6.2.1 Data arrival time

In *nxmap*, data arrival time includes the overall data delay, clock skew and setup/hold time (recovery/removal) for a critical timing path in a domain. Based on the setup/hold verification, data arrival time can be either maximum or minimum. In both cases, data arrival time is calculated as follows:

Maximum data arrival time =
 $\max\{\forall(\text{path delay} = \text{data delay} - \text{clock skew} + \text{setup or recovery}) : \text{path} \in \text{domain}\}$
 Minimum data arrival time =
 $\min\{\forall(\text{path delay} = \text{data delay} - \text{clock skew} - \text{hold or removal}) : \text{path} \in \text{domain}\}$

4.6.2.2 Setup/Recovery, Hold/Removal time relationship

The transfer of data signal from one sequential element to another requires certain timing checks for correct functionality. It involves hold and setup time verification with respect to launch and latch edge of the clock at the source and destination sequential element respectively.

Launch edge is the active clock edge at which data is sent from a source sequential element. Latch edge is the active clock edge at which the destination sequential element captures the data coming from the source.

Between launch and latch clock, hold and setup time relationship is established which is essentially the delay between the active edges of these clocks. This relationship is then used to determine positive or negative slack, validating the timing characteristics of the paths in the domain. Following are all the four cases of the active launch and latch clock edges.

- Case 1:

If both launch and latch clocks have active high edge, hold relationship is established between the current launch edge and the previous or current latch edge ensuring that the data is not captured by the latch edge coming before the launch edge. Similarly, setup relationship is established between the launch edge and the next latch edge ensuring that the data is captured by the latch edge coming after the launch edge.

In Figure 4, hold and setup relationship is shown between a launch and latch clock with no skew. In case of having different frequencies for launch and latch clock, hold relationship is established between the active launch and latch edges with the minimum difference. Thus, hold relationship gives the minimum delay allowed between launch and latch edges. Whereas, the setup relationship is established between launch and latch edges with minimum difference, hence giving the maximum allowed delay between launch and latch edges.

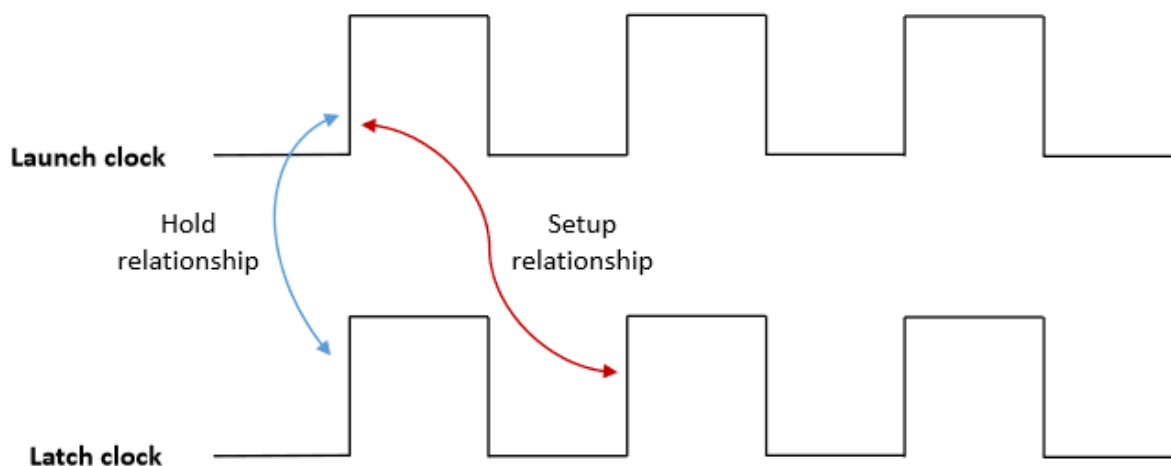


Figure 4: Hold/Setup time relationship for active high launch and active high latch clocks

- Case 2:

If launch and latch clocks have active high and active low edges respectively, in this case hold relationship is established between active high launch edge and the previous active low latch edge with the minimum difference. Similarly, setup relationship is established between launch edge and the next latch edge with the minimum difference.

Figure 5 shows hold and setup relationship between active high launch clock and active low latch clock.

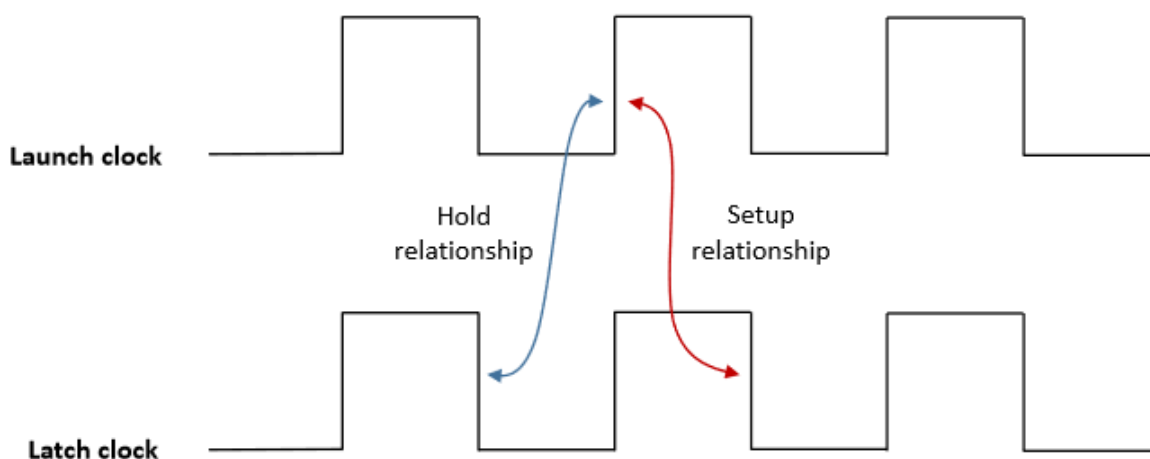


Figure 5: Hold/Setup time relationship for active high launch and active low latch clocks

- Case 3:

For the active low launch clock and active high latch clock, the hold relationship gives the minimum delay between active low launch edge and previous active high latch edge. Similarly, setup relationship gives the minimum delay between the active low launch edge and the very next active high latch edge as shown in Figure 6.

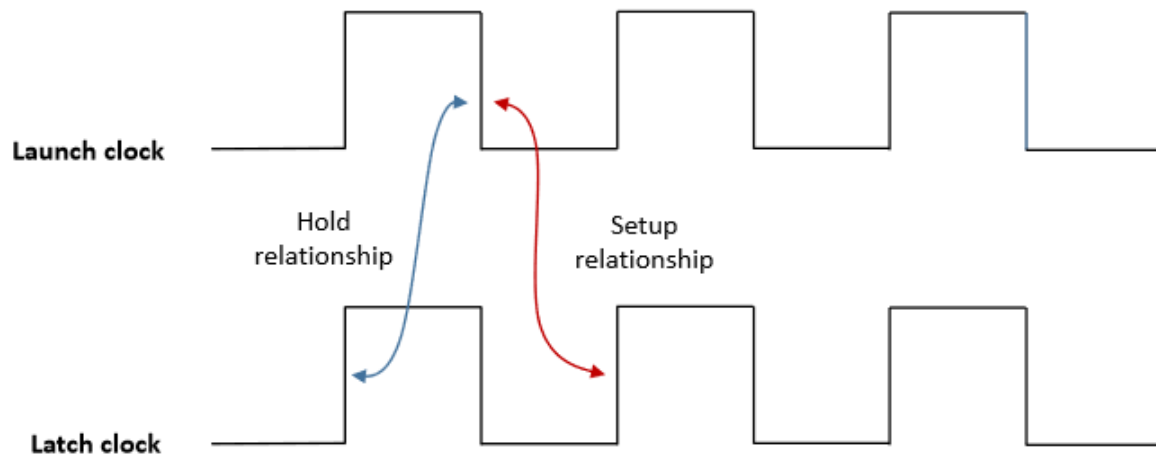


Figure 6: Hold/Setup time relationship for active low launch and active high latch clocks

- Case 4:

If both launch and latch clocks have active low edges, the hold relationship is established between active low launch edge and previous active low latch edge. Similarly, setup relationship is established between the active low launch edge and the very next active low latch edge as shown in Figure 7.

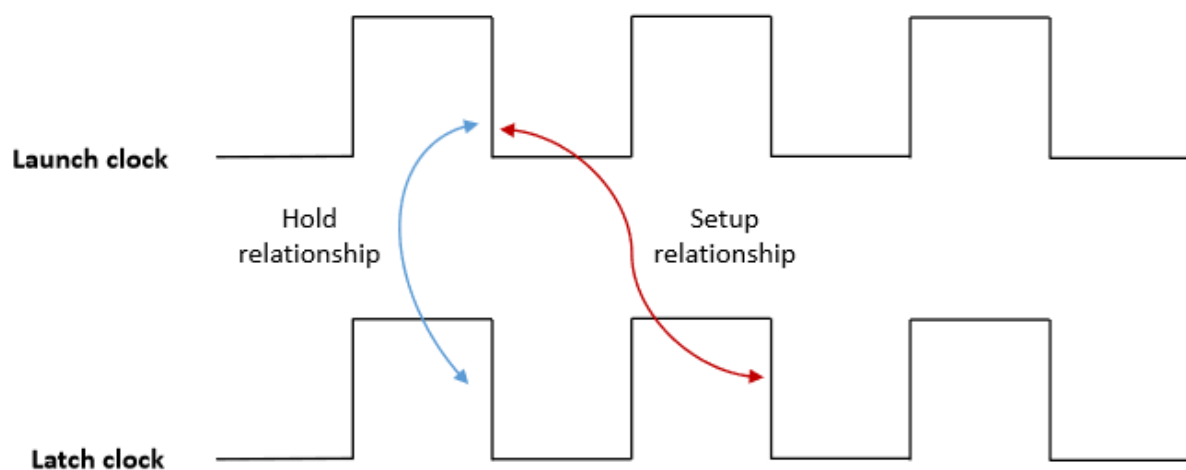


Figure 7: Hold/Setup time relationship for active low launch and active low latch clocks

4.6.2.3 Slack calculation

In *nxmap*, setup/recovery slack corresponds to the difference between maximum setup/recovery relationship and maximum data arrival time whereas hold/removal slack corresponds to the difference between minimum data arrival and minimum hold/removal relationship.

Setup/Recovery slack = Maximum setup/recovery relationship – Maximum data arrival time
 Hold/Removal slack = Minimum data arrival time – Minimum hold/removal relationship

4.6.3 Timing reports

In *nxmap* and *nxpython*, timing analyzer generates a set of reports presenting respectively the summary of analysis, critical paths of each domain, and violation information. All the timing reports are automatically saved in files (.timing) in 'logs' directory after a successful analyzing of the routed design.

4.6.3.1 STA summary

The file "Summary.timing" contains the report of the timing domains as well as hold/removal and setup/recovery summaries in the design. Such domains are grouped by categories:

- Domain
 - Source (beginning of the domain)
 - Target (ending of the domain)
- Frequency
 - Required (restricted frequency set by 'createClock' method)
 - Maximum (computed frequency)
- Hold/Removal Summary
 - Slack
 - Minimum Data Arrival Time
 - Minimum Required Relationship
- Setup/Recovery Summary
 - Slack
 - Maximum Data Arrival Time
 - Maximum Required Relationship

Reporting domains (Case Analysis: Typical)					
Domain			Frequency		
Source	Target		Required	Maximum	
Input	CLK (Rising)		–	–	
CLK (Rising)	CLK (Rising)		142,857MHz	138,543MHz	
CLK (Rising)	Output		–	–	
Total					
Hold/Removal Summary			Setup/Recovery Summary		
Slack	Minimum Data Arrival Time	Minimum Required Relationship	Slack	Maximum Data Arrival Time	Maximum Required Relationship
–	–968ps	–	–	16,501ns	–
1,253ns	1,253ns	0ps	–218ps	7,218ns	7,000ns
–	5,955ns	–	–	6,235ns	–
3					

Figure 8: Summary.timing example

Note

- The values of slack and of required relationship are only available for synchronous domains.
- Maximum frequency is only computed for paths where the source and target registers are driven by the same clock.

A domain whose source and target registers are driven by defined clocks or driven by the same clock is conceded as synchronous by default. For the Synchronous domains the required relationship can be

established for timing verifications. In the case of hold/removal timing verification, the slack represents the difference between the minimum data arrival time and the minimum required relationship. Similarly, in the case of setup/recovery timing verification, the slack represents the difference between maximum required relationship and maximum data arrival time. For the domains whose source and target registers are driven by the same undefined clock, the required relationships can't be established before analysis. In this case, the minimum required hold/removal relationship is considered as 0ps whereas maximum arrival time is considered as the maximum required setup/recovery relationship.

Generally, a negative slack indicates a violation of timing check. However, in the case where the required hold/removal relationship can't be established (hence, minimum required relationship is 0ps), the negative slack does not fully indicate the violation as the minimum data arrival time is negative. Whereas in the case where the relationship exists, the negative data arrival time producing a negative slack would be a timing violation.

After timing analysis, critical paths for each domains are detailed in a set of timing domain reports (.timing) which are named with the same domains' names. In a domain report, there are three parts:

1. a summary of the presenting domain (similar to Summary.timing);
2. two lists of critical longest paths and shortest paths;
3. detail tables of each found path.

4.6.3.2 Critical paths table

The critical paths tables show the lists of longest paths and shortest paths for the given timing domain of the design. The details of such critical paths include:

- Slack
- Source (beginning of the path)
- Target (ending of the path)
- Data Delay
- Clock Skew
- Setup/Recovery (longest only)
- Hold/Removal (shortest only)
- Depth
- Note (falling/rising edges of launch and latch clocks)

Reporting longest paths for domain CLK → CLK (Case Analysis: Typical)				
No.	Slack	Source	Target	
1	-679ps	{-}:UUT2 dout_reg[155].CK	{-}:sdrv_dataout[155].EI	
2	-610ps	{-}:UUT2 dout_reg[0].CK	{-}:sdrv_dataout[0].EI	
3	-556ps	{-}:UUT2 dout_reg[122].CK	{-}:sdrv_dataout[122].EI	
4	-531ps	{-}:UUT2 key_reg[0].CK	{-}:sdrv_dataout[0].EI	
5	-525ps	{-}:UUT2 key_reg[3].CK	{-}:sdrv_dataout[69].EI	
6	-512ps	{-}:UUT2 dout_reg[22].CK	{-}:sdrv_dataout[22].EI	
7	-493ps	{-}:UUT2 dout_reg[69].CK	{-}:sdrv_dataout[69].EI	
8	-463ps	{-}:UUT2 key_reg[1].CK	{-}:sdrv_dataout[22].EI	
9	-460ps	{-}:UUT2 dout_reg[126].CK	{-}:sdrv_dataout[126].EI	
10	-425ps	{-}:UUT2 dout_reg[139].CK	{-}:sdrv_dataout[139].EI	
Total				
Data Delay	Clock Skew	Setup/Recovery	Depth	Note
6,764ns	502ps	1,417ns	4	(RR)
7,059ns	475ps	1,026ns	4	(RR)
6,642ns	503ps	1,417ns	4	(RR)
6,979ns	474ps	1,026ns	4	(RR)
6,997ns	498ps	1,026ns	4	(RR)
6,985ns	499ps	1,026ns	4	(RR)
6,965ns	498ps	1,026ns	4	(RR)
6,938ns	501ps	1,026ns	4	(RR)
6,933ns	499ps	1,026ns	4	(RR)
6,511ns	503ps	1,417ns	4	(RR)
			10	

Figure 9: Example 1 of critical paths table

Reporting shortest paths for domain CLK → CLK (Case Analysis: Typical)				
No.	Slack	Source	Target	
1	1,249ns	{-}:UUT1 Gen_seq[2].seq_i temp_reg[1].CK	{-}:UUT1 Gen_seq[2].seq_i temp_reg[0].14	
2	1,249ns	{-}:UUT1 Gen_seq[2].seq_i temp_reg[1].CK	{-}:UUT2 dout_reg[33].14	
3	1,250ns	{-}:UUT1 Gen_seq[2].seq_i temp_reg[12].CK	{-}:UUT2 dout_reg[44].14	
4	1,250ns	{-}:UUT1 Gen_seq[5].seq_i temp_reg[29].CK	{-}:UUT1 Gen_seq[5].seq_i temp_reg[28].14	
5	1,250ns	{-}:UUT1 Gen_seq[5].seq_i temp_reg[29].CK	{-}:UUT2 dout_reg[157].14	
6	1,251ns	{-}:UUT1 Gen_seq[2].seq_i temp_reg[20].CK	{-}:UUT2 dout_reg[52].14	
7	1,252ns	{-}:UUT1 Gen_seq[2].seq_i temp_reg[20].CK	{-}:UUT1 Gen_seq[2].seq_i temp_reg[19].14	
8	1,253ns	{-}:UUT1 Gen_seq[2].seq_i temp_reg[31].CK	{-}:UUT1 Gen_seq[2].seq_i temp_reg[30].14	
9	1,253ns	{-}:UUT1 Gen_seq[2].seq_i temp_reg[31].CK	{-}:UUT2 dout_reg[63].14	
10	1,255ns	{-}:UUT1 Gen_seq[2].seq_i temp_reg[13].CK	{-}:UUT2 dout_reg[45].14	
Total				
Data Delay	Clock Skew	Hold/Removal	Depth	Note
1,048ns	95ps	-296ps	2	(RR)
1,048ns	95ps	-296ps	2	(RR)
1,052ns	95ps	-293ps	2	(RR)
1,053ns	96ps	-293ps	2	(RR)
1,054ns	96ps	-292ps	2	(RR)
1,052ns	95ps	-294ps	2	(RR)
1,052ns	94ps	-294ps	2	(RR)
1,052ns	96ps	-297ps	2	(RR)
1,051ns	94ps	-296ps	2	(RR)
1,056ns	95ps	-294ps	2	(RR)
			10	

Figure 10: Example 2 of critical paths table

4.6.3.3 Path detail

Following the critical paths table, all the listed critical paths are reported in detail. For each part of the path, the table contents the following information:

- source (beginning of the path)
- target (ending of the path)
- routing delay

- internal delay
- cumulated delay (from the source of the path)
- setup/recovery time (for data path)
- hold/removal time (for data path)
- clock skew (for data path)

Reporting detailed longest path No. 1 for domain CLK (Rising) -> CLK (Rising) (Case Analysis: Typical)

Launch clock path detail:

Source	Target	Routing delay	Internal delay	Cumulated delay
{-}:CLK O	{-}:wfg_B_CLK_ZI	1 ps	-	1 ps
{-}:wfg_B_CLK_ZI	{-}:wfg_B_CLK_ZO	-	566 ps	567 ps
{-}:wfg_B_CLK_ZO	{-}:si_CLK_I	659 ps	-	1,226 ns
{-}:si_CLK_I	{-}:si_CLK_O	-	0 ps	1,226 ns
{-}:si_CLK_O	{-}:UUT2[dout_reg[155]].CK	3,177 ns	-	4,403 ns
Total		3,837 ns	566 ps	4,403 ns

Latch clock path detail:

Source	Target	Routing delay	Internal delay	Cumulated delay
{-}:CLK O	{-}:wfg_B_CLK_ZI	1 ps	-	1 ps
{-}:wfg_B_CLK_ZI	{-}:wfg_B_CLK_ZO	-	566 ps	567 ps
{-}:wfg_B_CLK_ZO	{-}:si_CLK_I	659 ps	-	1,226 ns
{-}:si_CLK_I	{-}:si_CLK_O	-	0 ps	1,226 ns
{-}:si_CLK_O	{-}:so7_CLK_I	3,013 ns	-	4,239 ns
{-}:so7_CLK_I	{-}:so7_CLK_O	-	0 ps	4,239 ns
{-}:so7_CLK_O	{-}:sdrv_dataout[155].ECK	666 ps	-	4,905 ns
Total		4,339 ns	566 ps	4,905 ns

Data path detail:

Source	Target	Routing delay	Internal delay	Cumulated delay
{-}:UUT2[dout_reg[155]].CK	{-}:UUT2[dout_reg[155]].O	-	664 ps	664 ps
{-}:UUT2[dout_reg[155]].O	{-}:LOGIC[lut_339.12]	493 ps	-	1,157 ns
{-}:LOGIC[lut_339.12]	{-}:LOGIC[lut_339.O]	-	558 ps	1,715 ns
{-}:LOGIC[lut_339.O]	{-}:o1_UUT3[temp[155]].I	5,022 ns	-	6,737 ns
{-}:o1_UUT3[temp[155]].I	{-}:o1_UUT3[temp[155]].O	-	0 ps	6,737 ns
{-}:o1_UUT3[temp[155]].O	{-}:sdrv_dataout[155].EI	27 ps	-	6,764 ns
{-}:sdrv_dataout[155].EI	{-}:sdrv_dataout[155].ECK	-	-	6,764 ns
Total		5,542 ns	1,222 ns	6,764 ns

Setup/Recovery	Clock Skew
1,417 ns	502 ps

Figure 11: Example of path detail table

4.7 Logging

All messages generated by the application (generic information, warning, error, reports) are stored into several log files.

