ASTRA
A MelPUF Synthesis Tool

User Manual

User guide for successful MelPUF synthesis in a gate level netlist of a hardware intellectual property using the ASTRA tool [1].

Version: 0.1 (1June2k22)
Release Date: August 3, 2022

Contributors

Atri Chatterjee
Department of Electrical & Computer Engineering
University of Florida
Gainesville, Florida, USA.
Email: a.chatterjee@ufl.edu

Swarup Bhunia
Department of Electrical & Computer Engineering
University of Florida
Gainesville, Florida, USA.
Email: swarup@ece.ufl.edu

[1]This document is copyright protected and property of the entity mentioned above. Any disclosure, alteration, re-distribution, or commercial use is NOT allowed without the consent of the contributors mentioned above. For copyright issues, contact information can be found here.

Contents

List of Figures

List of Tables

1 Version Information & Revision History

This user guide corresponds to the latest ASTRA version: 0.1. The different version history of ASTRA is presented in Table 1.

Table 1: Revision history of **MeLPUF**

Version	**Release Date**	**Major Features/Changes**
0.1	1 June, 2022	- Initial release - Support for basic configurable parameters - Insert basic **MeLPUF** template into an gate level netlist - Support **MeLPUF** insertion in either normal mode or timing aware mode

2 Overview

ASTRA is a fully automated commercial quality tool which inserts basic **MeLPUF** structures in a gate level netlist. The tool runs in either normal mode or timing aware mode in which it identifies nodes where a PUF may be implemented. In normal mode it randomly selects gates of the design not caring for the effect to timing after a **MeLPUF** is inserted at the end of the said selected gate. Timing aware execution takes into account the effect of timing where timing closure is critical. The output of the tool is another gate level netlist which has the requested number of **MeLPUF** integrated in the logic of the hardware IP to facilitate an equivalent size of PUF signature. Fig: 1 shows the flow of the ASTRA tool, while Fig: 2 illustrates a sample full adder circuit before and after **MeLPUF** insertion.

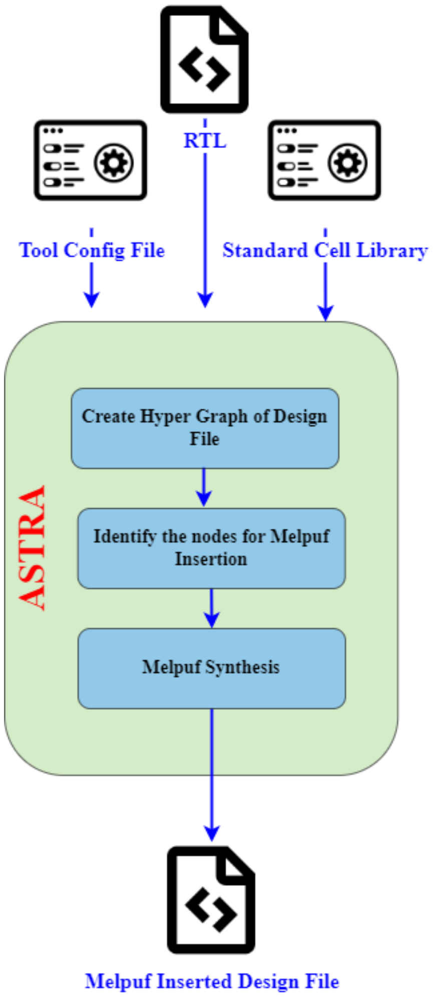
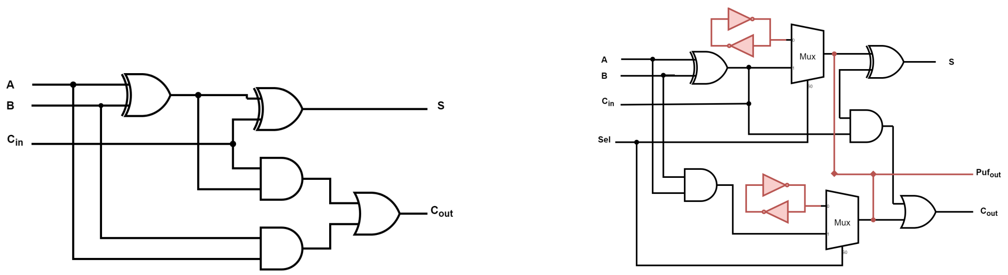

RTL
Tool Config File
Standard Cell Library
ASTRA
Create Hyper Graph of Design File
Identify the nodes for Melpuf Insertion
Melpuf Synthesis
Melpuf Inserted Design File

Figure 1: Fully automated basic EDA flow inside ASTRA.



(a) A full adder circuit.

(b) A full adder circuit with 2 bit **MeLPUF** inserted.

Figure 2: This illustrates a **MeLPUF** inserted system. The first sub-graph shows a normal full adder while the second sub-graph shows a 2 bit **MeLPUF** inserted system. The normal mode can be obtained by forcing '1' on the sel line, conversely the PUF signature can be retrieved by forcing sel to '1'.

