najaeda

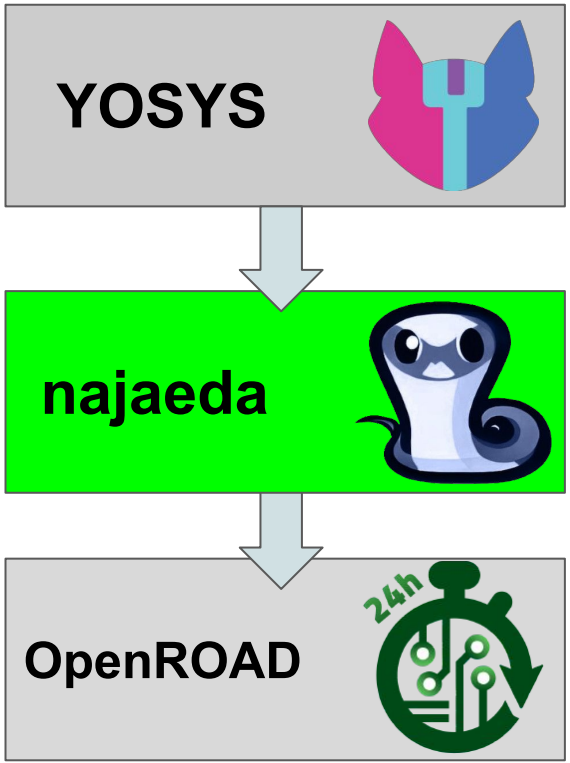
pip install najaeda

Christophe Alexandre
christophe.alexandre@keplertech.io

https://pypi.org/project/najaeda/
https://najaeda.readthedocs.io/en/latest/
https://github.com/najaeda/naja

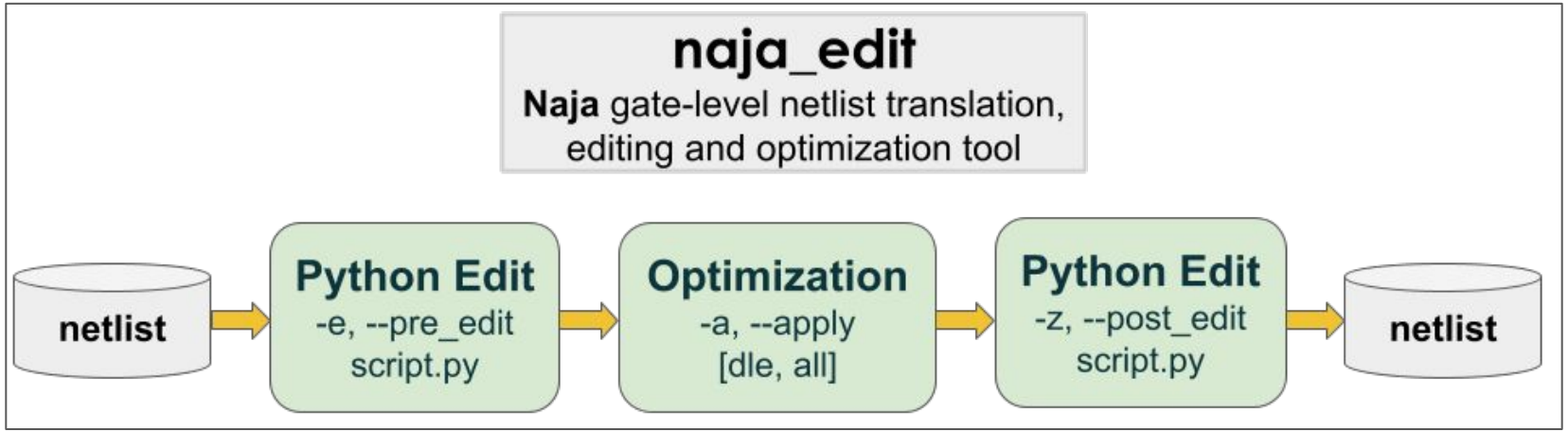
nlnet FOUNDATION

Introduction: **najaeda** purpose



- Introduced in December 2024.
- Simplify EDA tool development: **Enable Fast EDA**.
- Simplify EDA tool installation: '**pip install najaeda**'.
- **High Fidelity**: Preserve user data – minimize differences between input and output when handling netlists.
- **Scalability & Performance**: develop powerful algorithms for netlist optimization, place-and-route, and data mining on large-scale designs.
- Includes Engineering Change Order **(ECO)** management and netlist debugging.
- Collect structured netlist data with data/AI ecosystem, including Pandas, PyTorch,...

najaeda: the top layer of the **naja** ecosystem



Design name	Initial #gates	Post naja_edit #gates	#gates reduction	Run Time
megaboom	3871705	3042874	-21,40%	~15 mins

But **najaeda** is not just a python wrapper on top of naja C++

Instance

- Represents a specific occurrence within the hierarchy (**A/B/C1**, **A/B/C2**).
- Simplifies hierarchy traversal and management.
- Unifies multiple design views: logical, timing, and physical (in progress).
- Automates uniquification: Modifying **A/B/C** generates new instances of **B** and **A** as needed.

