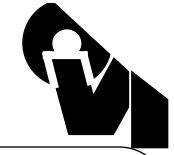
# ALF - Advanced Library Format for ASIC Cells, Blocks and Cores containing Timing, Power, Functional and Physical Information for Synthesis, Analysis and Test



# Outline

# Motivation

**Fast Introduction of new Technologies requires Efficient Library Generation**

Time to Market
Efficient Use of Resources
Cost Reduction
Automation
Quality Assignment

**Cost of ASIC Library Development**

Resources e.g. 60 people

Time e.g. 4 months

20 person years -> 50% productivity increase saves 10 person years

Freed-up resources can help to make your next product
even more competitive or bring it faster to market!

**Extensive Collaboration between EDA and ASIC Vendors necessary for Success**

Leverage for New EDA Tools

Fair Comparison Between Tools

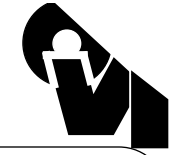
**The ALF standard can make life easier for everybody!**

# ALF Goals

* Generality
    - Addresses Needs for Generic Design Tools
    - Expandable for Advanced Modeling Capabilities
* Simplicity
    - Self-Explaining Format
    - Easy to Generate and to Parse
* Efficiency
    - Compact Representation of Pertinent Information
* Acceptance
    - Backward Compatibility
    - Added Value
* Flexibility
    - Does not Dictate Particular Modeling Style

# The ALF Story

| | |
|---|---|
| ALF work group started as OVI PS-TSC | 6/96 |
| ALF named and announced at IVC/VIUF | 3/97 |
| ALF integrated in ECSI OMILIBRES project | 9/97 |
| ALF1.0 approved by OVI | 12/97 |
| OLA project for integration of ALF and DCL approved by ASIC council and OVI | 2/98 |
| OLA demonstrator from ASIC council / SI2 | 6/98 (DAC) |
| ALF libraries from ASIC council companies | prototypes 4/98<br>production 9/98 |
| Commitment for ALF support from EDA Ambit, Avant!, Cadence, Mentor Graphics | 2/98 |
| ALF work group preparing ALF1.1 | ongoing |



# The ALF Story (cont.)

Companies currently active in ALF work group, coordinated by OVI

Ambit, Avant!, Cadence, Cadworx, Duet, Fujitsu, LSI Logic, Mentor Graphics, Motorola, NEC, Toshiba, VLSI Technologies

Companies involved in OLA (ALF-DCL integration project), coordinated by SI2

Ambit, Avant!, Cadence, Cadworx, Duet, IBM, LSI Logic, Lucent Technologies, Mentor Graphics, Motorola, NEC, Synopsys, TI, VLSI Technologies

Other EDA vendors interested in ALF

ASC, Sente Inc., Tera Systems, Verysys

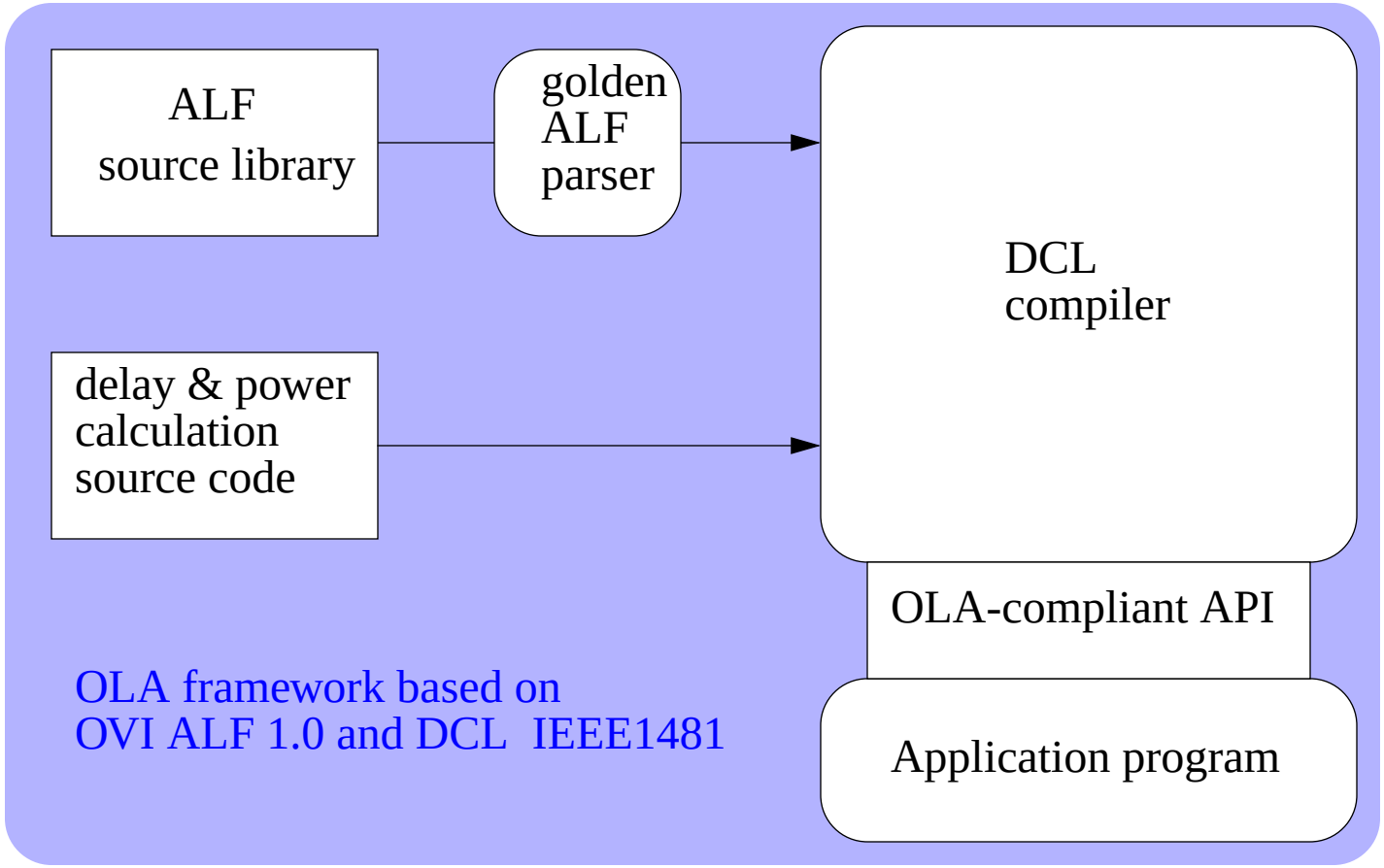
Other ASIC and System vendors interested in ALF

Alcatel, I&D Telefonica, Siemens, Sun Microsystems

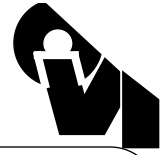
Other standardization bodies  interested in ALF

ECSI, VSIA

# Relation between ALF, DCL, OLA




*OLA framework development sponsored by ASIC council*



# Relation between ALF, DCL, OLA (cont.)

ALF (Advanced Library Format)
is the source for cell characterization data

Golden parser preprocesses ALF source for
correct syntax and semantics

DCL (Delay Calculation Language)
is the source for delay & power calculation algorithms

