Automated Tool for Hardware EvaluatioN

ATHENa Tutorial

Version 0.6.4

Table of Contents

ATHENa 0.6.4 Tutorial

The ATHENa Team

ECE Department, George Mason University, Fairfax, VA 22030, U.S.A.
http://cryptography.gmu.edu/athena

1 Tools Installation

ATHENa requires four types of programs to support the execution of its scripts: a Perl interpreter, Synthesis Tool(s), Implementation Tool(s) and Simulation Tool(s). Generally, basic versions of synthesis and implementation tools are provided freely by FPGA vendors. However, these free versions of tools are not able to perform synthesis and implementation targeting all FPGA devices available on the market. In order to generate results for a full range of FPGA devices, you may need to acquire appropriate licenses.

ATHENa supports the following operating systems:

- Windows: XP/VISTA/WINDOWS 7
- Linux: Ubuntu v10.1

The free versions of tools can be downloaded from the following web sites:

- Perl Interpreter
  - For Windows : Use Strawberry Perl v5.12.3.0 and above
  - For Linux : Install Perl with the following libraries - **Libipc-run3-perl** and **Libarchive-zip-perl**
- Synthesis and Implementation Toolsets (for Windows and Linux)
  - Xilinx - ISE
  - Altera - Quartus II
- Simulation Tools (for Windows and Linux)
  - Xilinx - ModelSIM
  - Altera - ModelSim for Altera comes with Altera toolset

Synthesis and Implementation tools do not support all operating systems. For a list of supported operating systems see:

- Xilinx
- Altera

New features of ATHENa added in version 0.6.4 include:

- Support for Linux,
- New application: *GMU_Optimization_1* (Section 3.5),
- Support for Generics Search (Section 5.1),
- Database report generator (see 12),
- Support for reducing the size of generated files (see Appendix A, under TRIM_MODE),
- Support for Xilinx Spartan 6 and Virtex 6, and Altera Cyclone IV, Stratix IV, and Arria families,
- Support for Altera MegaWizard Plug-in Manager and Xilinx CORE Generator (Section 5.2)
- Support for Verilog.

Limitations of the current version of ATHENa (to be eliminated in the future versions) include:

- No support for third party synthesis tools (i.e., synthesis tools other than the tools provided by Xilinx and Altera).

– Support for batch mode simulation using ModelSim only (to be extended to other simulators in the future). Additionally, functional simulation will be extended with post-synthesis and timing simulation in the future versions of tools.

The **root directory** is the directory called ATHENa, where you have unpacked the ATHENa toolset. Inside the root directory you will find several script files which are used to perform specific tasks in ATHENa. Windows users need to run **script_name.bat** (batch) files, where as Linux users execute **script_name.sh** (shell) scripts. Once the tools are installed, please make sure that all of them contain appropriate licenses by running the main program of each tool by itself (i.e., outside of ATHENa) at least once. Then, You can proceed to run **ATHENa_setup.bat/ATHENa_setup.sh** to set the version of tools for Xilinx and/or Altera. Once finished, you must save and exit before any changes become effective. In the case that user wants to modify the vendor installation license, start **ATHENa_setup.bat/ATHENa_setup.sh**. Then, select the second choice Change selected tools license to enter license modification page. Once there, user can switch between free or paid license type as needed. Finally, user will need to save before any changes are made.

Important note for Windows users before you proceed:

Oftentimes, you may want to resize the command line windows for better visibility of messages generated by ATHENa_setup and ATHENa. You can perform this by right-clicking at the top of the command line window. This is the area next to the minimize, full screen and exit buttons. Once there, you can select "Properties" and go to the "Layout tab". Under Layout, select a larger buffer size and change window size appropriately. Click "OK" once done. This change will remain effective for new windows as well.

Running **ATHENa_setup.bat/ATHENa_setup.sh** generates **ATHENa.bat /ATHENa.sh** in your root directory, and **tool_config.txt** in the **config** folder located inside of your root directory. It also selects appropriate **device library** to use, depending on your tool version and license type (i.e. ISE Webpack vs. ISE Design Suite for Xilinx tools). Additionally, **ATHENa_setup** allows you to select the maximum number of logical processors to be used by ATHENa.

If **ATHENa_setup.bat** is unable to execute, user can assume that there is a problem related to Perl installation. This problem is normally caused by a missing module from Perl distribution. To resolve this problem, execute the setup script in a command line environment (DOS). If a module is missing, an error will be shown detailing the name of the missing module. The missing module can be obtained from cpan with the installation procedures described in Module Installation.

ATHENa.bat/ATHENa.sh is a batch/script file that you use for starting ATHENa operation. **tool_config.txt** is a configuration file containing your tool settings. Please note that if a device library for a given version of tools does not exist, you may select a different library as a substitute. Basic functionality of ATHENa can be accomplished even if a device library version does not match exactly the version of tools installed on your computer. However, ATHENa will support only FPGA devices specified in the current library file, even if FPGA tools support an extended set of devices.

If you want to change tool versions or the maximum number of logical processors to be used by ATHENa, you can run **ATHENa_setup.bat/ATHENa _setup.sh** again, navigate to Manual Setup and make changes accordingly. Otherwise, ATHENa generally selects the appropriate settings for you. Note that ATHENa may not be able to always find your FPGA tool installations. If that happens, you can manually specify the location of tools by going to the respective tool's menu.

The number of **logical processors** (also known as virtual processors) available in your system is calculated based on the following equation:

$$\text{Number of Logical Processors} = \text{Number of Cores} * \text{Hyper} - \text{Threading Factor}$$

where, Hyper-Threading Factor is equal to 2 if Hyperthreading is enabled, and 1 otherwise.

Finally, ATHENa may select an incorrect tool's installation type (e.g., Xilinx WebPACK instead Xilinx Design Suite) during the first setup. Hence, the selected library file will be incorrect and it will cause the

best_match and/or all options to perform incorrectly. User can change the settings by restarting **ATHENa_setup**, and then choosing the following sequence of options from the menu: Manual Setup → Synthesize and Implementation Tools Setup → <specific tool version> → Update listed version/library.

2 General Project Setup and Reports

In order to prepare your code for evaluation using ATHENa, edit the file **design.config.txt** located in the **config** subdirectory of your root directory.

Inside the file, specify the desired workspace directory using the variable **WORK_DIR**. This is a directory used as a root for all intermediate and result file directories. Then, you need to specify the location of the source folder using the variable **SOURCE_DIR**. This is a folder containing your VHDL and/or Verilog source files. Specify the remaining parameters according to the meaning of options as explained in Appendix A.

