EDA-Schema

Release 1.0.0

Pratik Shrestha

Jun 13, 2024

CONTENTS:

EDA-schema is a property graph datamodel schema developed to represent digital circuit designs and the associated attributes. The contributions of this work include:

1. A standardized set of graph structures and feature sets representing a digital circuit and corresponding subcomponents
2. A dataset of physical designs generated from the IWLS'05 benchmark circuit suite utilizing the open-source 130 nm Process Design Kit (PDK) provided by Skywater and the open-source toolset OpenROAD

NOTE: This GitHub repository is still in the development phase. We are actively working on adding new features, fixing bugs, and enhancing the overall functionality. We appreciate your patience and understanding as we work to finalize the project.

CHAPTER

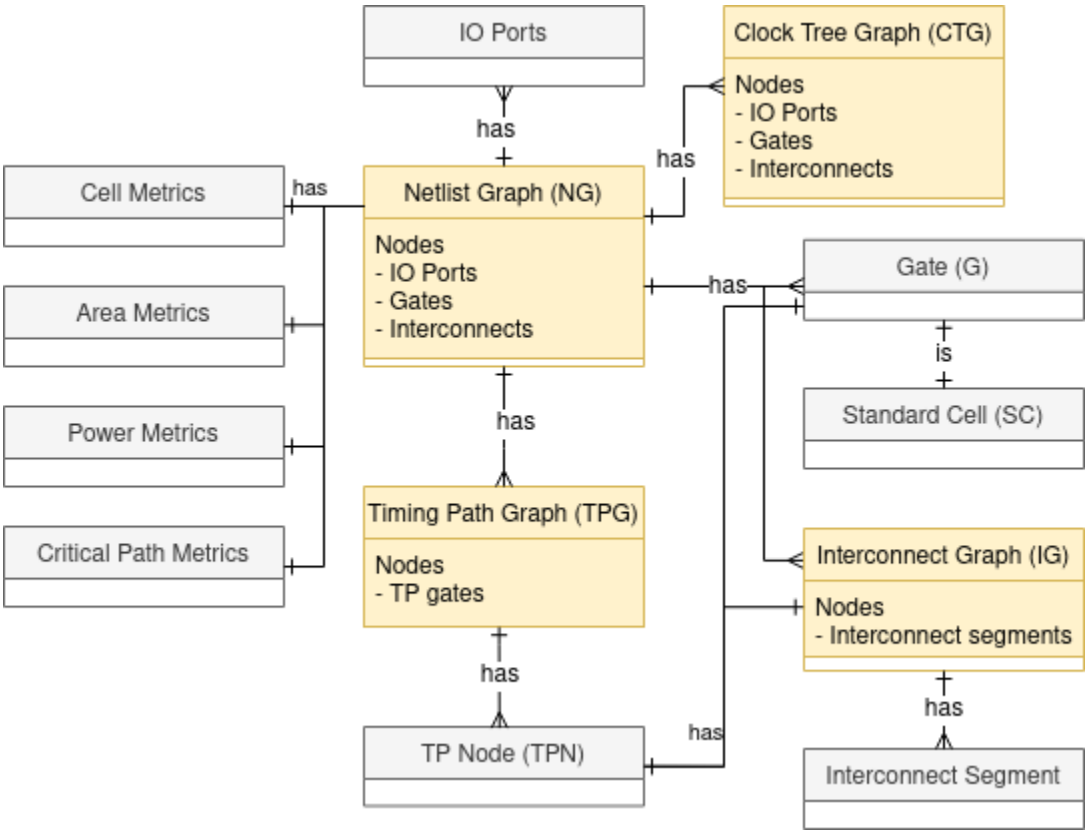
ONE

GRAPH DATAMODEL SCHEMA

EDA-schema is a property graph data model schema used to represent digital circuits. Netlists and performance reports are extracted from design tools after each stage of the EDA design flow. These files are utilized to derive structural information and performance metrics for each phase. While post-routing netlists and performance metrics are final and complete, earlier stages still provide incomplete yet useful structural information and estimated performance metrics. The data becomes more complete and accurate as the design flow progresses.

A circuit extracted from the design flow is represented as a graph where nodes are IO pins, gates, and wires, and edges are the connections between them. This primary netlist graph representation is complemented by more granular graph representations, such as a timing path graph and an interconnect graph. Together, these representations complete the schema.