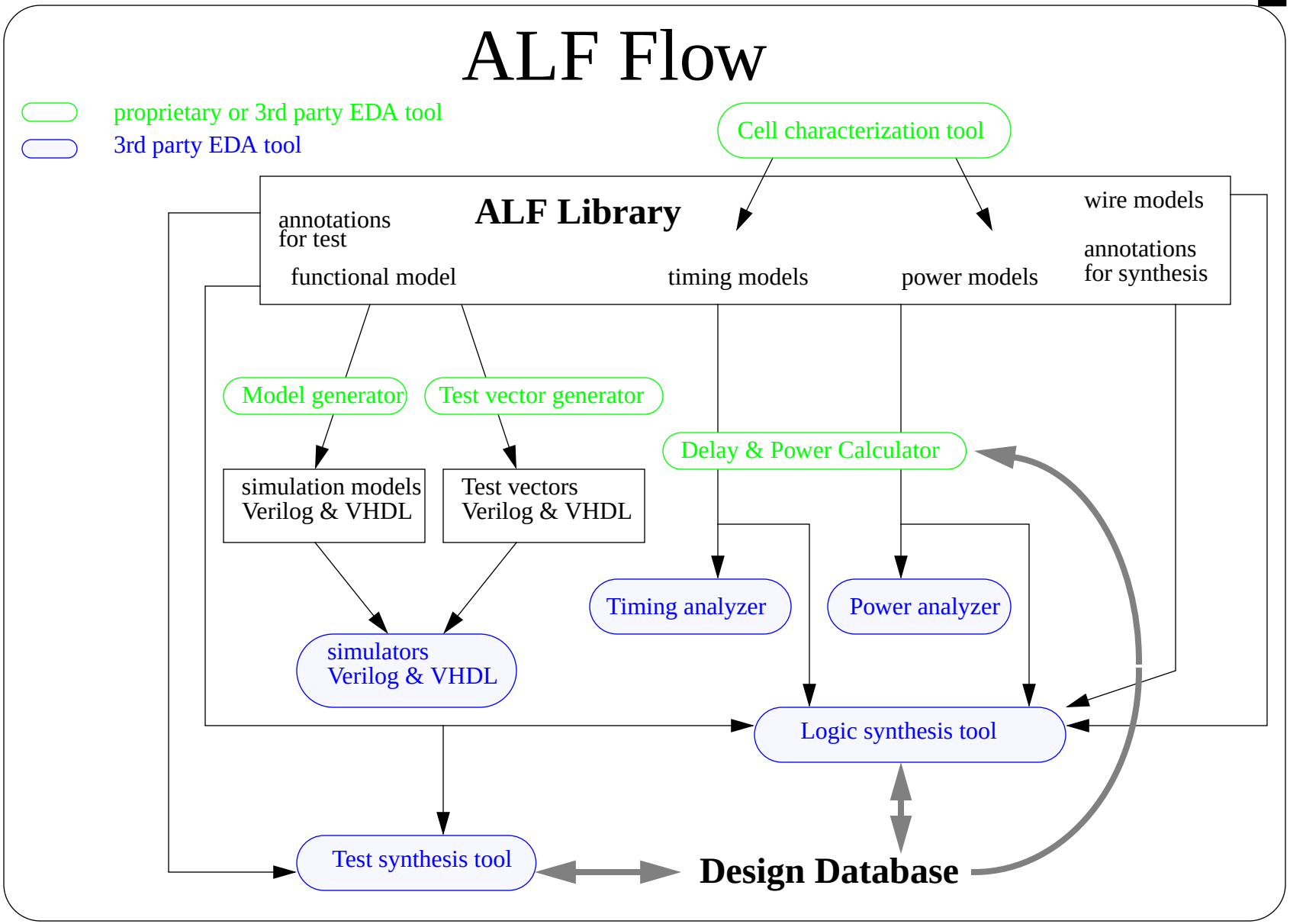
Delay & power calculation requires both *library* and *design* data as input
Results of calculation are required by application

DCL compiler digests both ALF and DCL source and transforms them
into optimized code for performance and memory usage

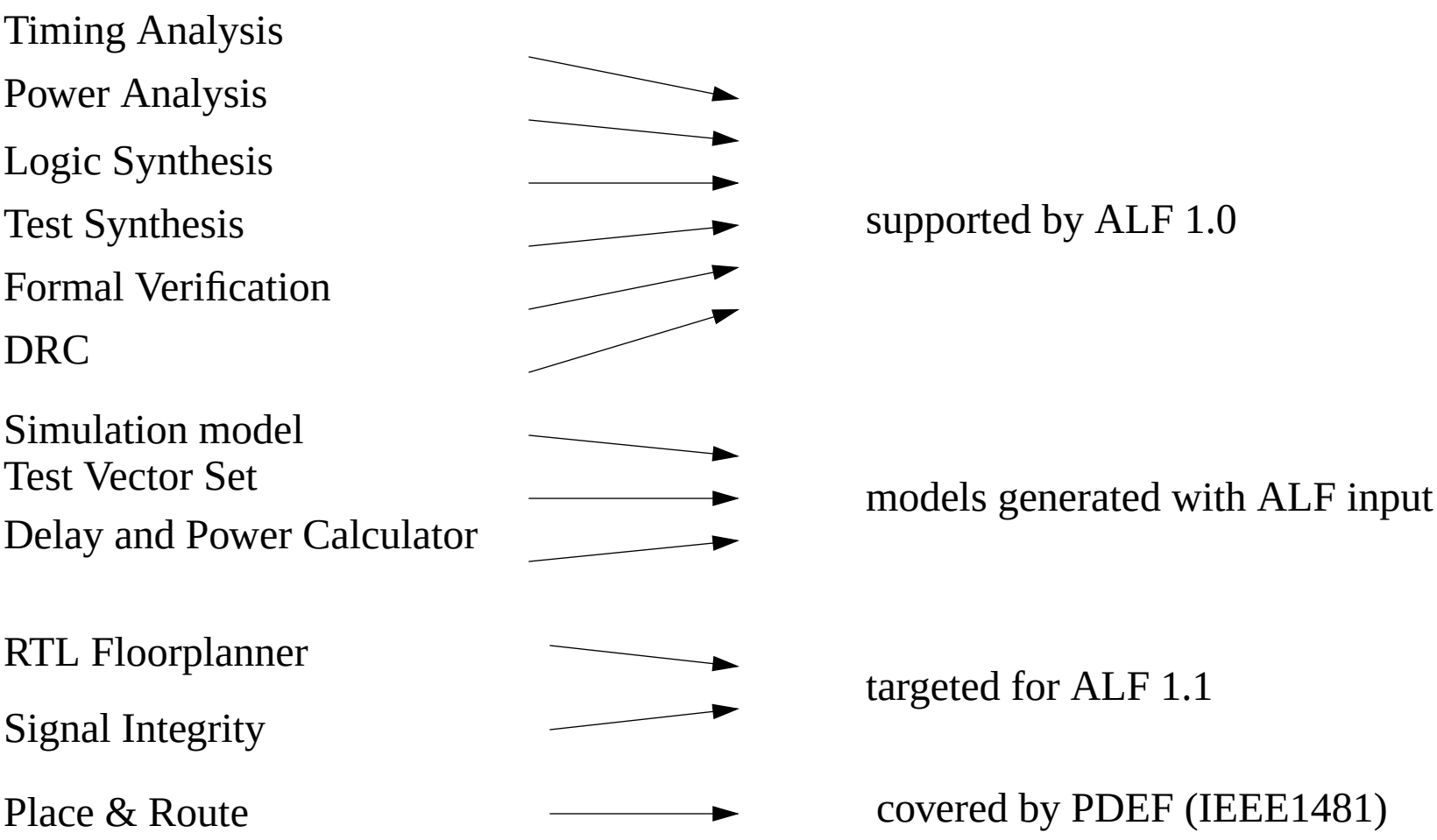
OLA (Open Library API) is the interface between ALF, DCL
and the application program

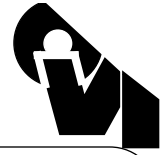
OLA makes delay & power data and calculation algorithms transparent
ALF and DCL sources cannot be retrieved technically

Functional models and annotations are accessible through OLA
ALF source can be retrieved
Source access protection can be embedded in API


ALF Flow
proprietary or 3rd party EDA tool
3rd party EDA tool
Cell characterization tool
ALF Library
annotations for test
functional model
timing models
power models
wire models
annotations for synthesis
Model generator
Test vector generator
Delay & Power Calculator
simulation models Verilog & VHDL
Test vectors Verilog & VHDL
Timing analyzer
Power analyzer
simulators Verilog & VHDL
Logic synthesis tool
Test synthesis tool
Design Database

# Target Applications





# ALF Features

**Modeling Scope**

Combinational and sequential cells
IO cells
Datapath: counters, adders, multipliers, comparators ...
Memories
Cores

**Generic Functional Model**

Canonical functional specification language
Supports logic expressions, event expressions, state and transition tables
Predefined and user-defined primitives
Hierarchical modeling, emulation of netlists

**Generic Model for CharacterizationData of Cells and Blocks**

Wide range of variables, including timing and power
Equation or Table Based Models
Vector-based modeling encompassed timing arcs, state-dependency and more

**Generic Physical Model**

Annotation of Physical Cell Properties for Logic and Test Synthesis
Equation or Table Based Wire Models
Support for DRC

# Outline

Introduction

- Motivation and Goals
- The ALF Story
- Relation between ALF, DCL, OLA
- ALF Flow, Target Applications and Features