A 'logs' subdirectory is created into the working directory, and log files are created inside it. Users can use *nxpython*'s command to print their own messages (see section 6).

Several log files are created:

general.log	contains all messages printed: it's the full log.
synthesize.log	contains messages printed during each one of the three main step of the process.
place.log	
route.log	
progress.log	contains reports printed at the end of each major process step.
python.log	contains user messages printed from the Python interface.

4.8 Report generations

In *nxmap*, several reports will be generated during the process of the design to give information about resources utilization (instances, ports, power consumption and timing).

4.8.1 Instances report

npython gives the possibility to the user to generate instances report utilization of the programmed design on the current variant. For more information about generating instances report with *npython*, please refer to section 6.2.37.

4.8.2 Ports report

npython gives the possibility to the user to generate report on the ports found in HDL source and the ones manually set by user in the project. For more information about generating ports report with *npython*, please refer to section 6.2.38.

4.8.3 Power consumption report

npython gives the possibility to the user to report an estimate of the power consumption of the programmed design. For more information about generating ports report with *npython*, please refer to section 6.2.39.

4.8.4 Timing report

The Timing analysis process also creates several report files (.timing), each one containing information about a specific clock domain. For more information about timing report, refer to section 4.6.3.

5 NXmap specification

5.1 Graphical User Interface overview (nxmap)

nxmap main window is divided into 3 sections (Figure 12):

1. The menu
2. The content
3. The log console

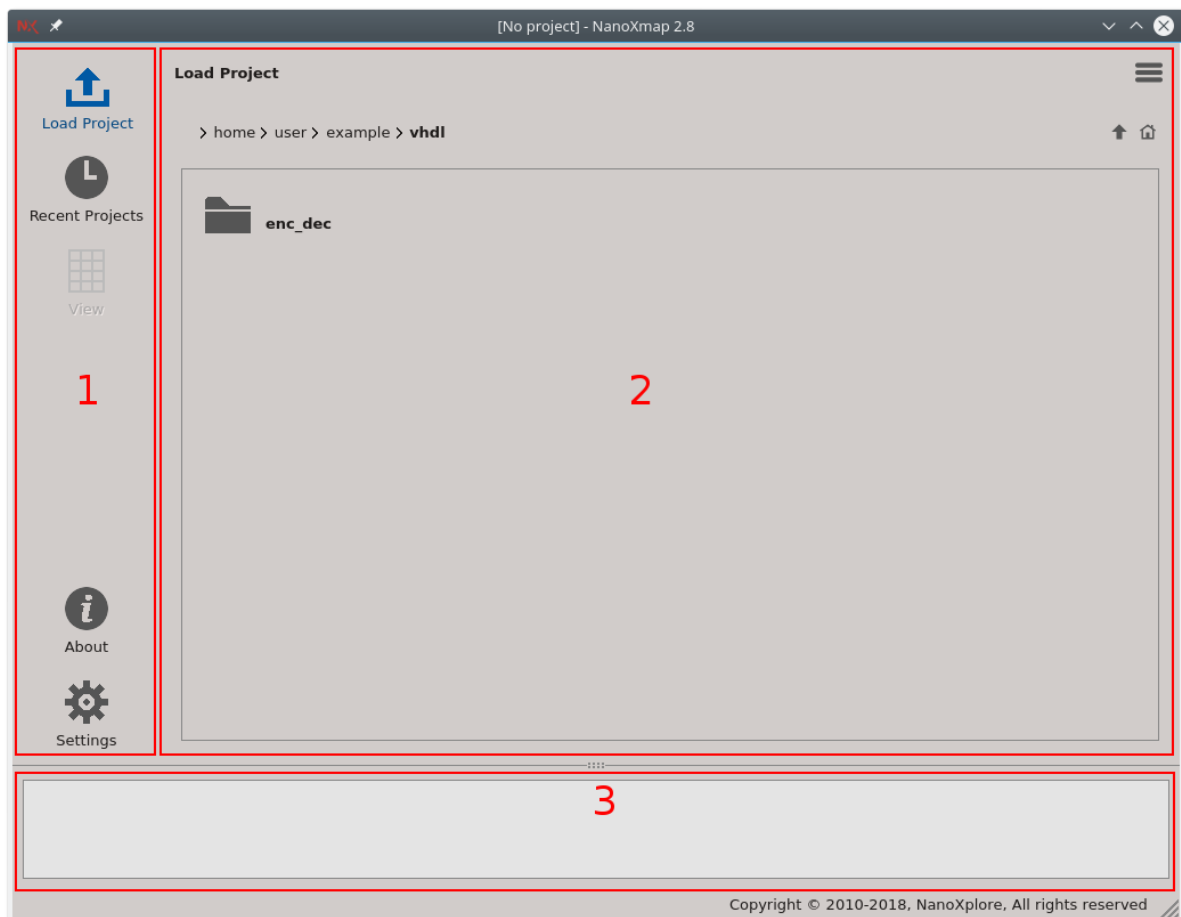


Figure 12: *nxmap* main window

The menu section allows user to access the following:

- **Load Project** load an existing project
- **Recent Projects** shows recently opened projects
- **View** graphical presentation of the loaded project
- **About** get information about *nxmap* release
- **Settings** change *nxmap* settings

When a tab of the menu is selected, the associated content is displayed in the content section.

In the top right corner of the content section, there is a help button that allows user to access the *nxmap* User manual or exit the application (Figure 13).

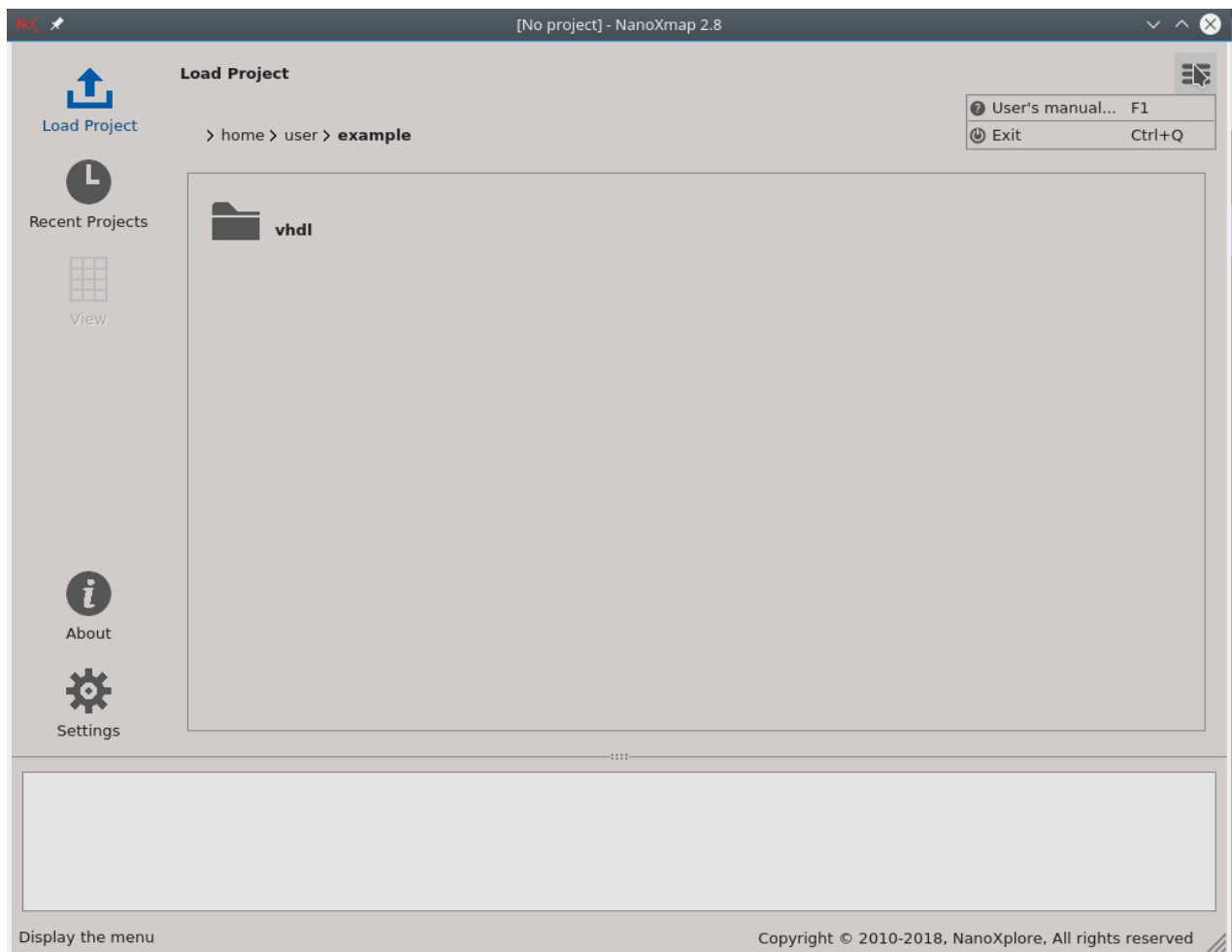


Figure 13: nxmap menu

In the following subsections, a brief introduction of each menu tab is given.

5.1.1 Load project

Using the "Load Project" tab and the associated file browser in the content section, user can load an existing project.

In the content section, each state of the selected project is represented by a name along with a short description containing the top cell and variant name.

In addition to this, a pie chart icon shows the completion state of the project. If the icon is blue, the archive can be loaded, while if it is red the archive is obsolete and cannot be loaded (Figure 14).

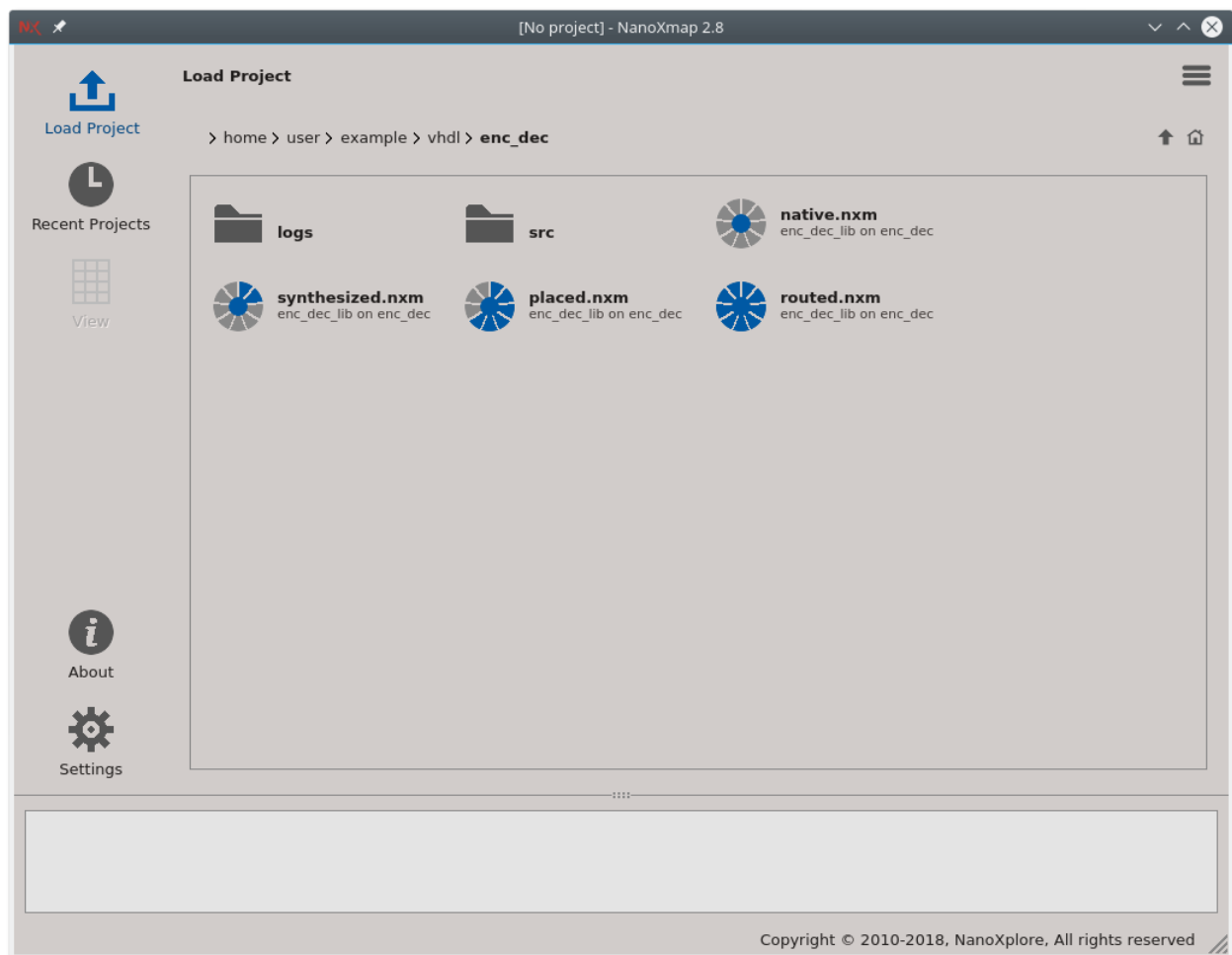


Figure 14: Load an existing project

5.1.2 Recent projects

"Recent Projects" tab shows recently loaded projects. The number of recent projects shown is limited to 10.

Like the "Load Project" tab, each project is represented by an icon, a name and a short description (Figure 15).

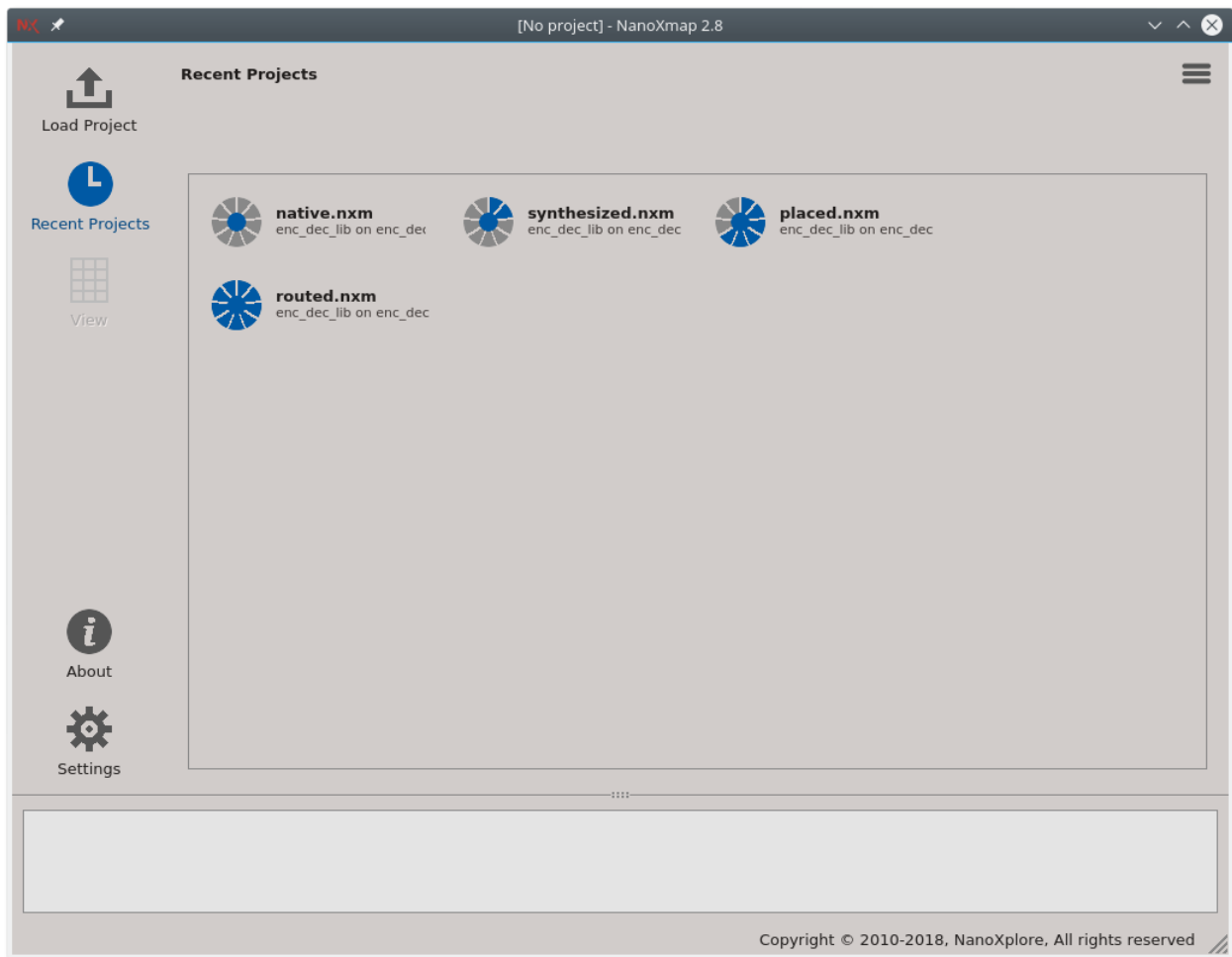


Figure 15: Load a recent project

5.1.3 View

The "View" tab provides a graphical representation of the FPGA architecture and the current design in the content section. The view is divided into 5 subsections (Figure 16):

- Progress bar:

The progress bar shows the current state of the project. Completed states are filled in green. By clicking on a state, the user can:

- Progress: perform computation up to a state, by clicking on a state that is not already completed.
- Regress: reload a previous state. Clicking on state will load the previous completed state (ie clicking on Place will load Synthesized).