Term, Net and Equipotential

- Term can be a top-level term or an instance term.
- Supports connection and disconnection operations.
- Provides a single-bit and multi-bit API, enabling bus support.
- Equipotential: flattening connectivity across hierarchies.

najaeda Code Examples

najaeda: loading an ASIC or FPGA design

```
from najaeda import netlist

netlist.load_liberty(['primitives.lib'])
top = netlist.load_verilog(['design.v'])
top.dump_verilog('design_naja.v')
```

```
from najaeda import netlist

netlist.load_primitives('xilinx')
top = netlist.load_verilog(['design.v'])
top.dump_verilog('design_naja.v')
```

najaeda: explore the design - print instances

```
def print_netlist(instance):
    for child_instance in instance.get_child_instances():
        print(f"{child_instance}:{child_instance.get_model_name()}")
        print_netlist(child_instance)
```

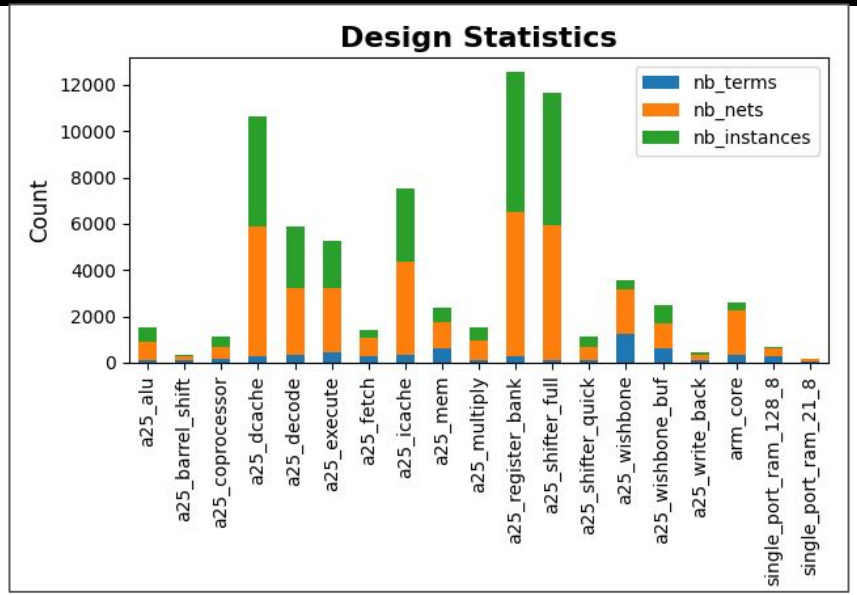
```
def print_instance(instance):
    print(f"{instance}:{instance.get_model_name()}")
visitor_config = instance_visitor.VisitorConfig(callback=print_instance)
instance_visitor.visit(top, visitor_config)
```

najaeda: simple statistics

```
leaves = {"count": 0, "assigns": 0, "constants": 0}
def count_leaves(instance, leaves):
    if instance.is_leaf():
        if instance.is_assign():
            leaves["assigns"] += 1
        elif instance.is_const():
            leaves["constants"] += 1
        else:
            leaves["count"] += 1
visitor_config = instance_visitor.VisitorConfig(callback=count_leaves, args=(leaves,))
instance_visitor.visit(top, visitor_config)
print(f"{top} leaves count")
print(f"nb_assigns={leaves['assigns']}")
print(f"nb constants={leaves['constants']}")
print(f"nb other leaves={leaves['count']}")
```

najaeda: extract statistics with Pandas

```
top = netlist.load_verilog(['design.v'])
design_stats_file = open('design.stats', 'w')
stats.dump_instance_stats_text(top, design_stats_file)
```