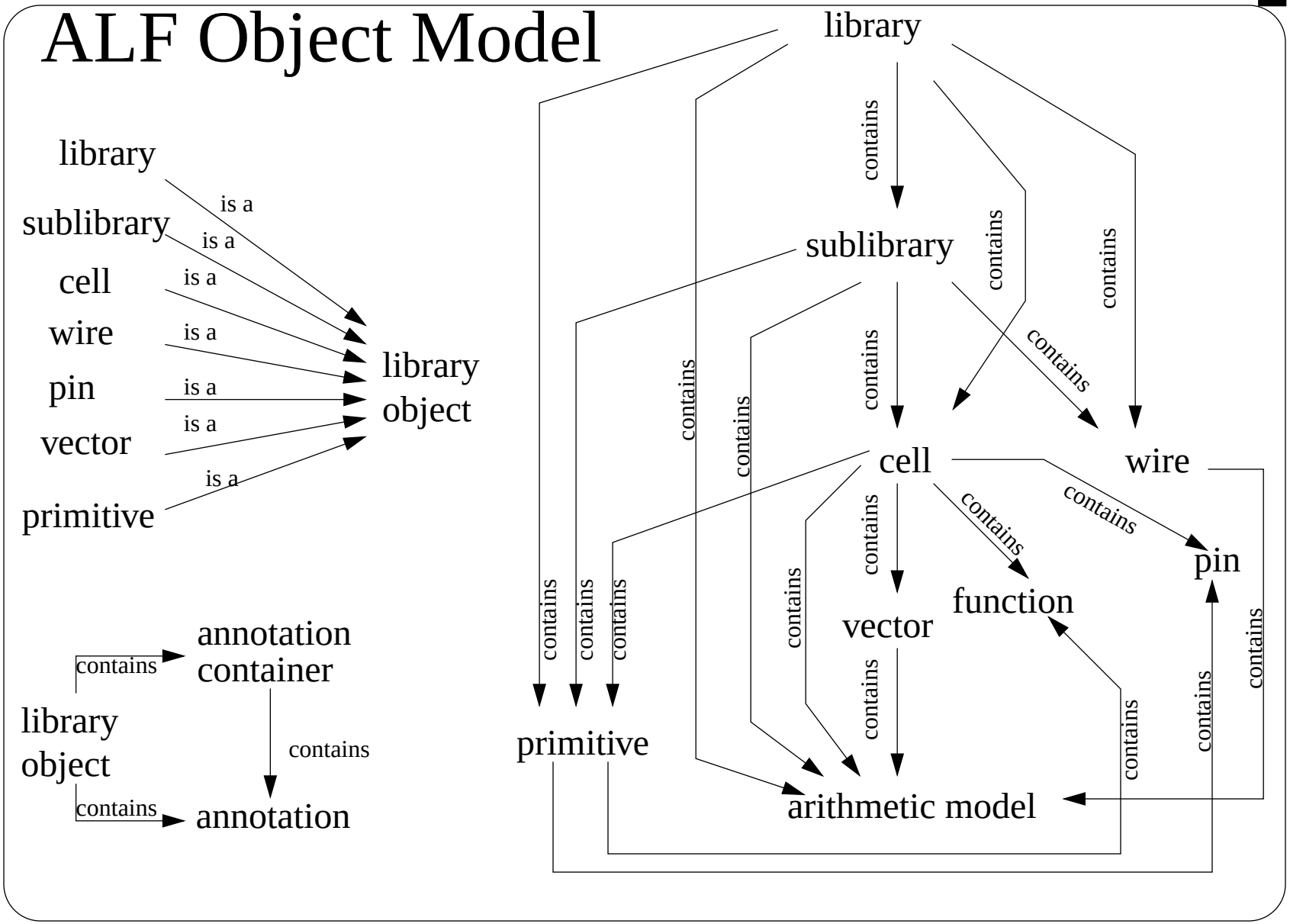
→ ALF Language

- Object Model and Semantic Features
- Meta Syntax and Scoping Rules
- General Purpose ALF Objects

ALF Applications

- Functional Modeling
- Timing and Power Modeling
- Modeling for Synthesis, Test and DRC
- Megacell and Core Modeling

Conclusion


ALF Object Model
library
sublibrary
cell
wire
pin
vector
primitive
is a
library object
annotation container
contains
library object
annotation
library
sublibrary
cell
wire
pin
vector
function
primitive
arithmetic model
contains

# A Simple Self-Explaining Example

```
LIBRARY sample_library {

// written by ALFRED CLEARMAN

     CAPACITANCE {UNIT = pF}
     SLEWRATE {UNIT = ns}
     DELAY {UNIT = ns}
     ENERGY {UNIT = pJ}

     CELL nand2 {
     // fill in cell information
     }

     CELL d_flipflop {
     // fill in cell information
     }

}
```

```
CELL nand2 {
     PIN a {
          DIRECTION = input
          CAPACITANCE = 20 {UNIT = fF}
     }
     PIN b {
          DIRECTION = input
          CAPACITANCE = 20 {UNIT = fF}
     }
     PIN z {
          DIRECTION = output
     }
     FUNCTION {
          BEHAVIOR {
               z = !(a && b);
          }
     }
     VECTOR (10 a -> 01 z){
     // fill in characterization data
     }
     VECTOR (01 a -> 10 z){
     // fill in characterization data
     }
     VECTOR (10 b -> 01 z){
     // fill in characterization data
     }
     VECTOR (01 b -> 10 z){
     // fill in characterization data
     }
}
```

# Semantic Features of ALF

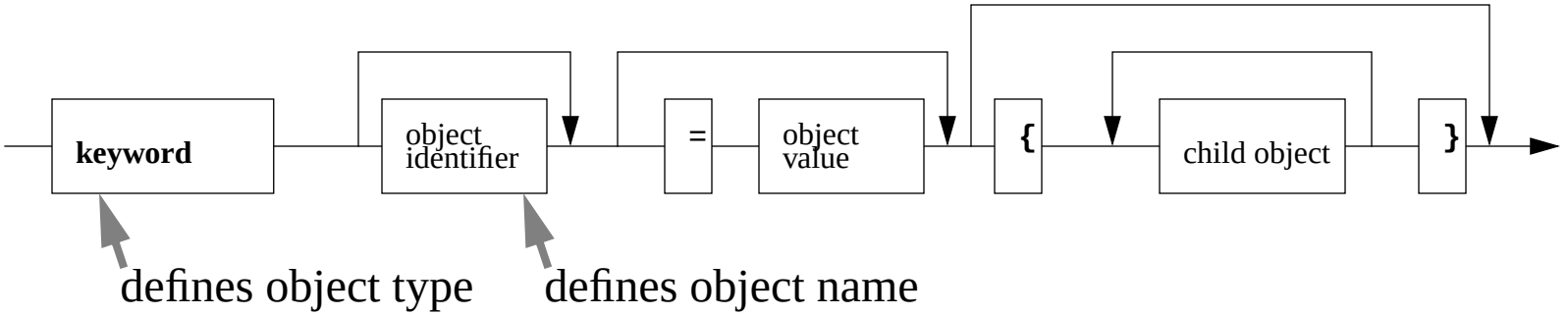
* Each object type is recognized by a keyword

* general purpose objects use hard keywords

* library objects, especially arithmetic models, use context-specific keywords
  - expandable modeling space for various applications
  - customized modeling possible

ALF is case-insensitive
However, keywords are written in upper case in all examples for clarity

# Meta Syntax




Any parser must be at least capable to identify begin and end of library objects

* unnamed object without value assignment

```
KEYWORD { /* fill in children objects */ }
```

* unnamed object with value assignment

```
KEYWORD = value { /* fill in children objects */ }
```

* named object without value assignment

```
KEYWORD object_id { /* fill in children objects */ }
```

* named object with value assignment

```
KEYWORD object_id = value { /* fill in children objects */ }
```

# Scoping Rules

* curly parenthesis { } change the scope

* children objects are not visible outside their scope

* parent objects propagate their visibility through their children objects

* no more than one unnamed object of the same type allowed within the same scope

* no unnamed object after a named object of the same type in the same scope (unnamed object, if any, must come first)

* unlimited number of named objects of same type allowed within the same scope

* named objects inherit properties of nearest visible unnamed object

* any inherited property can be overwritten by the object itself

# Scoping Rules (cont.)

Example:

```
LIBRARY my_library {
     CAPACITANCE { unit = 1e-12 }
     CELL cell1 {
          CAPACITANCE { unit = 1e-13}
          CAPACITANCE c1 = 5
     }
     CELL cell2 {
          CAPACITANCE = 7
          CAPACITANCE c2
     }
     CAPACITANCE c3 = 9
}
```

scope = my_library

scope = my_library.cell1

scope = my_library.cell2

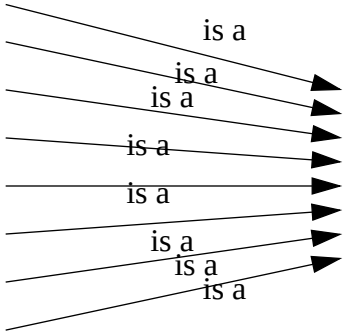
scope = my_library

Question: What are the units and values of c1, c2, and c3 ?

