OpenAccess Gear Functionality:

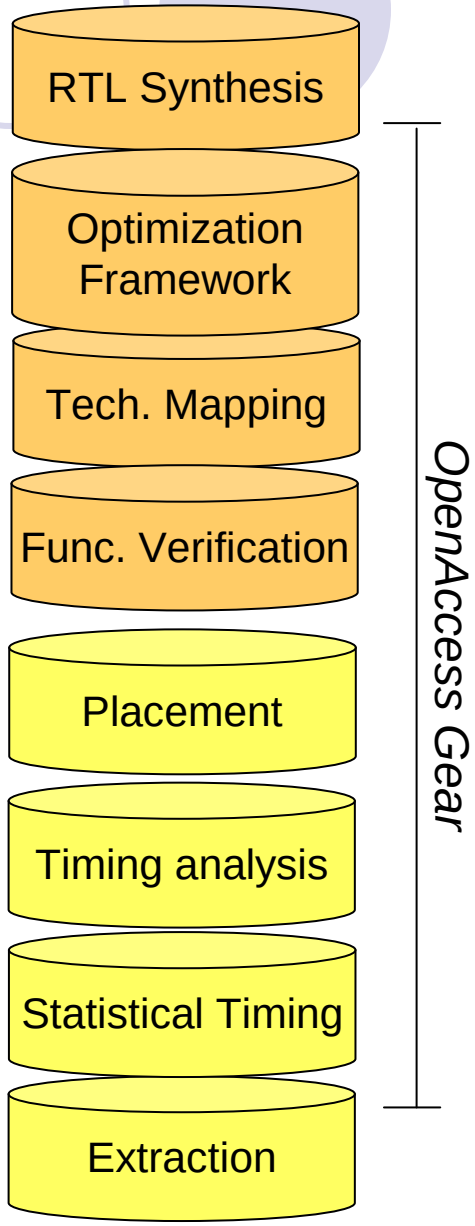
A Platform for Functional Representation, Synthesis, and Verification

Aaron P. Hurst [1,2]
Philip Chong [1]
Andreas Kuehlmann [1]
Christoph Albrecht [1]

[1] Cadence Berkeley Labs [2] University of California, Berkeley

Overview

- OpenAccess Gear (oaGear) is an open source library that extends the utility of the OpenAccess database with a set of common tools and applications
  - Includes static timer, placer, and GUI
  - Next release adds extraction, statistical timing, and multiple clock domains
- The Functionality package (oaGearFunc) provides a platform for representing, synthesizing, and reasoning about design functionality