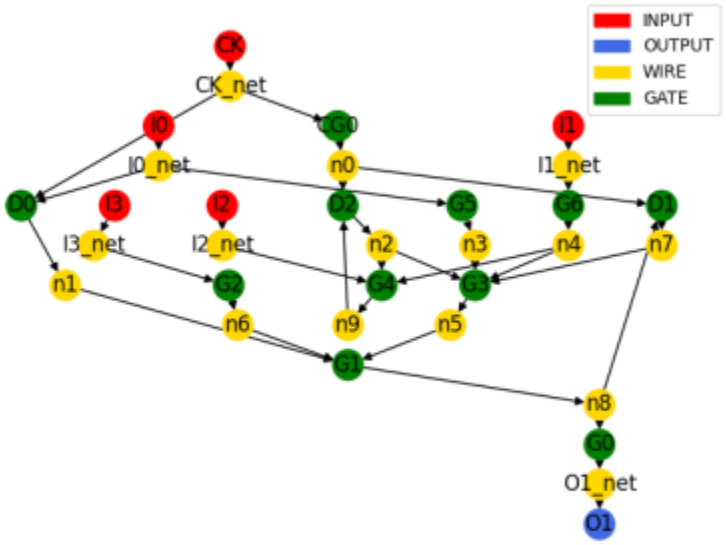
Presented is the entity relationship diagram (ERD) of EDA-schema. The primary graph entities—netlist, clock tree, timing path, and interconnect graphs—are highlighted in gold. Additional tabular entities associated with each graph are shown in silver. For each circuit, data is available for four states: post floorplan, post placement, post CTS, and post routing.



The primary graph entities of the circuit and schema include:

1.1 Netlist Graph (NG)

- The Netlist graph represents comprehensive structural details of digital circuits.
- **Nodes represent**:
  - Input/output ports (IO): These are the entry and exit points of the circuit for signals.
  - Gates (G): These are logical devices that perform basic operations like AND, OR, NOT, etc.
  - Interconnects (I): These are the wires that connect different components.
- **Edges denote** connections between devices and components.
- Gates (G) extend standard cell (SC) with design-specific structural and spatial attributes, capturing more detailed information about each gate's role and connections in the circuit.



Feature	Data Type	Unit	Source
Netlist Graph (NG)			
width	num	m	DEF file
height	num	m	DEF file
no. of inputs	num	#	DEF file
no. of outputs	num	#	DEF file
pin density	num	m^2	Calculated
cell density	num	m^2	Calculated
net density	num	m^2	Calculated
functionality name	str		

continues on next page

Table 1 – continued from previous page

Feature	Data Type	Unit	Source
Standard Cell (SC)			
width	num	m	LEF file
height	num	m	LEF file
no. of input pins	num	#	LEF file
no. of output pins	num	#	LEF file
input capacitance (min, max, avg)	num	pF	Liberty file
output capacitance (min, max, avg)	num	pF	Liberty file
leakage power (min, max, provided)	num	nW	Liberty file
is inverter gate	boolean		Calculated
is buffer gate	boolean		Calculated
is sequential gate	boolean		Calculated
drive strength	num		Calculated
Gate (G)			
name	str		DEF file
standard cell	str		DEF file
no. of fan-ins	num	#	DEF file
no. of fan-outs	num	#	DEF file
gate co-ordinates	(num, num)	(m, m)	DEF file
IO Port (IO)			
name	str		DEF file
direction	str		DEF file
co-ordinate	(num, num)	(m, m)	DEF file
capacitance	num	fF	DEF file

1.2 Quality Metrics

Quality Metrics are extracted from QoR reports to evaluate the performance and efficiency of the design.

Feature	Data Type	Unit	Source
Cell Metrics (CM)			
no. of combinational cell	num	#	QoR reports
no. of sequential cell	num	#	QoR reports
no. of buffer cell	num	#	QoR reports
no. of inverter cell	num	#	QoR reports
no. of macros	num	#	QoR reports
no. of total cell	num	#	QoR reports
Area Metrics (AM)			
combinational cell area	num	m^2	QoR reports
sequential cell area	num	m^2	QoR reports
buffer cell area	num	m^2	QoR reports
inverter cell area	num	m^2	QoR reports
macro area	num	m^2	QoR reports
cell area	num	m^2	QoR reports
net area	num	m^2	QoR reports
total area	num	m^2	QoR reports
Power Metrics (PM)			
combinational power	num	W	QoR reports