Answer: c1 = 5e-13  c2 = 7e-12  c3 = 9e-12

# General Purpose ALF Objects

alias
constant
include
class
attribute
template
property
group

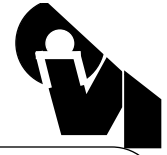



general purpose object

* each library object may contain general purpose objects as well

* general purpose objects may not contain arbitrary general purpose objects

* the meaning of a general purpose object must be understood by any parser, whereas some tool-specific parsers may skip particular library objects (e.g. scan insertion tool does not need power consumption information)



# General Purpose ALF Objects (cont.)

* alias: useful for customized renaming of context-sensitive keywords

```
ALIAS RAMPTIME = SLEWRATE
```

* constant: useful for constant numbers

```
CONSTANT vdd = 3.3
```

* include: inclusion of external ALF file

```
INCLUDE 'primitives.alf'
```

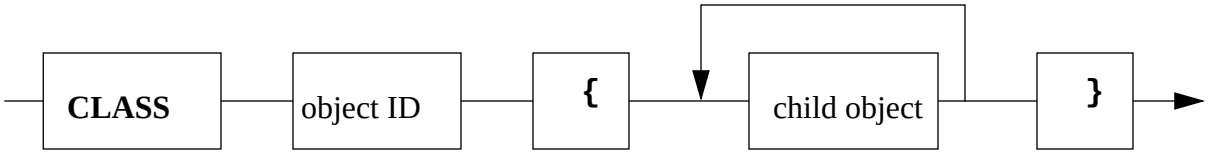
* attribute: association of arbitrary unordered attributes

```
CELL rom_8x128 {
     ATTRIBUTE {ROM ASYNCHRONOUS STATIC}
}
```

* property: useful for arbitrary parameter-value assignment

```
PROPERTY items {
     parameter1 = value1 ;
     parameter2 = value2 ;
}
```

# Class Object

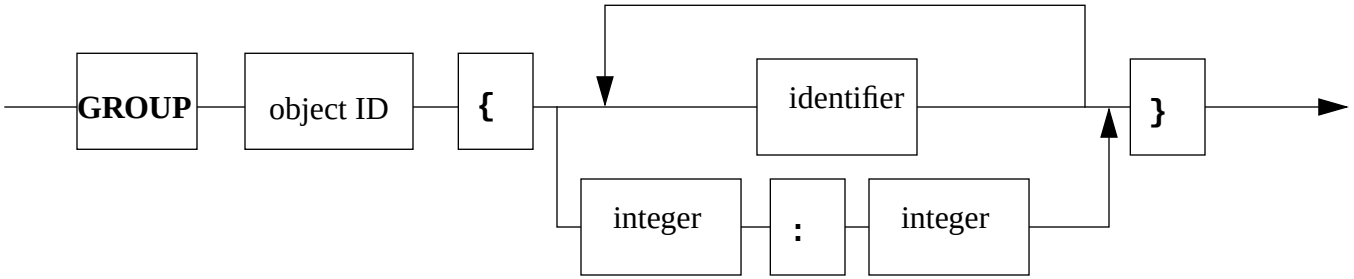



* Reference to a class inside an object allows inheritance of arbitrary properties
* One object can inherit from more than one class

Example:

```
CLASS io_pin_cap {
     UNIT = pF
     MEASUREMENT = average
}
CLASS internal_pin_cap {
     UNIT = fF
     MEASUREMENT = average
}

CELL nand2 {
     PIN a {
          DIRECTION = input
          CAPACITANCE = 20 {CLASS = internal_pin_cap}
     }
     ...
}
```

# Group Object




* purpose: expand one definition to multiple definitions

Example:

```
GROUP timing_measurements { DELAY SLEWRATE }
...
timing_measurements { UNIT = ns }
```

← group definition

← group instantiation

group instantiation replaces the following individual instantiations

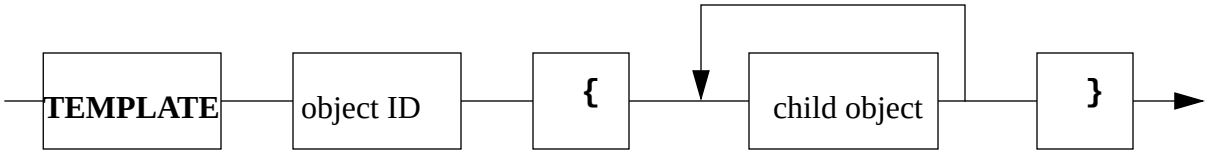
```
DELAY { UNIT = ns }
SLEWRATE { UNIT = ns }
```

* Can also be used for definitions within a range of numbers (useful for bus expansion)

Example:

```
GROUP bit { 0 : 3 }

VECTOR (a[bit] -> z[bit]) { ... }
// equivalent to 4 timing arc instantiations
```

# Template Object




* Inside the parenthesis, any literal enclosed by < and > is recognized as a placeholder

* purpose: make format more compact for repeated information
  define customized objects

* template instantiation uses the objectID

* reference to placeholders can be explicit (by annotation) or implicit (by order)

Example:

```
TEMPLATE input_pin {
     PIN <name> {
          DIRECTION = input
          CAPACITANCE = <cap> {UNIT = fF}
     }
}
CELL nand2 {
     input_pin { name = a; cap = 20; }
     input_pin { name = b; cap = 20; }
     ...
}
```

# Expressions

* Boolean expressions and vector expressions are used in functions and vectors

* Arithmetic expressions are used in arithmetic models

* Expressions can be nested

* Set of operators for arithmetic and boolean expressions adopted from IEEE standards

* Extended set of operators for vector expressions

* Standard priority rules for operators

# Outline

Introduction

- Motivation and Goals
- The ALF Story
- Relation between ALF, DCL, OLA
- ALF Flow, Target Applications and Features

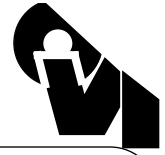
ALF Language

- Object Model and Semantic Features
- Meta Syntax and Scoping Rules
- General Purpose ALF Objects

→ ALF Applications

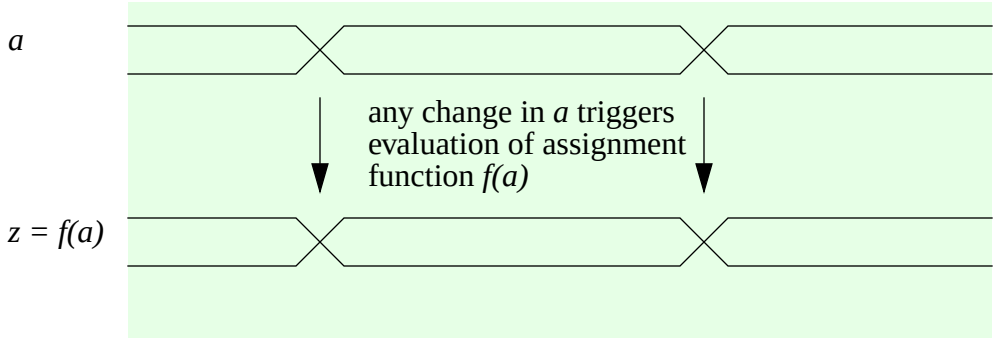
- Functional Modeling
- Timing and Power Modeling
- Modeling for Synthesis, Test and DRC
- Megacell and Core Modeling

Conclusion



# Function: Combinational Logic

* continuous assignments
  modeled with
  boolean expressions




* ALF uses the 9 value system from IEEE standard:
  - strong and weak 1, 0, X
  - highZ, don’t care and uninitialized

* ALF uses continuous assignment language from VERILOG standard:
  logical and, or, exor, if-then-else, unsigned and signed arithmetic ...
  unary, binary and bitwise operators ...