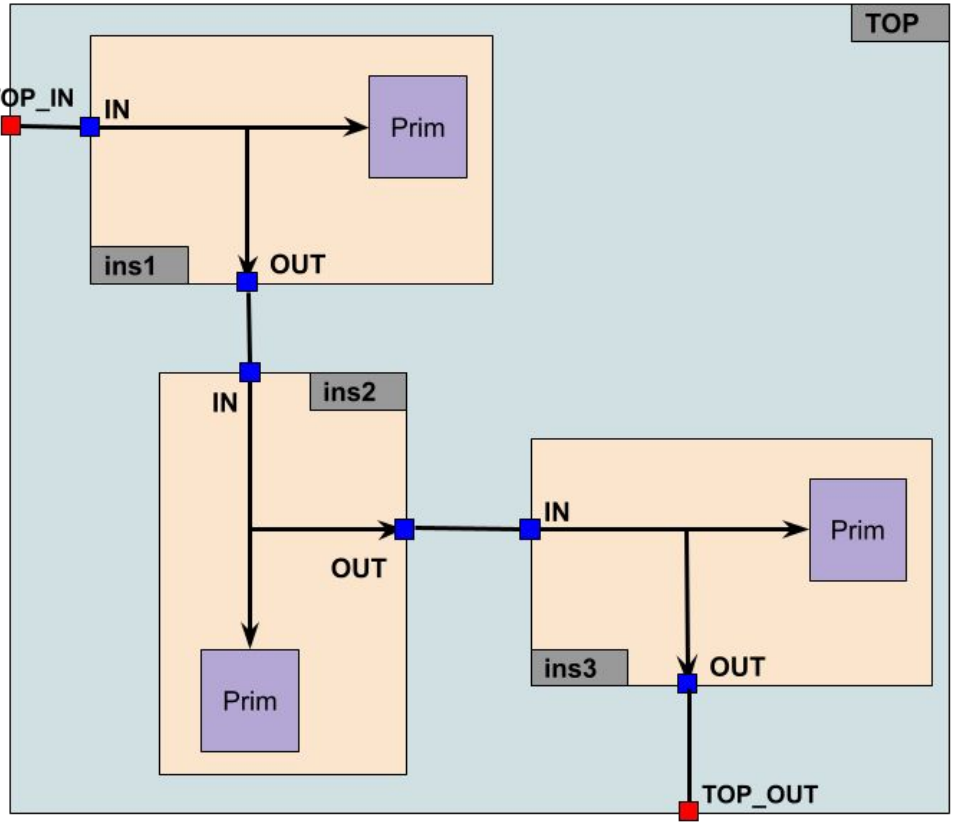


najaeda: DLE (Dead Logic Elimination) in 30 lines

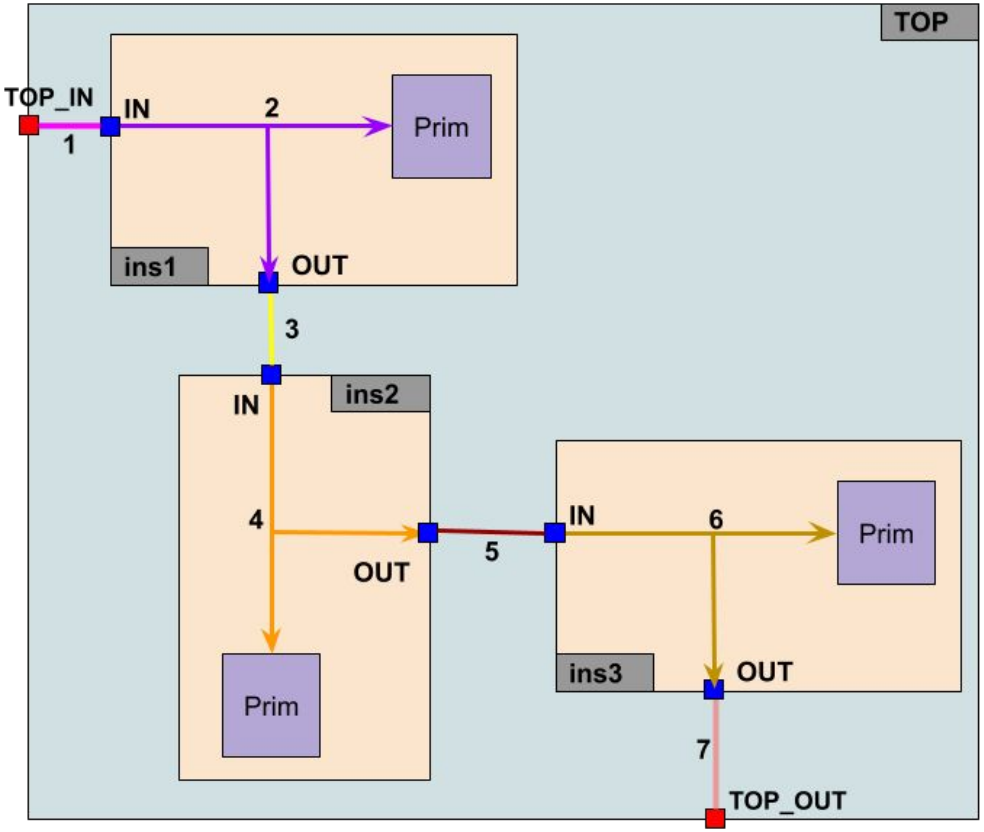
```python
def apply_dle(top, keep_attributes=None):
    # Trace back from design outputs
    visited = set()
    traced_terms = list(top.get_flat_output_terms())
    for leaf in top.get_leaf_children():
        attributes =  list(leaf.get_attributes())
        for attr in attributes:
            if attr in keep_attributes:
                for term in leaf.get_flat_input_terms():
                    traced_terms.append(term)
                break
    for termToTrace in traced_terms:
        queue = deque([termToTrace])
        while queue:
            term = queue.popleft()
            if term in visited:
                continue
            visited.add(term)
            equipotential = term.get_equipotential()
            leaf_drivers = equipotential.get_leaf_drivers()
            for driver in leaf_drivers:
                instance = driver.get_instance()
                instances.add(instance)
                input_terms = instance.get_flat_input_terms()
                queue.extend(input_terms)

    to_delete = [leaf for leaf in top.get_leaf_children() if leaf not in instances]
    for leaf in to_delete:
        leaf.delete()
    return to_delete
```