Once the configuration file is correctly prepared, execute **ATHENa.bat/ ATHENa.sh** located in your root folder to start. The results generated by the scripts will be shown in your command line window, as well as stored in text files located in the directory **WORK_DIR** under the following subfolder **${application}/${date}_${projectname}_${instance_no}**.

${application} is the type of application as specified in **design.config.txt**. $*date* is the date at which the project is run. *${projectname}* is the name of the project as specified in **design.config.txt**. *${instance_no}* is an instance number of the project. An instance number is used in order to distinguish folders created at different time during the same day.

In each run of ATHENa, five report files are generated. These are option, resource utilization, timing, execution time and summary reports. The option report contains information about the specific options of tools used in a given run (default tool options are not listed). The resource utilization report contains information about the use of FPGA resources. The timing report contains timing related results (such as maximum clock frequency after synthesis and after implementation, latency, throughput, etc.). The execution time report contains the execution time taken by each tool. Finally, the summary report combines information from all four aforementioned reports. Additionally, the same information is collected in two CSV (comma-separated values) files (one for Xilinx and one for Altera), which are more suitable for further electronic processing.

In addition to the report files, two log files called **athena_progress.txt** and **athena_log.txt** are generated. **athena_progress.txt** contains only major messages displayed on the screen during ATHENa run. **athena_log.txt** contains more detailed information about which specific FPGA tools were called, and with which options. This file should be also inspected for possible error and warning messages, in case an ATHENa run has not returned all expected results.

3 Application Setup

As of version 0.6, there are five supported ATHENa applications :

- **single_run**
- **placement_search**
- **exhaustive_search**
- **frequency_search**
- **GMU_Optimization_1**
- **GMU_Xilinx_optimization_1**

The tool selects an appropriate application based on the application name provided in the **APPLICATION = $APPLICATION_NAME** field in your **design.config.txt**. With the exception of the single_run

application, the tool will further look into a respective application configuration file contained in your **config** folder, which contains application specific settings. For instance, if you have your application set to

APPLICATION = placement_search

the tool will refer to application specific settings in **placement_search.txt**.

3.1 single_run

single_run is the most basic ATHENa application. It performs a single run through synthesis and implementation for all target FPGA devices specified in **design.config.txt**. Options of synthesis and implementation used for all these target devices are provided in the file:

options.<OPTIONS>_<OPTIMIZATION_TARGET>.txt

located in the subdirectory **config** of the root directory. **OPTIONS** and **OPTIMIZATION TARGET** are variables defined in **design.config.txt**. **OPTIONS** = **default** or **user.OPTIMIZATION_TARGET** = **speed**, **area**, or **balanced**.

See Appendix D for a list of groups of options that can be specified in the files **options.<OPTIONS>_<OPTIMIZATIO**

Note that, by default, the tool will also use the **OPTIONS** and **OPTIMIZATION_TARGET** provided in the design.config.txt as a starting point for other applications as well. Hence, you may want to create an option file for single_run to suit your needs and use this as a basis for starting more complex applications.

3.2 placement_search

placement_search is an application that allows running implementation automatically for multiple values of an option that determines a starting point of placement within a given FPGA device. These options include: **COST_TABLE** for Xilinx and **SEED** for Altera.

In placement_search.txt, user can specify the range of the given option values using the following format:

<number>; **or** <start_number>: <step> : <end_number>;

The first format allows specifying a single value of an option. The second format specifies a sequence of values starting from <start_number>, and ending with <end_number>, with the value incremented by <step> in each subsequent run. The specification of a value or a range has to be terminated by a semi-colon. A user can combine multiple specifications together, separated by semicolons. For instance,

XILINX_COST_TABLE_VALUES = 6; 13 : 6 : 30; 55;

means the values of the cost table equal to 6, 55 and the range from 13 to 30 with the step of 6. Thus, the implementation will be repeated for the following values of the cost table: 6, 13, 19, 25, and 55.

See Appendix E for a full list of variables of **placement_search.txt.**

3.3 exhaustive_search

exhaustive_search extends placement_search to allow a user to specify a range of option values for several types of options (beyond options that can be specified in placement_search).

In general, **exhaustive_search** is used to find optimum synthesis and implementation options for a given source code, target device, tools, tool versions, and optimization target (speed, area, or balanced).

The description of the exhaustive search strategy is given in the file: **exhaustive.<strategy_name>.txt**, where the string <strategy_name> can be chosen arbitrarily in the **exhaustive_search.txt**.

For each option, several possible values used in the exhaustive search are specified. All options are grouped into Level 1 options and Level 2 options. Level 1 options are investigated first in order to determine a number of best combinations given by the variable BEST_LEVEL_1_OPTION_SETS. For each set of options selected as best at Level 1, an exhaustive search is performed for all options at Level 2.

To select which strategy to use, simply use the **strategy_name** as a parameter to EXHAUSTIVE_SEARCH_STRATEGY option.

See Appendix F for a list of options of **exhaustive.<strategy_name>.txt**.

3.4 frequency_search

frequency_search is an optimization method that attempts to search for the best requested clock frequency value for a given design. The parameters of the search algorithms can be described in **frequency_search.txt** within the **ATHENa/config** directory. The **REQUESTED_FREQ_IMPROVEMENT_STEPS** option is a list of frequency improvement step sizes used by the algorithm. Proper care needs to be taken when choosing these values because the effectiveness of the search is closely tied to them. Some guidelines are outlined in Appendix H on how to choose these values for your particular design. User can specify improvement steps by the following format:

<improvement>% or <improvement 1>%, < improvement 2>%, ... < improvement X>%

The application steps through improvement steps from left to right. If the requested frequency is achieved, the same improvement step is reused. If not, then the next improvement step is used for the subsequent runs. The application will loop through all the specified improvements until no more improvement can be obtained.

This application is best suited for Xilinx FPGAs.