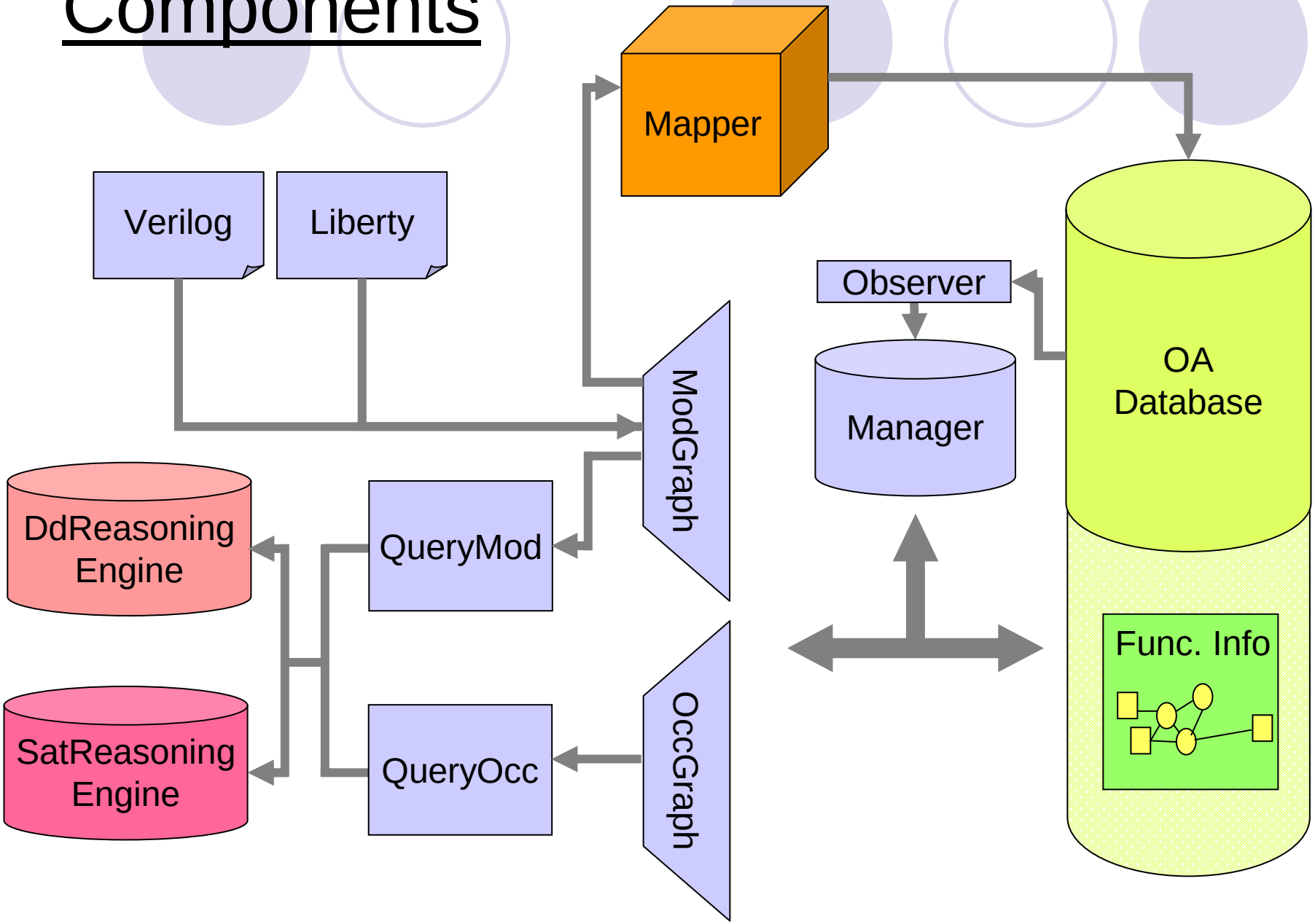


History and Motivation

- OpenAccess software provides…
  - Robust database
  - Rich but extensible data model
  - Common format with industrial tools
- Problem: model limited to structural and physical data
- Solution: extend data model to functionality
- Initially developed for a class at Berkeley
- Goal: provide a platform for EDA research that supplies…

1. **Context:** evaluation within a complete flow
2. **Cross-comparison:** standardized benchmarks, algorithms, and timing analysis

Components

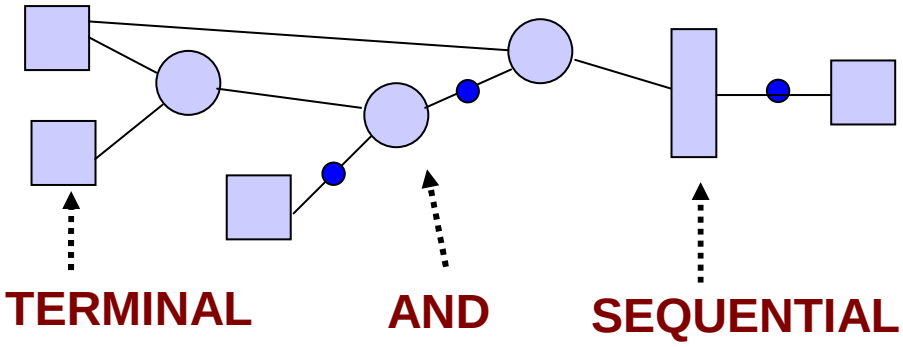

Mapper
Verilog
Liberty
Observer
Manager
OA
Database
ModGraph
DdReasoning
Engine
QueryMod
Func. Info
SatReasoning
Engine
QueryOcc
OccGraph

Front-end

- Functional descriptions can be imported from…
  - RTL Verilog
  - Liberty (.lib) libraries
- RTL Verilog parser understands a synthesizable subset of the commonly used 1995 standard
  - Structural / mapped designs
  - Hierarchical designs
  - Behavioral descriptions
  - Arithmetic operations
  - Sequential logic
- Functional description read into OpenAccess database
  - Automatically serialized and unserialized
  - No data conversion, import/export, etc.