najaeda Connectivity API

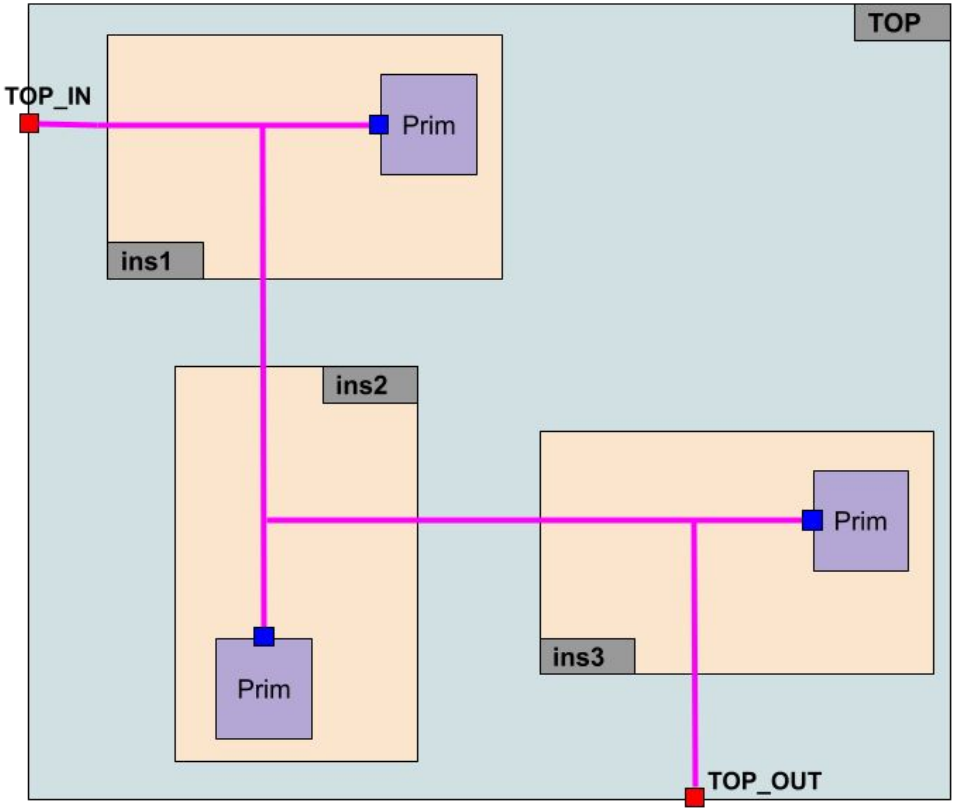
najaeda: Net or Equipotential, what is the difference ?



We can browse through 7 hierarchical nets



Or 1 flat equipotential with 5 connection points



najaeda: next steps

- Physical view
- LEF/DEF support
- Ongoing Experiments: Exploring data extraction workflows for seamless integration with Pandas, PyTorch, and other AI/data tools.