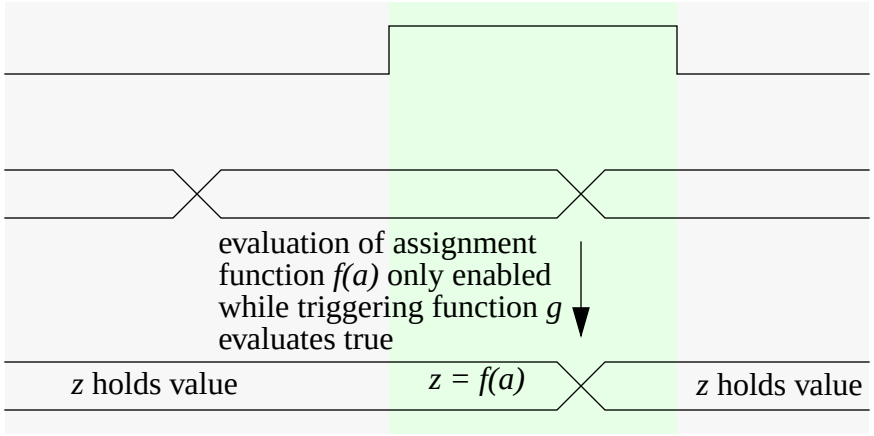
Examples:

```
z = a && b || c;                                          //and-or cell

q = (!s0 && !s1) ? d0 : (s0 && !s1) ? d1 : s1 ? d2 : ‘bx ;      //3-way mux

p[15:0] = a[7:0] * b[7:0];                              // 8x8 multiplier
```

# Level-Sensitive Sequential Logic

* conditional continuous assignments

* both assignment and condition modeled with boolean expressions




* Syntax: @ (*boolean expression*) { *continuous assignment* }

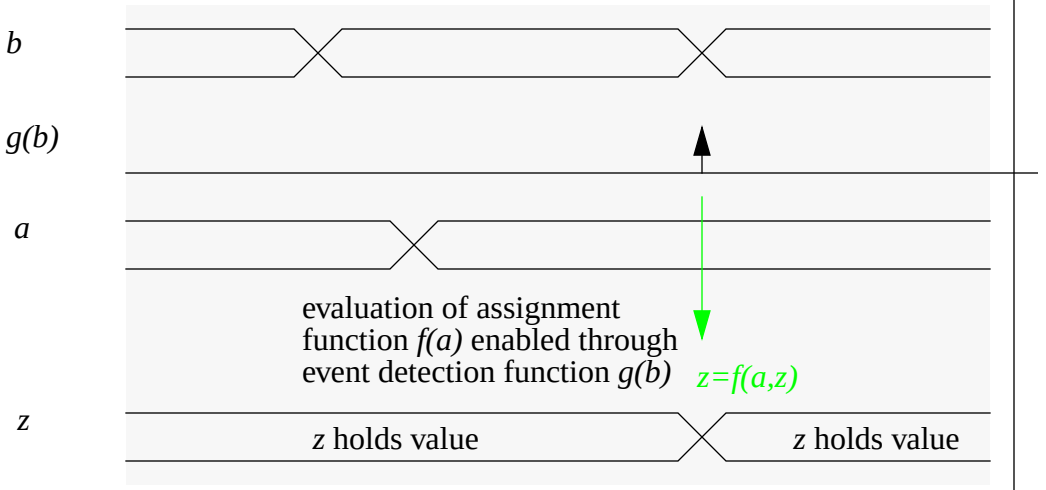
Example:

```
//latch with input scan mux and asynchronous clear

@ ( enb && !clear) {
     q = scan_enb ? test_in : data_in ;
}
@ (clear) {
     q = 0 ;
}
```

# Edge-Sensitive Sequential Logic

instantaneous assignment modeled with boolean expression

condition for assignment modeled with vector expression featuring edge operators




Syntax: @ (*vector expression*) { *continuous assignment* }

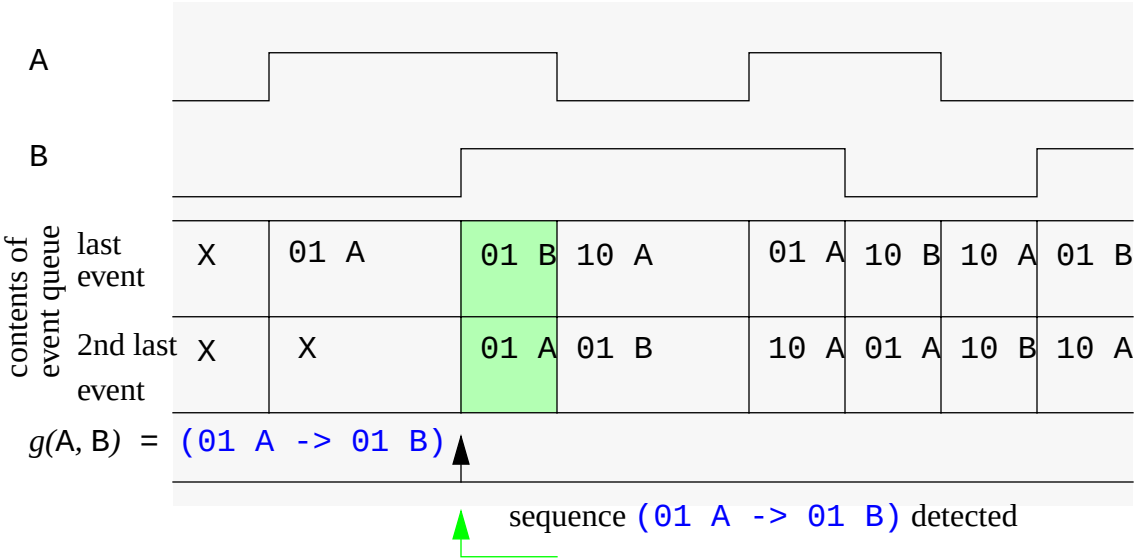
Example:

```
//positive-edge triggered flipflop with asynchronous clear

@ ( 01 cp && !clear) {
     q = d ;
}
@ (clear) {
     q = 0 ;
}
```

# General Event-Sensitive Logic

* General vector expression describes sequence of transitions

* vector expression evaluates true, when sequence is detected




Example:

```
//Mueller-C gate: output changes after two input events

@ ( ( 01 A || 01 B || 10 A || 10 B ) && event_count == 0 ) {
     event_count = 1 ;
}
@ ( 01 event_count -> ( 01 A || 01 B || 10 A || 10 B ) ) {
     event_count = 0 ;
     Q = !Q ; // now the output changes
}
```

# Choice of Functional Modeling Styles

logic behavior

logic statetable

primitive instantiation

hybrid modeling style is legal

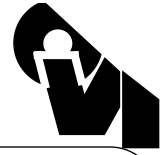
* a primitive is a user-defineable object, which has pins and function like a cell, but no arithmetic model

* modeler may define arbitrary primitives at different levels of scope for hierarchical netlist-style modeling

* ALF provides a set of predefined primitives, starting with ALF_ prefix

  ALF_BUF, ALF_NOT
  ALF_AND, ALF_OR, ALF_XOR, ALF_NAND, ALF_NOR, ALF_NXOR
  ALF_BUFIF1, ALF_BUFIF0, ALF_NOTIF1, ALF_NOTIF0
  ALF_MUX, ALF_LATCH, ALF_FLIPFLOP

* Names starting with ALF_ are reserved for future predefined primitives



# Functional Modeling Styles: Example

**NAND cell**

behavioral equation

```
BEHAVIOR {
     z = !(a && b);
}
```

state table

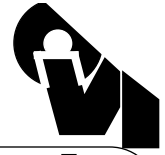
```
STATETABLE {
     a  b  : z ;
     0  ?  : 1 ;
     1  ?  : (!b);
}
```

primitive instantiation
pin reference by name

```
BEHAVIOR {
     ALF_NAND {
          out = z;
          in[0] = a;
          in[1] = b;
     }
}
```

primitive instantiation
pin reference by order

```
BEHAVIOR {
     ALF_NAND { z a b }
}
```



# Functional Modeling Styles: Example

**flipflop with asynchronous set and clear**

behavioral equation

```
BEHAVIOR {
     @(01 cp && cd && sd) {q = d;}
     @(!sd && cd) {q = 1;}
     @(!cd) {q = 0;}
}
```

concurrent style

```
BEHAVIOR {
     @(!cd) {q = 0;}
     :(!sd) {q = 1;}
     :(01 cp) {q = d;}
}
```

if-then-else style

state table

```
STATETABLE {
     cd sd cp d : q ;
     0  ?  ?? ? : 0 ;
     1  0  ?? ? : 1 ;
     1  1  1? ? :(q);
     1  1  ?0 ? :(q);
     1  1  01 ? :(d);
}
```