3.5 GMU_Optimization_1

GMU_Optimization_1 performs optimization specific to a vendor. For Xilinx, it combines frequency_search and placement_search with 3 optimization targets (Area, Speed and Balanced) and effort level. For Altera, only placement_search and optimization_target are combined as not much can be gained from frequency_search. The same syntax used by frequency_search and placement_search is applied here as well. For Xilinx, GMU_Optimization_1 works as follows. For each of the optimization target, an initial run of the design tools is generated using the default options as defined in **design.config.txt**. The frequency achieved in this initial run determines the starting point of the frequency search. After this initial frequency value is generated, the next run is executed with a requested frequency equal to the last achieved value increased by the percentage indicated by the first value in the **REQUESTED_FREQ_IMPROVEMENT_STEPS** list. The result from this run is used as the starting point for the next run and this process is continued until either zero or negative improvement is generated by the toolset. Once the increases in requested frequency no longer yield a positive effect on the achieved frequency, the highest achieved frequency is used as the requested frequency and the tool PAR options are set to a high effort. If the default tool options up to this point were equivalent to the high effort options this step is skipped. The previous incremental improvement process is continued using the high effort options until a positive effect on achieved frequency is no longer attainable. At this point the algorithm will change the placement options to try and generate a positive change in achieved frequency. The placement options are determined by the value chosen for **XILINX_COST_TABLE_VALUES**. The initial incremental improvement process is used again until no benefit from requested frequency increase is observed. At this point the highest achieved frequency is used as the basis for incremental improvement now using the step value indicated as the next value in the **REQUESTED_FREQ_IMPROVEMENT _STEPS** list. This process continues until all values within the list have been used. For Altera, placement_search application is applied for each optimization target.

The pseudo code for the Xilinx part of **GMU_Optimization_1** is provided below.

Pseudo Code:

F_{req} - Requested Frequency for the current run
F_{ach} - Frequency Achieved by the current run
F_{step} - Frequency Step Increase
$F_{step}(0)$ - first frequency step listed for **REQUESTED_FREQ_IMPROVEMENT_STEPS**
$F_{step}(i)$ - i^{th} frequency step listed for **REQUESTED_FREQ_IMPROVEMENT_STEPS**
Placement = ps_value_1, ps_value_N set of possible cost table values
ord (Placement) = cardinality of placement_set
Placement(j) j^{th} element from placement set

Run() Run the system with the currently selected settings

Step 1:
F_{req} = Default*;
Settings = Default*;
Run(F_{req},Default Effort);

Step 2:
While ($F_{ach} > F_{req}$)

$F_{req} = F_{ach} + (F_{ach} * F_{step}(0)/100)$;
Run(Freq,Default Effort);

Step 3:
Settings = High Effort;
Run(F_{req}, High Effort);
While ($F_{ach} > F_{req}$)

$F_{req} = F_{ach} + (F_{ach} * F_{step}(0)/100)$;
Run(F_{req}, High Effort);

Step 4:
Settings = High Effort;
Use Placement Search Options;

for j in 0 to ord (Placement)
Run(F_{req}, High Effort, Placement(j));
if ($F_{ach} > F_{req}$)
exit;
end for

While ($F_{ach} > F_{req}$)

$F_{req} = F_{ach} + (F_{ach} * Fstep(0)/100)$;
for j in 0 to ord (Placement)
Run(F_{req}, High Effort, Placement(j));
if ($F_{ach} > F_{req}$)
exit;
end for

Step 5:
Settings = High Effort;
Use Placement Search Options;
For(i = 0; i ¡ number of REQUESTED_F_req_IMPROVEMENT_STEPS; i++)

$F_{req} = F_{ach} + (F_{ach} * Fstep(i)/100)$;
for j in 0 to ord (Placement)
Run(F_{req}, High Effort, Placement(j));
if ($F_{ach} > F_{req}$)
exit;
end for

While ($F_{ach} > F_{req}$)

$F_{req} = F_{ach} + (F_{ach} * Fstep(i)/100)$;
for j in 0 to ord (Placement)
Run(F_{req}, High Effort, Placement(j));
if ($F_{ach} > F_{req}$)
exit;
end for

Default is either the default value, or the value specified by the user in **design.config.txt**.

The pseudo code for the Altera part of GMU_Optimization_1 is provided below:

F_{ach} - Frequency Achieved by the current run
$F_{highest}$ Highest Frequency achieved so far

Placement = ps_value_1, ps_value_N set of possible cost table values
Placement(j) j^{th} element from placement set

syn_effort_level = auto, fast
syn_opt_target = area, speed, balanced
fit_effort_level = auto, fast, standard
syn_effort_level(a) a^{th} element from syn_effort_level set
syn_opt_target(b) b^{th} element from syn_opt_target set
fit_effort_level(c) c^{th} element from fit_effort_level set
ord (set) - functions which returns cardinality of the set
Run() Run the system with the currently selected settings
for b in 1 to ord(syn_opt_target)
settings=(syn_effort_level=auto, syn_opt_target(b), fit_effort_level=standard);

for j in 0 to ord (Placement)
Run(settings, Placement(j));
if ($F_{highest} < F_{ach}$)
$F_{highest} = F_{ach}$
end if;

end for;
end for;

3.6 GMU_Xilinx_optimization_1

GMU_Xilinx_optimization_1 only works for Xilinx and performs similar operation to the Xilinx part in Section 3.5. However, instead of performing the frequency_search and placement_search for each of the optimization_targets, only one iteration is performed for the optimization target that provides the best result. The best optimization target is selected based on the throughput to area ratio of the single run result on each of the optimization target. Again, the same syntaxes used in previously described applications still applied here.

4 Functional verification of codes in ATHENa

4.1 Testbench Requirements

In order to allow for an automated functional verification of an HDL project in ATHENa using batch mode, an appropriate testbench must be prepared. ATHENa requires two specific features from the testbench in order to perform correctly.

First requirement:
The testbench must generate an output file called "athena_test_result.txt". This file should contain the keyword "pass". If it does not, then ATHENa will assume that the simulation has failed.

Note : Output file string should contain only the file name. For instance,

FILE output_file : TEXT OPEN WRITE_MODE is "athena_test_result.txt";

is fine. But,

FILE output_file : TEXT OPEN WRITE_MODE is "c:/athena_test_result.txt";

or

FILE output_file : TEXT OPEN WRITE_MODE is "some_folder/athena_test_result.txt";

will cause your simulation to fail in ATHENa.

Second Requirement:
A user must provide a simulation stop point for ATHENa. This can be done in two ways. First, the user can specify the maximum time up to which a simulation will run in the design configuration file. If there is no time specified, user must write a testbench in such a way that the end point is reached. The end point is reached only when there are no further changes in any signals inside the circuit. This means that all signals are stable from that point onwards; these signals include all inputs, outputs and clock signals. Normally, stopping the clock when there is no more input vectors is adequate in creating the required end point.

