EDA Analytics Central

Abdelrahman Hosny

Jun 08, 2020

CONTENTS:

EDAAC is a Python package to collect, store and analyze data coming out of EDA (*Electronic Design Automation*) Flows.

```
$ pip install -U edaac
```

Tutorial A quick tutorial extracting *Design Rule Violations (DRVs)* from Cadence Innovus log file to get you up and running with EDAAC.

User Guide The Full guide to EDAAC — from collecting metrics to storing them, from querying data to advanced analytics and *everything* in-between.

License EDAAC is open-source under BSD-3 license.

Feedback EDAAC is community-driven. Please, share with us your feedback and feature requests.

CHAPTER

ONE

COMMUNITY

To get help with using EDAAC, use the GitHub Issues and label the issue with *question*.

CHAPTER

TWO

CONTRIBUTING

Yes please!

EDAAC is a young project and we are looking for contributions, additions and improvements.

The source is available on GitHub and contributions are always encouraged. Contributions can be as simple as minor tweaks to this documentation, API names, or event the core architecture.

To contribute, fork the project on GitHub and send a pull request.

2.1 Tutorial

This tutorial introduces **EDAAC** by means of example — we will walk through how to extract metrics from a log file that comes out of an EDA tool.

Metrics are essential information that we extract about a circuit (hardware design) at a specific stage of the tape-out process. For example, after *Logic Synthesis*, we might be interested to know the total number of standard cells used after mapping the design to a standard cell library. After *Routing*, we would be concerned with the total number of *Design Rule Violations (DRVs)*. Instead of looking at the log files of EDA tools and searching through them for the important piece of information, **EDAAC** makes this straightforward for you.

To make use of the extracted metrics, **EDAAC** offers data models to store the collected metrics into a document-based database (MongoDB). This database can be used for further research and development of EDA tools.

2.1.1 Getting Started

If you haven't installed EDAAC, simply use pip to install it like so:

```
$ pip install edaac
```

To verify the installation:

```
>>> import edaac
>>> edaac.version()
EDA Analytics Central (EDAAC) v0.0.11
```

2.1.2 Extracting Metrics

In this example, we show how to extract *Design Rule Violations (DRVs)* from a log file saved by Cadence Innovus.

Note: Since Cadence tools are proprietary software, we are unable to publish raw log files outputted by the tool. We will update this tutorial with example log files once we support open-source EDA tools.

Assuming you have generated a DRC report using a proper command within Innovus to verify that the design meets the technology-defined constraints. The log file is located at `./test_design.drc.rpt`.

Now, you can use the below Python code to extract DRVs into a metrics dictionary:

```
from edaac.metrics.parsers import parse_innovus_drc_report

log_file = './test_design.drc.rpt'
metrics = parse_innovus_drc_report(log_file)
print(metrics)
```

An example output would be:

```
{
    'drv_total': 76,
    'drv_short_metal_total': 30,
    'drv_short_metal_area': 0.06930000,
    'drv_short_cut_total': 0,
    'drv_short_cut_area': 0.0,
    'drv_out_of_die_total': 0,
    'drv_out_of_die_area': 0.0,
    'drv_spacing_total': 32,
    'drv_spacing_parallel_run_length_total': 19,
    'drv_spacing_eol_total': 13,
    'drv_spacing_cut_total': 0,
    'drv_min_area_total': 14
}
```

What just happened? Underneath, the function mines the log files for a number of metrics that it registered. In its core, it heavily uses regular expressions to look for patterns.

Why I can't find the metric I'm looking for? Most probably, the metric is not yet registered in the parsing function. Help us improve the package by submitting an issue with label `enhancement`

2.1.3 What's Next?

EDAAC comes pre-loaded with a number of parsers (..and more under development). But that's not all. Storing metrics effeciently for post-processing is as important as *-if not more important than-* collecting the metrics themeselves.

In the *User Guide*, we show more examples of using **EDAAC** for metrics processing and storage.

2.2 User Guide

2.2.1 Installing EDAAC

To use EDAAC, you will need to download MongoDB and ensure it is running in an accessible location. You will also need MongoEngine to use EDAAC, but if you install EDAAC using setuptools, then the dependencies will be handled for you.