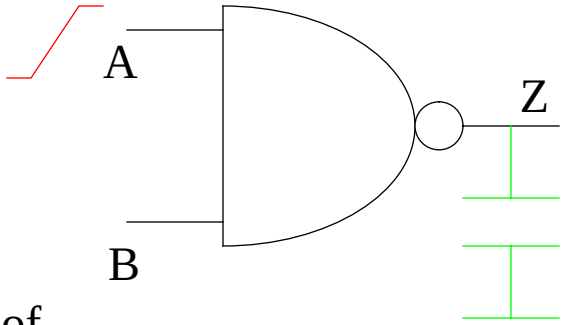
primitive instantiation
pin reference by name

```
BEHAVIOR {
     ALF_FLIPFLOP {
          Q = q ;
          D = d ;
          CLOCK = cp;
          CLEAR = !cd ;
          SET = !sd ;
     }
}
```

# Timing and Power Modeling

**NAND cell:**

Each operational mode of the cell, e.g. timing arc (01 A -> 10 Z)
is described by a vector expression



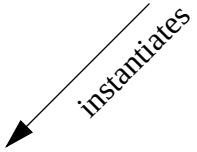

* delay and power for each timing arc are functions of
  slewrate @ pin A
  load capacitance @ pin Z

* other possible dimensions: Vdd, Temperature ...

* multi-dimensional tables or equations
  or combination of both

# Example

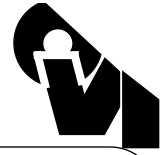
Template for Standard Characterization Table




Template for Standard Characterization Table Header

```
TEMPLATE std_header_2d {
     HEADER {
          CAPACITANCE {
               PIN = <out_pin>
               UNIT = pf
               TABLE
               {.01 .02 .04 .08 .16}
          }
          SLEWRATE {
               PIN = <in_pin>
               UNIT = ns
               FROM {THRESHOLD = 0.3}
               TO {THRESHOLD = 0.5}
               TABLE {.1 .3 .9}
          }
```

```
TEMPLATE std_char_2d {
     DELAY {
          UNIT = ns
          FROM{PIN = <in_pin>
               THRESHOLD = 0.4 }
          TO{  PIN = <out_pin>
               THRESHOLD = 0.6 }
          std_header_2d {
               in_pin = <in_pin>
               out_pin = <out_pin>
          }
          TABLE <delay_data>
     }SLEWRATE {
          UNIT = ns
          PIN = <out_pin>
          FROM {THRESHOLD = 0.3}
          TO {THRESHOLD = 0.5}
          std_header_2d {
               in_pin = <in_pin>
               out_pin = <out_pin>
          }
          TABLE <ramptime_data>
     }ENERGY {
          UNIT = pj
          std_header_2d {
               in_pin = <in_pin>
               out_pin = <out_pin>
          }
          TABLE <energy_data>
     }
}
```



# Example (cont.)

Use of template
in delay & power table
for NAND gate

```
std_char_2d {
     in_pin = a
     out_pin = z
     delay_data {
               0.1 0.2 0.4 0.8 1.6
               0.2 0.3 0.5 0.9 1.7
               0.4 0.5 0.7 1.1 1.9
     }
     ramptime_data {
               0.1 0.2 0.4 0.8 1.6
               0.1 0.2 0.4 0.8 1.6
               0.2 0.4 0.6 1.0 1.8
     }
     energy_data {
               0.21 0.32 0.64 0.98 1.96
               0.22 0.33 0.65 0.99 1.97
               0.31 0.42 0.74 1.08 2.06
     }
}
```

```
CELL nand2 {
     PIN a {
          DIRECTION = input
          CAPACITANCE = 0.02 {UNIT = pf}
     }
     PIN b {
          DIRECTION = input
          CAPACITANCE = 0.02 {UNIT = pf}
     }
     PIN z {
          DIRECTION = output
     }
     FUNCTION {
          BEHAVIOR {
               z = !(a && b);
          }
     }
     VECTOR (10 a -> 01 z){
          std_char_2d { ... }
     }
     VECTOR (01 a -> 10 z){
          std_char_2d { ... }
     }
     VECTOR (10 b -> 01 z){
          std_char_2d { ... }
     }
     VECTOR (01 b -> 10 z){
          std_char_2d { ... }
     }
}
```

# Rules for Table Format

* Table format is defined by the order of appearance of the arguments in the header
* Each argument must have a one- dimensional table itself
* First argument defines innermost index
* Target table contains  just data separated by whitespace

Example:

```
DELAY {
          HEADER {
               CAPACITANCE { TABLE { 0.5 1.0 2.0} }
               SLEWRATE { TABLE { 0.3 0.9 } }
               VOLTAGE { TABLE { 3.0 3.3 } }
          }
          TABLE {
               0.3  0.5  0.8
               0.6  1.0  1.4
               0.2  0.4  0.7
               0.5  0.8  1.2
          }
}
```

Question:  What is the delay for capacitance=1.0, slewrate=0.9, voltage=3.0 ?

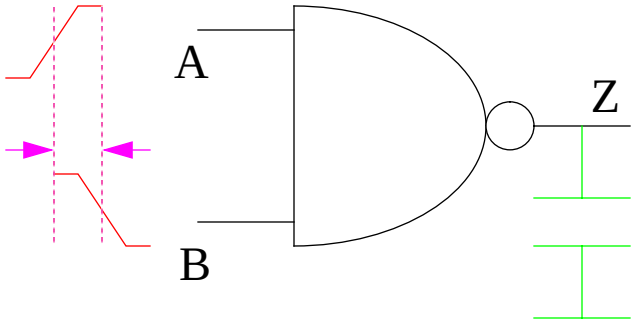
Answer:  delay = 1.0 (5th index)

# Glitch Power Modeling

**NAND cell:**

* glitch power for (01 A -> 10 B)

function of




slewrate @ pin A

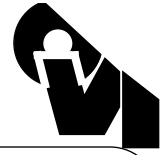
slewrate @ pin B

load capacitance @ pin Z

skew between transitions @ A and B

* 4-dimensional table not practical

-> use equations or combination of tables and equations



# Example

model glitch energy for a specific vector as a function of

- slewrate on each pin
- skew between input signals
- energy for each input transition which itself is a function of
  - slewrate on each pin

energy for each input is modeled as table

energy for resulting glitch is modeled as equation

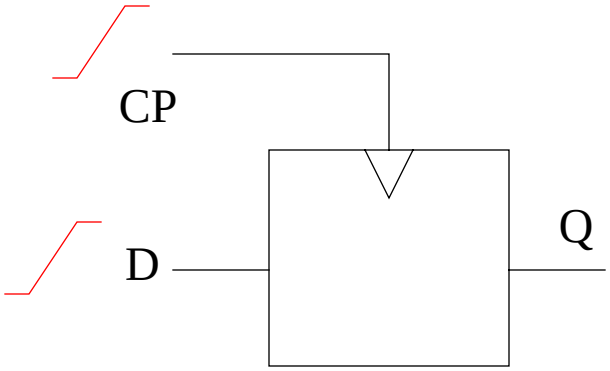
```
VECTOR (01 a -> 10 b) {
     ENERGY {
          HEADER {
               SLEWRATE s_a { PIN = a }
               SLEWRATE s_b { PIN = b }
               DELAY dt {
                    FROM { PIN = a }
                    TO   { PIN = b }
               }
               ENERGY e_a {
                    HEADER {
                         SLEWRATE {
                              PIN = a
                              TABLE {0.1  0.4  1.6}
                         }
                    TABLE {0.21 0.48 1.85}
               }
               ENERGY e_b {
                    HEADER {
                         SLEWRATE {
                              PIN = b
                              TABLE {0.1  0.4  1.6}
                         }
                    TABLE {0.23 0.46 1.79}
               }
          }
          EQUATION {
               (e_a + e_b) * dt / (s_a + s_b)
          }
     }
}
```

# Timing Constraint Modeling

**flipflop:**

* Timing constraint arc (01 D <&> 01 CP)

