Determining Circuit Characteristics for Aging Estimations of Digital Circuits.

Overview

Aging analysis is significant for reliability of digital circuits. Simulations using tools like SPICE or Cadence may take a very long time. This repository represents a Python tool that can be used to do simulations for large combinational circuits with thousands of gates at both the gate-level and the transistor-level in much less time. The tool uses Python classes to represent cells and construct a directed acyclic graph for analysis. The tool offers a command-line interface to simplify the interaction with users.

Dependencies

- Python
- Python NumPy
- Python Pandas
- Python Matplotlib
- Python Click
- Python Igraph
- Python Leidenalg
- Python Scipy
- Python VCDVCD
- Python PyParsing
- Python Setuptools
- Python Pybind11
- C++ compiler (supporting C++11 or newer)

Setup

- Create Python virtual environment
- Run the next commands

```
pip install numpy pandas matplotlib click igraph leidenalg scipy vcdvcd
pyparsing pybind11 setuptools
cd cpp
python setup.py build_ext --inplace
cd ../
python main.py
```

Example

```
python main.py parse-tr-lib "testing/standard cells" sp -g -n nfet -p pfet
python main.py parse-simulate "testing/ADDER_8BIT.sp" sp adder8 -r 1000
python main.py parse-simulate-tr "testing/ADDER_8BIT.sp" sp adder8_tr -r 1000
python main.py generate-aging adder8_tr bti -o adder8_bti
python main.py remove-lib sp
python main.py remove-tr-lib sp
```

Output files should be under the results folder

Help Docs

To display any command help docs run:

```
python main.py [COMMAND] --help
```

parse-lib

```
main.py parse-lib [OPTIONS] FOLDER LIB_NAME

  Parse .lib file, generate the corresponding gate-level .py file and save it
  in libs

  Arguments:
      FOLDER                the path to the folder of .lib files
      LIB_NAME        the name of the generated lib (should contain alpha-digit-
underscore chars only, starting with an alpha char)

  Note:
      the given file should match the following format,
  """
  cell (<cell-name>){
      ...
      pin (pin-name){
          ...
          direction: input;
          ...
      }
      ...
      pin (pin-name){
          ...
          direction: output;
          ...
          function: "<function>";
          ...
      }
      ...
  }
  """
      the file may contain more than one cell with more than one output pin

  Note:
      the files names should contain only alpha-digit-underscore chars, starting
with alpha char

Options:
  --help  Show this message and exit.
```

parse-tr-lib

```
main.py parse-tr-lib [OPTIONS] FOLDER LIB_NAME

  Parse .sp file of transistor-level cells, generating the corresponding .py
  file and save it in tr_libs

  Arguments:
      FOLDER                the path to the folder of .sp files
      LIB_NAME        the name of the generated tr_lib (should contain alpha-
digit-underscore chars only, starting with an alpha char)

  Note:
      the given file should match the following format,
  """
  .SUBCKT <cell-name> <pin-1> <pin-2> ...
  <instance-name> <drain> <gate> <source> <base> <instance-type(ntype|ptype)>
<...>
  ...
  .ENDS
  """
      the file may contain more than one cell

      VDD , VSS should be used for voltage supply pins


  Note:
      the files names should contain only alpha-digit-underscore chars, starting
with alpha char

Options:
  -g, --generate-gate-level-cells
                                  generate gate-level cells using lookup
                                  tables
  -e, --extend                    extend existing library
  -n, --ntype TEXT                the name used for n type transistors,
                                  default nmos
  -p, --ptype TEXT                the name used for p type transistors,
                                  default pmos
  --help                          Show this message and exit.
```

parse-sp

```
main.py parse-sp [OPTIONS] FILE LIB_NAME OUT_FILENAME

  Parse .sp file, generate the corresponding gate-level .py file and save it
  in libs

  Arguments:
      FILE                the path to .sp file
      LIB_NAME            the name of cells lib in libs folder
      OUT_FILENAME        the name of the generated .py file (should contain
alpha-digit-underscore chars only, starting with an alpha char)

  Note:
      the given file should match the following format,
  """
  .subckt <cell-name> <pin-1> <pin-2> ...
  <instance-name> <signal-1> <signal-2> ... <instance-type>
  ...
  .ends
  """
      the file may contain more than one cell
```

```
      VDD , VSS should be used for voltage supplies

  Note:
      use this command to add new cell to the libs.
      It 's not recommended to add large cell as this may make the tool slawer
every time it loads the libs
      If you want to just simulate a large circuit, use 'parse-simulate' command
instead.

Options:
  --help  Show this message and exit.
```