continues on next page

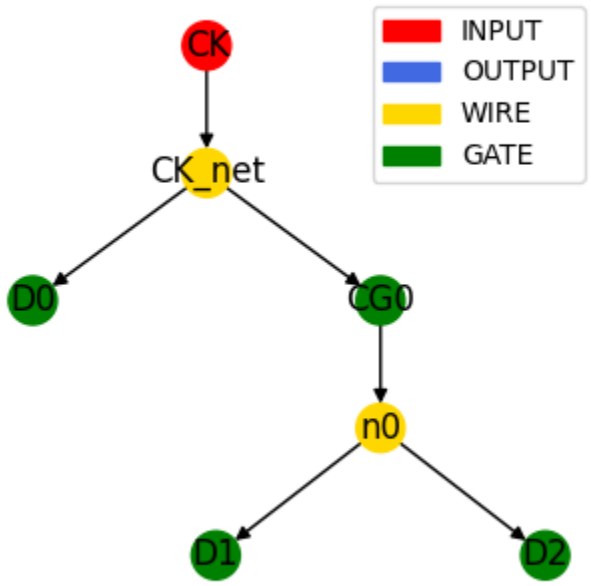
Table 2 – continued from previous page

Feature	Data Type	Unit	Source
sequential power	num	W	QoR reports
macro power	num	W	QoR reports
internal power	num	W	QoR reports
switching power	num	W	QoR reports
leakage power	num	W	QoR reports
total power	num	W	QoR reports
Critical Path Metric (CPM)			
start point	str		STA reports
end point	str		STA reports
path type	str		STA reports
no. of slack violations	num	#	STA reports
worst slack	num	ns	STA reports
worst negative slack	num	ns	STA reports
total negative slack	num	ns	STA reports
no. of hold violations	num	#	STA reports
worst hold violation	num	ns	STA reports
total hold violation	num	ns	STA reports

1.3 Clock Tree Graph (CTG)

Represents the clock distribution network within the circuit, including buffers, inverters, and the connections between them.

- Clock networks are substructures derived from the netlist
  - Follows the same structure
- **Nodes represent**:
  - Input/output ports (IO)
  - Gates (G)
  - Interconnects (I)
- **Edges denote** connections between devices and components

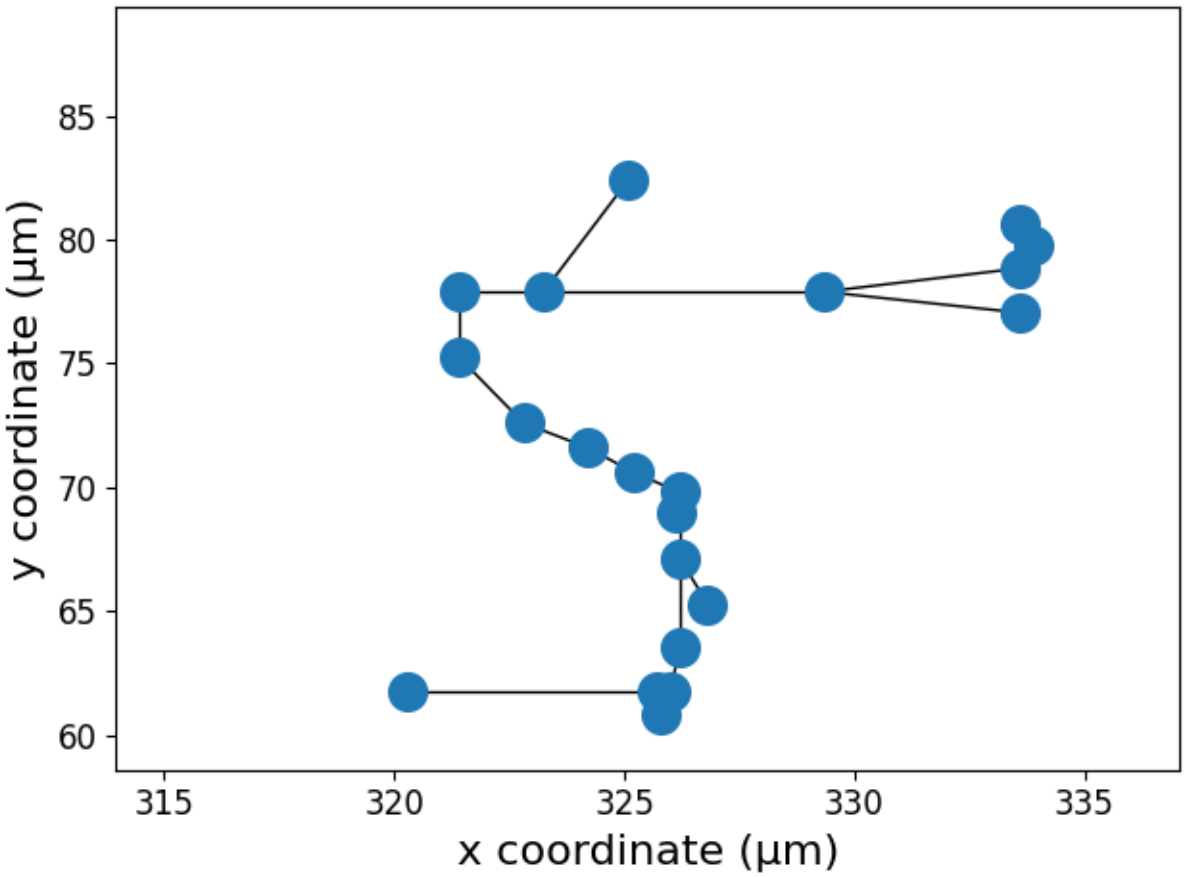


Feature	Data Type	Unit	Source
Clock Tree Graph (CG)			
no. of buffers	num	#	DEF file
no. of inverters	num	#	DEF file
no. of sinks	num	#	DEF file

1.4 Interconnect Graph

Represents the physical layout of the interconnections, including segments of wires and their parasitic impedances.