Graph Representation

- AND-INVERTER graphs + sequential nodes



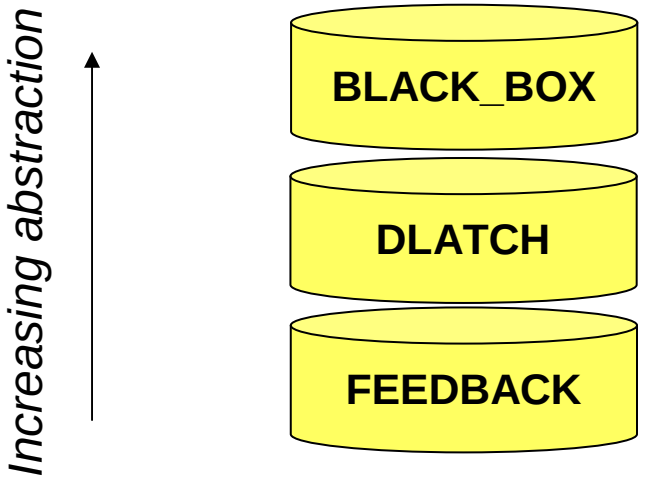
- Three node types
- Edge complementation

- Better than “boxes with pins” because…
  - No special cases
  - Fast to traverse
- Better than abstract representations of functionality…
  - Scale linearly size of design
  - Preserve topology of circuit

Sequential Elements

- All sequential nodes have an input, an output, and store exactly one bit of state

- The sequential behavior is specified using one of three available models…



- Hides non-synchronous behavior
- Mimics finite state machine model

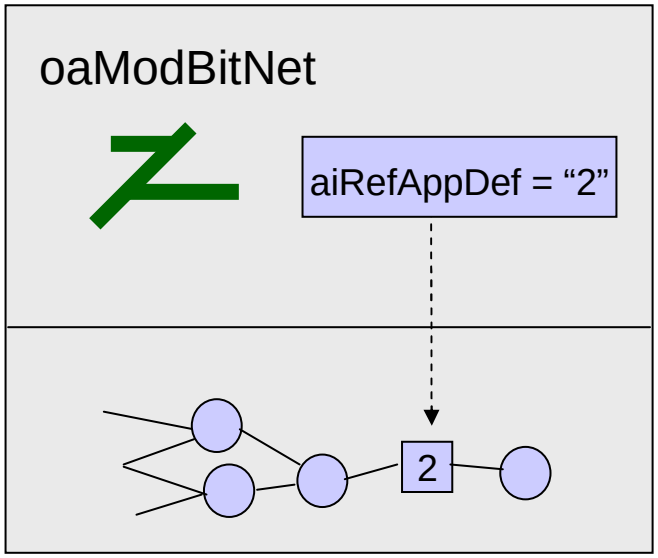
- Single primitive element: D latch
- Appropriate for mapping

- State storage as feedback loops
- Leaves combinational paths intact

Graphs and OpenAccess

- Graphs used to describe relationships between nets in modules

- ModBitNets are the working units
- Connections created to TERMINAL nodes
  - The reference is stored as a persistent AppDef on the net