EDAAC is available on PyPI, so you can use **pip**:

```
$ pip install edaac
```

Alternatively, if you don't have setuptools installed, download it from PyPi and run

```
$ python setup.py install
```

To use the bleeding-edge version of EDAAC, you can get the source from GitHub and install it as above:

```
$ git clone git://github.com/EDAAC/EDAAC
$ cd EDAAC
$ python setup.py install
```

2.2.2 Collecting Metrics

Metrics are characteristics of design artifacts, processes, and inter-process communications during the an SoC design flow. The main idea behind pervasively collecting metrics is to measure the design process and quantify its Quality of Results (QoR). This has always been a prerequisite to optimizing it and continuously achieving maximum productivity.

EDAAC implements *Metrics* collection functionality in `edaac.metrics` sub-package. Below, we document its functionality.

Synthsis Stats

We can extract useful statistics about a synthesized netlist that aid in the physical design process.

Supported Tools

- Yosys

Usage

1. Generate a report from Yosys using the *stat* command.
2. Use `edaac.metrics.parsers` to parse the report.

```
from edaac.metrics.parsers import parse_yosys_log
metrics = parse_yosys_log('/path/to/report')
```

3. `metrics` is a Python dictionary of `key: value` pairs.

```
print(metrics)
```

Dictionary

Key	Meaning
run__synth__yosys_version	Version of yosys build used
synth__inst__num__total	Total numner of standard cells
synth__inst__stdcell__area__total	Total area of standard cells
synth__wire__num__total	Total number of wires
synth__wirebits__num__total	Total number of wirebits
synth__memory__num__total	Total number of memories
synth__memorybits__num__total	Total number of memory bits
run__synth__warning__total	Total number of warnings
run__synth__warning__unique__total	Total number of unique warnings
run__synth__cpu__total	CPU usage
run__synth__mem__total	Memory usage

Example

```
metrics = {
    'run__synth__yosys_version': '0.9+1706 (git sha1 UNKNOWN, gcc 7.3.1 -fPIC -Os)',
    'synth__inst__num__total': 272,
    'synth__inst__stdcell__area__total': 407.512000,
    'synth__wire__num__total': 297,
    'synth__wirebits__num__total': 343,
    'synth__memory__num__total': 0,
    'synth__memorybits__num__total': 0,
    'run__synth__warning__total': 90,
    'run__synth__warning__unique__total': 26,
    'run__synth__cpu__total': 1.21,
    'run__synth__mem__total': 28.78
}
```

Design Rule Check

Design rules are geometric constraints imposed on an SoC to ensure that the design functions properly, reliably and can be manufactured by fabs.

A Design Rule Violation (DRV) is a record that represents a violation to the design rules defined by the technology library used.

Supported Tools

- Cadence Innovus

Usage

1. Generate a report from Innovus using the instructions here.
2. Use `edaac.metrics.parsers` to parse the report.

```
from edaac.metrics.parsers import parse_innovus_drc_report
metrics = parse_innovus_drc_report('/path/to/report')
```

3. `metrics` is a Python dictionary of `key: value` pairs.

```
print(metrics)
```

Dictionary

Key	Meaning
drv_total	The total number of DRVs
drv_short_metal_total	Total numner of short metal violations
drv_short_metal_area	Total area of short metal violations
drv_short_cut_total	Total number of cut spacing violations
drv_short_cut_area	Total area of cut spacing violations
drv_out_of_die_total	Total number of components placed/routed out of die
drv_out_of_die_area	Total area of components placed/routed out of die
drv_spacing_total	Total number of spacing violations
drv_spacing_parallel_run_length_total	Total number of parallel run length violations
drv_spacing_eol_total	Total number of end-of-line spacing violations
drv_spacing_cut_total	Total number of cut spacing violations
drv_min_area_total	Total number of min-area violations

Example

```
metrics = {
    'drv_total': 101,
    'drv_short_metal_total': 2,
    'drv_short_metal_area': 0.02382500,
    'drv_short_cut_total': 1,
    'drv_short_cut_area': 0.0012500,
    'drv_out_of_die_total': 0,
    'drv_out_of_die_area': 0.0,
    'drv_spacing_total': 41,
    'drv_spacing_parallel_run_length_total': 7,
    'drv_spacing_eol_total': 9,
    'drv_spacing_cut_total': 25,
    'drv_min_area_total': 57
}
```

Connectivity

This ensures that the circuit components are connected as in the schematic.