- Command bar:

The command bar defines the action (Select or Inspect) that will be performed when the user interacts with the view, and its associated options. For further details, consult 5.3.

- View window:

The view window allows user to survey the design fitted for hardware architecture and also to inspect, debug and diagnose current design by using select and inspect commands.

- Dashboard:

The dashboard section provides an overview of the architecture. User can interact with the dashboard by scrolling and/or zooming with a mouse. The main view is scrolled/zoomed the same way.

When on the view widget or the dashboard widget, the mouse has the following behavior:

- **Left click** selects an element based on the current command
 - **Right click** centers the view on the cursor position
 - **Right click + drag** zooms on the selected area
 - **Middle click + drag** scrolls the view
 - **Wheel up/down** zooms out/in the view
 - **Ctrl + Wheel up/down** scrolls up/down the view
 - **Ctrl + Shift + Wheel up/down** scrolls right/left the view
- Matching elements:

Once a group of elements has been selected, user can use the 'Name' field to enter any string or regular expression to filter the matching elements. Moreover, by clicking the 'Details' button at the bottom of this subsection, a detailed report of the selected element is shown in the log console section.

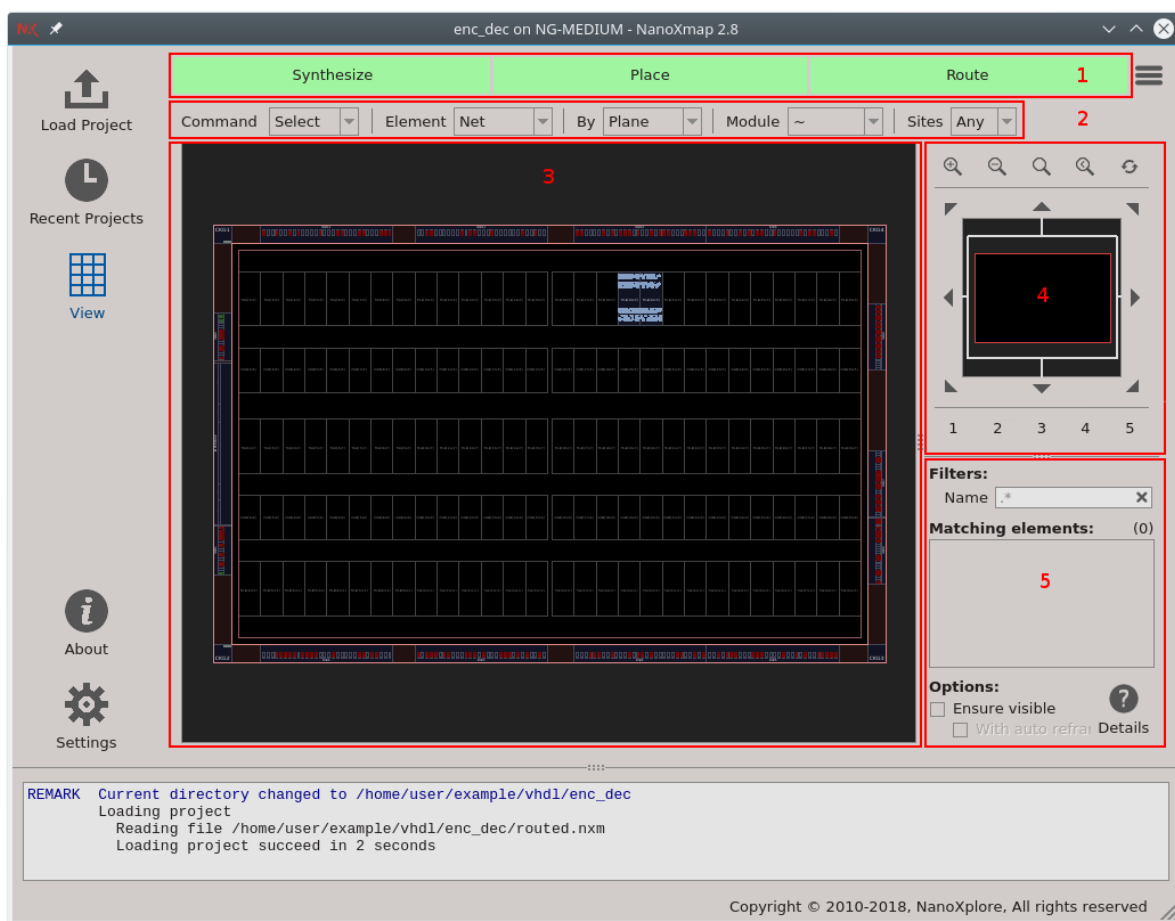
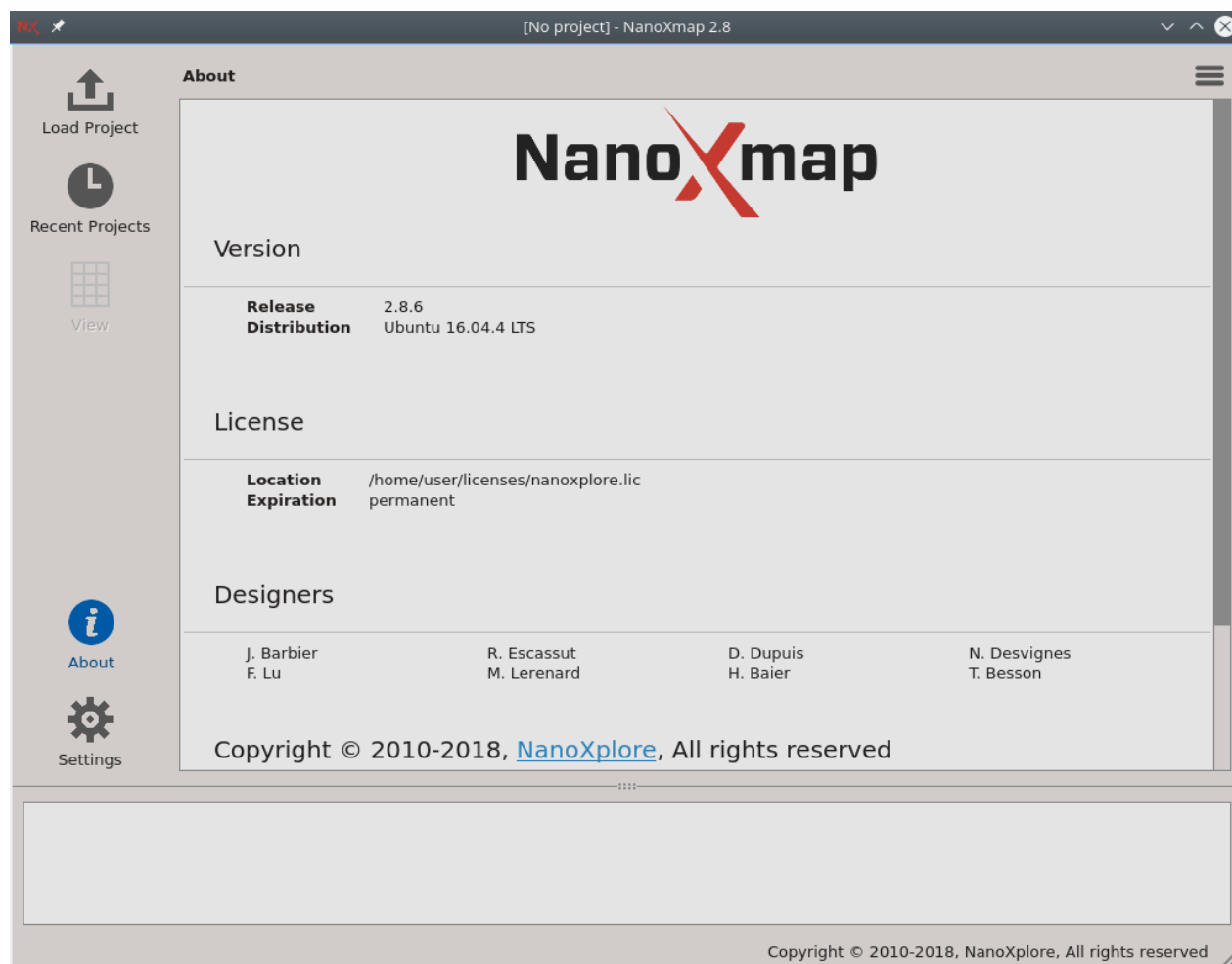


Figure 16: Graphical representation of a design on *nxmap*

5.1.4 About

The "About" tab shows information of the *nxmap* version, License, designers and copyright statement (Figure 17).

Figure 17: About *nxmap*

5.1.5 Settings

The settings tab allows user to change some parameters of *nxmap* (Figure 18).

The "General" subsection allows user to change the graphical theme and choose whether a warning message should be displayed when a user tries to load an obsolete archive.

Note

For the theme change takes effect, user must restart *nxmap*. Others parameters are applied immediately.

The "View" subsection allows user to change parameters about the view. User can change the mouse button associated to "zoom to area" action and can invert zoom direction associated to the wheel button.

The "Confirmation" subsection allows user to choose whether to be warned when exiting the application, closing or regressing a project and overwriting a file.

Note

All parameters can be reset by clicking on the "Restore defaults" button or by launching *nxmap* with the "-clear" option.

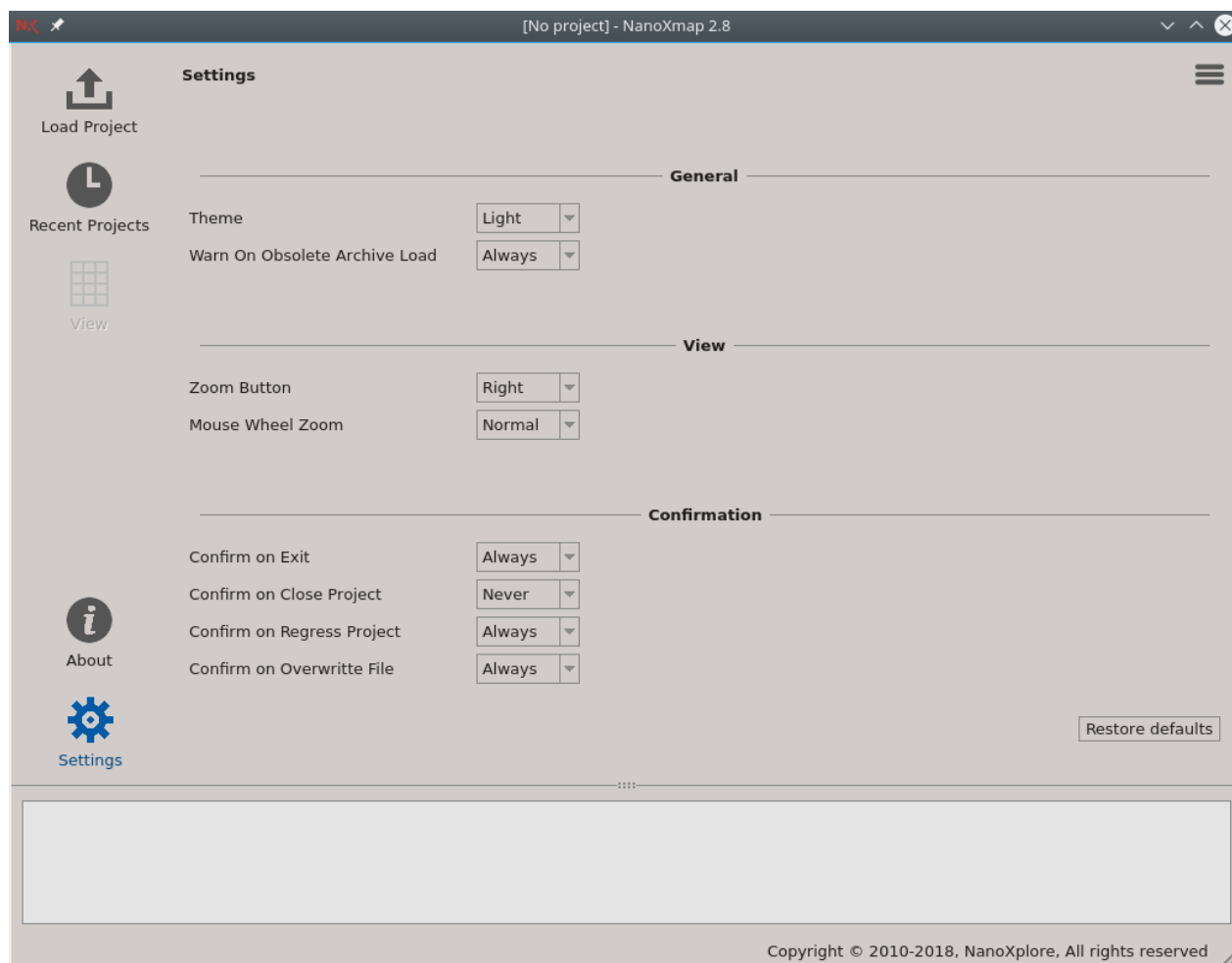


Figure 18: Settings tab

5.1.6 Command lines

There are several ways to start *nxmap* software:

```
$> nxmap
```

will run the graphical user interface and let the user load a project.

```
$> nxmap file.nxm
```

will run the graphical user interface and open the project le passed as argument.

```
$> nxmap --clear
```

will restore default user settings and run the graphical interface.

```
$> nxmap -version or -v
```

prints *nxmap* version and exits.

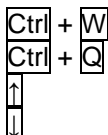
```
$> nxmap -help or -h
```

prints *nxmap* help and exits.

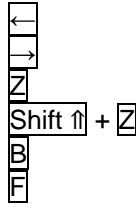
5.1.7 Keyboard shortcuts

nxmap software provides the following keyboard shortcuts:

- Close project
- Exit
- Scroll up
- Scroll down



- Scroll left
- Scroll right
- Zoom in
- Zoom out
- Zoom back
- Zoom full
- *nxmap* help F1
- *npython* help F2



5.2 Design flow

5.2.1 Synthesize

In *nxmap*, project can be loaded by clicking on "native.nxm" which will turn on the "View" section. Clicking on "Synthesize" in Progress bar will start the synthesis of the design.

To launch "Synthesize" step from *npython*, please refer to section 6.2.52.

5.2.2 Place & Route

In *nxmap*, if a project is already synthesized, user can launch place and route by clicking on "Place" and "Route" in the progress bar respectively. Otherwise clicking on any advanced step automatically performs the previous undone steps as well.

To launch "Place" and "Route" steps from *npython*, please refer to section 6.2.36 and 6.2.41 respectively.

5.2.3 Static Timing Analysis

In *nxmap*, static timing analysis can be launched after the successful routing of the design by selecting 'Analyze' from the help button at the top right corner of the content section (Figure 19).

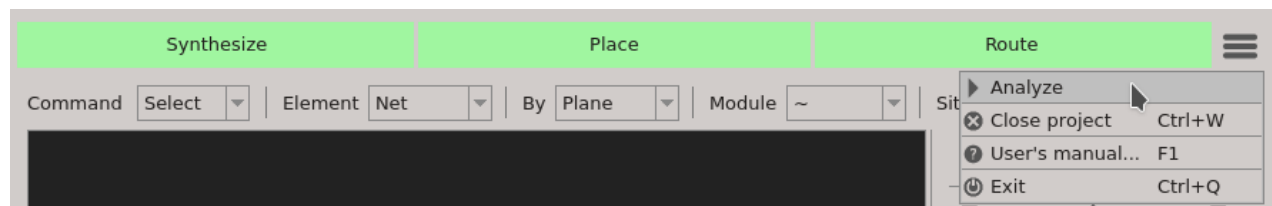


Figure 19: Static Timing Analysis option in *nxmap*

Note

Currently Static Timing Analysis can only be performed after a successful routing of the design.

To perform a Static Timing Analysis from *npython*, please refer to section 6.3.

5.3 Graphical inspection

As mentioned earlier, user can select or inspect the design in the "View" section of *nxmap*.

The command bar allows user to interact with the graphical view of the design. The command bar contains several drop-down menus as shown in Figure 20.

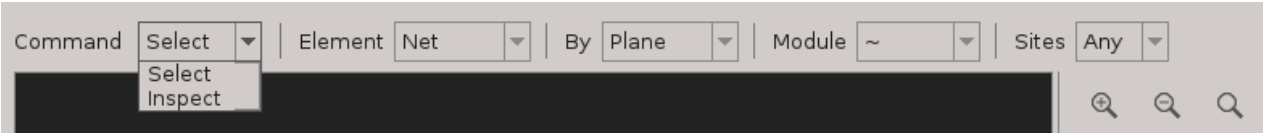


Figure 20: Command bar options

5.3.1 Select command

The "Select" command allows user to select nets, instances or paths based on different criteria (Figure 21).