2.1 Interface & Dependencies

ASTRA takes a Verilog/VHDL design file that contains a gate-level netlist and other optional configuration files to generate a **MeLPUF** inserted design which contains a PUF signature of equivalent size. The current version of the tool cannot handle RTL files directly so the input design file needs to mapped to a specific technology library file : gscl45nm. Future version of this tool will use several industrial EDA tools to perform various actions.

Table 2: Tool interface: Input/Output

Inputs	**Outputs**
- Flattened Gate-Level Netlist (Original RTl will be supported in a future version) - Standard Cell Library - Configuration File	- Flattened Gate-Level Netlist (**MeLPUF** inserted design)

Table 3: Tool Flow: Dependency

FPGA/ASIC	**Other EDA Tools**	**EDA Toolset**
ASIC	- Simulation Tool - Synthesis Tool	Synopsys Toolset Design Compiler vcs

3 Installation & Setup

In order to run ASTRA, the following dependencies must be met or arranged by the user:

3.1 Installation

Operating System: UNIX
Executable: astra
Library: gscl45nm.lib, gscl45nm.db, gscl45nm.v
EDA tool set: Synopsys Design Compiler (DC), Synopsys Verilog Compiler & Simulator (VCS)

3.2 Environment Setup

EDA tool setup: Future versions of ASTRA will have dependency over multiple commercial EDA tools. The executable for these tools needs to be either placed in the working directory or the paths to be included to the current working directory.
Compiling ASTRA binary: ASTRA release comes with a pre-compiled binary. Otherwise, run ”*make*” command in the ”*src*” directory of the tool package to compile the binary executable.
ASTRA setup: ASTRA uses its own library *gscl45nm.lib* that comes with the tool package. The executable *and* library both needs to be placed in the src and lib directories respectively.

4 How to Run

Assuming the required environment is being set up, ASTRA usage is as following:

Usage: **./astra** [**optional argument**]
Options:
-source <Location/of/Configuration/File.txt>
-help

4.1 Input Arguments

-source <Location/of/Configuration/File.v>
Optional All of the tool commands that a user requires can be provided to ASTRA as a configuration file directly from the command line using this flag. Alternatively, the same command -"source" may be used after running the the tool binary directly. Otherwise, the necessary commands may also be provided by the user manually after the tool binary is executed.

-help **Optional** Outputs a message on how to run the "astra" binary.

5 Configurable Parameters

ASTRA uses a large pool of user configurable parameters that allow user to have control over the **MeLPUF** synthesis steps. Of-course, most of these parameters are optional inputs to the tool as ASTRA will use default values for each of these parameters, if not specified by the user. A brief description along with relevant examples and default values [2] are given in the following subsections for each of the user configurable parameters. A sample configuration file comes with the tool package (check "root" directory of the tool).

5.1 source

source defines the configuration file from which the ASTRA tool can read all of the tool commands.

5.2 set_hdl

set_hdl defines the flattened gate level netlist file. The file has to be a verilog file.

5.3 set_library

set_library defines the the gscl45nm.lib file. This file should be utilized to generate the flattened netlist. Future version of the tool will utilize this file to auto-generate a flattened netlist from a given RTL file.

5.4 set_tool_library

set_tool_library defines the the gscl45nm.db file. This file will be utilized in a future version of the tool where the .db format is required.

5.5 set_library_primitive

set_library_primitive defines the the gscl45nm.v file. This file is used for test-bench verification. This file contains all of the library cells as verilog modules.

5.6 set_timing_file

set_timing_file defines the timing file that records the timing details of all of the nodes of the input design. This file **should be** generated using Synopsis Design Compiler tool. This file is only required when running the tool in timing aware mode.

5.7 set_top

set_top defines the top module of the provided design file.

[2] Values with "*" defines the default value. If not specified, read the corresponding example to find the default value.

5.8 set_clk

set_clock defines the clock port of the provided design file.

5.9 set_rst

set_rst defines the reset port of the provided design file.

5.10 set_puf_size

set_puf_size defines the size of the **MeLPUF** signature.

5.11 set_regression

set_regression if many different sized **MeLPUF** inserted output files are required of the same input design file, this command may be used instead of re-running the tool multiple times.

5.12 set_dont_touch

set_dont_touch defines EDA tool specific pragmas that preserve the extra logic elements introduced due to **MeLPUF** insertion.

Value	Interpretation	Impact
cadence*	Use Cadence tool specific pragmas.	Re-synthesis of the output file using Cadence tools like genus will preserve the extra logic elements introduced due to **MeLPUF** insertion.
genus	Use Synopsis tool specific pragmas.	Re-synthesis of the output file using Synopsis tools like Design Compiler will preserve the extra logic elements introduced due to **MeLPUF** insertion.
quartus	Use Intel tool specific pragmas.	Re-synthesis of the output file using Intel tools like Quartus will preserve the extra logic elements introduced due to **MeLPUF** insertion.
vivado	Use AMD Xilinx tool specific pragmas.	Re-synthesis of the output file using AMD Xilinx tools like Vivado will preserve the extra logic elements introduced due to **MeLPUF** insertion.