- Captures spatial layout and connections
  - **Nodes represent** Interconnect Segments (IS)
  - **Edges denote** connections between segments
- Spatial Graph
  - Captures the spatial layout and connections of the circuit

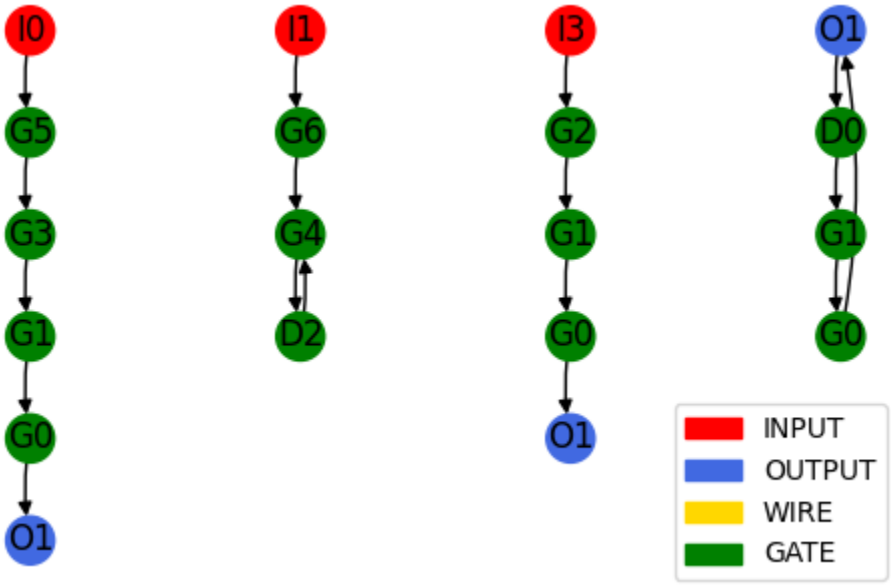


Feature	Data Type	Unit	Source
Interconnect Graph (IG)			
name	str		DEF file
no. of inputs	num	#	DEF file
no. of outputs	num	#	DEF file
bounding box co-ordinates	(num, num, num, num)	(xmin, ymin, xmax, ymax)	DEF file
half perimeter width length	num	m	Calculated
RUDY	num		Calculated
resistance	num	f	SPEF file
capacitance	num	fF	SPEF file
Interconnect Segment (IS)			DEF file
name	str		DEF file
length	num	m	DEF file
start point co-ordinate	(num, num)	m	DEF file
end point co-ordinate	(num, num)	m	DEF file
midpoint	(num, num)	m	Calculated
RUDY	num		Calculated
resistance	num	f	SPEF file
capacitance	num	fF	SPEF file

1.5 Timing Path Graph

Represents the critical timing paths in the circuit, capturing the timing constraints and the delays associated with each path.

- Static Timing Analysis (STA) Timing Path
  - Sequence of circuit elements through which a signal travels
- Timing Path Graphs are subgraphs of netlist graphs
  - **Nodes represent** Timing Path Nodes (TPN)
  - **Edges denote** connections between TPNs



Feature	Data Type	Unit	Source
Timing Path Graph (TPG)			
start point	str		STA reports
end point	str		STA reports
path type	str		STA reports
arrival time	num	ns	STA reports
required time	num	ns	STA reports
slack	num	ns	STA reports
no. of gates	num	#	STA reports
is critical path	boolean		STA reports
Timing Path Node (TPN)			
cell delay	num	ns	STA reports
arrival time	num	ns	STA reports
slew	num	ns	STA reports
is rising transition	boolean		STA reports
is falling transition	boolean		STA reports

CHAPTER

TWO

OPEN DATASET

A comprehensive dataset for physical design is comprised of four key components: 1) Design circuits: selected IWLS'05 benchmark circuits 2) Process Development Kit (PDK): Skywater 130 nm 3) Physical Design Toolset: OpenROAD 4) Design parameters and Constraints:

Parameters	Values or Ranges	# of Samples
Clock periods (ns)	{0.5, 1, 2, 5}	4
Aspect ratio	{0.5, 0.75, 1}	3
Max utilization	{0.3, 0.5}	2
Max skew (ns)	{0.01 - 0.2}	2
Max fanout	{50 - 250}	2
Max clock network capacitance (pF)	{0.05 - 0.3}	2
Max latency (ns)	{0 - 1}	2
Total circuits per design		**48**

The open dataset is available publicly as a zip file of mongoDB data dump.