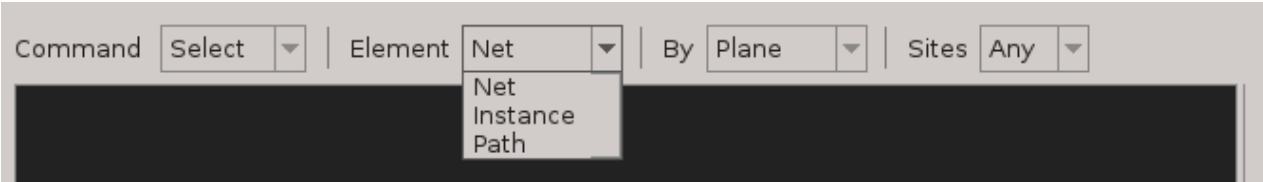


Figure 21: Element options in Command bar

For example, setting 'Command' drop-down menu to "Select", 'Element' drop-down menu to "Net" and then clicking on the architecture representation in "View" section, all matching nets for the selected menu will be added in the "Matching elements" section (right bottom). Items in "Matching elements" list can be highlighted in the "View" section upon selection (Figure 22).

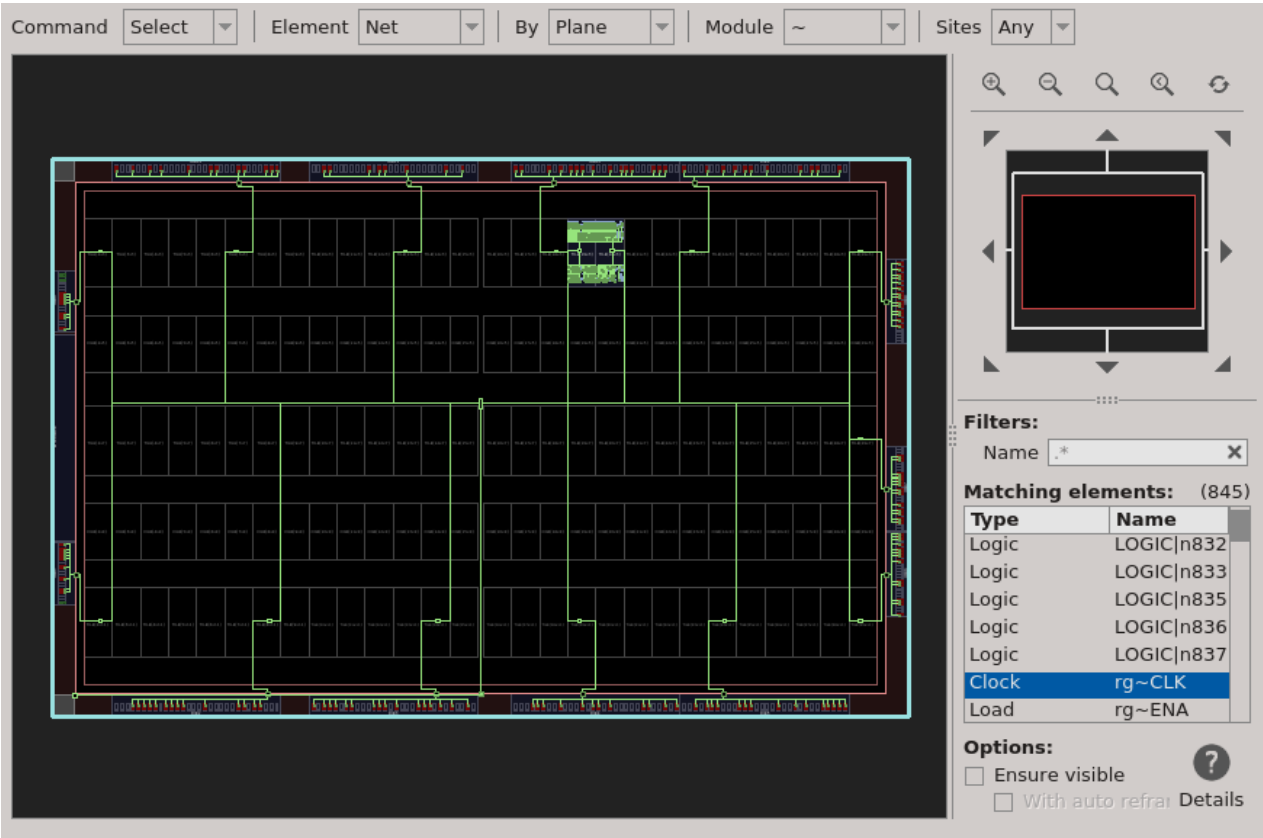


Figure 22: List of matching elements

When selecting 'Net', the available criteria are:

- By** the type of architecture's element, the net is connected to [Plane, Site, Instance or Pin]
- Module** the top module or sub-modules in the design [top module is indicated by '~']
- Sites** the number of sites, the net is connected to [Any, 1, 2, 3, 4, 5, 6, 7, 8, 9+]

When selecting 'Instance', the available criteria are:

By the type of architecture's element, the net is connected to [Plane, Site, Instance or Pin]
Module the top module or sub-modules in the design [top module is indicated by '~']
Type the design interface, the instance is connected to [Any, IOB, IOD, CKG, FE, CY, RF, CKS, RAM, DSP]

When selecting 'Path', the available criteria are:

By the type of architecture's element, the net is connected to [Plane, Site, Instance or Pin]
Domain the timing domains in the design
Type the longest or shortest paths

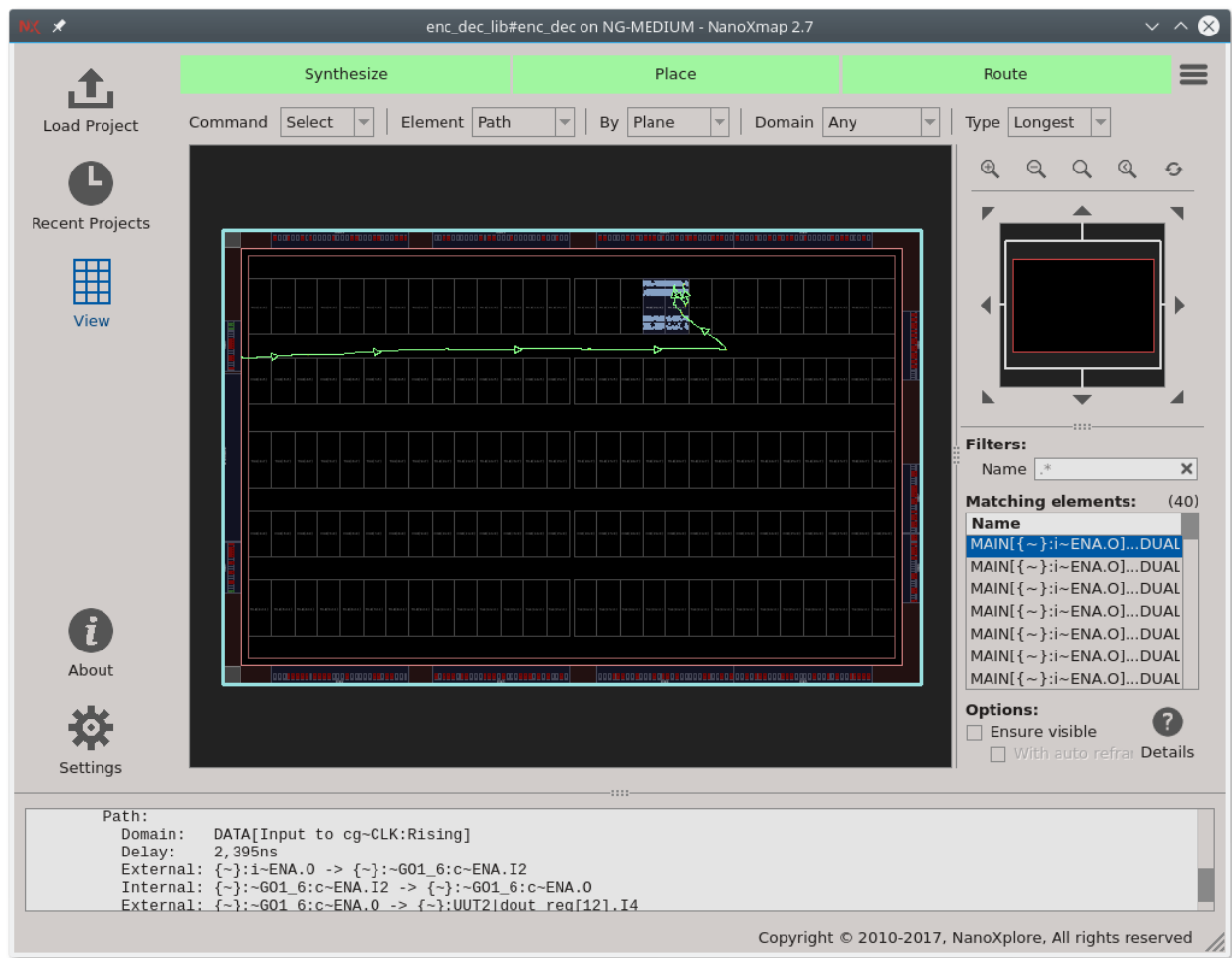


Figure 23: A critical path in a timing domain

By selecting an element (Net, Instance or Path) and the corresponding criteria in the command bar, a list of elements appears in the "Matching elements" on the right. The detailed report on each element selected from the "Matching elements" list can be seen in the log console section by clicking on the 'Details' button at the bottom of the 'Matching element' subsection.

In case of selecting a path and corresponding domain in the command bar, a list of first ten critical paths in that domain appears in the "Matching elements". Each of these critical paths can be selected and seen in the main view. Similarly, to see a detailed timing report of a selected path in the log console section, click the 'Details' button (Figure 23).

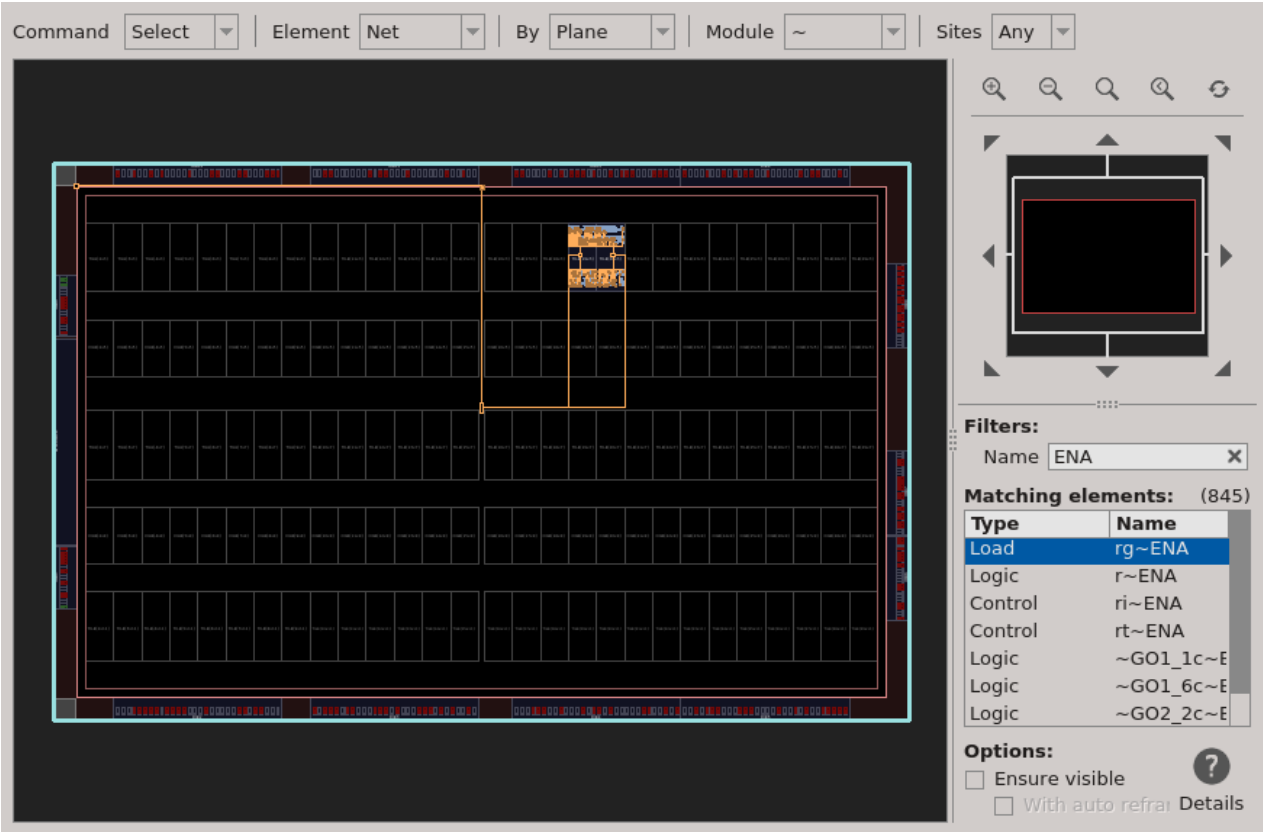


Figure 24: Filtering the Matching elements

User can use the "Name" field to filter the items in "Matching elements" by entering any string or regular expression (Figure 24).

5.3.2 Inspect command

The "Inspect" command allows user to select any element of the architecture and get information about it.

For example, when selecting an instance of register, the inspect command displays the clock sensitivity as shown in Figure 25.

The available criterion for the "Inspect" command is:

By the type of architecture's element to be inspected [Cell, Zone, Instance or Pin]

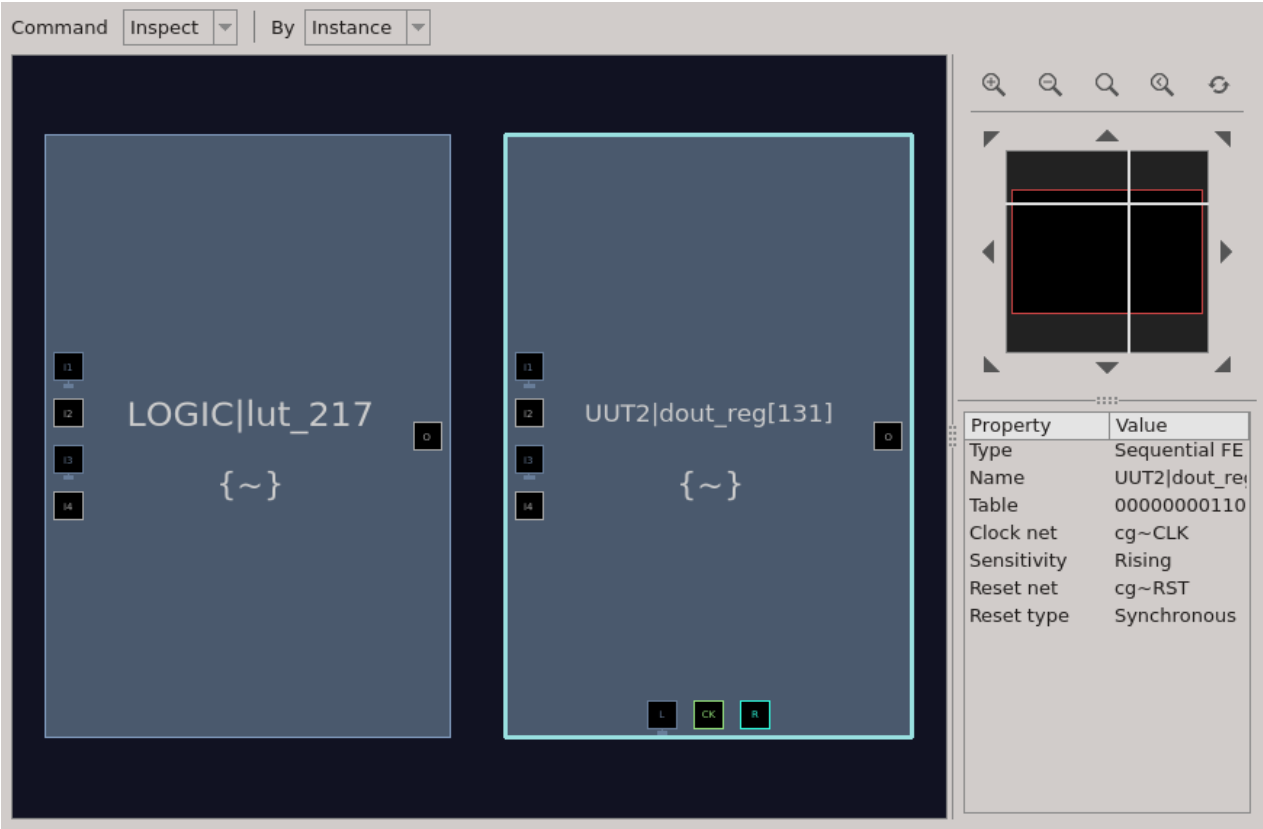


Figure 25: Inspect command

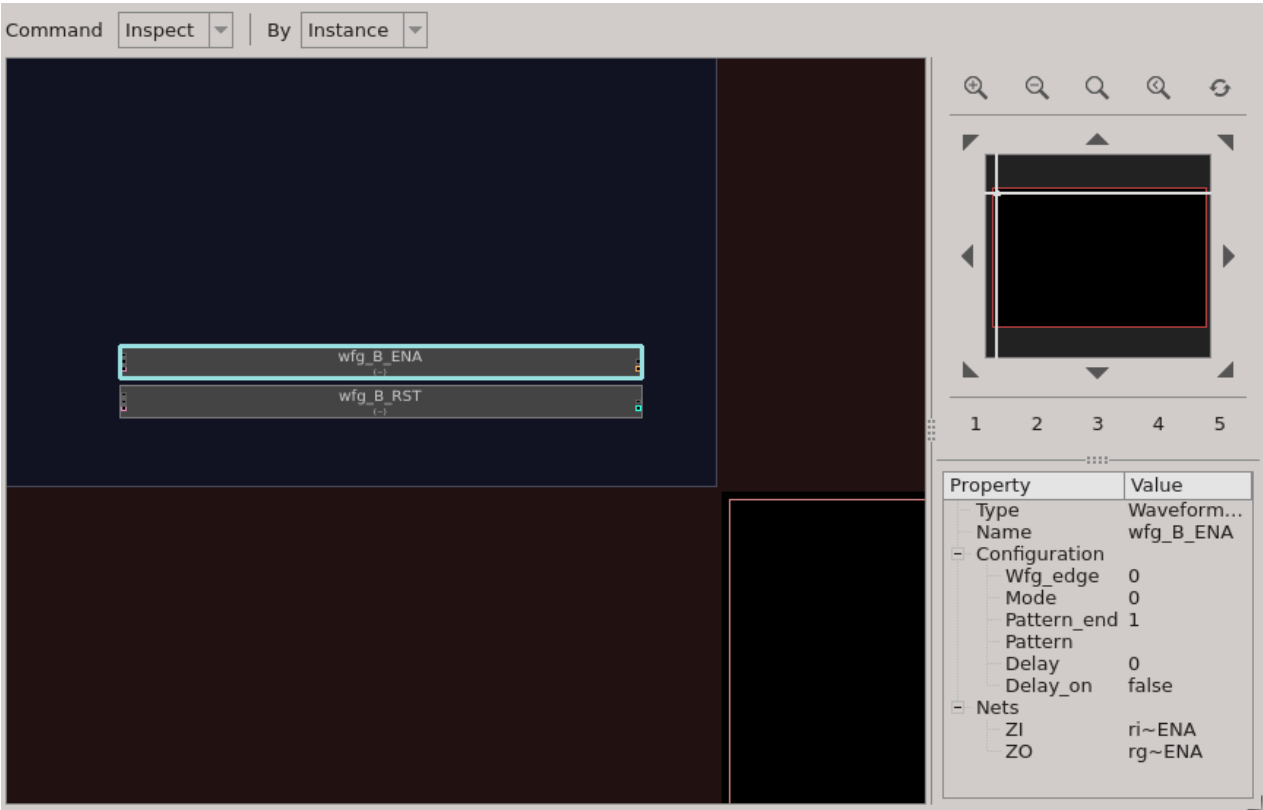


Figure 26: Inspect WFG

6 Nxpython specification

6.1 General commands

This chapter presents the general purpose commands. These commands do not need any object to be called as they are directly defined in the nanoxmap module.

