OpenAccess Gear Timing

Luis Guerra e Silva[1,2], Afshin Abdollahi[1,3], Philip Chong[1], Christoph Albrecht[1], Joel Phillips[1], Andreas Kuehlmann[1]

[1]Cadence Berkeley Labs, [2]IST/INESC-ID, [3]USC

April 11, 2006

Outline

- What is OA Gear ? OA Gear goals. OA Gear Components.
- OA Gear Timing: Overview. Internal Data Structures. Incremental TA.
- Additions in the Last Release
- Upcoming Features
  - Better Library Support;
  - Simple Timing Graph;
  - Generic Timing Algebras;
  - Interconnect Delay Calculation;
- Future Work

What OA Gear is ?

- library of **utilities and components** built on top of OpenAccess, with **features and performance sufficient for academic research** in EDA;
- project started in the Summer of 2004, with 2 student interns, at Cadence Berkeley Labs, and has since been supported by Cadence, with additional contributions from Universities.
- **open-source** development model;
- **fixes/improvements** continuously being implemented from user feedback; some users have even provided complete patches.

OA Gear Goals

- make OpenAccess a **useful** and **accessible** platform for academia;
- provide a **common infrastructure** on top of OA for research and benchmarking;

OA Gear Components

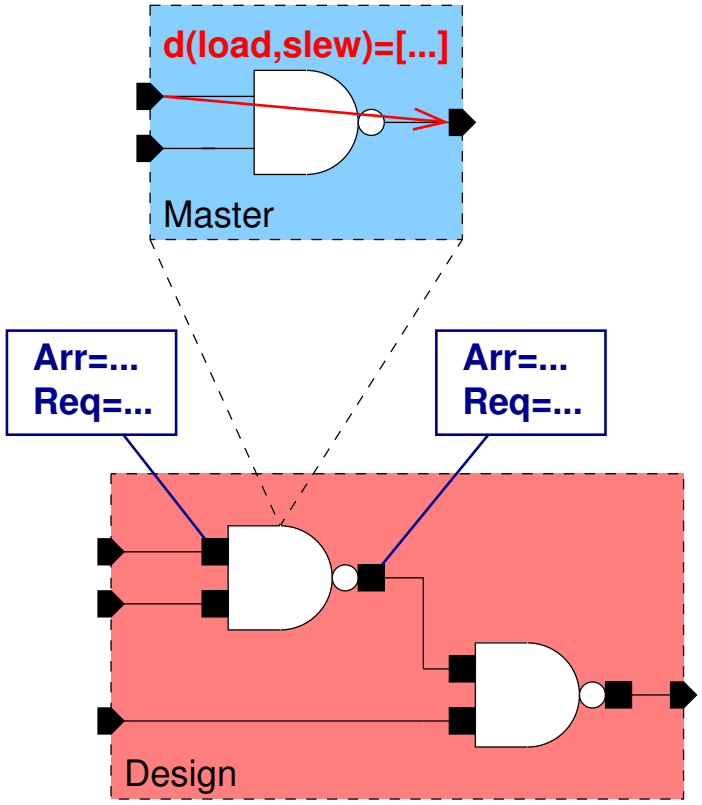
- Bazaar: extendable graphical user interface, with layout and schematic viewer, and scripting capabilities;
- Functional: infrastructure for logical functional representation, synthesis and verification;
- Timing: bare-bones static timing analysis engine;
- Placement: wrapper interface for Capo placement tool;
- Benchmarks: designs for academic research.