Following are the details on the open dataset

2.1 Full dataset

- consists of all the generated circuits
- Overall circuits in dataset: 48 * 20 = 960
- Number of gates in dataset: 7,468,228
- Number of nets in dataset: 7,726,920
- Number of timing paths in dataset: 1,561,975
- Total dataset size: 82.836G

2.2 Minimal dataset

- consists of a small subset generated circuits
- Target parameters
  - operating frequency: 1 GHz
  - aspect ratio: 0.5
  - utilizatopm: 70%
- Overall circuits in dataset: 20
- Number of gates in dataset: 272,568
- Number of nets in dataset: 227,148
- Number of timing paths in dataset: 121,298
- Total dataset size: 1.194GB

CHAPTER

THREE

GETTING STARTED

3.1 Installation

The key dependencies are requred by EDA-schema are the following.

- python3.6 or beyond
- pip3
- mongoDB

Clone the repository and use pip for installation.

```
$ git clone git@github.com:drexel-ice/eda-schema.git
$ cd eda-schema
$ pip install -e .
```

3.2 Get Open Dataset

To use the dataset, first download the zip file of mongoDB data dump, unzip it, and execute the follwing command.

```
$ sudo systemctl stop mongod
$ mongod --dbpath <path_to_the_data_dump>
```

CHAPTER

FOUR

CITE THIS WORK

- P. Shrestha, A. Aversa, S. Phatharodom, and I. Savidis, "**EDA-schema: A graph datamodel schema and open dataset for digital design automation**", Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI), pp. 69–77, Jun. 2024.

CHAPTER

FIVE

CONTACT

- For any questions, support, or if you have issues accessing the dataset, please send us an email at ps937@drexel.edu, is338@drexel.edu.
- If you encounter any bugs or have any issues, please feel free to open an issue. We appreciate your feedback and will work to address any problems as quickly as possible.

CHAPTER

SIX

INDEX

6.1 eda_schema package

6.1.1 Submodules

6.1.2 eda_schema.base module

class eda_schema.base.**BaseEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: `object`

Base class for JSON schema validation and attribute setting.

asdict() → Dict[str, Any]

Convert the object to a dictionary representation.

Returns: dict: Dictionary representation of the object attributes.

load(*json_data: Dict[str, Any]*) → None

Load and validate JSON data, setting attributes.

Args: json_data (dict): JSON data to load and validate.

validate(*json_data: Dict[str, Any]*) → None

Validate JSON data against the schema.

Args: json_data (dict): JSON data to validate.

Raises: ValidationError: If JSON data does not conform to the schema.

class eda_schema.base.**GraphEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: `networkx.classes.digraph.DiGraph`, *eda_schema.base.BaseEntity*

Base class for JSON schema validation and attribute setting.

graph_dict()

6.1.3 eda_schema.dataset module

class eda_schema.dataset.**Dataset**(*db_obj*)

Bases: dict

Dictionary class representing an EDA dataset.

Attributes: standard_cells (dict): Dictionary to store standard cell data. db (FileDB): File-based database for storing EDA-related data.

dump_dataset()

Dump the entire dataset into the database.

dump_netlist(*circuit*, *netlist_id*, *phase*)

Dump netlist data into the database.

Args: circuit (str): Circuit name. netlist_id (str): Netlist ID. phase (str): Circuit design phase.

dump_standard_cells()

Dump standard cell data into the database.

load_dataset()

Load the dataset from the database.

load_interconnect(*net_key*, *_net_data=None*, *_net_segment_dict=None*)

load_netlist(*key*, *netlist_data=None*)

load_standard_cells()

load_timing_path(*timing_path_key*, *_timing_path_data=None*, *_timing_point_df=None*, *_timing_point_dict=None*)

standard_cells = {}

class eda_schema.dataset.**StandardCellData**

Bases: dict

Dictionary class for storing standard cell data.

Attributes: seq_cells (list): List of sequential cells.

seq_cells = []

6.1.4 eda_schema.db module

class eda_schema.db.**BaseDB**

Bases: object

Base class for managing data tables and graph data storage.

add_graph_data(*entity_name*, *graph*, *key*)

Add graph data to the database.

Args: entity_name (str): Name of the entity. graph: Graph object containing the data. key (str): Unique identifier for the graph data.

add_table_data(*entity_name*, *data*)

Add multiple rows to a data table in the database.

Args: entity_name (str): Name of the entity. data (list): List of dictionaries containing data for multiple rows.

add_table_row(*entity_name*, *row*)

Add a row to a data table in the database.

Args: entity_name (str): Name of the entity. row (dict): Data for the new row.

area_metric_columns = ['combinational_cell_area', 'sequential_cell_area', 'buffer_area', 'inverter_area', 'macro_area', 'cell_area', 'net_area', 'total_area']

cell_metric_columns = ['no_of_combinational_cells', 'no_of_sequential_cells', 'no_of_buffers', 'no_of_inverters', 'no_of_macros', 'no_of_total_cells']

clock_tree_columns = ['no_of_buffers', 'no_of_clock_sinks']

create_dataset_tables()

Create tables for all entities in the dataset.

critical_path_metric_columns = ['startpoint', 'endpoint', 'worst_arrival_time', 'worst_slack', 'total_negative_slack', 'no_of_timing_paths', 'no_of_slack_violations']

gate_columns = ['name', 'standard_cell', 'no_of_fanins', 'no_of_fanouts', 'x', 'y']

get_graph_data(*entity_name*, *key*)

Retrieve graph data from the database.