6.1.1 *createProject*([*workingDirectory*])

This method is used to create an empty project object. The project can then be loaded or built (see section 6.2).

Arguments:

Name	Type	Description
workingDirectory	string	the working directory in which files will be saved

When no argument is supplied, the working directory will be the one *nxpython* is launched from.

Example:

```
project = createProject("/path/to/working/directory/")
```

6.1.2 *getErrorCount()*

This function returns the number of errors.

This function takes no argument.

Example:

```
print 'Errors: ', getErrorCount()
```

Output:

```
Errors: 0
```


6.1.3 *getWarningCount()*

This function returns the number of warnings.

This function takes no argument.

Example:

```
print 'Warnings: ', getWarningCount()
```

Output:

```
Warnings: 0
```

6.1.4 *info([object])*

This function prints help about the given object. If no argument is given, it prints top level help with a list of available generic functions and class objects.

Arguments:

Name	Type	Description
object	string	object to get information about

Example:

```
info()
or
info('Project')
or
info('Project.setVariantName')
```

Output:

```
| Method
|   Project.setVariantName(variant)
|
|   This method is used to set the variant of the project.
|   The default value of the variant is 'NG-MEDIUM'.
|
|   Arguments:
|   variant
|     type:      string
|     description: the name of the variant
|
|   Example:
|   project = createProject()
|
|   project.setVariantName('NG-MEDIUM')
```

6.1.5 *printError(message)*

This function is used to print an error message in the *nxmap* output. This includes terminal output and log file if set. An error message is prefixed by "ERROR" string so it can be easily found.

When using this method, the global number of errors is incremented by 1.

Arguments:

Name	Type	Description
message	string	error message to print

Example:

```
printError("It's a trap!")
```

Output:

```
ERROR | It's a trap!
```

6.1.6 *printText*([message])

This function is used to print a message in the *nxmap* output. This includes terminal output and log file if set.

When no message is given, method prints a new line in the *nxmap* output.

Arguments:

Name	Type	Description
message	string	message to print

Example:

```
printText('There is no spoon.')
```

Output:

```
| There is no spoon.
```

6.1.7 *printWarning(message)*

This function is used to print a warning message in the *nxmap* output. This includes terminal output and log file if set. A warning message is prefixed by "WARNING" string so it can be easily found.

When using this method, the global number of warnings is incremented by 1.

Arguments:

Name	Type	Description
message	string	warnig message to print

Example:

```
printWarning('The cake is a lie.')
```

Output:

```
WARNING | The cake is a lie.
```

6.2 Project related methods

This section presents the methods related to the project object.

6.2.1 *addBlackBox(name, type, mapping, interface)*

The method 'addBlackBox' allows users to define customized mapping of certain HDL modules on the specific FPGA blocks and to dispose the interface of the blocks.

Arguments:

Name	Type	Description
name	string	the name of HDL module that is intended to be custom-mapped.
type	string	the <i>npython</i> object type (further details below).
mapping	string	the mapping target (further details below).
interface	string	the interface mapping between HDL module and <i>npython</i> (further details below).

The 'type' can only be overridden for specific object types. The available types are:

Type	Description
ROM	Represents a ROM
RAM	Represents a RAM

For each type, there are some specific instances in the FPGA onto which it can be mapped. This can be specified using 'mapping' argument. The following table presents all the available mapping targets for each type:

Type	Mapping targets	Description
ROM	RF, RAM	A ROM can be mapped onto RF blocks or RAM blocks
RAM	RF, RAM, RAM_ECC, DFF	A RAM can be mapped onto register files or RAM blocks or RAM blocks with error detection/correction code or registers

The argument 'interface' specifies the interface mapping between the HDL module and *npython*.

Important

Space character is not allowed in argument 'interface'.

For 'ROM' type, the *npython* interface is:

Interface	Description
AD	Address to read data from
DO	Output data
CK	Clock

Important

Used only when creating synchronous memory (with one clock).

For 'RAM' type, the *npython* interface depends on the mode RAM is being used in:

- RAM mode: "Simple port"

Interface	Description
CK	Clock
AD	Address
DI	Input data
DO	Output data
WE	Write enable
RE	Read enable (optional)

- RAM mode: "Split read/write port" (1 port for read and 1 port for write)

Interface	Description
CK0	Read clock
AD0	Read address
DO0	Read output data
RE0	Read enable (optional)
CK1	Write clock
AD1	Write address
DI1	Write input data
WE1	Write enable

- RAM mode: "True dual port" (2 ports for read/write)
First port:

Interface	Description
CK0	Clock
AD0	Address
DI0	Input data
DO0	Output data
WE0	Write enable
RE0	Read enable (optional)

Second port:

Interface	Description
CK1	Clock
AD1	Address
DI1	Input data
DO1	Output data
WE1	Write enable
RE1	Read enable (optional)

Example:

If the HDL design contains a micro code ROM define in VHDL as:

```
component myMicroCode
port (
  addr  : in std_logic_vector(5 downto 0);
  clk   : in std_logic;
  micro : out std_logic_vector(5 downto 0)
);
end component;
```

User can add a black box to force *nxpython* to recognize this module as a ROM.

```
project.addBlackBox('myMicroCode', 'ROM', 'RF', 'AD(addr),DO(micro),CK(clk)')
```

6.2.2 *addFalsePath(from_list, to_list)*

This method is used to specify the false path for the timing paths. It is used by timing driver algorithms and static timing analysis.

Arguments:

Name	Type	Description
from_list	string	the command which specifies how to get a list of timing path starting points. A valid timing starting point is a register.
to_list	string	the command which specifies how to get a list of timing path ending points. A valid timing ending point is a register.

Example:

```
project = createProject()
project.load('routed.nxm')
project.addFalsePath('getRegister(UUT1|Gen_seq[3].seq_i|temp_reg[1]]',
                    'getRegister(UUT2|dout_reg[61]]')
```


6.2.3 *addFile([library], file)*

This method is used to add a HDL source file to the project. The file can be added to a specific library by passing its name as the first optional argument 'library'. The default library is 'work'.

Arguments:

Name	Type	Description
library	string	name of the HDL library that contains the source file.
file	string	HDL source file, the path can be absolute or relative to the directory of the project.

Example:

```
project.addFile('src/simple.vhd')
```

or

```
project.addFile('myLib', 'src/simple.vhd')
```

6.2.4 *addFiles([library], fileList)*

This method is used to add several HDL source files to the project. The files can be added to a specific library by passing its name as the first optional argument 'library'. The default library is 'work'.

Arguments:

Name	Type	Description
library	string	name of the HDL library that contains the source files.
fileList	list	list of HDL source files, the path can be absolute or relative to the directory of the project.

Example:

```
project.addFiles(['src/simple.vhd', 'src/test.vhd'])  
  
or  
  
project.addFiles('myLib', ['src/simple.vhd', 'src/test.vhd'])
```

Note

List of HDL source files can be in any order. *nxpython* will sort them such that dependencies are satisfied.

6.2.5 *addInterface(name, location)*

This method allow user to configure an input/output of the HDL module into specific interface of the FPGA.

Arguments:

Name	Type	Description
name	string	the HDL input/output name.
location	string	the FPGA interface location (e.g. "USER_D0").

Example:

```
project.addInterface('LED', 'USER_D0')
```

6.2.6 *addInterfaces(interfaces)*

This method is used to configure inputs/outputs of the HDL module into specific interfaces of the FPGA. The argument is a dictionary that associate the HDL input/output name to the interface location.

Arguments:

Name	Type	Description
interfaces	dictionary	the keys are the HDL input/output names and the values are the interfaces locations.

Example:

```
interfaces = {  
    'LED0': 'USER_D0'  
    , 'LED1': 'USER_D1'  
}  
  
project.addInterfaces(interfaces)
```

6.2.7 *addIP(name)*

This method is used to add a NX IP source file to the project.

The name must be a valid NX IP name (file name without extension). The available NX IP can be found in install/share/ips directory.

Arguments:

Name	Type	Description
name	string	the name of the NX IP.

Example:

```
project = createProject()
project.addFile('src/test.vhd')
project.addIP('IP_SPW_BANK')
```

6.2.8 *addMappingDirective(name, type, mapping)*

This method is used to override default *nxpython* mapping behavior for a specific HDL module or instance.

Arguments:

Name	Type	Description
name	string	the name of HDL module or instance that is intended to be custom-mapped.
type	string	the <i>nxpython</i> object type (further details below).
mapping	string	the mapping target (further details below).

The 'name' argument can be a regular expression matching a HDL module or instance name. In order to specify whether to search for module or instance, the user has to use the 'getModels(...)' or 'getInstances(...)' keyword.

The 'type' can only be overridden for specific object types. For each type, there are some specific instances in the FPGA onto which it can be mapped. This can be specified using 'mapping' argument. The following table presents all the available mapping targets for each type:

Type	Mapping targets
ADD	CY, DSP
LTN	LUT, CY, DSP
MUL	CY,DSP
RAM	RF, RAM, RAM_ECC, DFF
ROM	LUT, RF, RAM, RAM_ECC

Note

Instance name must match the following pattern: 'PATHNAME|INSTANCENAME'.

Example:

If the HDL design named myTest contains a small 3 write ports and 3 read ports FIFO named myFifo, *nxpython* cannot automatically map it into RF or RAM, but user can add a mapping directive to map it into DFF:

```
project.addMappingDirective('getModels(myTest_myFifo)', 'RAM', 'DFF')
```

Important

It can be useful to use "." in getModels and getInstances method as following:

```
project.addMappingDirective('getModels(.myFifo.*)', 'RAM', 'RAM')
```

The first "." is used because your model is defined in a VHD library.

The second "." is used because your "myFifo" model uses some generic map and thus is renamed to myFifo(#id) by *nxmap*.

6.2.9 *addMaxDelayPath(from_list, to_list, delay)*

This method is used to specify the maximum delay path for the timing paths. It is used by timing driver algorithms and static timing analysis.

Arguments:

Name	Type	Description
from_list	string	the command which specifies how to get a list of timing path starting points. A valid timing starting point is a register.
to_list	string	the command which specifies how to get a list of timing path ending points. A valid timing ending point is a register.
delay	integer	the required maximum delay value in ps for specified paths.

Example:

```
project = createProject()

project.load('routed.nxm')

project.addMaxDelayPath('getRegister(UUT1|Gen_seq[3].seq_i|temp_reg[1]]',
                        'getRegister(UUT2|dout_reg[61]]', 3900)
```

6.2.10 *addMemoryInitialization(name, type, file)*

This method is used to specify a file that will be used to initialize a specific memory instance.

Arguments:

Name	Type	Description
name	string	the name of the instance that is intended to be initialized from a file.
type	string	the file format (further details below).
mapping	string	the name of the file to load (further details below).

The 'name' argument can be a regular expression matching a HDL module or instance name. In order to specify whether to search for module or instance, the user has to use the 'getModels(...)' or 'getInstances(...)' keyword.

The available file formats are:

Type	Format
NX	NanoXplore Proprietary format

Note

The 'name' parameter must match the following pattern: TOPCELLNAME_INSTANCENAME.

The NX File format is a simple format in which initialization bits are written in ASCII. Each line of the file represents one vector in the memory. NULL MSBs and empty (only 0's) lines at the end of the file are NOT mandatory.

Example:

To initialize a memory called IMG_1' in a design called 'test_init' using data from a file called "initdata.nx", call:

```
Project.addMemoryInitialization('getModels(test_init_IMG_1)', 'NX', 'initdata.nx')
```

If you declare a memory array 'array(0 to 5) of std_logic_vector(0 to 3)' and want to load an identity matrix, the file should contain:

```
0000
1000
0100
0010
0001
0000
```

Since it is not mandatory to initialize:

- the end of the array if it contains only zeros
- MSBs if they are null

The previous example can be written this way:

```
0000
1000
100
10
1
```


6.2.11 *addMinDelayPath(from_list, to_list, delay)*

This method is used to specify the minimum delay path for the timing paths. It is used by timing driver algorithms and static timing analysis.

Arguments:

Name	Type	Description
from_list	string	the command which specifies how to get a list of timing path starting points. A valid timing starting point is a register.
to_list	string	the command which specifies how to get a list of timing path ending points. A valid timing ending point is a register.
delay	integer	the required minimum delay value in ps for specified paths.

Example:

```
project = createProject()

project.load('routed.nxm')

project.addMinDelayPath('getRegister(UUT1|Gen_seq[3].seq_i|temp_reg[23]]',
                        'getRegister(UUT1|Gen_seq[3].seq_i|temp_reg[22]]', 1200)
```

6.2.12 *addMulticyclePath(from_list, to_list, cycle_count)*

This method is used to specify the multicycle path for the timing paths. It is used by timing driver algorithms and static timing analysis.

Arguments:

Name	Type	Description
from_list	string	the command which specifies how to get a list of timing path starting points. A valid timing starting point is a register.
to_list	string	the command which specifies how to get a list of timing path ending points. A valid timing ending point is a register.
cycle_count	unsigned	An unsigned value that represents a number of cycles the data path must have for setup check.

Example:

```
project = createProject()

project.load('routed.nxm')

project.addMulticyclePath('getRegister(UUT1|Gen_seq[3].seq_i|temp_reg[1]]',
                          'getRegister(UUT2|dout_reg[61]]', 2)
```

Important

This method is only available for path(s) whose source and target registers are clocked by the same clock!

6.2.13 addPad(name, parameters)

This method allows user to configure an input/output of the HDL module into specific pad of the FPGA. For the given pad, the configuration has two basic argument (name and parameters).

Arguments:

Name	Type	Description
name	string	the HDL input/output name.
parameters	dictionary	the keys are the parameters names and the values are the parameters values.

Parameters is a dictionary which can take the following keys:

• location (string)	• termination (string)	• terminationReference (string)
• type (string)	• inputDelayLine (integer)	• turbo (boolean)
• weakTermination (string)	• outputDelayLine (integer)	• inputSignalSlope (integer)
• slewRate (string)	• differential (boolean)	• outputCapacity (integer)
• registered (string)		

The type must be one of the following values:

- LVCMOS_1.5V_2mA
- LVCMOS_1.8V_2mA
- LVCMOS_2.5V_2mA
- LVCMOS_3.3V_2mA
- LVCMOS_1.5V_4mA
- LVCMOS_1.8V_4mA
- LVCMOS_2.5V_4mA
- LVCMOS_3.3V_4mA
- LVCMOS_1.5V_8mA
- LVCMOS_1.8V_8mA
- LVCMOS_2.5V_8mA
- LVCMOS_3.3V_8mA
- LVCMOS_1.5V_16mA
- LVCMOS_1.8V_16mA
- LVCMOS_2.5V_16mA
- LVCMOS_3.3V_16mA
- LVDS_2.5V
- SSTL_1.8V_I
- SSTL_2.5V_I
- SSTL_1.8V_II
- SSTL_2.5V_II
- HSTL_1.5V_I
- HSTL_1.8V_I
- HSTL_1.5V_II
- HSTL_1.8V_II

The weakTermination must be one of the following values:

- None
- PullUp
- PullDown
- Keeper

The slewRate must be one of the following values:

- Slow
- Medium
- Fast

The registered parameter defines if pad uses registers and must be one of the following values:

- I
- IO
- IOC
- O
- OC

If another value is given, then registers for the specific pad are not used.

The termination specifies the output impedance of the pad in Ohms.

Input and output delay line should be between 0 and 63. This number correspond to a number of step.

The terminationReference must be one of the following values:

- floating
- VT

Unit of inputSignalSlope is V/ns.

Unit of outputCapacity is pF.

Important

LVDS type can only be differential.

When a key is not in the dictionary, its value is set to default, i.e. 0 for inputDelayLine/outputDelayLine/inputSignalSlope/outputCapacity and False for all the boolean arguments.

Example:

```
project.addPad('A', {'location': 'IO_B10D09P', 'type': 'LVCMOS_2.5V_2mA'})

project.addPad('B', {'location': 'IO_B10D09N', 'type': 'LVCMOS_2.5V_2mA', 'weakTermination': 'PullUp',
                    'slewRate': 'Slow', 'termination': '50', 'inputDelayLine': 1, 'outputDelayLine': 2,
                    'differential': False, 'turbo': True})
```

Note

There are 16 special system pads which can be used to access to the clock tree architecture:

- | | |
|-------------|--------------|
| • IOB0_D10P | • IOB6_D14P |
| • IOB0_D11P | • IOB6_D15P |
| • IOB1_D01P | • IOB8_D01P |
| • IOB1_D02P | • IOB8_D02P |
| • IOB2_D08P | • IOB9_D08P |
| • IOB2_D09P | • IOB9_D09P |
| • IOB5_D08P | • IOB12_D08P |
| • IOB5_D09P | • IOB12_D09P |

6.2.14 *addPads(pads)*

This method is used to configure inputs/outputs of the HDL module into specific pads of the FPGA. The argument is a dictionary that associates the HDL input/output name to a dictionary, which can contain some keys (location, type, weakTermination, slewRate, termination, inputDelayLine, outputDelayLine, differential, terminationReference, turbo) as defined in 6.2.13.

Arguments:

Name	Type	Description
pads	dictionary	the keys are the HDL input/output names and the values are dictionaries of parameters.