OA Gear Timing: Overview

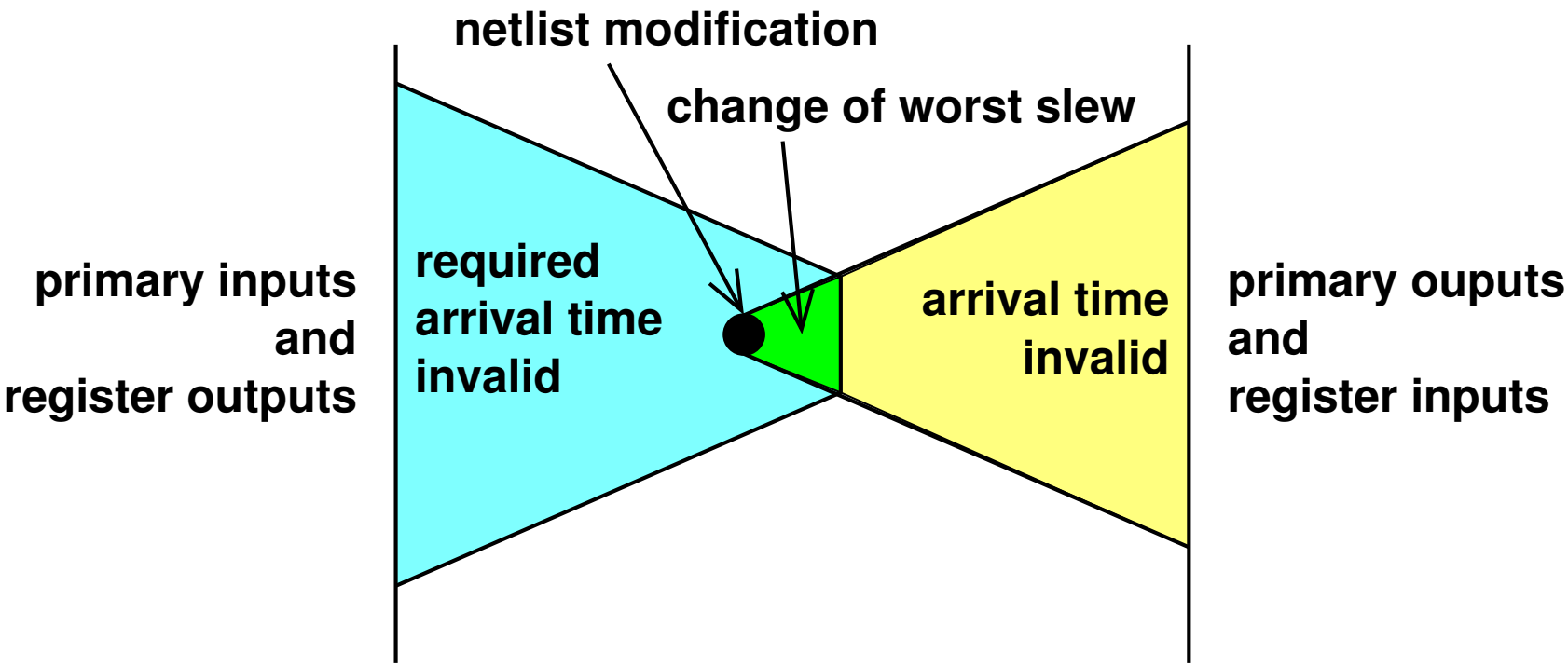
- built on OpenAccess for **integration** into other tools (e.g. placement);
- **incremental** (callback-based) timing analysis;
- wire models: **0-wire delay** and a linear delay model based on **bounding box** wire length estimate; can be easily extended to more accurate models;
- library formats: Synopsys **.lib**;
- timing constraints: subset of **.sdc** constraints,
  - clock period, input delay/drive strength, output load;
- standardized timing reports.

Internal Data Structures

- timing data stored using the OpenAccess extension mechanism (AppDefs);
- in **cell library**, *Term*s annotated with delay paths through each standard cell and corresponding 2-D lookup table models;
- in **design**, *InstTerm*s and *Term*s annotated with delay information (arrival/required times, slew, etc);



Incremental TA



Additions in the Last Release

- support for **multicycle paths**;
- support for **multiple clock domains**;
- operation in the Occurrence domain, introducing support for **hierarchical designs**;
- extended suite of **regression tests**;
- minor bug fixes and better memory management.

Upcoming Features

- Better Library Support;
- Simplified Timing Graph;
- Generic Timing Algebras (under consideration);
- Interconnect Delay Calculation;

Some of these feature are under implementation, while others are just planned. They may or may not be available in the next release of OA Gear.