4.2 Specifying Design Configuration

All parameters inside "Verification Settings" of your design configuration are required. Only MAX_TIME_FUNCTIONAL_V is optional.

4.3 Specifying Test Vector Locations

If in your testbench, you specify your test vector location as a fixed path, you do need to include test vector files inside VERIFICATION_LIST_FILE. Otherwise, you must include the test vectors as a part of this list. Please note that a test vector file cannot have .vhdl, .vhd or .v extension if you are specifying it as a part of the list.

4.4 Examples

For examples on how to write the testbench according to the ATHENa's requirement, please refer to the **a**rraymult or **sha256_rs** projects inside of the **e**xamples folder. The corresponding design configuration files are also included in the respective project folders to allow immediate testing.

5 Synthesize and Implementation Features

5.1 Generics

ATHENa accepts generics in the design configuration file via two types of keywords:

```
GLOBAL_GENERICS_BEGIN
...
GLOBAL_GENERICS_END
and
GENERICS_BEGIN
...
GENERICS_END
```

The global generic keyword must be specified outside the FPGA_VENDOR clause. On the other hand, the basic generic keyword must be specified within the FPGA_FAMILY clause, where the keyword is applied to the selected family only. If the same generic name is specified in both location, the local generic values will overwrite the global generic values.

Inside the keyword, generic name and its corresponding value(s) can be specified as follows: name = value1, value2, value3, ..., valueX if specific combination of generics is required, user can specify a generic combo as shown below: (name1, name2, ..., nameX) = (name1_value, name2_value, ..., nameX_value), ...

If more than one generic name or combo are specified, ATHENa will automatically go through all possible combinations of the specified values and combos. Note that different format of generic specified in global and local generic keywords may cause ATHENa not to function properly. For instance, the following method of generic description should be avoided:

```
GLOBAL_GENERICS_BEGIN
(unrolled, h) =  (4, 256), (8,256)
GLOBAL_GENERICS_END

GENERICS_BEGIN
h = 512
GENERICS_END
```

Furthermore, a generic of type integer should be specified by its actual value and not by the constant value (in case you have it specified elsewhere), as the synthesis tool may not be able to recognize HDL constants properly. Finally, user should make sure that the tool is able to capture the generic parameter passed by ATHENa before letting it run autonomously. This can be done by looking at the synthesis report file and make sure that there are no errors/warnings generated.

5.2 Using ATHENa with Xilinx CORE Generator and Altera MegaWizard Plug-in Manager

Both Xilinx and Altera allow designers to use the libraries of Intellectual Property (IP) Cores. The use of components from these libraries is supported by special interactive programs, typically referred to as wizards. These programs allow users to choose a generic library component that best matches the required function, and then parameterize this component by answering a series of questions appearing on subsequent screens. Based on the provided input, the program generates a set of files that can be used to simulate, synthesize, and implement a higher-level circuit that instantiates a given IP core.

Xilinx: A wizard available with Xilinx tools is called CORE Generator, and can be started by choosing it from the $Start \rightarrow AllPrograms$ menu of Windows, under $Xilinx\ ISE < version > \rightarrow ISE \rightarrow Accessories \rightarrow CORE\ Generator$. After creating a new project, choosing an appropriate function and a target FPGA device, and setting all necessary parameters, the program generates several output files. The most important out of these files are:

<Component_name>.ngc:
Binary Xilinx implementation netlist file containing the information required to implement a given IP Core in a specified Xilinx FPGA. This is the only file required for the synthesis and implementation of the given core. The name of this file should be placed in the source list file of ATHENa, just before the name of a file instantiating a given IP core.

<Component_name>.vhd:
VHDL wrapper file provided to support functional simulation only. This file contains instantiation of the simulation model for the core. The simulation model is customized through multiple parameters of the core, which are passed as generic values during instantiation. This file is required only in the verification mode of ATHENa, and it should be added to the verification list file, if ATHENa is used for functional verification in batch mode.

<Component_name>.vho:
A template file containing code that can be used as a model for instantiating a given CORE Generator module in a VHDL design. This file is not used by ATHENa.

<Component_name>_readme.txt:
Text file listing all output files, and explaining how they should be used. This file is not used by ATHENa.

Altera: A wizard available with Altera tools is called MegaWizard Plug-in Manager, and can be started by choosing it from the $Start \rightarrow AllPrograms$ menu of Windows, under **Altera→Quartus II** <version>**→Quartus II**<version> **MegaWizard Plug-in Manager**.

After creating a new project (called custom megafunction variation), choosing an appropriate function and a target FPGA family, and setting all necessary parameters, the program generates several output files. The most important out of these files are (assuming that VHDL was selected as an output file type):

<Component_name>.vhd:
VHDL wrapper file containing an instantiation of a given IP core, with values of generics specified using the MegaWizard Plug-in Manager. This is the only file required for the synthesis, implementation, and simulation of the given core. The name of this file should be placed in the source list file of ATHENa and in the verification source list file of ATHENa, just before the name of a file instantiating a given IP core.

<Component_name>_inst.vhd:
An instantiation template file containing code that can be used as a model for instantiating a given IP core

in a VHDL design. This file is not used by ATHENa.

<Component_name>.cmp:
A component declaration file containing code that can be used as a model for declaring a given IP component in a VHDL design. This file is not used by ATHENa.

<Component_name>_waveforms.html:
An HTML files containing sample timing waveforms demonstrating proper operation of a given IP core.

6 Example Run

We have included example projects in our ATHENa package. These projects are located inside of the folder **examples** of the root directory. The default project is currently sha256_rs, which is our implementation of the current hash function standard SHA-256, using architecture with rescheduling proposed by Chaves et al. at CHES 2006. In execute this project, simply run **ATHENa.bat/ ATHENa.sh**. The result files will be generated in a default work directory called **ATHENa_workspace**, located in the root directory. For the remaining examples, please copy the respective **design.config.txt** file, located in the same directory as the source codes of the given example, namely **examples/<project_name>** to the folder **config**. Overwrite the previous contents of the file **config/design.config.txt**. Once completed, you can navigate to the root directory of ATHENa and run **ATHENa.bat/ATHENa.sh** to start running ATHENa for the given project. The most important options of design.config.txt for the sha256_rs example are listed below. Please feel free to modify any of these options in order to investigate different FPGA families and different optimization strategies.