parse-tr-sp

```
main.py parse-tr-sp [OPTIONS] FILE LIB_NAME OUT_FILENAME

  Parse .sp file, generate the corresponding transistor-level .py file and
  save it in tr-libs for transistor-level analysis

  Arguments:
      FILE                the path to .sp file
      LIB_NAME            the name of cells lib in tr_libs folder
      OUT_FILENAME        the name of the generated .py file (should contain
alpha-digit-underscore chars only, starting with an alpha char)

  Note:
      the given file should match the following format,
  """
  .subckt <cell-name> <pin-1> <pin-2> ...
  <instance-name> <signal-1> <signal-2> ... <instance-type>
  ...
  .ends
  """
      the file may contain more than one cell

      VDD , VSS should be used for voltage supplies

  Note:
      use this command to add new cell to the tr_libs.
      It 's not recommended to add large cell as this may make the tool slower
every time it loads the tr_libs
      If you want to just simulate a large circuit, use 'parse-simulate-tr'
command instead.

Options:
  -g, --generate-gate-level-cells
                                  generate the corresponding gate-level cells
  --help  Show this message and exit.
```

simulate

```
main.py simulate [OPTIONS] CELL_NAME LIB_NAME OUT_FILENAME

  Simulate the given cell name from libs (gate-level simulation) and save
  results to 'results' folder

  Arguments:
      CELL_NAME           the name of cell in the libs
      LIB_NAME            the name of cells lib in tr_libs folder
      NO_VECTORS          the no of random input vectors to apply
      OUT_FILENAME        the name of the generated .csv file (should contain
```

```
alpha-digit-underscore chars only, starting with an alpha char)

  Note:
      inputs file (if provided) should be of binary inputs in CSV format

Options:
  -r, --random-vectors-count INTEGER RANGE
                                  number of random input vectors for
                                  simulation  [x>=0]
  -i, --inputs-file PATH          path to the binary input sequence CSV file
                                  for simulation
  -v, --vcd-file PATH             path to the VCD file for simulation
  -s, --saif-file PATH            path to the SAIF file for simulation
  -lz, --lazy                     lazy loading or generating input vectors
  -p, --parallel                  parallel simulations
  -t, --threads-count INTEGER RANGE
                                  number of threads  [x>=1]
  --help                          Show this message and exit.
```

simulate-tr

```
main.py simulate-tr [OPTIONS] CELL_NAME LIB_NAME OUT_FILENAME

  Simulate the given cell name from tr-libs (transistor-level simulation) and
  save results to 'results' folder for gate-level analysis

  Arguments:
      CELL_NAME           the name of cell in the libs
      LIB_NAME            the name of cells lib in tr_libs folder
      NO_VECTORS          the no of random input vectors to apply
      OUT_FILENAME        the name of the generated .csv file (should contain
alpha-digit-underscore chars only, starting with an alpha char)

  Note:
      inputs file (if provided) should be of binary inputs in CSV format

Options:
  -r, --random-vectors-count INTEGER RANGE
                                  number of random input vectors for
                                  simulation  [x>=0]
  -i, --inputs-file PATH          path to the binary input sequence CSV file
                                  for simulation
  -v, --vcd-file PATH             path to the VCD file for simulation
  -s, --saif-file PATH            path to the SAIF file for simulation
  -lz, --lazy                     lazy loading or generating input vectors
  -p, --parallel                  parallel simulations
  -t, --threads-count INTEGER RANGE
                                  number of threads  [x>=1]
  --help                          Show this message and exit.
```

parse-simulate

```
main.py parse-simulate [OPTIONS] FILE LIB_NAME OUT_FILENAME

  Parse .sp file, and simulate (gate-level) it with the given no of vectors.

  Arguments:
      FILE                the path to .sp file
      LIB_NAME            the name of cells lib in libs folder
      OUT_FILENAME        the name of the generated .py file (should contain
alpha-digit-underscore chars only, starting with an alpha char)
```

```
  Note:
      the given file should match the following format,
  """
  .subckt <cell-name> <pin-1> <pin-2> ...
  <instance-name> <signal-1> <signal-2> ... <instance-type>
  ...
  .ends
  """
      the file may contain more than one cell, only one will be simulated.

      VDD , VSS should be used for voltage supplies

  Note:
      inputs file (if provided) should be of binary inputs in CSV format

Options:
  -n, --no-simulations INTEGER RANGE
                                  no of simulations to do, default 1  [x>=1]
  -r, --random-vectors-count INTEGER RANGE
                                  number of random input vectors for
                                  simulation  [x>=0]
  -i, --inputs-file PATH          path to the binary input sequence CSV file
                                  for simulation
  -v, --vcd-file PATH             path to the VCD file for simulation
  -s, --saif-file PATH            path to the SAIF file for simulation
  -lz, --lazy                     lazy loading or generating input vectors
  -p, --parallel                  parallel simulations
  -t, --threads-count INTEGER RANGE
                                  number of threads  [x>=1]
  --help                          Show this message and exit.
```