Args: entity_name (str): Name of the entity. key (str): Unique identifier for the graph data.

Returns: dict: Graph data as a dictionary.

get_table_data(*entity_name*, ***kwargs*)

Retrieve data from a data table in the database.

Args: entity_name (str): Name of the entity. **kwargs: Filtering criteria for retrieving data.

Returns: pd.DataFrame: Data from the specified data table.

get_table_row(*entity_name*, ***kwargs*)

Retrieve a specific row from a data table in the database.

Args: entity_name (str): Name of the entity. **kwargs: Filtering criteria for retrieving data.

Returns: pd.Series: A specific row from the specified data table.

io_port_columns = ['name', 'direction', 'x', 'y', 'capacitance']

net_columns = ['name', 'no_of_inputs', 'no_of_outputs', 'x_min', 'y_min', 'x_max', 'y_max', 'hwpl', 'rudy', 'resistance', 'capacitance']

net_segment_columns = ['name', 'length', 'x1', 'y1', 'x2', 'y2', 'x', 'y', 'rudy', 'resistance', 'capacitance']

netlist_columns = ['width', 'height', 'no_of_inputs', 'no_of_outputs', 'cell_density', 'pin_density', 'net_density']

```
power_metric_columns = ['combinational_power', 'sequential_power', 'macro_power',
'internal_power', 'switching_power', 'leakage_power', 'total_power']

standard_cell_columns = ['name', 'width', 'height', 'no_of_input_pins',
'no_of_output_pins', 'is_sequential', 'is_inverter', 'is_buffer', 'drive_strength',
'input_capacitance_min', 'input_capacitance_max', 'input_capacitance_mean',
'output_capacitance_min', 'output_capacitance_max', 'output_capacitance_mean',
'leakage_power_min', 'leakage_power_max', 'leakage_power_provided']

timing_path_columns = ['startpoint', 'endpoint', 'path_type', 'arrival_time',
'required_time', 'slack', 'no_of_gates', 'is_critical_path']

timing_point_columns = ['name', 'cell_delay', 'arrival_time', 'slew',
'is_rise_transition', 'is_fall_transition', 'node_depth']
```

class eda_schema.db.**FileDB**(*data_home*)

Bases: *eda_schema.db.BaseDB*

File-based database for storing EDA-related data.

Attributes: data_home (str): Directory path where data tables and graph data are stored.

add_graph_data(*entity_name*, *graph*, *key*)

Add graph data to the database.

Args: entity_name (str): Name of the entity. graph: Graph object containing the data. key (str): Unique identifier for the graph data.

add_table_data(*entity_name*, *data*)

Add multiple rows to a data table in the database.

Args: entity_name (str): Name of the entity. data (list): List of dictionaries containing data for multiple rows.

add_table_row(*entity_name*, *row*)

Add a row to a data table in the database.

Args: entity_name (str): Name of the entity. row (dict): Data for the new row.

create_dataset_tables()

Create tables for all entities in the dataset.

get_graph_data(*entity_name*, *key*)

Retrieve graph data from the database.

Args: entity_name (str): Name of the entity. key (str): Unique identifier for the graph data.

Returns: dict: Graph data as a dictionary.

get_table_data(*entity_name*, ***kwargs*)

Retrieve data from a data table in the database.

Args: entity_name (str): Name of the entity. **kwargs: Filtering criteria for retrieving data.

Returns: pd.DataFrame: Data from the specified data table.

get_table_row(*entity_name*, ***kwargs*)

Retrieve a specific row from a data table in the database.

Args: entity_name (str): Name of the entity. **kwargs: Filtering criteria for retrieving data.

Returns: pd.Series: A specific row from the specified data table.

class eda_schema.db.**MongoDB**(*db_uri*, *db_name*)

Bases: *eda_schema.db.BaseDB*, pymongo.mongo_client.MongoClient

MongoDB database for storing EDA-related data.

Attributes: data_home (str): Directory path where data tables and graph data are stored.

add_graph_data(*entity_name*, *graph*, *key*)

Add graph data to the database.

Args: entity_name (str): Name of the entity. graph: Graph object containing the data. key (str): Unique identifier for the graph data.

add_table_data(*entity_name*, *data*)

Add multiple rows to a data table in the database.

Args: entity_name (str): Name of the entity. data (list): List of dictionaries containing data for multiple rows.

add_table_row(*entity_name*, *row*)

Add a row to a data table in the database.

Args: entity_name (str): Name of the entity. row (dict): Data for the new row.

create_dataset_tables()

Create tables for all entities in the dataset.

get_graph_data(*entity_name*, *key*)

Retrieve graph data from the database.

Args: entity_name (str): Name of the entity. key (str): Unique identifier for the graph data.