Supported Tools

- Cadence Innovus

Usage

1. Generate a report from Innovus using the instructions here.
2. Use `edaac.metrics.parsers` to parse the report.

```
from edaac.metrics.parsers import parse_innovus_conn_report
metrics = parse_innovus_conn_report('/path/to/report')
```

3. `metrics` is a Python dictionary of `key: value` pairs.

```
print(metrics)
```

Dictionary

Key	Meaning
conn_open_nets	Total number of open nets

Example

```
metrics = {
    'conn_open_nets': 22
}
```

Static Timing Analysis (STA)

Static Timing Analysis validates the timing performance of a design by checking all possible paths for timing violations under worst-case conditions.

The *arrival time* of a signal is the time elapsed for a signal to arrive at a certain point.

The *required time* is the latest time at which a signal can arrive without making the clock cycle longer than desired.

The *slack* associated with each connection is the difference between the required time and the arrival time. A positive slack *s* at some node implies that the arrival time at that node may be increased by s, without affecting the overall delay of the circuit. Conversely, negative slack implies that a path is too slow, and the path must be sped up (or the reference signal delayed) if the whole circuit is to work at the desired speed.

The critical path is defined as the path between an input and an output with the maximum delay. The critical path is sometimes referred to as the worst path. If this path has a negative slack, the circuit won't work as expected at the desired speed.

Supported Tools

- Cadence Innovus
- OpenSTA

Usage

1. Generate a report from Innovus using the appropriate command. Or generate a report from OpenSTA using `report_tns`, `report_wns` and `report_design_area`.
2. Use `edaac.metrics.parsers` to parse the report.

```
from edaac.metrics.parsers import parse_innovus_timing_report
metrics = parse_innovus_timing_report('/path/to/report')
```

```
from edaac.metrics.parsers import parse_openroad_log
metrics = parse_openroad_log('/path/to/report', 'OpenSTA')
```

3. `metrics` is a Python dictionary of `key: value` pairs.

```
print(metrics)
```

Dictionary from Innovus

Key	Meaning
`timing_wns`	Worst negative slack
`timing_tns`	Total negative slack
`timing_violating_paths`	Number of violating paths

Example

```
metrics = {
    'timing_tns': -27.496,
    'timing_wns': -0.851,
    'timing_violating_paths': 35
}
```

Dictionary from OpenSTA

Key	Meaning
`slack__negative__total`	Total negative slack
`slack__negative__worst`	Worst negative slack
`std__area__total`	Total standard cell area
`util`	Core utilization

Example

```
metrics = {
    'slack__negative__total': 0.00,
    'slack__negative__worst': 0.00,
    'std__area__total': 491.0,
    'util': 8.0
}
```

Power

This reports the power consumption of the circuit.

Supported Tools

- Cadence Innovus

Usage

1. Generate a report from Innovus using the appropriate command.
2. Use `edaac.metrics.parsers` to parse the report.

```
from edaac.metrics.parsers import parse_innovus_power_report
metrics = parse_innovus_power_report('/path/to/report')
```

3. `metrics` is a Python dictionary of `key: value` pairs.

```
print(metrics)
```

Dictionary

Key	Meaning
`power_internal_total`	Total internal power
`power_switching_total`	Total switching power
`power_leakage_total`	Total leakage power
`power_total`	Total power (sumof the above)
`power_internal_percentage`	Internal power / Total * 100.0
`power_switching_percentage`	Swithing power / Total * 100.0
`power_leakage_percentage`	Leakage power / Total * 100.0

Example

```
metrics = {
    'power_internal_total': 26.31116662,
    'power_switching_total': 21.61735782,
    'power_leakage_total': 13.58182182,
    'power_total': 61.51034631,
    'power_internal_percentage': 42.7752,
    'power_switching_percentage': 35.1443,
    'power_leakage_percentage': 22.0805
}
```

Area

This reports the area of the standard cells in addition to the cell count.

Supported Tools

- Cadence Innovus

Usage

1. Generate the area report from Innovus using the appropriate command.
2. Use `edaac.metrics.parsers` to parse the report.

```
from edaac.metrics.parsers import parse_innovus_area
metrics = parse_innovus_area_report('/path/to/report')
```

3. `metrics` is a Python dictionary of `key: value` pairs.

```
print(metrics)
```

Dictionary

Key	Meaning
`area_stdcell`	Total area of standard cells (um^2)
`area_stdcell_count`	Total number of standard cells

Example

```
metrics = {
    'area_stdcell': 48191.040,
    'area_stdcell_count': 11306
}
```

Compute Resources

This reports the compute resources (cpu, memory) used by a flow process.