SHA2-256 (architecture with rescheduling)

```
WORK_DIR = <ATHENA_workspace>
SOURCE_DIR = examples\sha256_rs
SOURCE_LIST_FILE = source_list.txt
PROJECT_NAME = sha256
TOP_LEVEL_ENTITY = sha256
TOP_LEVEL_ARCH = rs_arch
CLOCK_NET = clk

LATENCY = TCLK*65
THROUGHPUT = 512/(TCLK*65)

OPTIMIZATION_TARGET = speed
OPTIONS = default

APPLICATION = single_run

FPGA_VENDOR = xilinx
FPGA_FAMILY = spartan3
FPGA_DEVICES = all
SYN_CONSTRAINT_FILE = default
IMP_CONSTRAINT_FILE = default
REQ_SYN_FREQ = 120
REQ_IMP_FREQ = 100
END FAMILY
END VENDOR
```

```
FPGA_VENDOR = altera
FPGA_FAMILY = Cyclone III
FPGA_DEVICES = best_match
SYN_CONSTRAINT_FILE = default
IMP_CONSTRAINT_FILE = default
REQ_IMP_FREQ =  120
MAX_LE_UTILIZATION = 0.8
MAX_MEMORY_UTILIZATION = 0.8
MAX_DSP_UTILIZATION = 1
MAX_MULT_UTILIZATION = 1
MAX_PIN_UTILIZATION = 0.8
END FAMILY
END VENDOR
```

7 Workspace

Workspace is the place where ATHENa creates an instance of your project, including result directory. An instance of your project is created under the path described in Part 2:

$workspace/$application/${date}_${projectname}_${instance_no}

An ATHENa benchmarking project may contain many runs for each device. Hence, to view original information as created by vendors tool before ATHENa processes this information, the user needs to navigate to the following path of the project folder:

${vendor}/${family}/${device}${run_no}

${vendor}*, *${family} and *${device}* are the name of the vendor, family and device used in a given run, respectively. *${run_no}* is the run number, starting from 1. This folder numbering is used to distinguish between subsequent runs using different options.

8 Constraint File

At this point, constraint files are supported for Xilinx only. The constraint file must be located inside of the respective project source directory. The user can specify the name of the constraint file in the design.config.txt under *SYN_ CONSTRAINT_FILE* and *IMP_CONSTRAINT_FILE* for synthesis and implementation, respectively.

9 Purely Combinational Circuit

If a design is purely combinational, user will need to make sure that *CLOCK_ NET* is set to empty i.e. "CLOCK_NET = ". This will ensure that ATHENa will not incorrectly treat any port as a clock signal and does not incorrectly request any clock frequency. Also, since timing is not a requirement, user should set application to *single_run* with no frequency request as well.

10 Project Termination, Report Generation and Cleaning Workspace

In case a user wants to terminate an ATHENa run, the user can stop the run by pressing **CTRL+C** in the console window. If the user wants to stop the ATHENa_spooler, he/she has to press **CTRL+C** multiple times until spooler stops all the remaining design configuration files that are in the queue. If the project is

terminated prematurely, the user will not be able to find report files as the report script has not been run yet. To generate the files, user can run the **report_generator.bat/report_generator.sh** script to create the report files. Note that user will need to know which application and project the user wants to generate the report files for.

Also, be aware that a project folder corresponding to the task that the user started is normally created inside of the specified workspace even if the project is terminated mid-way. These undesired project folders can take up space on your hard drive. To clean up the unwanted directories, the user can either delete them manually or by running the **clean_workspace.bat/clean_workspace.sh** script located inside your root directory. Please be aware that **clean_workspace.bat/ clean_workspace.sh** file will also remove directories corresponding to any uncompleted projects and projects that do not produce any results. Hence, please be careful when you are running the script as you may accidentally delete some directories that you would like to use for debugging.

11 Spooler

A simple script, called **ATHENa_spooler.bat/ATHENa_spooler.sh**, is provided when a user wants to run several projects consecutively. The script automatically replaces the design.config.txt in the config folder by a file from the spool directory, **ATHENa/config/spool**. Files inside of this directory should have the *.txt extension. They can have arbitrary names, but they need to follow internally the same format as design.config.txt.

Once the spooler script starts, it chooses the next configuration file in alphabetical order, and moves this file into the **ATHENa/config/spool/processing** directory. When a project is finished, the respective configuration file is moved from the processing directory into the **ATHENa/config/spool/completed** directory. In case the program is halted mid-way due to an arbitrary reason (blue screen, power outage, insufficient memory, etc.), simply copy the configuration file corresponding to an unfinished run from the processing directory to the main spool directory, and restart the spooler script again.

Note that any files added to the spool directory when the spooler is running will be automatically selected (in the alphabetical order) when the previous project is finished.

It is also possible to run several spoolers simultaneously. This is similar to running several ATHENa projects at the same time. A user should set an appropriate number of logical processors before using this method as ATHENa cannot manage processor distribution across different projects.

12 Database report generator and result replication

ATHENa provides an easy way to generate a zip file containing queried/selected results to be uploaded to the ATHENa database. This zip file contains CSV file(s) of the results. Each zip file contains a unique id which have a corresponding zip file based on vendor associated with it. This file contains necessary files to replicate the same result, assuming the same source codes and tools are used.

Results selection can be done by starting the **db_report_generator.bat/ db_report_generator.sh** script. The script will automatically bring you to the workspace as specified by the design.config.txt so that the user can navigate to the project of interest.

Once user entered the project settings, there are several options that user can choose. In a simple form, user can simply select one of the five criterions to generate the zip file to upload to ATHENa database. This can be done by performing the following steps:

- Modify design.config.txt to point to the correct workspace
- Start db_report_generator.bat/.sh
- Select application
- Select project of choice
- Select desired criterion(s) (enter number(s) between **1** and **5**)
- Generate zip file to upload for database (enter **g**)

– Navigate to the specified folder to obtain the zip file and upload to ATHENa database

However, user can perform a more sophisticated querying method. This can be done by toggle through the *query modes* (enter 'q'). Table 1 describes the meaning of each query mode :

Table 1. Query modes definition

Query Mode	Definition
Best overall	Query the best result based on the specified criterion(s) for each family
Best per generic	Query the best result based on the specified criterion(s) for each device and generic inside a family
Best per device	Query the best result based on the specified criterion(s) for each device inside a family
None	Generate zip file with all the selected results

If query mode selection is not enough, user can **select data to be queried** by entering '**m**'. This will bring user to a new menu with the **Selection Mode** being **Navigate**. User can then traverse the data to select/deselect as required. To change the **Selection Mode** to **Modify**, simply press '**m**' again.