Example:

```
pads = {  
    'A': {'location': 'IO_B10D09P', 'type': 'LVCMOS_2.5V_2mA'},  
    'B': {'location': 'IO_B10D09N', 'type': 'LVCMOS_2.5V_2mA', 'weakTermination': 'PullUp',  
        'slewRate': 'Slow', 'termination': '50', 'inputDelayLine': 1, 'outputDelayLine': 2, 'differential': False,  
        'turbo': True}  
}  
  
project.addPads(pads)
```

Note

If "pads" are not configured or configured incompletely, *nxmap* will randomly place non assigned ports on the FPGA.

6.2.15 *addPin(name, location)*

This method allow user to configure an input/output of the HDL module into specific pin of the embedded FPGA.

Arguments:

Name	Type	Description
name	string	the HDL input/output name.
location	string	the location of the input/output.

Example:

```
project.addPin('A', 'TUI200')
```

6.2.16 *addPins(pins)*

This method allow user to configure inputs/outputs of the HDL module into specific pins of the embedded FPGA.

Arguments:

Name	Type	Description
pins	dictionary	the keys are the HDL input/output names and the values are the locations.

Example:

```
pins = {  
    'A': 'TUI200',  
    'B': 'TUI201'  
}  
  
project.addPins(pins)
```

6.2.17 *addParameter(name, value)*

This method allows user to specify the value of his design generic parameters.

Arguments:

Name	Type	Description
name	string	the name of the generic parameter.
value	string	the value of the generic parameter.

Example:

```
project.addParameter('BusWidth', '16')  
project.addParameter('Key_size', '2')
```

Important

- Each parameter must be in the 'generic' list of the design top entity.
- All uninitialized parameters in the design must be set with the value at this step.
- Already initialized parameters in the design can be overridden with a new value.

6.2.18 *addParameters(parameters)*

In case of a large number of parameters, a 'parameter dictionary' can be created consisting of the name of the parameter as a key and its respective value. Both name and value must be string.

Arguments:

Name	Type	Description
parameters	dictionary	parameters dictionary where keys are the parameters names and values the parameters values.

Example:

```
parameters = {'BusWidth': '16', 'Key_size': '2', ...}  
project.addParameters(parameters)
```

Important

- Each parameter must be in the 'generic' list of the design top entity.
- All uninitialized parameters in the design must be set with the value at this step.
- Already initialized parameters in the design can be overridden with a new value.

6.2.19 *addPLLLocation(name, location)*

This method is used to set the spot in which a PLL should be set.

Arguments:

Name	Type	Description
name	string	the name of the instance (can be a regular expression).
location	string	the requested location. Valid locations are CKG[1-4].PLL1

Example:

```
project.addPLLLocation('pll_0', 'CKG2.PLL1')
```

6.2.20 *addVerilogIncludeDirectories(directoryList)*

This method is used to add several extra directories containing Verilog files to the project. This is used when the included files are external to the working directory.

Arguments:

Name	Type	Description
directoryList	list	list of the directories containing Verilog files. Can be in absolute or relative.

Example:

```
project.addVerilogIncludeDirectories(['include', '../lib'])
```

6.2.21 *addVerilogIncludeDirectory(directory)*

This method is used to add an extra directory containing Verilog files to the project. This is used when the included files are external to the working directory.

Arguments:

Name	Type	Description
directory	string	name of the directory containing Verilog files. Can be in absolute or relative.

Example:

```
project.addVerilogIncludeDirectory('include')  
  
or  
  
project.addVerilogIncludeDirectory('../lib')
```

6.2.22 *addWFGLocation(name, location)*

This method is used to set the spot in which a WFG should be set.

Arguments:

Name	Type	Description
name	string	the name of the instance (can be a regular expression).
location	string	the requested location.

Valid locations depend on the variant and are:

NG-MEDIUM	CKG[1-4].WFG_R[1-2] CKG[1-4].WFG_M[1-3] CKG[1-4].WFG_C[1-3]
NG-LARGE	CKG[1-4].WFG_R[1-3] CKG[1-4].WFG_M[1-3] CKG[1-4].WFG_C[1-4]

Example:

```
project.addWFGLocation('wfg_0', 'CKG2.WFG_M3')
```

6.2.23 *createAnalyzer()*

If a targeted design is already routed, user can launch static timing analysis by creating a timing analyzer. This method takes no argument.

Example:

```
project = createProject()  
project.load('/home/user/example/vhdl/simple/routed.nxm')  
analyzer = project.createAnalyzer()
```

6.2.24 createClock(target, name, period, [rising, falling])

This method is used to create a clock constraint of a timing point. This constraint is used by timing driver algorithms and static timing analysis.

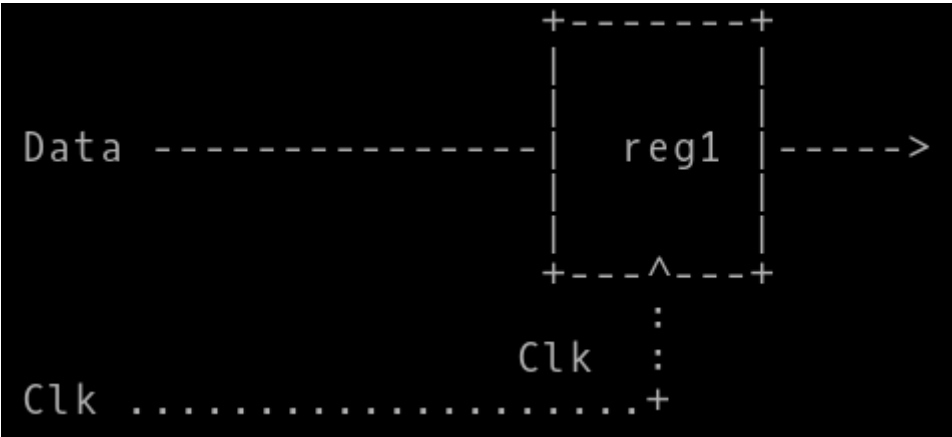
Arguments:

Name	Type	Description
target	string	the command which specifies how to get a clock related point. A valid command can be: getPort(port_name), getRegisterClock(register_name) (or getRegister(register_name)) and getClockNet(clock_net_name).
name	string	user clock name of the created clock.
period	unsigned	the period value in ps.
rising	unsigned	specific rising edge in ps for clock waveform. Range: [0, period[(default value is 0).
falling	unsigned	specific falling edge in ps for clock waveform. Range:]rising, rising + period] (default value is period/2).

Important

The name of the clock net is case sensitive.

Example:



In the example above, to define a clock with 100MHz for net "Clk", the following three commands are equivalent:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getRegisterClock(reg1)', 'clk', 10000)
    or
project.createClock('getPort(Clk)', 'clk', 10000)
    or
project.createClock('getClockNet(Clk)', 'clk', 10000, 0, 5000)
```

6.2.25 createGeneratedClock(source, target, name, relationship)

This method is used to create an internally generated clock constraint at a timing point. This constraint is used by timing driver algorithms and static timing analysis.

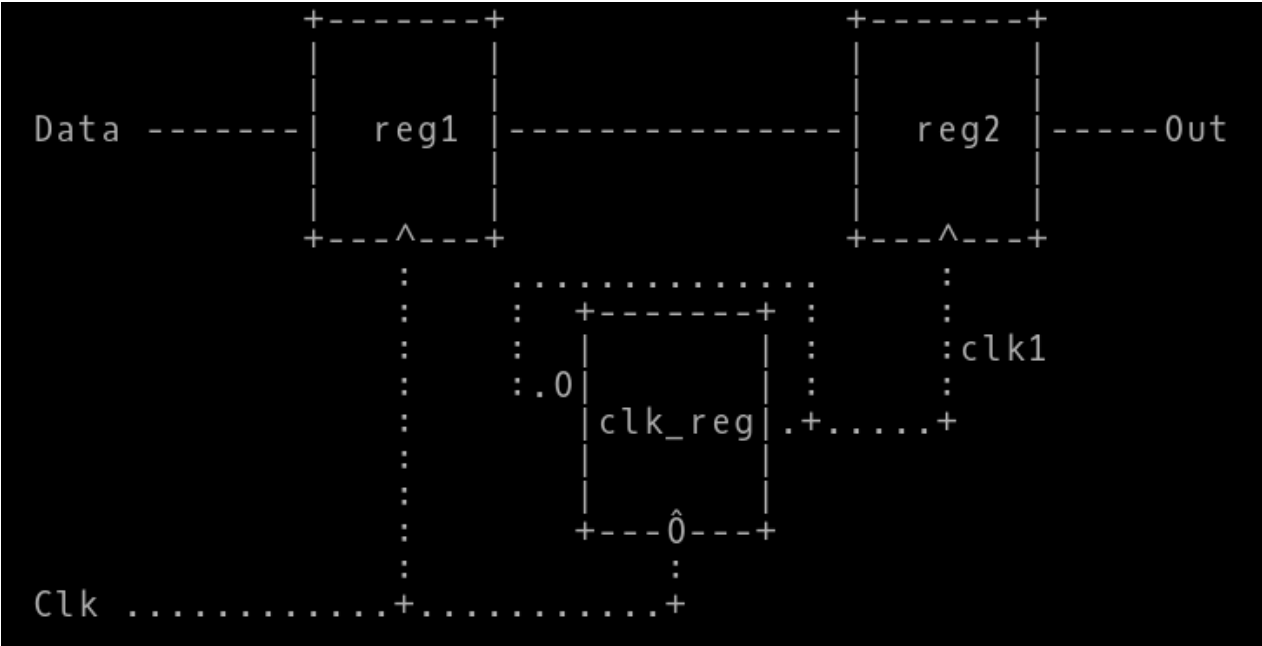
Arguments:

Name	Type	Description
source	string	the command which specifies how to get a source clock related point. A valid command can be: getClock(), getPort(port_name), getRegisterClock(register_name) (or getRegister(register_name)), getClockNet(clock_net_name), getWFGOutput(wfg_name)
target	string	the command which specifies how to get a clock related point. A valid command can be: getRegisterClock(register_name) (or getRegister(register_name)), getClockNet(clock_net_name)
Name	string	user clock name of the generated clock
relationship	dictionnary	the relationships for computing clock wave of generated clock from master clock. All valid parameters can be: MultiplyBy : unsigned DivideBy : unsigned DutyCycle : unsigned (1 to 99) Phase : unsigned (0 to 359) Offset : integer (delay in ps) Edges : list [unsigned, unsigned, unsigned] (in non-decreasing order) EdgeShift : list [integer, integer, integer] (delay in ps)

Important

Frequency-based and edge-based relationships are mutually exclusive.

Example:



In the example above, the master clock "Clk" was created as 100MHz and the generated clock "clk1" is divided by 2 from the master clock. But noted that the "clk_reg" is driven by falling edge of master clock, the relation between the master clock and generated clock is following:



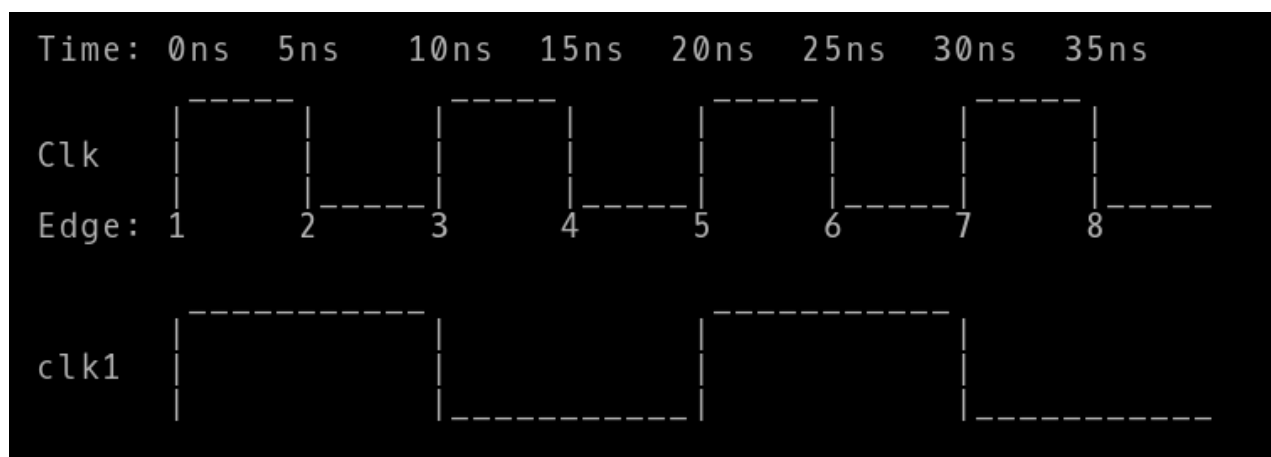
```
project = createProject()
project.load('routed.nxm')
project.createClock('getPort(Clk)', 'Clk', 10000)
or
project.createGeneratedClock('getRegisterClock(clk_reg)', 'getRegisterClock(reg2)', 'clk1', {'DivideBy': 2})
or
project.createGeneratedClock('getClock(Clk)', 'getRegisterClock(reg2)', 'clk1', {'Edges': [2, 4, 6]})
```

Important

The following script is incorrect for this case:

```
##### INCORRECT SCRIPT #####
project.createGeneratedClock('getClock(Clk)', 'getRegisterClock(reg2)', 'clk1', {'DivideBy': 2})
```

The relationship generated by above command is:



6.2.26 *createSimulator()*

This method is used to create a simulator object. A simulator object is used to simulate the project with a specified testbench.

This method takes no argument.

This method returns an object of class Simulator.

Example:

```
project = createProject()  
project.load('home/user/example/vhdl/simple/routed.nxm')  
analyzer = project.createSimulator()
```

6.2.27 *developCKGs()*

This method automatically creates a generated clock constraint on each output of the PLLs and WFGs in current project.

This constraint is used by timing driver algorithms and static timing analysis. This method takes no argument.

Note

"developCKGs()" allows the user to create generated clocks for CKGs' output pins. For example, before activating a generated clock whose base clock is driven by a WFG, user needs to launch this method for generating the base clock.

Without this method, *nxmap* automatically derives a clock on each output of the CKGs after activating all the given timing constraints.

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLK)', 'clk', 8000, 0, 4000)
project.developCKGs()
project.createGeneratedClock('getWFGOutput(wfg_clk[1])', 'getRegister(data_reg[0])', 'clk1_div2',
{'DivideBy': 2})
```

6.2.28 *destroy()*

This method is used to destroy the project.

This method takes no argument.

Note

Once the project is destroyed, every python variable referencing it becomes obsolete so *nxpython* behavior is safe.

Example:

```
project = createProject()
project.load('/home/user/example/vhdl/simple/synthesized.nxm')
project.display()
project.destroy()
print 'Display after destruction'
project.display()
```

Output:

```
Display after destruction
Traceback (most recent call last):
  File "test.py", line 22, in <module>
    project.display()
nanoxmap.error: Invalid Request: Obsolete object
```

6.2.29 display()

This method is used to open current project in *nxmap* graphic user interface. It can be called several times from the same Python script.

This method takes no argument.

Note

The project must have been at least synthesized before using this method.

Example:

```
project = createProject()
project.load('/home/user/example/vhdl/simple/native.nxm')

project.synthesize()
project.display()

project.place()
project.display()

project.route()
project.display()
```

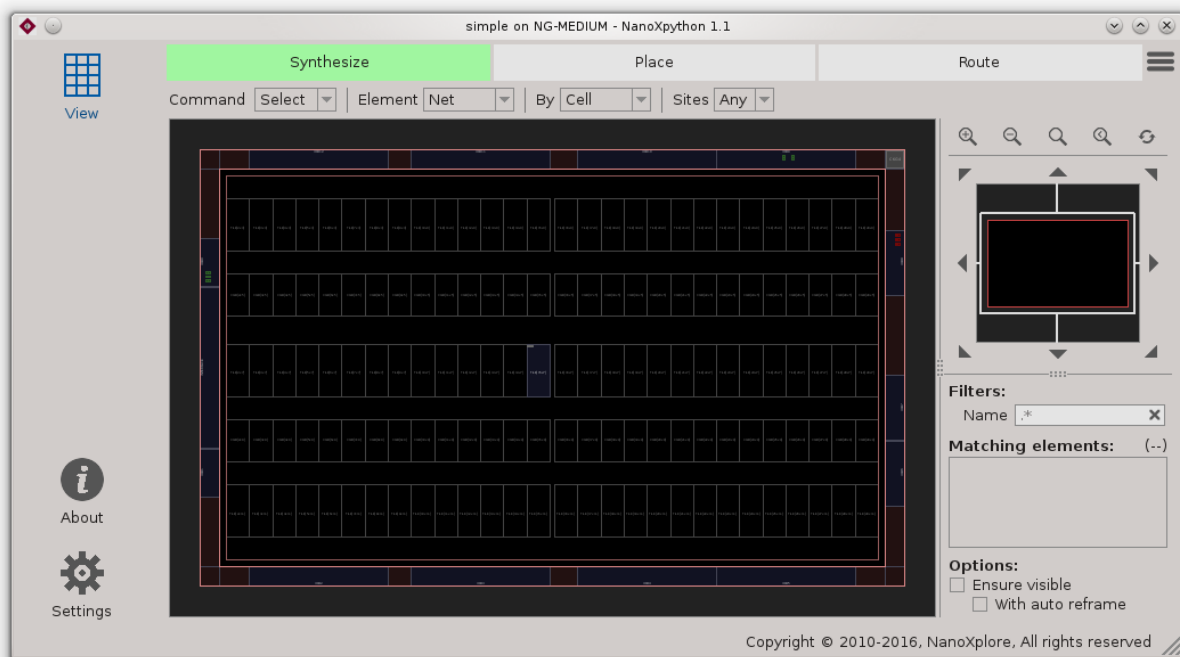


Figure 27: *nxmap* GUI called by `display()` method

Important

nxpython does not need an X11 environment to be run, but the `display()` method does. In order to use the `display()` method, user must add the `-gui` or `-g` option on the *nxpython* command line.

This option must be placed as option of the tool (not the script) as shown is the following example:
\$> *nxpython* -gui script.py

6.2.30 exportAsIPCore(file, options)

This method is used to export the synthesized design to an IPCore. An IPCore is a HDL description (at NX components level) that can be given as part of new source design to *nxmap*.

The export is different from the `save()` method as it is done during the second step of synthesis.