parse-simulate-tr

```
main.py parse-simulate-tr [OPTIONS] FILE LIB_NAME OUT_FILENAME

  Parse .sp file, and simulate (transistor-level) it with the given no of
  vectors.

  Arguments:
      FILE                the path to .sp file
      LIB_NAME            the name of cells lib in tr_libs folder
      OUT_FILENAME        the name of the generated .py file (should contain
alpha-digit-underscore chars only, starting with an alpha char)

  Note:
      the given file should match the following format,
  """
  .subckt <cell-name> <pin-1> <pin-2> ...
  <instance-name> <signal-1> <signal-2> ... <instance-type>
  ...
  .ends
  """
      the file may contain more than one cell, only one will be simulated.

      VDD , VSS should be used for voltage supplies

  Note:
      inputs file (if provided) should be of binary inputs in CSV format

Options:
  -n, --no-simulations INTEGER RANGE
                                  no of simulations to do, default 1  [x>=1]
  -r, --random-vectors-count INTEGER RANGE
```

```
                                  number of random input vectors for
                                  simulation  [x>=0]
  -i, --inputs-file PATH          path to the binary input sequence CSV file
                                  for simulation
  -v, --vcd-file PATH             path to the VCD file for simulation
  -s, --saif-file PATH            path to the SAIF file for simulation
  -lz, --lazy                     lazy loading or generating input vectors
  -p, --parallel                  parallel simulations
  -t, --threads-count INTEGER RANGE
                                  number of threads  [x>=1]
  --help                          Show this message and exit.
```

test

```
main.py test [OPTIONS] CELL_NAME LIB_NAME

  Test the given cell name from libs (gate-level simulation) with all possible
  input vectors and save results in the 'results' folder if out-filename is
  given or just print it in the prompt

  Arguments:
      CELL_NAME           the name of cell in the libs
      LIB_NAME            the name of cells lib in libs folder

Options:
  -f, --out-filename NAME  out filename, default None (print the output in the
                           prompt)
  --help                   Show this message and exit.
```

test-tr

```
main.py test-tr [OPTIONS] CELL_NAME LIB_NAME

  Test the given cell name from tr_libs (transistor-level simulation) with all
  possible input vectors and save results in the 'results' folder if out-
  filename is given or just print it in the prompt

  Arguments:
      CELL_NAME           the name of cell in the libs
      LIB_NAME            the name of cells lib in tr_libs folder

Options:
  -f, --out-filename NAME  out filename, default None (print the output in the
                           prompt)
  --help                   Show this message and exit.
```

list-libs

```
main.py list-libs [OPTIONS]

  List all gate-level libraries in the libs folder

Options:
  --help  Show this message and exit.
```

list-tr-libs

```
main.py list-tr-libs [OPTIONS]

  List all transistor-level libraries in the tr_libs folder
```

```
Options:
  --help  Show this message and exit.
```

remove-lib

```
main.py remove-lib [OPTIONS] LIB_NAME

  Remove a library from the libs folder

  Arguments:
      LIB_NAME        the name of the library to be removed

Options:
  --help  Show this message and exit.
```

remove-tr-lib

```
main.py remove-tr-lib [OPTIONS] LIB_NAME

  Remove a library from the tr_libs folder

  Arguments:
      LIB_NAME        the name of the library to be removed

Options:
  --help  Show this message and exit.
```

copy-results

```
main.py copy-results [OPTIONS] FOLDER

  Copy the results files to the specified folder

  Argument:
      FOLDER          the path of the folder

Options:
  --help  Show this message and exit.
```

remove-results

```
main.py remove-results [OPTIONS]

  Remove all results files

Options:
  --help  Show this message and exit.
```