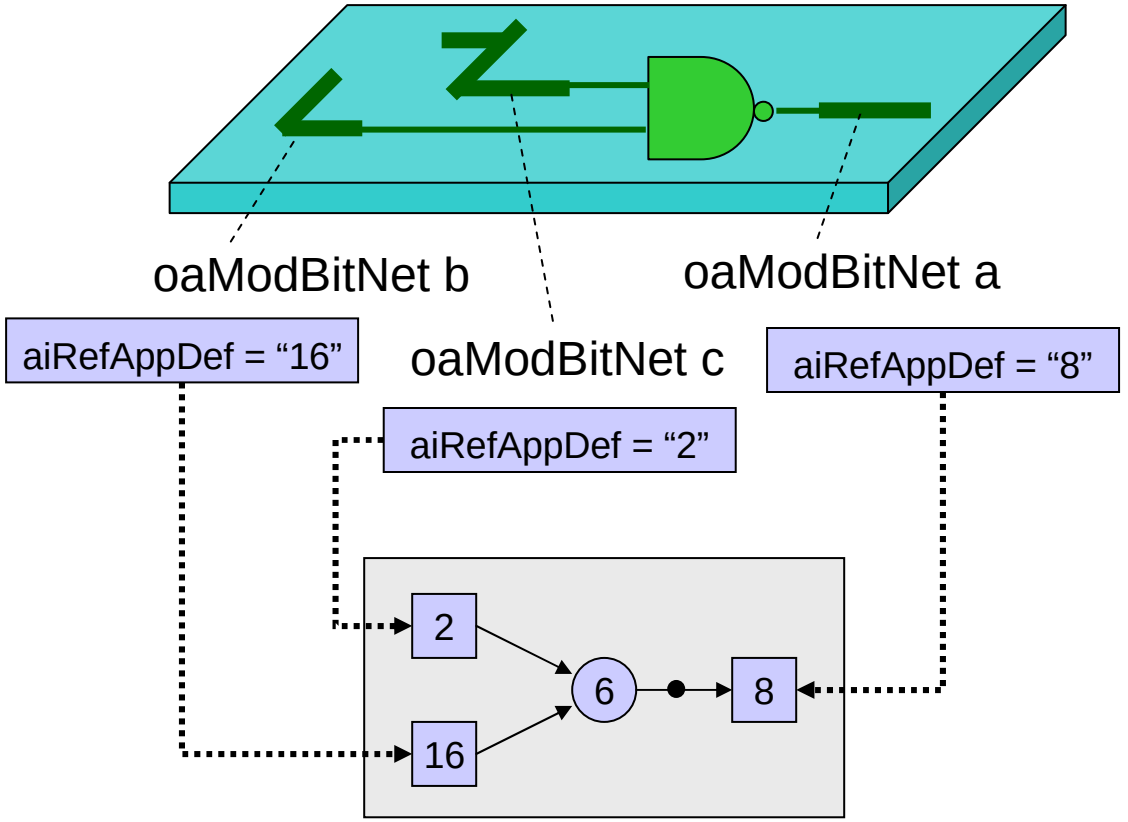
- The graph structures are allocated and maintained by a Manager object, one per design
  - i.e. getNetToAiConnection(ModBitNet or TERMINAL)