Database report generator also comes with several options to view the results. Table 2 summarizes the viewing options:

Table 2. Database Report Generator Viewing Options

Query Mode	Command	Definition
View data in query	v	Display querying data (deselected data is not shown)
View all available data	va	Displays all available data (deselected data is shown with surrounding parenthesis)
View best results	vb	Displays best results based on the selected query mode with the assumption that all criterions are selected
View best queried results	vq	Displays best results based on the selected query mode with the selected criterion(s)

13 Troubleshooting ATHENa runs

At this point, ATHENa is not yet sophisticated enough to pinpoint errors related to the operation of synthesis and implementation tools. Potential problems can vary from missing a file in a source list, misspelling a source file name, not being able to compile a file, not being able to map a design, etc. Luckily, these problems are not so hard to troubleshoot (except for the first few times) as some general methods of analyzing the synthesis and/or implementation reports still apply. Specifically, a user needs to analyze selected warnings and error messages included in these reports.

The report files can be found in any run folder located inside of your project directory. The generic path is **$workspace/$application/$project/$vendor/ $family_$generic/$device/$runnumber**. For example, if the first step of your project is to implement a design using Xilinx Virtex 5 with xc5vlx50ff1153-3 device, with the single_run application, your folder path will be

$workspace/single_run/$project/xilinx/virtex5_1/xc5vlx50ff1153-3/$runnumber.

The $workspace denotes the path to your workspace as specified in the design configuration file. The $project denotes the name of your project, which is normally preceded by a date and appended with the project number for that day. For instance, if your project is named test, you should see your project labeled as $date_test_$projectnumber. Family's name also contain a generic number appended to the path. If you do not have any generic specified, then this number is one. Otherwise, there will be multiple folders equal to the number of your generic combinations. The generic used for that particular family is specified in the **generic.txt** file within that folder, which is the same level as the device folder. Finally, $runnumber denotes the run number for a given device. Normally if a problem occurs, the corresponding run number is generally the first one, which is run_1.

Once you have located the correct folder, simply look inside the report files. The latest generated synthesis and/or implementation report typically contains an error reported by ATHENa. For Xilinx and Altera, these files are saved as *.log, *.twr and *.rpt, respectively.

Note that if ATHENa spots a possible problem in the design configuration file, the tool may stop the program prematurely. Hence, no $vendor will be found in the project folder. This kind of error can be found by looking at athena_log.txt.

Appendices

A design.config.txt Options

Table 3: design.config.txt Options

Option	Explanation
WORK_DIR	Directory of your workspace. ATHENa_workspace is selected as your default workspace if this option is not specified.
SOURCE_DIR	Directory of your source files for the project.
SOURCE_LIST_FILE	Contains the list of files to be compiled starting from the lowest level to the highest level of file hierarchy. Each file name should be separated by new line.
PROJECT_NAME	Project's name. The name of the project is directly associated with the generated project name in your specified workspace. For Altera, project's name must be the same as your top level entity's name.
FUNCATIONAL _VERIFICATION_MODE	Turning functional verification on or off.
VERIFICATION_DIR	Directory containing HDL source files and test vectors of the test-bench.
VERIFICATION _LIST_FILE	A list of testbench files in the order suitable for compilation from the lowest level (top line) to the highest level (bottom line). Each file name should be separated by new line. The *.vhd, *.v and *.vhdl extensions cannot be used to specify text (non-VHDL) test vector files.
TB_TOP_LEVEL _ENTITY	Top level entity of the testbench.
TB_TOP_LEVEL_ARCH	Top level architecture of the testbench.
MAX_TIME_ FUNC-TIONAL_VERIFICATION	Maximum simulation time for functional verification. Acceptable units are ps, ns, us, and ms. If unspecified, verification will run until end point is reached.
VERIFICATION_ONLY	This option specifies the tool to perform functional verification only, without performing synthesis or implementation.
TOP_LEVEL_ENTITY	Name of top level entity.
TOP_LEVEL_ARCH	Name of the architecture of your top level entity.
CLOCK_NET	Name of the global clock in your design.
LATENCY	The equation for calculating circuit's latency. This should be in terms of TCLK, where TCLK is the minimum clock period of the circuit.
THROUGHPUT	The equation for calculating circuit's throughput. This should be in terms of TCLK, where TCLK is the minimum clock period of the circuit.
OPTIMIZATION_ TAR-GET	Synthesis and implementation strategy. Optimization for area, speed, or balanced.

Continued on next page ...

Table 3 – Continued

Option	Explanation
OPTIONS	Option mode (default or user). In the default mode, default options of tools, as specified in the file options.default_<OPTIMIZATION_TARGET> will be used. If you want to use non-default options of tools, please change this variable to user, and modify the file options.user_<OPTIMIZATION_TARGET>.
APPLICATION	Name of an application. See Part 3 - Application Setup for more details.
FPGA_VENDOR END VENDOR	Target FPGA vendor, i.e. Xilinx or Altera.
FPGA_FAMILY END FAMILY	Target FPGA family of a specified vendor, i.e. Spartan3.
FPGA_DEVICES ... END DEVICES	List of target FPGA devices based on a specified family and vendor. Device names must be separated by comma. Two special modes of operation exist for this option, best_match and all. For best_match, the script will search for the smallest device that passes all criteria as specify by the user. For all, the script will go through all the available devices of the specified family. See Table 2 for details of available parameters. Note : The special modes will search through the FPGA devices specified in the library file located in the device_lib folder.
TRIM_MODE	Data trimming mode. This mode is used to reduce the size of your workspace directory. The following values are supported : off, *zip*, *tiny_zip* and *delete*. *zip* mode compresses all uncritical ATHENa files and all the files created by synthesis and implementation tools with the exception of report files, which will be used by ATHENa for result extraction. *tiny_zip* performs the same operation as *zip* mode with the exception that all the subfolders in the working run directory are deleted. If intermediate result files are not required, delete *mode* can permanently remove these files. This option should be set to *tiny_zip* or *delete* depending on your preference if you are operating on a computer with limited disk space. By default, the TRIM_MODE is set to off.
DB_QUERY_MODE	This mode is used for data base report generation. The following values are supported : off, *overall*, *generic*, *device*. off mode turns off the database query mode. *overall* mode queries the results within the same device family. *generic* mode queries all results with the same family and generic. *device* mode queries all results with the same family, generic and device

Continued on next page ...