generate-annotations

```
main.py generate-annotations [OPTIONS] FILE LIB_NAME RESULTS_FILENAME

  Replace cells instances by its name and its inputs signal probabilities

  Arguments:
      FILE                the path to .sp file
      LIB_NAME            the name of cells lib in libs folder
      RESULTS_FILENAME    the name of the simulations results file in the
results folder

Options:
```

```
  -o, --out-filename NAME  the name of the generated file in the results
                           folder
  --help  Show this message and exit.
```

generate-aging

```
main.py generate-aging [OPTIONS] RESULTS_FILENAME {bti|hci|bti-hci}

  Generate aging values using the specified model

  Arguments:
      RESULTS_FILENAME    the simulations results filename in the results folder
      MODEL               the name of the mode to use(bti, hci, bti-hci)

Options:
  -o, --out-filename NAME  the name of the generated file in the results
                           folder
  --help                   Show this message and exit.
```

find-paths

```
main.py find-paths [OPTIONS] FILE LIB_NAME AGING_FILENAME

  Find the path (input pin Ĭ¢ĬŁ¢â‚¬Ĭ Ĭ¢â‚¬â„¢ output pin) which sees the
highest/lower aging
  * Highest average threshold voltage shift per transistor in a path Find the
  X paths (e.g., 10 paths) with the highest/lowest aging

  Arguments:
      FILE                the path to .sp file
      LIB_NAME            the name of cells lib in tr_libs folder
      AGING_FILENAME        the name of the aging CSV file in the results folder

Note:
    [X] should be replaced by _X in pins names provided by '-ip' and '-op'
options

Options:
  -o, --out-filename NAME      the name of the generated file in the results
                               folder
  -h, --highest INTEGER RANGE  number of paths with the highest avg  [x>=0]
  -l, --lowest INTEGER RANGE   number of paths with the lowest avg  [x>=0]
  -ip, --input-name NAME       name of an input pin
  -op, --output-name NAME      name of an output pin
  -t, --n-iterations INTEGER RANGE
                                  number of iterations of the binary search
                                  [x>=0]
  --help                       Show this message and exit.
```

generate-clusters

```
main.py generate-clusters [OPTIONS] FILE LIB_NAME AGING_FILENAME

  Find the cluster in the circuit (use clustering algorithms) * Connected (by
  circuit topology) transistors which all share similar threshold voltages

  Arguments:
      FILE                the path to .sp file
      LIB_NAME            the name of cells lib in tr_libs folder
      AGING_FILENAME        the name of the aging CSV file in the results folder

Options:
```

```
  -o, --out-filename NAME  the name of the generated file in the results
                           folder
  --help                   Show this message and exit.
```

generate-opt-seq

```
main.py generate-opt-seq [OPTIONS] FILE LIB_NAME {bti|hci|bti-hci}

  Find the input sequence, which results in the highest/lowest amount of aging
  in the circuit (maximize/minimize average threshold voltage per transistor)

  Arguments:
      FILE                the path to .sp file
      LIB_NAME            the name of cells lib in tr_libs folder
      MODEL               the name of the mode to use(bti, hci, bti-hci)

Options:
  -o, --out-filename NAME         the name of the generated file in the
                                  results folder
  -s, --seq-len INTEGER RANGE     the length of the generated sequence  [x>=1]
  -l, --lowest                    generate sequence to minimize average vth
                                  shift
  -n, --n-iterations INTEGER RANGE
                                  no of iterations to do, default 1  [x>=1]
  -t, --threads-count INTEGER RANGE
                                  number of threads  [x>=1]
  --help                          Show this message and exit.
```

analyze

```
main.py analyze [OPTIONS] [FILES]...

  Analyze results files

  Arguments:
      FILES       names of results files in the results folder separated by
space

  Note:
      files should be of the same type(gate-level|transistor-level|...)

Options:
  -o, --out-path PATH  the path to outputs folder where results will be saved
  --help               Show this message and exit.
```

histograms

```
main.py histograms [OPTIONS] GL_RESULTS TL_RESULTS

  Generate histograms of the followed properties:     - Signal Probability
  - Output Transition Probability     - Output Switching Frequency     - Duty
  Cycle     - Vth Shift

  Arguments:
      GL_RESULTS          the name of gate-level simulation results file
      TL_RESULTS          the name of transistor-level simulation results file
after aging analysis

  Note:
      transistor-level simulation results file should be the output of generate-
aging command
```

```
Options:
  -o, --out-path PATH          the path to outputs folder where results will
                               be saved
  -n, --no-bins INTEGER RANGE  no of bins in histograms, default 10  [x>=1]
  --help                       Show this message and exit.
```