Better Library Support

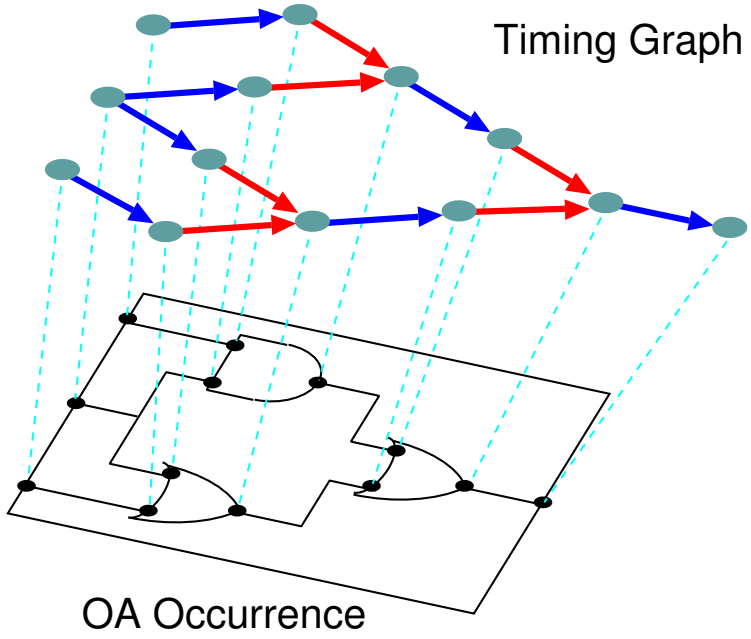
- **timing information** accessed through an easy-to-use **OO data model**, built on top of the TAP-in open source *.lib* parser, for robustness and maintainability;
- **timing constraints** are mapped to handler functions, that can be customized, through the Tcl interface and the TAP-in open source *.sdc* parser;

Simple Timing Graph

- some **academic users** want an abstraction layer on top of OA data structures;
- they just want a **simple way** to prototype their algorithms, test them on designs stored in OA databases, and possibly annotate back their results;
- in timing analysis they would rather have:
  - a **simplified annotated timing graph**, with all the timing information necessary for timing analysis;
  - an easy way to **traverse** the graph, in topological order;
  - an easy way to **read/write information** from/into the graph;

Simple Timing Graph

- view the circuit as a **timing graph**;
- a **Vertex** maps a *Term* or *instTerm*;
- an **Edge** maps a pin-to-pin delay in an *Inst* or *Net*; can have **interconnect** and **cell** edges;
- all the graph elements keep **references** to their original OA objects;



- any graph element can be **annotated with properties**, and some properties may be written back to the OA database.

Generic Timing Algebras

- typical timing quantities:
  - **delay**: *double*;
  - **time**: *double*;
- typical timing algebras:
  - **max**: *atY = (atA > atB) ? atA : atB;*
  - **min**: *atY = (atA < atB) ? atA : atB;*
  - **sum**: *atY = atX + d;*
  - **sub**: *atY = atX − d;*

Generic Timing Algebras

- **however**, the impact of variability on emergent deep submicron technologies pushed the development of new timing models, where these assumption may no longer be valid:
  - **timing quantities** can be sets of values, distributions, intervals, analytical functions, etc;
  - **timing algebras** can be arbitrarily complex functions, necessary to adequately manipulate the timing quantities;
- **nontheless**, the overall procedure for timing analysis remains:
  - perform **forward traversal** to compute arrival times;
  - perform **backward traversal** to compute slacks;
  - check for **violations**;

Generic Timing Algebras

cadence

- need to **decouple** the timing analysis procedure from the timing quantities and timing algebras used;
- convert delay and time quantities, previously typed *double*, to the pure virtual classes or template arguments:
  - *Delay*;
  - *Time*;
- any derived class must implement a **minimum set of operators**:
  - $+$, $-$, *max*, *min*;

Generic Timing Algebras

cadence

- two ways to ”plug-in” the appropriate classes for delay and time:
  - **templating**: the timer is templated and *Delay* and *Time* are arguments of the template (more performance, less versatility);
  - **dynamic loading**: derived classes from *Delay* and *Time* are dynamically loaded into the timer, as plugins (less performance, more versatility);
- the overall timing analysis procedure may also be **customized**, to a certain extent, by deriving a new class from *SimpleTimer*, and overriding the appropriate methods;

Interconnect Delay Calculator

- **parasitic information** can be already available in a design or can be loaded using *spef2oa*;
- an OA Gear **Extractor** is under development;
- using this information it is possible to compute **accurate interconnect delay estimates**;
- implement an **interconnect delay calculator**, to be invoked by the timer that, for a given net, loads the parasitic network from the OA database and computes pin-to-pin interconnect delays and output slews, given the input slews and operating conditions.

Future Work

- add support for **transparent latches**;
- add support for **new library types** that support more advanced characterizations, compatible with more complex timing quantities and algebras;
- improve the support for **Tcl scripting**.

OA Gear Website

http://openedatools.si2.org/oagear

Thank You !