Arguments:

Name	Type	Description
file	string	path of the output file can be absolute or relative to the directory of the project.
options	dictionary	dictionary that contains optional configuration

The options dictionary must contain at least the 'coreName' key. The following keys are available:

Name	Description
coreName	the name of the top entity in the IPCore (default is top cell name during synthesis)
obfuscate	if set to 'Yes', the generated code is obfuscated (default is 'No')
encrypt	used to encrypt the generated file using IEEE P1735 standard. If set to 'None', the file is not encrypted, otherwise it is and the value corresponds to the software targeted: 'NX': to reuse file in <i>nxmap</i> 'MGC': to reuse file in QuestaSim 'All': to reuse the file in <i>nxmap</i> and / or QuestaSim
author	author information used in the encrypted file header
license	license information used in the encrypted file header

Example:

```
project = createProject()

project.load('/home/user/example/vhdl/simple/native.nxm')

options = { 'coreName': 'IPCore', 'encrypt': 'All' }

p.exportAsIPCore('IPCore.vhd', options)

p.synthesize()
```

6.2.31 *generateBitstream(file, [frames1], [frames2])*

This method is used to save the bitstream configuration file to disk. The extension of the target file must be .nxb

Arguments:

Name	Type	Description
file	string	path of the output file can be absolute or relative to the directory of the project.
frames1	list of string	bitstream frames to inject before configuration
frames2	list of string	bitstream frames to inject after configuration

Example:

```
project = createProject()

project.load('/home/user/example/vhdl/simple/routed.nxm')

frame1 = []
frame2 = ['LF(255, CMIC_DELAY, 000002FF)'] # set CMIC_DELAY to 2FF FFFF = ~1s @50MHz
                                           # the 16 lower bits are forced to 1 in hardware

project.generateBitstream('bitstream.nxb', frame1, frame2)
```

6.2.32 *getDirectory()*

This method returns the project current directory. This directory is used as root for all relative paths passed as argument of generic functions and/or methods. The default value of the directory is the current working directory.

This method takes no argument.

Example:

```
project = createProject()  
directory = project.getDirectory()
```


6.2.33 *getTopCellName()*

This method returns the name of the top HDL module. If the HDL module is defined inside a library, the name is prefixed by the name of the library followed by a # character.

This method takes no argument.

Example:

```
project = createProject()
project.setTopCellName('myLib', 'simple')
printText(project.getTopCellName())
```

Output:

```
myLib#simple
```

6.2.34 *getVariantName()*

This method returns the variant of the project. The default value of the variant is 'NG-MEDIUM'.

This method takes no argument.

Example:

```
project = createProject()  
printText(project.getVariantName())
```

Output:

```
NG-MEDIUM
```

6.2.35 *load(file)*

This method is used to load a previously saved project from disk. The input format (file extension) must be .nxm (*nxmap* archive).

Arguments:

Name	Type	Description
file	string	<i>nxmap</i> archive file (extension must be .nxm).

Note

Path of the input file can be absolute or relative to the directory of the project.

Example:

```
project = createProject('/home/user/example/vhdl/simple')  
project.load('synthesized.nxm')
```

6.2.36 *place()*

This method is used to run the place algorithm on a project.

This method takes no argument.

Note

- The project must have been created using the "createProject()" method and loaded with the load method or configured before using this method.
- The option 'TimingDriven' may be assigned before placement.
- If the project is not synthesized, the "synthesize()" method will be called first.

Example:

```
project = createProject()  
project.load('/home/user/example/vhdl/simple/synthesized.nxm')  
project.setOption('TimingDriven', 'Yes')  
project.place()
```

6.2.37 reportInstances()

This method reports the instances utilization of the programmed design on the current variant. Three tables are printed. The first one contains values for all types of instances. The instances are grouped by categories:

4-LUT	Look up table with a maximum of 4 connected entries
DFF	DFF register
X-LUT	"X-LUT" as defined in datasheet
4-Bits Carry	"4-bits carry look ahead" as defined in datasheet
Register file block	"Register files" as defined in datasheet
Cross domain clock	A two "DFF" pipeline optimized for metastability problem
Clock switch	Elements which logically interrupt a clock without shortening current cycle
Digital signal processor	"DSP" as defined in datasheet
Memory block	"Memory" as defined in datasheet
WFG	"WFG" as defined in datasheet
PLL	"PLL" as defined in datasheet

The second table gives more detailed about 4-LUTs utilization:

Synthesized from HDL	Number of LUTs that have been instantiated from the design
Used by DFF	Number of LUTs that are used by DFFs
Used by CY	Number of LUTs that are occupied by Carrys
Used by RF	Number of LUTs that are occupied by RFs
Used by CKS	Number of LUTs that are occupied by CKSs
Used by Internal	Number of LUTs created while processing the design to solve architectural constrains
Used by: Total	Total number of LUTs NOT instanciated from the design
4-LUT count	Overall use of 4-LUT for the current design

The last table gives detailed information about the use of registers:

SFE	Number of DFF registers used by LUTs
Carry	Number of DFF registers used by Carries
DFF sub-total	Total count of DFF registers
CKS	Number of registers used in CKSs
RF	Number of registers used in RFs
DSP	Number of registers used in DSPs
RAM	Number of registers used in RAMs
Pads	Number of DFFs merged in pads
Total	Overall use of registers for the current design

This method takes no argument.

Example:

```
project = createProject()

project.load('/home/user/example/vhdl/simple/synthesized.nxm')

project.reportInstances()
```

Output:

Reporting instances									
4-LUT	DFF	XLUT	4-bits carry	Register file block					
8177/32256 (26%)	1623/32256 (6%)	339/2016 (17%)	95/2016 (5%)	56/168 (34%)					
Cross domain clock	Clock switch	Digital signal processor	Memory block	WFG	PLL				
0/168 (0%)	0/336 (0%)	0/112 (0%)	0/56 (0%)	2/32 (7%)	0/4 (0%)				
Reporting Detail of 4-LUTs use									
Synthesized From HDL	Used by								Total 4-LUT Count
	DFF	CY	RF	CDC	CKS	Internal	Total		
5192	749	347	1764	0	0	125	2985	8177/32256 (26%)	
Reporting Detail of Register use									
SFE	Carry	Dff sub-total	CDC	CKS	RF	DSP	RAM	Pads	Total
1621	2	1623/32256 (6%)	0	0	1764	0	0	0/1122 (0%)	3387

Figure 28: Output tables from reportInstances() method

6.2.38 reportPorts()

This method reports the ports found in HDL source and the ones manually added by user in the project. The report displays the following information:

Name	Name of the HDL port
Direction	Direction of the port (can be input, output or inout)
Location	Pad location associated to the port (can be set by the user or automatically set by <i>nxmap</i>)
Type	Type of the associated pad
SlewRate	Slew rate value of the associated pad
InputDelayLine	Input delay line of the associated pad
OutputDelayLine	Output delay line of the associated pad
Differential	True if the associated pad is in differential mode
Turbo	True if the associated pad is in turbo mode

This method takes no argument.

Example:

```
project = createProject()
project.load('/home/user/example/vhdl/simple/placed.nxm')
project.reportPorts()
```

6.2.39 *reportPowerConsumption()*

This method reports an estimate of the power consumption of the programmed design. The power consumption is split in two values:

- static Power consumed by the chip when idle.
- dynamic Power consumed when the design is running, given in W/Mhz.

This method takes no argument.

Example:

```
project = createProject()  
project.load('/home/user/simple/routed.nxm')  
project.reportPowerConsumption()
```


6.2.40 *resetTimingConstraints()*

This method is used to reset all timing constraints for current project.

This includes clocks, generated clocks, derived clocks of PLLs and WFGs, input delays, output delays, clock groups, analysis case, false paths, multicycle paths, min delay paths, and max delays paths.

This method takes no argument.

Note

- After `resetTimingconstraints()` is called, the timing analyzer need to be recreated by "**Project.createAnalyzer()**".

Example:

```
project = createProject()  
project.load('/home/user/simple/routed.nxm')  
project.resetTimingConstraints()
```

6.2.41 route()

This method is used to run the route algorithm on a project.

This method takes no argument.

Note

- The project must have been created using the "createProject()" method and loaded with the load method or configured before using this method.
- The option 'TimingDriven' may be assigned before routing.
- If the project is not placed, the "place()" method will be called first.

Example:

```
project = createProject()  
project.load('/home/user/example/vhdl/simple/placed.nxm')  
project.setOption('TimingDriven', 'Yes')  
project.route()
```

6.2.42 *save(file)*

This method is used to save the current project to a *nxmap* archive (.nxm) or to a design netlist (.v or .vhd).

Arguments:

Name	Type	Description
file	string	Target file (extension must be .nxm, .v, .vhd or .sdf)

Note

Project must be routed to save a .sdf file.

Example:

```
project = createProject()

project.load('native.nxm')

project.synthesize()
project.save('synthesized.nxm')
project.save('synthesized.vhd')

project.route()

project.save('routed.vhd')
project.save('routed.sdf')
```

6.2.43 *savePorts(filename)*

This method is used to save the current port placement of the project to a Python script file.

Arguments:

Name	Type	Description
filename	string	Name of the output file

Example:

```
project = createProject()
project.load('native.nxm')
project.place()
project.savePorts('ports.py')
```

Note

This method is useful to save the automatic pad placement done by the *nxpython* place algorithm.

6.2.44 *setCaseAnalysis(value, net_list)*

This method is used to specify a constant logic value to assign to the given nets. This constraint is used by timing driver algorithms and static timing analysis.

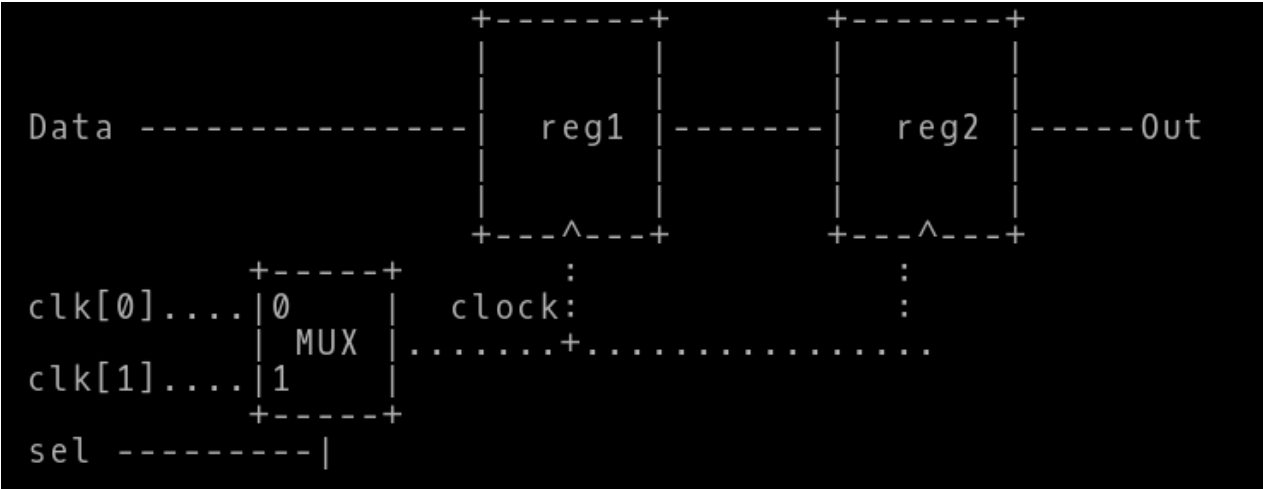
Arguments:

Name	Type	Description
value	unsigned	the valid constant values are 0 or 1
net_list	string	the command which specifies how to get one or several nets. A valid command can be: getNet(net_name), getNets(net_name_expression), getPort(port_name), getPorts(port_name_expression)

Note

Setting a case value on a net results in disabling timing analysis through the emitter pin and all the receiver pins of the net. It means that timing paths through those pins are not considered. The constant value is propagated through the network as long as a controlling value for the traversed logic is at the constant value.

Example:



In the example above, two clocks (clk[0] and clk[1]) are connected to the inputs of the multiplexer, but only clk[1] is propagated through the output after setting the constant value on the selection signal (sel).

```
project = createProject()
project.load('routed.nxm')
project.setCaseAnalysis(1, 'getNet(sel)')
```

6.2.45 **setClockGroup(group1_list, group2_list, option)**

This method is used to specify which clocks are not related. This constraint is used by timing driver algorithms and static timing analysis.

Note

A clock cannot be in a different group from itself.

Arguments:

Name	Type	Description
group1_list	string	the command which specifies how to get a group of clocks. A valid clock should be a clock created by command "createClock". A valid command can be: getClock(clock_name) and getClocks(name_expression).
group2_list	string	same as the argument "group1_list"
option	string	a valid option can be 'asynchronous' or 'exclusive': Asynchronous clocks are those that are completely unrelated. Exclusive clocks are not actively used in the design at the same time

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLOCK[1])', 'clk1', 2700)
project.createClock('getClockNet(CLOCK[2])', 'clk2', 5000, 0, 2000)
project.setClockGroup('getClock(clk1)', 'getClock(clk2)', 'exclusive')
```

6.2.46 *setInputDelay(clock, clock_mode, minimum_delay, maximum_delay, port_list)*

This method specifies the data arrival times at the specified input ports relative to the clock. The clock must refer to a clock name in the design. This constraint is used by timing driven algorithms and static timing analysis.

Arguments:

Name	Type	Description
clock	string	the command which specifies how to get a clock specified. A valid clock should be a clock created by command "createClock". A valid command: getClock(clock_name).
clock_node	string	specifies that input delay is relative to the falling or rising edge of the clock. It must be "rise" or "fall".
minimum_delay	integer	applies value as minimum data arrival time.
maximum_delay	integer	applies value as maximum data arrival time.
port_list	string	the command which specifies how to get a list of input pads. A valid command can be: getPort(port_name), getPorts(name_expression).

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLK)', 'CLK', 8000)
project.setInputDelay('getClock(CLK)', 'rise', 1000, 1500, 'getPort(RST)')
```

6.2.47 setOption(name, value)

This method is used to set the value of a *nxpython* option. An option can be set at any time of the design flow.

Arguments:

Name	Type	Description
name	string	the name of the option
value	string	the value of the option

The available options are:

Name	Default	Description
'AdderToDSPMapThreshold'	'0'	Is obsolete as of release 2.9.0 of nxmap
'DefaultFSMEncoding'	'OneHot'	Default encoding of finite state machine (can be 'OneHot', 'OneHotSafe', 'OneHotSafeExtra' or 'Binary').
'DefaultRAMMapping'	'RF'	Default mapping of RAM (can be onto 'RF' or 'RAM').
'DefaultROMMapping'	'LUT'	Default mapping of ROM (can be onto 'LUT', 'RF' or 'RAM').
'DisableAdderBasicMerge'	'No'	Disable carry optimization around adders and subtractors.
'DisableAdderTreeOptimization'	'No'	Disable adder mux reordering and adder tree balancing.
'DisableAdderTrivialRemoval'	'No'	Disable simplification of adder that could fit in 1 or 2 LUTs.
'DisableDSPAluOperator'	'No'	Disable merge of ALU within inferred DSP.
'DisableDSPFullRecognition'	'No'	Disable inference of DSP.
'DisableDSPPreOperator'	'No'	Disable merge of pre-operator within inferred DSP.
'DisableDSPRegisters'	'No'	Disable merge of registers within inferred DSP.
'DisableLoadAndResetBypass'	'No'	Disable load and reset signal bypass on DFF.
'DisableRAMAlternateForm'	'No'	Disable recognition of registered address read port.
'DisableRAMRegisters'	'No'	Disable merge of registers within inferred RAM.
'ExhaustiveBitstream'	'No'	Can be used to force generation of all configurations and contexts in bitstream (can be 'No', 'Config', 'Context' or 'ConfigContext').
'ExportAsIPCore'	"	Is obsolete as of release 2.9.0. Use exportAsIPCore method of project instead.
'GenerateIntermediateArchives'	'Never'	If set to 'Always', .nxi archives are generated for each intermediate step of the flow (for debug purpose).
'GenerateBitstreamCMIC'	'Always'	Enable/Disable CMIC in bitstream.
'IgnoreRAMFlashClear'	'No'	Do not output error when recognizing a RAM with flash clear.
'LessThanToDSPMapThreshold'	'0'	Is obsolete as of release 2.9.0 of nxmap
'ManageAsynchronousReadPort'	'No'	If 'Yes', detect asynchronous read port in memories and repair it in synchronous read port. The read port receive the reversed write clock. It can slow down the design and sometimes may cause invalid behavior.
'ManageUnconnectedOutputs'	'Error'	Undriven outputs of HDL modules are treated as 'Error', 'Ground' or 'Power'.
'ManageUnconnectedSignals'	'Error'	Undriven internal signals of HDL modules are treated as 'Error', 'Ground' or 'Power'.
'ManageUninitializedLoops'	'Never'	Remove reset-less looped DFF causing extra-mapping and 'X' values in simulation (can be 'Never', 'Ground', 'Always').
'MappingEffort'	'High'	Effort for an optimized mapping (can be 'Low', 'Medium' or 'High').

'MaxRegisterCount'	'2500'	Maximum number of registers handled per HDL module (not the whole design) by the synthesizer.
'MergeRegisterToPad'	'Never'	Automatically merge registers into IO buffers (can be 'Always', 'Never', 'Input' or 'Output').
'MultiplierToDSPMapThreshold'	'0'	Is obsolete as of release 2.9.0 of nxmap
'OptimizedMux'	'Yes'	If set to 'Yes', nxmap will identify and convert every mux in the corresponding optimized 4-LUT structure.
'Rollback'	'Yes'	Allow the application to roll-back to a previous state when an error is encountered during Routing, and resume the process with new parameters.
'TimingDriven'	'No'	If set to 'Yes', algorithms are timing driven (can be 'Yes' or 'No').
'UnusedPads'	'Floating'	State in which the pads must be set when not used. Values can be 'Floating', 'WeakPullUp', 'WeakPullDown'. Note that when the state is different from 'floating', all the pads are serialized in the bitstream. Note that 'WeakPullDown' value is not available on NX1H35S component.
'UseNxLibrary'	'Yes'	If set to 'Yes', physical elements from nxLibrary.vhdp can be directly instantiate in the source file.
'UseSynthesisRetiming'	'No'	If set to 'Yes', retiming algorithms will be enabled during synthesis.
'UseXLUT'	'No'	If set to 'Yes', Place stage will use XLUTs to reduce interconnexion timing between LUTs (in some cases, using XLUTs can degrade final frequency).
'VariantAwareSynthesis'	'Yes'	If set to 'Yes' synthesis will automatically map to equivalent resource when specific resource is depleted. For example using DSP when there are no more CY available (can be 'Yes' or 'No').

Important

Bigger the 'MaxRegisterCount' value, bigger the memory consumption by the synthesizer.