map-libs

```
main.py map-libs [OPTIONS] FROM_LIB TO_LIB

  Map Standard Cells from one library to another library and generate a new
  library

  Arguments:
      FROM_LIB        the name of the source library
      TO_LIB          the name of the destination library

Note:
    to ensure cells will be mapped correctly, cells names should match one of
the following:
    * [A-Za-z0-9]+_x[0-9]+..   (Ex. AND2_X1, or3_x2, xor2_x1_asap7)
    * [A-Za-z0-9]+x[0-9]+..   (Ex. AND2x1, OR3x1_asap7)
    * [A-Za-z0-9]+..   (Ex. AND2, xor2_asap7)

Options:
  -o, --out-lib NAME  the name of the generated library
  -sp, --source-prefix NAME  the prefix used for source library cells
  -tp, --target-prefix NAME  the prefix used for target library cells
  --help              Show this message and exit.
```

map-cells

```
main.py map-cells [OPTIONS] FILE LIB_NAME

  Map cells in the input file according to the map generated by map-libs
  command

  Arguments:
      FILE            the path to the input .sp file
      LIB_NAME        the name of the library generated by map-libs command

Options:
  -o, --out-file PATH  the path of the generated file
  --help               Show this message and exit.
```

Flow Process

This section shows only the flow of commands and how they are related to each other. To know more about any command and how it can be used, refer to the help docs.

Inputs

- Library Folder
- Netlist File
-

Library Parsing

- **parse-tr-lib**

Inputs

- Library Folder
- Name used for P-type and N-type transistors
- '-g' flag

Outputs

- Python Library
-

Gate-Level Analysis

PreProcess

- Library Parsing

Commands

- **parse-simulate**

Inputs

- Netlist File
- Python Library
- Binary inputs CSV | VCD file or
- Number of random vectors
- '-lz' flag with large number of random vectors (or file)

Outputs

- Gate-Level Simulations Results
-

- **generate-annotations**

Inputs

- Netlist File
- Gate-Level Simulations Results

Outputs

- Netlist File with annotations

- **copy results**

Inputs

- Path to destination

Transistor-Level Analysis

PreProcess

- Library Parsing

Commands

- **parse-simulate-tr**

Inputs

- Netlist File
- Python Library
- Binary inputs CSV | VCD file or
- Number of random vectors
- '-lz' flag with large number of random vectors (or file)

Outputs

- Transistor-Level Simulations Results
-

- **generate-aging**

Inputs

- Transistor-Level Simulations Results
- Model name

Outputs

- Aging Results
-

- **generate-clusters**

Inputs

- Netlist File
- Aging Results

Outputs

- Clusters CSV file

- **find-paths**

Inputs

- Netlist File
- Aging Results
- Inputs pins names, default all
- Outputs pins names, default all

- Number of paths with one of the flags '-h' or '-l'

Outputs

- Paths CSV file

- **copy results**

Inputs

- Path to destination

Optimization

PreProcess

- Library Parsing

Commands

- **generate-opt-seq**

Inputs

- Netlist File
- Python Library
- Model name
- Sequence length
- Number of iterations (Ex. 10)
- 'lowest' flag if needed

Outputs

- Optimization results file

- **copy results**

Inputs

- Path to destination

Results Analysis

The tool offers only basic analysis. Users may do any further analysis as desired.

PreProcess

- Gate-Level Analysis
- Transistor-Level Analysis

Commands

- **analyze**

Inputs

- Gate-Level Simulations Results or
- Transistor-Level Simulations Results

Outputs

- Graphs

This is command is useful to compare results after multiple simulations of the same circuit.

- **histograms**

Inputs

- Gate-Level Simulations Results
- Transistor-Level Simulations Results

Outputs

- Histograms

Library Extend

PreProcess

- Library Parsing

Standard Cells Commands

- **parse-tr-lib**

Inputs

- Python Library
- Folder contains standard cells
- Name used for P-type and N-type transistors
- '-g' flag
- '-e' flag

Composite Cells Commands

- **parse-tr-sp**

Inputs

- Python Library
- File contains composite cells
- '-g' flag

Mapping

PreProcess

- Library Parsing

Commands

- **map-libs**

Inputs

- Source Library
- Target Library

Regarding missed standard cells, there are two options:

- copy missed cells and continue
- add new cells to target library using Library Extend, and rerun this command

Outputs

- Python Library
-

- **map-cells**

Inputs

- Python Library
- Netlist file that uses cells of source library

Outputs

- Netlist file