Table 3 – Continued

Option	Explanation
DB_CRITERIA	Selection criterion for database query. This option will be ignored if DB_QUERY_MODE is set to off. Area used for query are Slice and ALUT for Xilinx and Altera, respectively. The following values are supported : *throughput*, $throughput_area$, *area*, *latency*, $latency_area$. *throughput* option queries the database for highest throughput. $throughput_area$ option queries the database for highest throughput over area ratio *area* option queries the database for smallest area *latency* option queries the database for shortest latency $Latency_area$ option queries the database for smallest latency times area
GLOBAL_GENERICS _BEGIN ... GLOBAL_GENERICS _END	Key phrases used to specify generics for all of the implemented devices.
GENERICS_BEGIN ... GENERICS_END	Key phrases used to specify local generics specific to a given family of FPGAs. This section must reside within FPGA_FAMILY END FAMILY section describing devices and parameters specific for a given family.

B Xilinx FPGA_DEVICES specific options

Table 4. Family independent options

Options	Explanation
MAX_SLICE _UTILIZATION	Maximum CLB slice utilization ratio.
MAX_BRAM _UTILIZATION	Maximum BRAM utilization ratio. If this variable is set to 0, then no BRAMs will be used in the implementation.
MAX_PIN _UTILIZATION	Maximum pin utilization ratio.
SYN_CONSTRAINT _FILE	Path to a synthesis constraint file (*.xcf). If a constraint file is not used, specify default.
IMP_CONSTRAINT _FILE	Path to an implementation constraint file (*.ucf). If a constraint file is not used, specify default.
REQ_SYN_FREQ	Requested synthesis clock frequency in MHz.
REQ_IMP_FREQ	Requested implementation clock frequency in MHz.

Table 5. Family specific options applicable to Families : Spartan 3, Virtex 2 and Virtex 2 Pro (and equivalent) [encoded as spartan3, virtex2, virtex2p]

Options	Explanation
MAX_MULT _UTILIZATION	Maximum multiplier utilization ratio. If this variable is set to 0, then no multipliers will be used in the implementation.

Table 6. Family specific options applicable to Families : Spartan 6, Virtex 4, Virtex 5, Virtex 6 (and equivalent) [encoded as spartan6, virtex4, virtex5, virtex6]
6

Options	Explanation
MAX_DSP _UTILIZATION	Maximum DSP utilization ratio. If this variable is set to 0, then no DSP units will be used in the implementation.

Note : These options are located inside FPGA_DEVICES ... END_DEVICES clause. Options MAX_<RESOURCE>_UTIL
have effect only in the best_match and all option modes.

C Altera FPGA_DEVICES specific options

Table 7. Family independent options

Options	Explanation
MAX_MEMORY _UTILIZATION	Maximum memory utilization ratio. If this variable is set to 0, then no memory blocks will be used in the implementation.
MAX_PIN _UTILIZATION	Maximum pin utilization ratio.
REQ_IMP _FREQ	Requested implementation clock frequency.

Note : If MAX_MEMORY_UTILIZATION is set to 0, ATHENa will specify the settings for MAX_RAM_BLOCKS_<types> to 0, where types are M4k, M512 and MRAM. This can cause the design not to pass synthesis or implementation stage.

Table 8. Family specific options applicable to the following Families: Cyclone Families and Stratix

Options	Explanation
MAX_LE _UTILIZATION	Maximum Logic Element (LE) utilization ratio.
MAX_MULT _UTILIZATION	Maximum multiplier utilization ratio. If this variable is set to 0, then no multipliers will be used in the implementation.

Table 9. Family specific options applicable to the following Families : Stratix II, Stratix III, Stratix IV, and Arria

Options	Explanation
MAX_LOGIC _UTILIZATION	Maximum Logic Utilization ratio, as calculated by the tools.
MAX_DSP _UTILIZATION	Maximum DSP utilization ratio. If this variable is set to 0, then no DSP units will be used in the implementation.

Note : These options are located inside FPGA_DEVICES ... END_DEVICES clause. Options MAX_<RESOURCE>_UTIL have effect only in the best_match and all option modes.

D "option.<option>_<optimization_target>.txt" options

Table 10.

Options	Explanation
XILINX_SYNTHESIS _TOOL	Xilinx synthesis tool <only Xilinx XST is currently supported>
ALTERA_SYNTHESIS _TOOL	Altera synthesis tool <only quartus_map is currently supported>
ACTEL_SYNTHESIS _TOOL	<currently unsupported>
XILINX_SYNPLIFY_OPT ... END_OPT	<currently unsupported>
XILINX_XST_OPT ... END_OPT	Options for Xilinx XST (synthesis)
ALTERA_QUARTUS _MAP_OPT ... END_OPT	Options for Altera Mapping Tool (synthesis)
ALTERA_QUARTUS _FIT_OPT ... END_OPT	Options for Altera Fitting Tool (implementation)
ACTEL_SYNPLIFY _OPT ... END_OPT	<currently unsupported>
XILINX_NGDBUILD _OPT ... END_OPT	Options for Xilinx NGDBUILD
XILINX_MAP _OPT ... END_OPT	Options for Xilinx MAP
XILINX_PAR _OPT ... END_OPT	Options for Xilinx PAR
XILINX_TRACE _OPT ... END_OPT	Options for Xilinx TRACE

E "placement_search.txt" options

Table 11.

Options	Explanation
XILINX_COST _TABLE_VALUES	Xilinx cost table. Possible range is from 1 to 100
ALTERA_SEED _VALUES	Altera seed. Possible range is from 1 to 232-1

F "exhaustive.<strategy_name>.txt" options

Table 12.

Options	Explanation
General Options	
TARGET_CLK_FREQ	Requested synthesis and implementation frequency. If Altera is used, only implementation frequency is used.
RUN_ALL_OPTIONS	The tool will loop through all specified options if YES is selected. Otherwise, it will stop whenever the target clock frequency is reached.
BEST_LEVEL _1_OPTION_SETS	Number of best combinations of options from Level 1 that will be used for runs at Level 2.
Level 1 Options	
XILINX_SYNTHESIS _TOOL	<currently unsupported>
XILINX_SYNPLIFY_OPT	<currently unsupported>
XILINX_XST_OPT	Xilinx XST options
XILINX_MAP_OPT	Xilinx MAP options
XILINX_PAR_OPT	Xilinx PAR options
Level 2 Options	
XILINX_COST _TABLE_VALUES	Xilinx parameter determining the starting placement point.