Some options can affect only specific processes of the design flow. Following, a table of the specific processes per options:

Name	Synthesize	Place	Route	Bitstream
'DefaultFSMEncoding'	X			
'DefaultRAMMapping'	X			
'DefaultROMMapping'	X			
'DisableAdderBasicMerge'	X			
'DisableAdderTreeOptimization'	X			
'DisableAdderTrivialRemoval'	X			
'DisableDSPAluOperator'	X			
'DisableDSPFullRecognition'	X			
'DisableDSPPreOperator'	X			
'DisableDSPRegisters'	X			
'DisableLoadAndResetBypass'			X	
'DisableRAMAlternateForm'	X			
'DisableRAMRegisters'	X			
'ExhaustiveBitstream'				X
'MaxRegisterCount'	X			
'MergeRegisterToPad'	X			
'OptimizedMux'	X			
'Rollback'		X	X	

'TimingDriven'	X	X	X	
'UnusedPads'				X
'UseNxLibrary'	X			
'UseSynthesisRetiming'	X			
'UseXLUT'		X		
'VariantAwareSynthesis'	X			

Example:

```
project.setOption('MappingEffort', 'Medium')
```

6.2.48 *setOptions(options)*

This method allows user to set the values of several *nxpathon* options. The options can be set at any time of the design flow.

Arguments:

Name	Type	Description
options	dictionary	the keys of the dictionary are the options names and the values are the options values.

For a list of all available options, see 6.2.47.

Example:

```
options = {  
    'MappingEffort': 'Medium',  
    'AdderToDSPMapThreshold': '12'  
}  
  
project.setOptions(options)
```

6.2.49 ***setOutputDelay(clock, clock_mode, minimum_delay, maximum_delay, port_list)***

This method specifies the data arrival times at the specified output ports relative to the clock. The clock must refer to a clock name in the design. This constraint is used by timing driven algorithms and static timing analysis.

Arguments:

Name	Type	Description
clock	string	the command which specifies how to get a clock specified. A valid clock should be a clock created by command "createClock". A valid command can be: getClock(clock_name).
clock_mode	string	specifies that output delay is relative to the falling or rising edge of the clock. It must be "rise" or "fall".
minimum_delay	integer	applies value as minimum data arrival time.
maximum_delay	integer	applies value as maximum data arrival time.
port_list	string	the command which specifies how to get a list of output pads. A valid command can be: getPort(port_name), getPorts(name_expression).

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLK)', 'CLK', 8000)
project.setOutputDelay('getClock(CLK)', 'rise', 1000, 1500, 'getPorts(dataout\[0-5\])')
```

6.2.50 *setTopCellName*([library], name)

This method is used to set the name of the HDL top module. If the HDL module is defined inside a library, the name of the library can be passed as the first argument.

Arguments:

Name	Type	Description
library	string	name of the HDL library that contains the top module.
name	string	name of the HDL top module.

Example:

```
project.setTopCellName('simple')  
  
or  
  
project.setTopCellName('myLib', 'simple')
```

Important

The name of the HDL top module and the name of library are case sensitive.

6.2.51 *setVariantName(variant)*

This method is used to set the variant of the project. The default value of the variant is 'NG-MEDIUM'.

Arguments:

Name	Type	Description
variant	string	the name of the variant.

The following variants are available:

Family	Registers	4-LUTs	XLUTs	Carries	RAMs	RAM bits	DSPs	Clocks
NG-MEDIUM	32256	32256	2016	8064	56	2016 K	112	24
NG-LARGE	129024	129024	8064	32256	192	6912 K	384	36

Here is a table reporting the amount of available pads in normal variants:

Variant	IO Pads	HSSL
NG-MEDIUM	374	0
NG-LARGE	640	4 x 6 links

Here is a table reporting the amount of available pins in embedded variants:

Variant	Input Pins	Output Pins
NG-MEDIUM-EMBEDDED	8440	8440
NG-LARGE-EMBEDDED	15402	15402

Example:

```
project = createProject()
project.setVariantName('NG-MEDIUM')
```

6.2.52 *synthesize()*

This method is used to synthesize the HDL source files and realize the technology mapping against *nxython* library.

This method takes no argument.

This method uses the following options:

- 'MappingEffort',
- 'MaxRegisterCount',
- 'MergeRegisterToPad',
- 'DefaultFSMEncoding',
- 'AdderToDSPMapThreshold',
- 'ManageUnconnectedOutputs',
- 'ManageUnconnectedSignals',
- 'LessThanToDSPMapThreshold',
- 'MultiplierToDSPMapThreshold'

This method also uses all the blackboxes and mapping directives set.

Note

The project must have been created using the "createProject()" function and loaded with the 'Project.load' method or configured before using this method.

Example:

```
project = createProject()
project.load('native.nxm')
project.synthesize()
```

6.3 Static Timing Analysis related methods

This section presents the methods related to the analyzer object. This object controls the static timing analysis.

6.3.1 *destroy()*

This method is used to destroy the analyzer object.

This method takes no argument.

Note

Once the analyzer is destroyed every python variable referencing it becomes obsolete so *nxpython* behavior is safe.

Example:

```
project = createProject()
project.load('/home/user/example/vhdl/simple/routed.nxm')
analyzer = project.createAnalyzer()
analyzer.launch()
analyzer.destroy()
```


6.3.2 *launch(parameters)*

This method is used to run the static timing analysis.

The analyzer computes a maximum of 'searchPathsLimit' paths for each domain, considering only the paths which slack is less than or equal to 'maximumSlack'.

Temperature and voltage can also be supplied to adjust timing values.

When not given, the parameters take their default values.

Arguments:

Name	Type	Description
parameters	dictionary	Keys are the names of the parameters to set (see following table), values must match the type specified.

Available Parameters:

Name	Type	Description (Default value in bold)
searchPathsLimit	unsigned	maximum number of paths computed for each domain. (default value is 10)
maximumSlack	integer	maximum reportable slack in ps. (default is unlimited)
temperature	float	Temperature used to apply derating on timings. (authorized values are -40, 25 , 125)
voltage	float	Voltage used to apply derating on timings. (authorized values are 1.1, 1.2 , 1.3)

Example:

```
analyzer = project.createAnalyzer()

parameters = { 'searchPathsLimit': 15, 'temperature': 125 }
analyzer.launch (parameters)

parameters = { 'searchPathsLimit': 15, 'maximumSlack' : 500, 'temperature': -40 }
analyzer.launch (parameters)
```

Note

The project must be routed to run this method.

6.3.3 Launching timing analyzer steps

The steps involved in *nxpython* STA are the following:

Step 1. Creating Analyzer:

If a targeted design is already routed, user can launch static timing analysis by creating a timing analyzer. The analyzer can be created using the following command:

```
analyzer = project.createAnalyzer()
```

Step 2. Launching Static Timing Analysis:

Static Timing Analysis can be launched using the following command:

```
analyzer.launch()
```

or

```
analyzer.launch(parameters)
```

Please check the previous section 6.3.2 for more information on how to use the `launch()` function.

Step 3. Generating Static Timing Analysis reports:

In *nxpython*, a number of reports (.timing) can be produced presenting the results in the form of timing domains, critical paths, violation details, etc... The explanation of these reports can be found in the section 4.6.3.

Note

Currently, Static Timing Analysis can only be performed after a successful routing of the design.

6.4 Simulation related methods

This section presents the methods of the simulator object. This object controls the simulation of the design.

6.4.1 *addTestBench*(*[library]*, *testBench*)

This method is used to add a testBench file. The file can be added to a specific library by passing its name as the first optional argument 'library'. The default library is 'work'.

Arguments:

Name	Type	Description
library	string	Name of the HDL library that contains the testbench file.
testBench	string	Testbench file relative to the project directory.

Example:

```
simulator.addTestBench('testbench.vhd')
```

or

```
simulator.addTestBench('myLib', 'testbench.vhd')
```

6.4.2 *addTestBenchs([library], fileList)*

This method is used to add several testbench files. The files can be added to a specific library by passing its name as the first optional argument 'library'. The default library is 'work'.

Arguments:

Name	Type	Description
library	string	Name of the HDL library that contains the testbench file.
fileList	list	List of testbench files relative to the project directory.

Example:

```
simulator.addTestBenchs(['testbench1.vhd', 'testbench2.vhd'])  
  
or  
  
simulator.addTestBenchs('myLib', ['testbench1.vhd', 'testbench2.vhd'])
```

6.4.3 *addWave(wave)*

This method is used to add a signal which will be displayed in vsim. It's also possible to add all the available waves by passing '*' as argument.

Arguments:

Name	Type	Description
wave	string	Name of the signal.

Example:

```
simulator.addWave('*')
```

or

```
simulator.addWave('A')
```

6.4.4 *addWaves(waveList)*

This method is used to add several signals which will be displayed in vsim.

Arguments:

Name	Type	Description
waveList	list	List of signal.

Example:

```
simulator.addWaves(['A', 'B', 'O'])
```

6.4.5 *destroy()*

This method is used to destroy the simulator object.

This method takes no argument.

Note

Once the simulator is destroyed every python variable referencing it becomes obsolete so *nxpython* behavior is safe.

Example:

```
simulator.destroy()
```

6.4.6 *launch(graphical)*

This method is used to run the simulation.

To run the simulation, the simulator object must be fully initialized. You must at least have specified the testbench files and the testbench top cell.

The simulation is launched on a netlist representing the current state of the project.

Note

If a simulation is launched when the project is on its native state, the source files must have been added to the project in the right compile order. Otherwise the simulation will fail.

Arguments:

Name	Type	Description
graphical	boolean	Specify if the vsim GUI should be launched or not.

Example:

```
project = createProject()  
  
project.load('routed.nxm')  
  
simulator = project.createSimulator()  
  
simulator.setTestBenchTop('testbench')  
simulator.addTestBench('testbench.vhd')  
  
simulator.launch(True)
```


6.4.7 *setTestBenchTop(testBenchTop)*

This method is used to set the top cell name of the testbench.

Arguments:

Name	Type	Description
testBenchTop	string	Top cell of the testbench.

Example:

```
simulator.setTestBenchTop('testbench')
```

6.4.8 *setWorkingDirectory(workingDirectory)*

This method is used to set the working directory of the simulation.

Arguments:

Name	Type	Description
workingDirectory	string	Working directory, relative to the project directory.

Example:

```
simulator.setWorkingDirectory('simulation')
```

6.5 Argument related methods

6.5.1 *getClock(clock_name)*

This argument method is used to get the clock from a given clock name. A valid clock should be a clock created by command "createClock".

Arguments:

Name	Type	Description
clock_name	string	the name of the clock used to get the valid clock.

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLOCK[1])', 'clk1', 2700)
project.createClock('getClockNet(CLOCK[2])', 'clk2', 5000, 0, 2000)
project.setClockGroup('getClock(clk1)', 'getClock(clk2)', 'exclusive')
```

6.5.2 *getClockNet(clock_net_name)*

This argument method is used to get the net from a given clock net name. It is used by “createClock” method to specify a clock related point as argument.

Arguments:

Name	Type	Description
clock_net_name	string	the name of the net clock.

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLOCK[1])', 'clk1', 2700)
project.createClock('getClockNet(CLOCK[2])', 'clk2', 5000, 0, 2000)
```

6.5.3 *getClocks(clock_name_expression)*

This argument method is used to get the clocks from a given clock name expression. A valid clock should be a clock created by command "createClock".

Arguments:

Name	Type	Description
clock_name_expression	string	the name expression of the clock used to get the list of clocks.

Example:

```
project = createProject()

project.load('routed.nxm')

project.createClock('getClockNet(CLOCK[1])', 'clk1', 2700)
project.createClock('getClockNet(CLOCK[2])', 'clk2', 5000, 0, 2000)
project.createClock('getClockNet(CLOCK[3])', 'test_clkA', 5000, 0, 2000)
project.createClock('getClockNet(CLOCK[4])', 'test_clkB', 5000, 0, 2000)

project.setClockGroup('getClocks(clk*)', 'getClocks(test_clk*)', 'exclusive')
(project.setClockGroup('getClocks(clk1, clk2)', 'getClocks(test_clkA, test_clkB)', 'exclusive'))
(project.setClockGroup('getClocks(clk[1-2])', 'getClocks(test_clk[AB])', 'exclusive'))
```

6.5.4 *getInstances(instance_name)*

This argument method is used to get the instances from an instance name. It can be used by some *nxython* methods (“addMappingDirective”, “addMemoryInitialization”).

Arguments:

Name	Type	Description
instance_name	string	the name of the instance used to get the valid instance.

Example:

```
project = createProject()
project.addMappingDirective('getInstances(smul_4|mult_S)', 'MUL', 'DSP')
```

Note

Instance name must match the following pattern: 'PATHNAME|INSTANCENAME'.

It can be useful to use ".*" in getInstances method as following:

```
project.addMappingDirective('getInstances(*myInstance)', 'RAM', 'RAM')
```

The first ".*" is used because your instance is defined in a path.

Note

The list of all the instances identified by *nxmap* from your design is available in the log file operators.rpt in logs directory.

It is also possible to get the list of all the instances identified by *nxmap* using ".*" expression as instance name in getInstances method, as following:

```
project.addMappingDirective('getInstances(.*)', 'RAM', 'RAM')
```

This way, you will see all the available instances in the general.log file in logs directory.

6.5.5 *getModels(model_name)*

This argument method is used to get the models from a model name. It can be used by some *nxython* methods ("addMappingDirective", "addMemoryInitialization").

Arguments:

Name	Type	Description
model_name	string	the name of the model used to get the valid model.

Example:

```
project = createProject()
project.addMappingDirective('getModels(myTest_myFifo)', 'RAM', 'DFF')
```

Note

It can be useful to use "." in getModels method as following:

```
project.addMappingDirective('getModels(.myFifo.)', 'RAM', 'RAM')
```

The first "." is used because your model is defined in a VHD library.

The second "." is used because your "myFifo" model uses some generic map and thus is renamed to myFifo(#id) by *nxmap*.

Note

The list of all the models identified by *nxmap* from your design is available in the log file operators.rpt in logs directory.

It is also possible to get the list of all the models identified by *nxmap* using "." expression as model name in getModels method, as following:

```
project.addMappingDirective('getModels(.)', 'RAM', 'RAM')
```

This way, you will see all the available models in the general.log file in logs directory.

6.5.6 *getPort(port_name)*

This argument method is used to get the port from a given port name. Some *nxython* methods need a port as argument.

Arguments:

Name	Type	Description
port_name	string	the name of the port used to get the valid port.

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLK)', 'CLK', 8000)
project.setInputDelay('getClock(CLK)', 'rise', 1000, 1500, 'getPort(RST))
```


6.5.7 *getPorts(port_name_expression)*

This argument method is used to get the ports from a given port name expression. Some *nxpython* methods need a list of ports as argument.

Arguments:

Name	Type	Description
port_name_expression	string	the name expression of the port used to get the list of ports.

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLK)', 'CLK', 8000)
project.setOutputDelay('getClock(CLK)', 'rise', 1000, 1500, 'getPorts(dataout\[[0-5]\])')
```

6.5.8 *getRegister(register_name)*

This argument method is used to get the register from a given register name. Some *npython* methods need a register as argument.

Arguments:

Name	Type	Description
register_name	string	the name of the register used to get the valid register.

Example:

```
project = createProject()
project.load('routed.nxm')
project.addFalsePath('getRegister(UUT1|Gen_seq[3].seq_i|temp_reg[1])',
                    'getRegister(UUT2|dout_reg[61])')
```

6.5.9 *getRegisterClock(register_name)*

This argument method is used to get the register clock from a given register name. Some *nxpython* methods need a clock as argument.

Arguments:

Name	Type	Description
register_name	string	the name of the register used to get the valid clock.

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getRegisterClock(reg1)', 'clk', 10000)
```

6.5.10 *getRegisters(register_name_expression)*

This argument method is used to get the registers from a given register name expression. Some *nxython* methods need a list of registers as argument.

Arguments:

Name	Type	Description
register_name_expression	string	the name expression of the register used to get the list of registers.

Example:

```
project = createProject()
project.load('routed.nxm')
project.addFalsePath('getRegisters(UUT1\Gen_seq[3]\.seq_i\temp_reg[[0-9]])',
                    'getRegister(UUT2\dout_reg[61])')
```

6.5.11 *getWFGOutput(wfg_name)*

This argument method is used to get the wfg clock from a given wfg name. Some *nxpathon* methods need a clock as argument.

Arguments:

Name	Type	Description
wfg_name	string	the name of the wfg used to get the valid clock.

Example:

```
project = createProject()
project.load('routed.nxm')
project.createClock('getClockNet(CLK)', 'clk', 8000, 0, 4000)
project.developCKGs()
project.createGeneratedClock('getWFGOutput(wfg_clk[1])', 'getRegister(data_reg[0]), 'clk1_div2',
                             {'DivideBy': 2})
```

6.6 Full script example

This chapter presents the full Python script of the provided VHDL 'simple' example.

Example:

```
import os
import sys

from nanoxmap import *

dir = os.path.dirname(os.path.realpath(__file__))

project = createProject(dir)

project.setVariantName('NG-MEDIUM')

project.setTopCellName('simple')

project.addFile('src/simple.vhd')

project.setOption('MappingEffort', 'Medium')

project.addPad('I1',    {'location': 'IO_B0D02P', 'type': 'LVCMOS_2.5V_2mA'})
project.addPad('I2',    {'location': 'IO_B0D03P', 'type': 'LVCMOS_2.5V_2mA'})
project.addPad('I3',    {'location': 'IO_B0D04P', 'type': 'LVCMOS_2.5V_2mA'})
project.addPad('CK1',   {'location': 'IO_B9D08P', 'type': 'LVCMOS_2.5V_2mA'})
project.addPad('CK2',   {'location': 'IO_B9D09P', 'type': 'LVCMOS_2.5V_2mA'})
project.addPad('O[0]',  {'location': 'IO_B8D02P', 'type': 'LVCMOS_2.5V_2mA', 'outputDelayLine': 5,
                        'differential': False, 'weakTermination': 'None'})
project.addPad('O[1]',  {'location': 'IO_B8D03P', 'type': 'LVCMOS_2.5V_2mA', 'outputDelayLine': 5,
                        'differential': False, 'weakTermination': 'None'})
project.addPad('O[2]',  {'location': 'IO_B8D04P', 'type': 'LVCMOS_2.5V_2mA', 'outputDelayLine': 5,
                        'differential': False, 'weakTermination': 'None'})

project.save('native.nxm')

if not project.synthesize():
    sys.exit(1)

project.save('synthesized.nxm')

if not project.place():
    sys.exit(1)

project.reportPorts()

project.save('placed.nxm')

if not project.route():
    sys.exit(1)

project.save('routed.nxm')

project.reportInstances()

#STA
analyzer = project.createAnalyzer()
```

```
analyzer.launch()  
  
#bitstream  
project.generateBitstream('simple.nxb')  
  
project.destroy()  
  
print 'Errors: ', getErrorCount()  
print 'Warnings: ', getWarningCount()
```