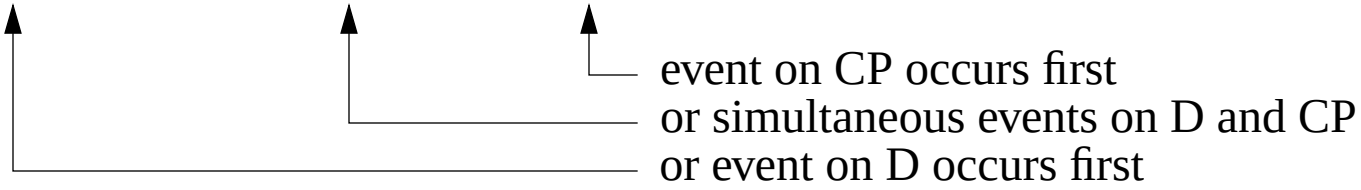
* setup time and hold time are function of
  - slewrate @ pin D
  - slewrate @ pin CP




(01 D <&> 01 CP)

is a shorthand expression for

( (01 D -> 01 CP) || (01 D & 01 CP) || (01 CP -> 01 D) )



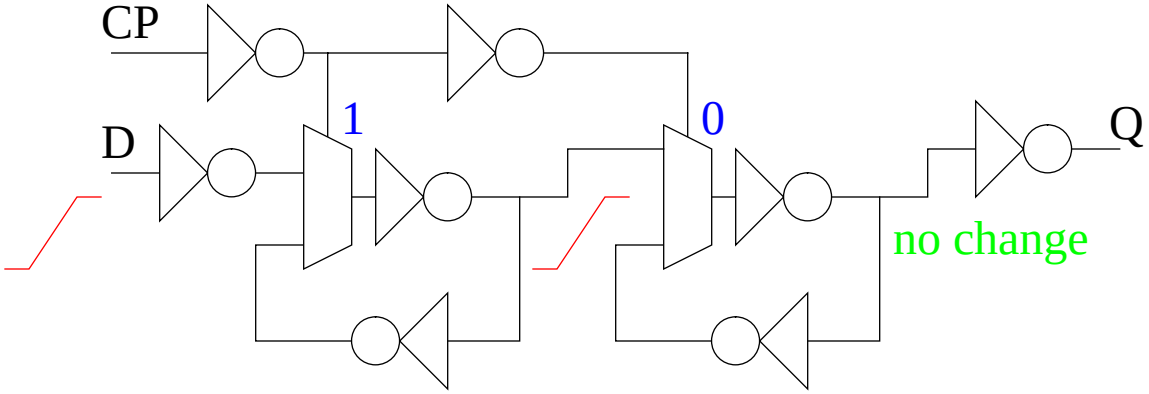
# Example

```
VECTOR (01 d <&> 01 cp)
     SETUP {
          FROM { PIN = d;  THRESHOLD = 0.35; }
          TO   { PIN = cp; THRESHOLD = 0.35; }
          HEADER {
               SLEWRATE s_d {
                    PIN = d
                    TABLE { 0.2 0.4 0.8 }
               }
               SLEWRATE s_cp {
                    PIN = cp
                    TABLE { 0.2 0.4 0.8 }
               }
          }
          TABLE {
                0.1 0.4 0.9
               -0.2 0.1 0.6
               -0.3 0.0 0.4
          }
          VIOLATION {
               BEHAVIOR { q = ‘bx; }
               MESSAGE_TYPE = error
               MESSAGE = “setup violation rising d to rising cp“
          }
     }
}
```

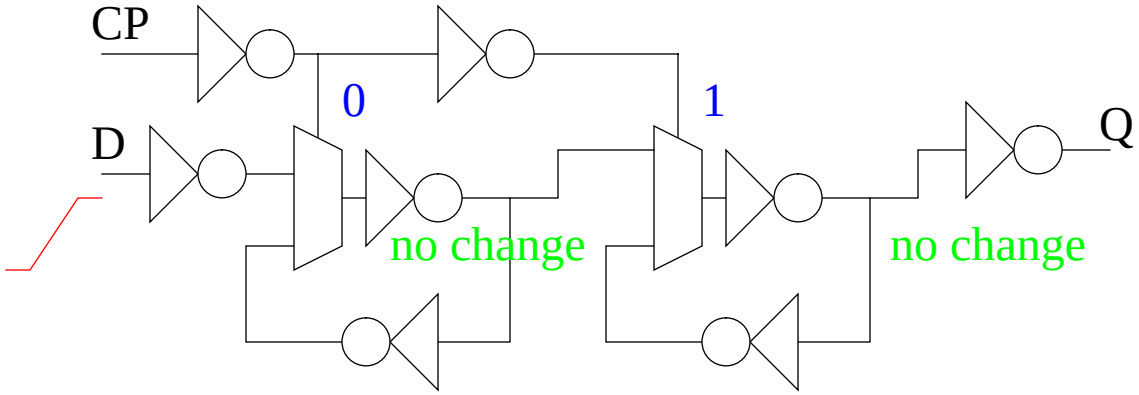
# State-dependent Power Modeling

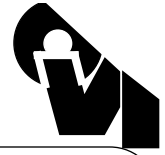
**flipflop:** Internal switching power is a state-dependent function of slewrate @ pin D

(01 D && !CP)



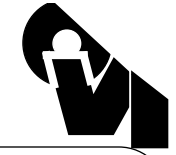

(01 D && CP)





# Example

```
VECTOR (01 d && !cp) {
     ENERGY {
          HEADER {
               SLEWRATE {
                    PIN = d
                    TABLE {
                         0.1 0.4 1.6
                    }
               }
          }
          TABLE {
               0.9 1.4 4.7
          }
     }
}
VECTOR (01 d && cp) {
     ENERGY {
          HEADER {
               SLEWRATE {
                    PIN = d
                    TABLE {
                         0.1 0.4 1.6
                    }
               }
          }
          TABLE {
               0.3 0.5 1.8
          }
     }
}
```



# Modeling of Delay (and Power) Derating

*Voltage and Temperature can be used as continuous variables in any linear or non-linear model

* Process is a discrete variable, therefore process corners must be in a table

* For linear equation-based derating, put the process corner *definition* in the *header* and the *value* of the derating factor in the *table*

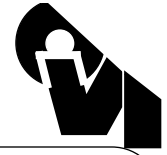
```
HEADER {
     PROCESS { HEADER {nom wnwp snsp} TABLE  {0.0 0.25 -0.18} }
     // other variables to follow
}
EQUATION { ... 0.24 + 1.53 * PROCESS ... }
```

* For nonlinear equation-based derating, use multiple named process objects and do for each the same as for linear derating

```
HEADER {
     PROCESS p1{ HEADER {nom wnwp snsp} TABLE  {0.0 0.25 -0.18} }
     PROCESS p2{ HEADER {nom wnwp snsp} TABLE  {0.0 0.03 -0.04} }
     // other variables to follow
}
EQUATION { ... 0.24 + 1.53 * p1 + 0.07 * p2 ** 2 ...}
```

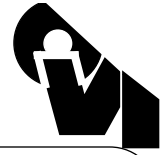
* For lookup table model, process corners are used as non-interpolatable indices. Put process corner *definition* in the *table*, derating factors are not needed

```
HEADER {
     PROCESS { TABLE {nom wnwp snsp} }
     // other variables to follow
}
TABLE { /* multiple of 3 values */ }
```



# Example

```
DELAY {
     HEADER {
          PROCESS {
               HEADER {nom snsp snwp wnsp wnwp}
               TABLE  {0.0 -0.1 -0.2 +0.3 +0.2}
          }
          VOLTAGE { ... }
          TEMPERATURE { ... }
          DELAY {
               HEADER {
                    CAPACITANCE { TABLE {0.03 0.06 0.12 0.24} }
                    SLEWRATE { TABLE {0.1 0.3 0.9} }
               }
               TABLE {
                    0.07 0.10 0.14 0.22
                    0.09 0.13 0.19 0.30
                    0.10 0.15 0.25 0.41
               }
          }
     }
     EQUATION {
          DELAY
          * (1 + PROCESS)
          * (1 + (TEMPERATURE - 25)*0.001)
          * (1 + (VOLTAGE - 3.3)*(-0.3))
     }
}
```



# Wire Modeling for Synthesis

* Estimation of wire capacitance and resistance as a function of fanout

* expressed in tables or equations in the same way as timing & power models

* Table used for fanout inside range

* Equation used for fanout outside range

```
WIRE my_wire {
     CAPACITANCE {
          UNIT = pF
          HEADER {
               FANOUT {
                    TABLE { 0 1 2 3 }
               }
          }
          TABLE { 0.0 1.7 2.5 3.7 }
          EQUATION { (FANOUT + 1) * 1.2 + 0.7}
     }
     RESISTANCE {
          UNIT = mOhm
          HEADER {
               FANOUT {
                    TABLE { 0 1 2 3 }
               }
          }
          TABLE { 0.0 9.5 12.5 15.0 }
          EQUATION { (FANOUT + 1) * 4.5 + 0.8}
     }
}
```