Returns: dict: Graph data as a dictionary.

get_table_data(*entity_name*, ***kwargs*)

Retrieve data from a data table in the database.

Args: entity_name (str): Name of the entity. **kwargs: Filtering criteria for retrieving data.

Returns: pd.DataFrame: Data from the specified data table.

get_table_row(*entity_name*, ***kwargs*)

Retrieve a specific row from a data table in the database.

Args: entity_name (str): Name of the entity. **kwargs: Filtering criteria for retrieving data.

Returns: pd.Series: A specific row from the specified data table.

6.1.5 eda_schema.entity module

class eda_schema.entity.**AreaMetricsEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing area metric data using a JSON schema.

```
schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items':
{'properties': {'buffer_area': {'type': 'number'}, 'cell_area': {'type':
'number'}, 'combinational_cell_area': {'type': 'number'}, 'inverter_area':
{'type': 'number'}, 'macro_area': {'type': 'number'}, 'net_area': {'type':
'number'}, 'sequential_cell_area': {'type': 'number'}, 'total_area': {'type':
'number'}}, 'type': 'object'}, 'type': 'array'}
```

title = 'area_metrics'

class eda_schema.entity.**CellMetricsEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing cell metric data using a JSON schema.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'no_of_buffers': {'type': 'number'}, 'no_of_combinational_cells': {'type': 'number'}, 'no_of_inverters': {'type': 'number'}, 'no_of_macros': {'type': 'number'}, 'no_of_sequential_cells': {'type': 'number'}, 'no_of_total_cells': {'type': 'number'}}, 'type': 'object'}, 'type': 'array'}

title = 'cell_metrics'

class eda_schema.entity.**ClockTreeEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.GraphEntity*

Class for representing clock tree using a JSON schema.

load_from_netlist(*netlist*, *clock_source*, *dffs*)

Load clock tree data from a netlist.

Args: netlist: Netlist data. clock_source: Clock source information. dffs: D flip-flop information.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'no_of_buffers': {'type': 'number'}, 'no_of_clock_sinks': {'type': 'number'}}, 'type': 'object'}, 'type': 'array'}

title = 'clock_tree'

traverse_cts(*node*)

Traverse the clock tree structure starting from a given node.

Args: node: Starting node for traversal.

Returns: list: List of traversed nodes in the clock tree.

class eda_schema.entity.**CriticalPathMetricsEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing critical path metric data using a JSON schema.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'endpoint': {'type': 'string'}, 'no_of_slack_violations': {'type': 'number'}, 'no_of_timing_paths': {'type': 'number'}, 'startpoint': {'type': 'string'}, 'total_negative_slack': {'type': 'number'}, 'worst_arrival_time': {'type': 'number'}, 'worst_slack': {'type': 'number'}}, 'type': 'object'}, 'type': 'array'}

title = 'critical_path_metrics'

class eda_schema.entity.**GateEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing gate data using a JSON schema.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'name': {'type': 'string'}, 'no_of_fanins': {'type': 'number'}, 'no_of_fanouts': {'type': 'number'}, 'standard_cell': {'type': 'string'}, 'x': {'type': ['number', 'null']}, 'y': {'type': ['number', 'null']}}, 'type': 'object'}, 'type': 'array'}

title = 'gate'

class eda_schema.entity.**IOPortEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing input/output port data using a JSON schema.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'capacitance': {'type': ['number', 'null']}, 'direction': {'type': 'string'}, 'name': {'type': 'string'}, 'x': {'type': ['number', 'null']}, 'y': {'type': ['number', 'null']}}, 'type': 'object'}, 'type': 'array'}

title = 'io_port'

class eda_schema.entity.**InterconnectEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.GraphEntity*

Class for representing interconnect data using a JSON schema.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'capacitance': {'type': ['number', 'null']}, 'hwpl': {'type': ['number', 'null']}, 'name': {'type': 'string'}, 'no_of_inputs': {'type': 'number'}, 'no_of_outputs': {'type': 'number'}, 'resistance': {'type': ['number', 'null']}, 'rudy': {'type': ['number', 'null']}, 'x_max': {'type': ['number', 'null']}, 'x_min': {'type': ['number', 'null']}, 'y_max': {'type': ['number', 'null']}, 'y_min': {'type': ['number', 'null']}}, 'type': 'object'}, 'type': 'array'}

title = 'interconnect'

class eda_schema.entity.**InterconnectSegmentEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing interconnect segment data using a JSON schema.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'capacitance': {'type': ['number', 'null']}, 'length': {'type': ['number', 'null']}, 'name': {'type': 'string'}, 'resistance': {'type': ['number', 'null']}, 'rudy': {'type': ['number', 'null']}, 'x': {'type': ['number', 'null']}, 'x1': {'type': ['number', 'null']}, 'x2': {'type': ['number', 'null']}, 'y': {'type': ['number', 'null']}, 'y1': {'type': ['number', 'null']}, 'y2': {'type': ['number', 'null']}}, 'type': 'object'}, 'type': 'array'}

title = 'interconnect_segment'

class eda_schema.entity.**NetlistEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.GraphEntity*