Graphs and OpenAccess

- An example snippet of Verilog

```
wire a, b, c;
assign a = ~(b&c);
```

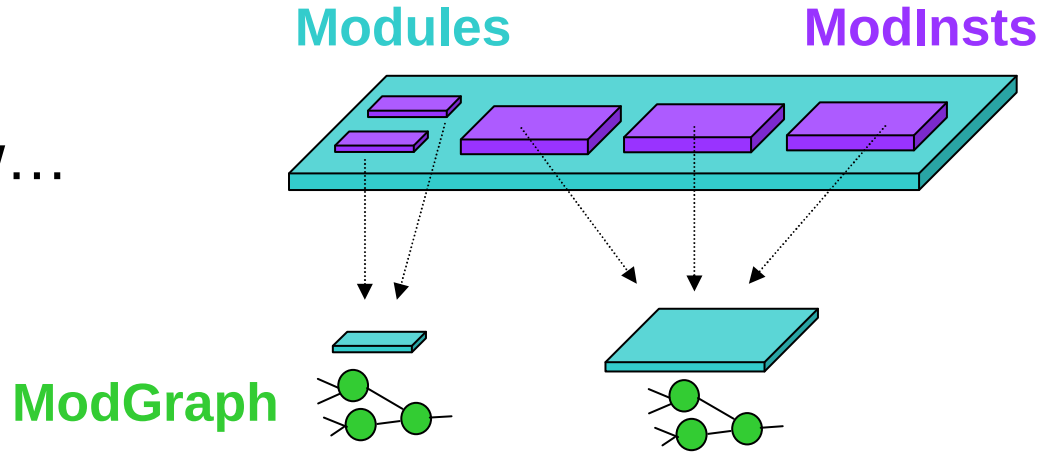
  - NAND gate



OA Domains: Graphs and Modules

- OA defines several “domains” : views of a design

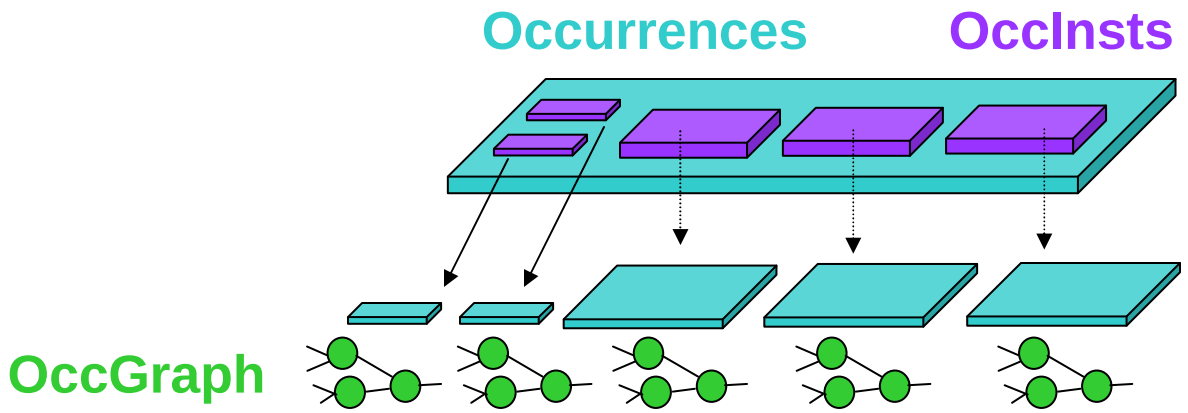
In the module (folded) view…



- Modules objects (including graphs) elements characterize all instances of that object
- All instances implement the same local behavior
  - Functional descriptions “live” in the module domain
- ModGraph structure can be modified
  - Changes affect all instances of that module

OA Domains: Graphs and Occurrences

In the occurrence (unfolded) view…



- Occurrence objects (including graphs) elements identify one instance of an object
- The hierarchal context of instance is known
- OccGraphs are natural for global functional behavior
- OccGraphs can not be modified
  - Objects must be *uniquified* first