Supported Tools

- Cadence Innovus

Usage

1. Dump Innovus logs (that are shown on stdout) to a file.
2. Use `edaac.metrics.parsers` to parse the report.

```
from edaac.metrics.parsers import parse_innovus_log
metrics = parse_innovus_log('/path/to/report')
```

3. `metrics` is a Python dictionary of `key: value` pairs.

```
print(metrics)
```

Dictionary

Key	Meaning
compute_cpu_time_total	Total time from all CPU cores (seconds)
compute_real_time_total	Total wall clock time (seconds)
compute_mem_total	Total memory usage (MB)

Example

```
metrics = {
    'compute_cpu_time_total': 540,
    'compute_real_time_total':184,
    'compute_mem_total': 2287.4
}
```

2.2.3 Data Model

Managing the storage of the collected *Metrics* is a challenging task. Collecting metrics from hundreds, or even thousands, of EDA flows introduces the pondering question: **How should we structure the data to make efficient use of it in predictive analytics applications?**

EDAAC implements a general-purpose data model in `edaac.models` sub-package. Below, we document its functionality.

Documents

EDAAC's data model is an unstructured document that represents an SoC project during its different life stages (from logic synthesis to routing). The root of the data model is a `Project` document. A `Project` is a container for all related artifacts of the design lifecycle.

Below is a complete birds-eye view of the `Project` document.

Usage: `from edaac.models import Project`

Every embedded document in the project has a class representation in `edaac.models`. For example, the `technology` key in the project should be an instance of `edaac.models.Technology`. Similarly, `design`, `flow`, `stage` and `tool` should be instances of `edaac.models.Design`, `edaac.models.Flow`, `edaac.models.Stage` and `edaac.models.Tool` respectively.

```
{
    "name": "<Project Name>",
    "description": "<Project Description>",
    "technology": {
        "foundry": "<Example: TSMC>",
        "process": 65,
        "beol": "<Back End Of Line>",
        "tracl": "<The height of the track>",
        "opv": "<Operating voltage>",
        "vt": "<Voltage threshold>",
        "channel_width": "<Channel width>",
        "config": "<Configuration>",
        "version": "<Version>",
        "rag": "<Red | Amber | Green>"
    },
    "design": {
        "name": "<Design Name>",
        "rtl_files": [
            "<Absolute Path of verilog RTL file>",
            "<Absolute Path of verilog RTL file>"
        ],
        "netlist_file": "<Absolute path of netlist file (after synthesis)>",
        "sdc_file": "<Absolute path of constraints file (ins .sdc format)>",
        "runset_tag": "<RTL related tags for the release candidates>",
        "runset_id": "<ID of the release candidates>",
        "rtl_config": "<Configuration of RTL>",
        "rtl_tag": "<Tags for the RTL>",
        "rtl_rag": "<Red | Amber | Green>"
    },
    "flows": [
        {
            "flow_directory": "<Directory of the Flow>",
            "params": {
                "<Flow parameter>": "<value>",
                "<Flow parameter>": "<value>"
            },
            "stages": [
                {
                    "name": "<Stage name>",
                    "tool": {
                        "name": "<Tool name>",
                        "version": "<Tool version>"
                    },
```

(continues on next page)

(continued from previous page)

```
                    "machine": "<Host name running this stage>",
                    "collection_mode": "<OFFLINE_FROM_LOGS | DURING_RUN_TIME>",
                    "status": "<NOT_STARTED | RUNNING | COMPLETED_SUCCESSFULLY | COMPLETED_WITH_ERRORS>",
                    "log_files": [
                        "<Absolute path of log file>",
                        "<Absolute path of log file>"
                    ],
                    "metrics": {
                        "<Metric key>": "<value>",
                        "<Metric key>": "<value>"
                    }
                }
            ],
            "log_files": [
                "<Absolute path of log file>",
                "<Absolute path of log file>"
            ]
        }
    ]
}
```

Note: **rag:** red: no verification ran on the design. amber: alpha or beta release with some levels of verifications. green: release candidate

Database

The question now is where do we store this these information? Answer: MongoDB.