5.13 set_insertion_type

set_insertion_type this is for running the tool in either normal mode or timing aware mode where **MeLPUF** insertion should not affect the timing closure.

Value	Interpretation	Impact
normal*	Run ASTRA in normal mode.	**MeLPUF** insertion does not care about the timing closure of the output file.
slack_aware constrained	Run ASTRA in timing aware mode.	**MeLPUF** insertion does not violate the timing closure of the output file.

5.14 set_reuse_logic

set_reuse_logic sets if the internal nodes of the netlist are to be utilized for **MeLPUF** insertion. This saves some overhead cost for **MeLPUF** synthesis.

5.15 set_puf_template

set_puf_template is used for choosing which template to use for **MeLPUF** when reusing existing logic.

Value	Token	Impact
set_puf_template	inverter*	Will only target inverter cells in the netlist during **MeLPUF** insertion when logic reuse is enabled.
set_puf_template	nand	Will only target nand cells in the netlist during **MeLPUF** insertion when logic reuse is enabled.
set_puf_template	nor	Will only target nor cells in the netlist during **MeLPUF** insertion when logic reuse is enabled.
set_puf_template	mixed_mode	Will target inverter, nand and nor cells in the netlist during **MeLPUF** insertion when logic reuse is enabled.

5.16 set_random_seed

set_random_seed sets the seed value for random number generation.

5.17 set_debug

set_debug prints extra information when tool is in operation.

5.18 set_verification

set_verification generates a test-bench setup and verifies the original design to the **MeLPUF** inserted design for functional correctness.

5.19 set_trace

set_trace generates vcd dump files for viewing the traces driven the testbench setup.

5.20 set_number_of_tests

set_number_of_tests sets number of times random input vectors are generated to test the **MeLPUF** inserted design against the original design for functional correctness. **Default:** 1000.

5.21 set_control_ports

set_control_ports identifies the control ports of a sequential design. This helps in properly triggering the design under test.

5.22 set_status_ports

set_status_ports identifies the status ports of a sequential design. This helps in properly polling for completion of an operation of the design under under test.

5.23 start

start starts the **MeLPUF** synthesis process.

5.24 report_params

report_params reports all of the parameter values to the terminal.

5.25 clear

clear clears the terminal.

5.26 remove_design

remove_design remove the current design and also resets all other parameters.

5.27 set_default_output_directory

set_default_output_directory defines the directory path where the output files are to generated. If the directory does not exist, a directory with that name will be created. **Default:** reports.

5.28 set_suppress_messages

set_suppress_messages suppresses the targeted messages when the tool is in operation.

5.29 reset_suppress_messages

reset_suppress_messages un-suppresses the targeted messages when the tool is in operation.

5.30 help

help reports how to use the tool in the terminal.

6 Outputs

6.1 Directory Structure

ASTRA generates one directories per design. Assuming that the input design file name is *Design.v* (excluding the location part). ASTRA will generate:

- Design/design_puf_<number of puf bits>.v

The current version of the tool generates a **MeLPUF** inserted design and a test-bench setup for post-synthesis correctness. Future versions of the tool will support various other report such as area, power, timing overheads and test-bench simulation etc.

7 Types of Messages

ASTRA generates three types of messages for user interpretation of each message:

- **INFO[message-number]**
- **WARNING[message-number]**
- **ERROR[message-number]**

7.1 INFO

Type of message that gives information to the user to keep track of the process. Couple of example INFOs are given below:
>>INFO[25]:Config File: config.txt
>>INFO[7]: Reading puf template file -> puf_template/puf_template.v

7.2 WARNING

Type of message that alerts the user for potential anomalies. WARNINGs do not stop the **MeLPUF** synthesis process. However, may lead to unexpected outputs from the tool. In most cases, WARNINGs are taken care of by **ASTRA**. If not, reasons of WARNINGs may need to be taken care of by the user to get expected results from **MeLPUF** synthesis. Couple of example WARNINGs are given below:
>>WARNING[2]: Some node timing has been violated in the fan out cone of the node:U6561. Puf will not be inserted in this timing path..

7.3 ERROR

Type of message that tells the user about something severe happened. ERROR stops the **MeLPUF** synthesis process. User needs to avoid the ERRORs in order to run the tool further. Couple of example ERRORs are given below:
- When the requested PUF signature size is greater than the number of nodes in the design:
>>ERROR[6]: The number of PUF cannot exceed the number of nodes in the given design. Cannot insert PUFs in the design.
- When the PUF signature size if not specified:
>>ERROR: Flag error: -number_of_puf not specified.

8 Acknowledgement

This tool development is supported by grant from the AISS Program.

9 Contributors

Swarup Bhunia
Atri Chatterjee

10 Support

For any question, comments, concerns, or bug reporting, contact:
Atri Chatterjee
Email: ***a.chatterjee@ufl.edu***