G "GMU_Optimization_1" and "GMU_Xilinx_Optimization_1.txt" options

Table 13.

Options	Explanation
General Options	
TARGET_CLK_FREQ	Target implementation frequency.
RUN_ALL_OPTIONS	The tool will loop through all specified options if YES is selected. Otherwise, it will stop whenever the target clock frequency is reached.
BEST_LEVEL_1 _OPTION_SETS	Number of best combinations of options from Level 1 that will be used for runs at Level 2.
Level 1 Options	
ALTERA_SYNTHESIS _TOOL	<currently unsupported>
ALTERA_SYNPLIFY _OPT	<currently unsupported>
ALTERA_QUARTUS _MAP_OPT	Altera mapping tool options
ALTERA_QUARTUS _FIT_OPT	Altera fitting tool options
Level 2 Options	
ALTERA_SEED _VALUES	Altera parameter determining the starting point for placement.

H Guidelines for Choosing REQUESTED_FREQ_IMPROVEMENT_STEPS

There are some general guidelines to be followed when selecting values for the **REQUESTED_FREQ_IMPROVEMEN** list. These guidelines are based on general experimentation and may not be completely effective for any given design, but provide good results for a variety of designs. Some experimentation with these values may be necessary to yield the best results.

The first guideline is that the first value should be a relatively large step, at least 15%. This allows the algorithm to quickly squeeze gains from the system with a few runs. After this successively smaller values allow the algorithm to zero in on an efficient value with fewer runs. Whether this method is effective at minimizing the achieved clock period depends on the relationship between the requested frequency and achieved frequency, which is inherent to a specific design. The more complex this relationship, the more difficult it is to determine the most effective values.

For example if this relationship equates to a function with a single local maximum as in series 1 below, this algorithm works very effectively. If the relationship has multiple local maxima, as in series 2 below, then this method could yield good results but may not be adequate.

I Important notes on new families

Xilinx : Spartan 6, and Virtex 6

Spartan 6 and Virtex 6 utilize a new set of tool options. Hence, options used by old devices may be obsolete for these new devices, and new options may not be accepted for the old ones. This feature can cause your ATHENa run to end with failure. At this point, ATHENa does not automatically turn off all the new/old options based on the family used, so it may be prudent for a user to run implementations on old and new families as separate projects. RAMB16E1 and RAMB8BWER are considered as a single Block RAM.

Altera: Arria Families, Stratix IV, and Cyclone IV

Timequest timing analyzer is used for these new families. Because of the complex nature of this tool, the timing report generated by ATHENa may be inaccurate. At this point ATHENa selects only the first timing result it finds in the timing report. This timing model is the slowest (85°C) model available. Hence, the result may not reflect the result that can be obtained in the actual operating conditions of the device.

J History of Changes since ATHENa 0.6

2010/12/14: ATHENa 0.6.1

- ATHENa_setup.bat revised in the following way: User is asked during ATHENa setup whether he/she uses a free version of Xilinx tools (Webpack) or a commercial/educational version (Design Suite). If the trial version of commercial tools is used, the proper answer should be Design Suite, during the trial period, and Webpack after the end of the trial period (when the extended license expired).
- Added support for using ATHENa together with Altera MegaWizard Plug-in Manager and Xilinx CORE Generator. For details, please see Section 5.2 of this tutorial.
- Added two new examples: coregenerator_example and megawizard_example, demonstrating the new capabilities of ATHENa described above. See the example folder of ATHENa.
- ATHENa device libraries for Xilinx and Altera have been revised in order to remove bugs and inaccuracies reported by users of version 0.6.
- The best_match feature of ATHENa has been fixed in order to remove incorrect behavior reported for Virtex 5, Virtex 6, and Spartan 6.
- Support for Verilog source files has been added and documented in the tutorial.
- The sha256_rs example has been updated by adding missing data_in/data_out files in the folder examples/sha256_rs/tb.
- Corrected the problem in the db_report_generator.bat associated with missing fields describing the implementation clock period and the implementation clock frequency in the output .csv file.

2011/06/16 : ATHENa 0.6.2

- Fixed a bug in GMU_Optimization_1 for Altera
- Fixed a bug for Altera's memory settings. This means that the following settings are applied when MEM_UTILIZATION_RATIO is set to 0:
  - set_global_assignment -name MAX_RAM_BLOCKS_M4K 0
  - set_global_assignment -name MAX_RAM_BLOCKS_M512 0
  - set_global_assignment -name MAX_RAM_BLOCKS_MRAM 0
  - set_global_assignment -name AUTO_ROM_RECOGNITION OFF
  - set_global_assignment -name AUTO_RAM_RECOGNITION OFF
  - set_global_assignment -name AUTO_SHIFT_REGISTER_RECOGNITION OFF
- Fixed a bug for Altera's DSP settings. This means that the following settings are applied when DSP_UTILIZATION_RATIO or MULT_UTILIZATION_RATIO is set to 0:
  - set_global_assignment -name AUTO_DSP_RECOGNITION OFF
- Added FF extraction for Xiilnx and Altera
- Overhauled db_report_generator.bat/.sh functionality to make it fully compatible with the website
- Reduce severity level when missing clock net to allow purely combinational circuit mode
- Fixed a bug in TRIM_MODE where subfolders were not included in the compression.
- Added additional option, tiny_zip, in TRIM_MODE to allow a compression mode that ignores the subfolders of working directory.
- Added library for Xilinx ISE 12.4
- Addressing issue if a circuit is purely combinational

2012/05/08 : ATHENa 0.6.3

- Recommended Perl distribution for Windows OS is changed to Strawberry Perl.
- Added Library for Xilinx 13.1, 13.2, 13.3 and Altera Quartus II 11.0, 11.1
- Fixed a bug in desing_config parser.
  - Bug: Clock net name mismatch due to case sensitivity issues.
- Fixed a bug in BRAM resource extraction for Xilinx
  - Bug: Incorrect BRAM resource extraction in Spartan 3A devices.
- Added automated database report extraction feature from the design_config.

2012/12/06 : ATHENa 0.6.4

- Added Library for Xilinx 14.1, 14.2, 14.3 and Altera Quartus II 12.0, 12.1.
- Fixed a bug in sha256_rs example source code.
  - Bug: Incorrect passing of generic value at line 204 of rs_datapath.vhd.