Starting a MongoDB Server

Option 1: Docker

This is the easiest option to get started. Use the following command to start a local database server

```
docker run -d -p 27017:27017 -v /path/to/local/folder:/data/db --name edaac_db mongo
```

This will start a local MongoDB server on port `27017` (the default port for MongoDB). It will also mount a folder at `/path/to/local/folder` to the container to persist data when the container is stopped.

Option 2: Install Locally

Follow the instructions on the official documentation.

Option 3: Cloud Instance

Create a MongoDB instance on your cloud provider account using MongoDB Atlas.

Connecting to MongoDB

After starting the server, download MongoDB Compass to graphically connect to the database and ensure that it is running correctly.

Next, create a database with a give it a name (e.g. *test_db*) using MongoDB Compass.

From Python, connect to the database using:

```
import mongoengine as mongo

mongo.connect('test_db')
```

The above code will connect automatically to a MongoDB server running on the localhost with the default port, username and password.

If you are running a remote MongoDB instance, provide the credentials as below:

```
import mongoengine as mongo

mongo.connect('test_db', host='', port='', username='', password='')
```

Note: `mongoengine` package is installed as part of `edaac` dependencies.

Examples

Creating a Project

The only required key of a project document is its `name`. All other keys can be updated later by retrieving the project, modifying it and then saving it back.

```
import mongoengine as mongo
from edaac.models import Project, Technology, Design

mongo.connect('test_db')

# create project
project = Project(
    name='test-project',
    description='demonstrates the use of edaac models',
    technology=Technology(
        foundry='TestFoundry',
        process=45
    ),
    design=Design(
        name='test-design',
        rtl_files=['/path/to/rtl1.v', '/path/to/rtl2.v'],
        netlist_file='/path/to/netlist.v',
        sdc_file='/path/to/const.sdc'
    )
)
project.save()
mongo.disconnect()
```

Update Project Data

The below code retrieves an existing project and updates its data.

```
import mongoengine as mongo
from edaac.models import Project, Flow, Stage, Design, Tool
from edaac.enum import StageStatus, DataCollectionMode

mongo.connect('test_db')

# retrieve project
project = Project.objects(name='test-project-flows').first()
self.assertIsNotNone(project)

project.design = Design(
    name='test-design',
    rtl_files=['/path/to/rtl1.v', '/path/to/rtl2.v'],
    netlist_file='/path/to/netlist.v',
    sdc_file='/path/to/const.sdc'
)
project.flows.append(
    Flow(
        flow_directory='/path/to/flow/directory',
        params={
            'param1': 'value1',
            'param2': 'value2'
        },
        stages=[
            Stage(
                name='synth',
                tool=Tool(
                    name='synth_tool',
                    version='0.0.0'
                ),
                machine='test-machine',
                collection_mode=DataCollectionMode.OFFLINE_FROM_LOGS.name,
                status=StageStatus.COMPLETED_SUCCESSFULLY.name,
                log_files=['/path/to/log1',
                           '/path/to/drc', '/path/to/timing'],
                metrics={}       # should be extracted using edaac.parsers
            ),
            Stage(
                name='placement',
                tool=Tool(
                    name='placement_tool',
                    version='0.0.0'
                ),
                machine='test-machine',
                collection_mode=DataCollectionMode.OFFLINE_FROM_LOGS.name,
                status=StageStatus.COMPLETED_SUCCESSFULLY.name,
                log_files=['/path/to/log1',
                           '/path/to/drc', '/path/to/timing'],
                metrics={}       # should be extracted using edaac.parsers
            ),
            Stage(
                name='routing',
                tool=Tool(
```

(continues on next page)

(continued from previous page)

```
                    name='routing_tool',
                    version='0.0.0'
                ),
                machine='test-machine',
                collection_mode=DataCollectionMode.OFFLINE_FROM_LOGS.name,
                status=StageStatus.COMPLETED_SUCCESSFULLY.name,
                log_files=['/path/to/log1',
                            '/path/to/drc', '/path/to/timing'],
                metrics={}      # should be extracted using edaac.parsers
            )
        ],
        log_files=['/path/to/log1', '/path/to/log2']
    )
)

result = project.save()
mongo.disconnect()
```

2.3 License

EDA Analytics Central (EDAAC)

2.4 Feedback

EDAAC is an on-going effort and we would like it to be shaped by our community. If you have feedback for the project (e.g. feature request, metrics collection capabilities, data model improvement), please create an issue on GitHub.

You can also reach out to Abdelrahman, the main contributor, at abdelrahman_hosny@brown.edu.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search