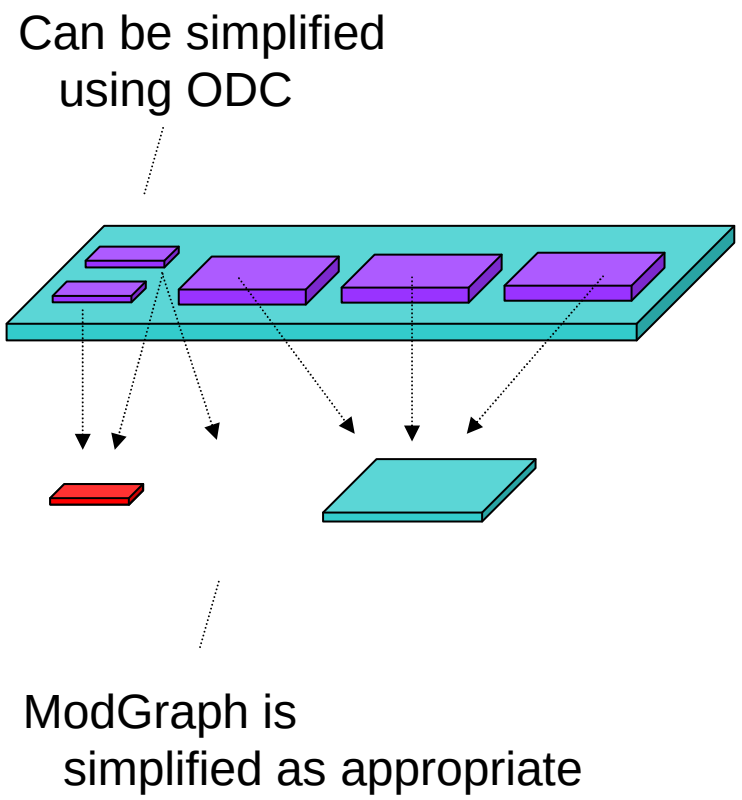
Toolbox of Algorithms

- Parallel ModGraph and OccGraph interfaces
- AI graph package
  - Paging, garbage collection, hashing
  - Graph manipulations
    - Transitive fan-in and fan-out
    - K-way cut enumeration
    - Equivalent node marking and substitution
- Manipulating design hierarchy (flatten, uniquify…)
- Managing connections between OA and graphs
- Query objects and swappable reasoning engines

Example: Simplifying with an ODC

- An observability don't care (ODC) is a subset of node functionality that is irrelevant due to the subsequent logic
- ODCs only apply to a specific instance

1. Uniquify instance
2. Simplify new module

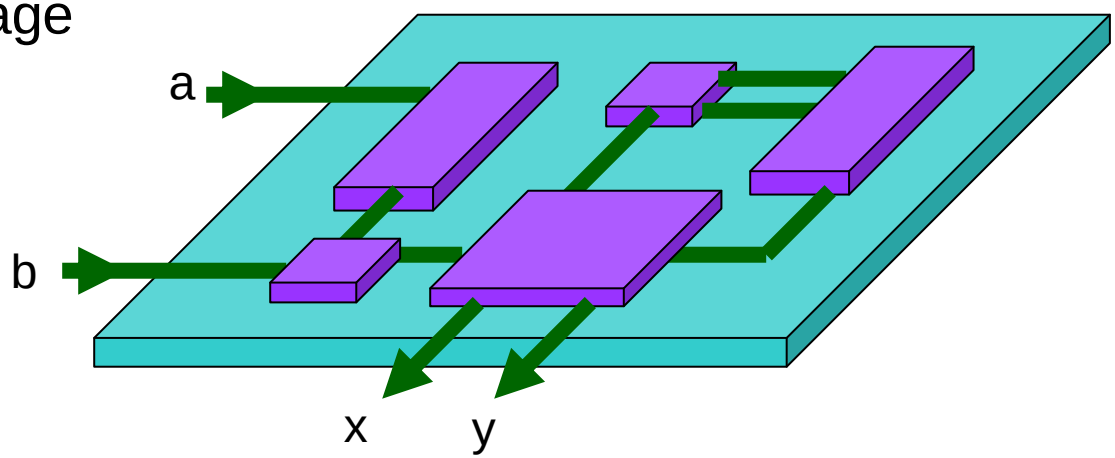


Example: BDD of an Output

What function is being implemented at output x?

1. Initialize a reasoning engine
   - Knows “AND” and “NOT”
   - i.e. CUDD BDD package
2. Create a QueryOcc
3. *Set* all inputs

```
myQuery.set(a,aVar);
```



4. *Get* the output

```
myQuery.get(x);
```

x = f(aVar , bVar)

- Basis for the included logical equivalence checker

Mapping

- In a mapped design…
  - All functionality is implemented in the leaf cells
  - Leaf cells are drawn from a set of library cells
- Technology mapping removes pieces of the graph and replaces them with functionally equivalent library instances
- Hand-off to purely OA flow
- Included SimpleMapper performs a direct mapping from graph elements into three library cells

Conclusions

OA Gear Functionality…

- A simple but complete representation for design functionality
- Transparently integrated into OpenAccess data model
- A set of reference tools to take a design from an RTL description to a mapped netlist
- The foundation for others to incrementally build and improve a synthesis flow