Class for representing netlist data using a JSON schema.

area_metrics = None

cell_metrics = None

clock_trees = {}

critical_path_metrics = None

power_metrics = None

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'cell_density': {'type': ['number', 'null']}, 'height': {'type': ['number', 'null']}, 'net_density': {'type': ['number', 'null']}, 'no_of_inputs': {'type': 'number'}, 'no_of_outputs': {'type': 'number'}, 'pin_density': {'type': ['number', 'null']}, 'width': {'type': ['number', 'null']}}, 'type': 'object'}, 'type': 'array'}

timing_paths = {}

title = 'netlist'

class eda_schema.entity.**PowerMetricsEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing power metric data using a JSON schema.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'combinational_power': {'type': 'number'}, 'internal_power': {'type': 'number'}, 'leakage_power': {'type': 'number'}, 'macro_power': {'type': 'number'}, 'sequential_power': {'type': 'number'}, 'switching_power': {'type': 'number'}, 'total_power': {'type': 'number'}}, 'type': 'object'}, 'type': 'array'}

title = 'power_metrics'

class eda_schema.entity.**StandardCellEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing standard cell data using a JSON schema.

schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items': {'properties': {'drive_strength': {'type': ['number', 'null']}, 'height': {'type': 'number'}, 'input_capacitance_max': {'type': ['number', 'null']}, 'input_capacitance_mean': {'type': ['number', 'null']}, 'input_capacitance_min': {'type': ['number', 'null']}, 'is_buffer': {'type': 'boolean'}, 'is_inverter': {'type': 'boolean'}, 'is_sequential': {'type': 'boolean'}, 'leakage_power_max': {'type': ['number', 'null']}, 'leakage_power_min': {'type': ['number', 'null']}, 'leakage_power_provided': {'type': 'number'}, 'name': {'type': 'string'}, 'no_of_input_pins': {'type': 'number'}, 'no_of_output_pins': {'type': 'number'}, 'output_capacitance_max': {'type': ['number', 'null']}, 'output_capacitance_mean': {'type': ['number', 'null']}, 'output_capacitance_min': {'type': ['number', 'null']}, 'width': {'type': 'number'}}, 'type': 'object'}, 'type': 'array'}

title = 'standard_cell'

class eda_schema.entity.**TimingPathEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.GraphEntity*

Class for representing timing path data using a JSON schema.

```
schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items':
{'properties': {'arrival_time': {'type': 'number'}, 'endpoint': {'type':
'string'}, 'is_critical_path': {'type': 'boolean'}, 'no_of_gates': {'type':
'number'}, 'path_type': {'type': 'string'}, 'required_time': {'type': 'number'},
'slack': {'type': 'number'}, 'startpoint': {'type': 'string'}}, 'type':
'object'}, 'type': 'array'}

title = 'timing_path'
```

class eda_schema.entity.**TimingPointEntity**(*json_data: Optional[Dict[str, Any]] = None*)

Bases: *eda_schema.base.BaseEntity*

Class for representing timing path point data using a JSON schema.

```
schema = {'$schema': 'http://json-schema.org/draft-04/schema#', 'items':
{'properties': {'arrival_time': {'type': 'number'}, 'cell_delay': {'type':
'number'}, 'is_fall_transition': {'type': 'boolean'}, 'is_rise_transition':
{'type': 'boolean'}, 'name': {'type': 'string'}, 'node_depth': {'type':
'number'}, 'slew': {'type': 'number'}}, 'type': 'object'}, 'type': 'array'}

title = 'timing_point'
```

6.1.6 eda_schema.errors module

exception eda_schema.errors.**EDASchemaError**

Bases: Exception

Base exception class for EDA schema-related errors.

exception eda_schema.errors.**ValidationError**

Bases: *eda_schema.errors.EDASchemaError*

Exception raised for validation errors.

6.1.7 eda_schema.json_utils module

eda_schema.json_utils.**dump_json**(*home: str*, *schema_object:* eda_schema.base.BaseEntity, *suffix: str*) → None

Dump schema object data into a JSON file.

Args: home (str): Directory where JSON file will be saved. schema_object (BaseEntity): The schema object to be serialized into JSON.

eda_schema.json_utils.**load_json**(*home: str*, *schema_class: Type[*eda_schema.base.BaseEntity*]*) → *eda_schema.base.BaseEntity*

Load JSON data from a file, validate it using the given schema, and create a schema object.

Args: home (str): Directory where JSON file is located. schema_class (class): The schema class to use for validation and object creation.

Returns: BaseEntity: Instance of the schema object loaded with validated JSON data.

6.1.8 Module contents

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

G

I

L

M

N

P

S

T

V