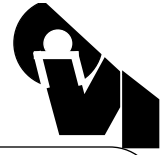
# Modeling for Test

* Annotations of Cell and Pin Properties for Scan Insertion and Test
* Reference to non-scan replacement model

Example:

```
CELL DFF2SC {
     PIN Q  {DIRECTION=output}
     PIN QN {DIRECTION=output}
     PIN D  {DIRECTION=input }
     PIN CP {DIRECTION=input }
     PIN CD {DIRECTION=input }
     PIN SD {DIRECTION=input }
     FUNCTION {
          @(CD) {QN = 1;}
          @(SD) {Q = 1;}
          @(CD && !SD) {Q = 0;}
          @(SD && !CD) {QN = 0;}
          @(01 CP && !CD && !SD)
               {Q = D; QN = !D;}
     }
}
```

```
CELL S000 {
          PIN H01 {...}
          PIN H02 {...}
          PIN H03 {...}
          PIN H04 {...}
          PIN H05 {...}
          PIN H06 {...}
          PIN N01 {...}
          PIN N02 {...}

     SCAN {
          scan_type = mux_scan
          non_scan_cell = DFF2SC {
               D  = H06
               CP = H02
               CD = H05
               SD = H04
               Q  = N01
               QN = N02
          }
     }
}
```



# Modeling for DRC

Modeling concepts:

* Definition of connect classes at library level

* Each pin of a cell belongs to a connect class by annotation

* Connectivity function:

  lookup table with connect-classes used as non-interpolatable indices

  values of connectivity function:
  1 connect rule is true
  0 connect rule is false
  ? connect rule is don’t care

  input variables of lookup table:
  driver
  receiver

* Connect rule must be associated with each connectivity function

  values of connect_rule:
  must_short
  can_short
  cannot_short

# Example

```
LIBRARY example_library {
     CLASS default
     CLASS clock
     CLASS enable
     CLASS reset
     CLASS tristate
     CONNECTIVITY driver_to_driver {
          CONNECT_RULE = can_short
          HEADER {
               DRIVER {  TABLE     { default clock enable reset tristate } }
          }              TABLE     { 0       0     0      0     1        }
     }
     CONNECTIVITY receiver_to_receiver {
          CONNECT_RULE = can_short
          HEADER {
               RECEIVER{ TABLE { default clock enable reset } }
          }              TABLE { 1       1     1      1     }
     }
     CONNECTIVITY driver_to_receiver {
          CONNECT_RULE = can_short
          HEADER {
               DRIVER {  TABLE { default clock enable reset tristate } }
               RECEIVER{ TABLE { default clock enable reset } }
          }
          TABLE {// default   clock     enable    reset     tristate
                    1         1         1         1         0         // default
                    0         1         0         0         0         // clock
                    0         0         1         0         0         // enable
                    0         0         0         1         0         // reset
}    }    }
```

# Megacell Modeling

* Support for bus modeling and two-dimensional arrays (same as in VERILOG)

* Edge operators are supported also for words

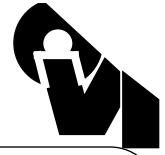
Example: (‘b0100‘b1100 d[3:0]) is a short form of
(01 d[3] && d[2] && !d[1] && !d[0])

* Additional support for symbolic edge operators

| | |
|---|---|
| ?- | no switching activity on bus |
| ?? | arbitary switching activity on bus |
| ?! | at least 1 bit switches on bus |
| ?~ | all bits switch on bus |

* For Timing modeling, groups can be used to do timing arc expansion

* For Power modeling, number of switching bits can be used as a variable



# Example

Asynchronous dual-port memory

```
CELL async_2port_ram {
     PIN enable_write {
          DIRECTION = input
     }
     PIN [4:0] adr_write {
          DIRECTION = input
     }
     PIN [4:0] adr_read {
          DIRECTION = input
     }
     PIN [7:0] data_write {
          DIRECTION = input
     }
     PIN [7:0] data_read {
          DIRECTION = output
     }
     PIN [7:0] data_store [0:31]{
          DIRECTION = output; VIEW = none; SCOPE = modeling;
     } // array of virtual pins

     FUNCTION {
          BEHAVIOR {
               data_read = data_store[adr_read];
               @(enable_write) {
                    data_store[adr_write] = data_write;
               }
     }
     // to be followed by timing and power arcs
}
```

# Example (cont.)

expansion of timing and power arcs using groups

use of symbolic edge operator “?!” for non-expanded power arc

```
GROUP adr_bits { 1 2 3 }
GROUP data_bits { 1 2 }

VECTOR ( 01 adr_read[adr_bits] -> 01 data_read[data_bits] ) {
     DELAY { ... }
} // this timing arc is expanded from all to all

VECTOR ( 01 din[data_bits] -> 01 dout[data_bits] ) {
     DELAY { ... }
} // this timing arc is expanded bitwise

VECTOR (?! data_write[7:0] && enable_write) {
     ENERGY {
          HEADER { SWITCHING_BITS { PIN = data_write } }
          EQUATION { 1.3 * SWITCHING_BITS }
     }
} // this power arc is not expanded

VECTOR (?! adr_read[4:0] -> 01 data_read[data_bits]) {
     ENERGY {
          HEADER { CAPACITANCE { PIN = data_read } }
          EQUATION { 2.5 + 3.3 * CAPACITANCE }
     }
} // this power arc is expanded to all data bits
}
```

# Core Modeling

Approach: define lower level components as primitives

Instantiate primitives for top level function

Define vector objects for timing shell

Example: 2nd order digital IIR filter for DSP
shows datapath and control building blocks

Algorithm:

data_out(t) = state(t) + b1*state(t-1) + b2* state(t-2)
state(t) = data_in(t) - a1*state(t-1) - a2*state(t-2)

Resources:

16-bit wide data bus

16bit x 4words register file for coefficients a1, a2, b1, b2

16 bit adder/subtractor/accumulator, 16x16 bit multiplier

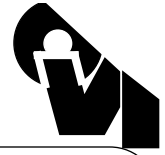
Schedule: 4 multiplications performed in 4 clock cycles

Controller: 2 bit counter counts 4 cycles

# Core Modeling: Example

top-level view

```
CELL digital_filter {
// external pins
     PIN [15:0] data_out  {DIRECTION = output;}
     PIN [15:0] data_in   {DIRECTION = input; }
     PIN [1:0] index_coeff{DIRECTION = input; }
     PIN [15:0] coeff_in  {DIRECTION = input; }
     PIN clock            {DIRECTION = input; }
     PIN reset            {DIRECTION = input; }
     PIN write_coeff      {DIRECTION = input; }
     PIN data_strobe      {DIRECTION = input; }
// internal pins
     PIN [1:0]  count     {DIRECTION = output; VIEW = none;}
     PIN [15:0] coeff_out {DIRECTION = output; VIEW = none;}
// user-defined primitives
     PRIMITIVE CNT2    { ... }
     PRIMITIVE REG16X4 { ... }
     PRIMITIVE IIR2    { ... }
// function
     FUNCTION { ... }
// timing shell
     GROUP index {15 : 0}
     VECTOR (01 clock -> 01 data_out[index]) { DELAY { ... } }
     VECTOR (01 clock -> 10 data_out[index]) { DELAY { ... } }
     VECTOR (01 clock <&> 01 data_in[index]) { SETUP { ... } HOLD { ... } }
     VECTOR (01 clock <&> 10 data_in[index]) { SETUP { ... } HOLD { ... } }
}
```



# Core Modeling: Example (cont.)

2 bit counter

```
PRIMITIVE CNT2 {
     PIN cd { DIRECTION = input}
     PIN cp { DIRECTION = input}
     PIN start { DIRECTION = input}
     PIN[1:0] count { DIRECTION = output}
     FUNCTION {
          BEHAVIOR {
               @ (!cd)  {count = 2'b0;}
               : (01 cp){count = start ? 2'b0 : count + 1;}
          }
     }
}
```

16bit x 4 word register file

```
PRIMITIVE REG16X4 {
     PIN we                {DIRECTION = input}
     PIN [15:0] din        {DIRECTION = input}
     PIN [15:0] dout       {DIRECTION = output}
     PIN [15:0] dmem [1:4] {DIRECTION = output; VIEW = none;}
     FUNCTION {
          BEHAVIOR {
               dout = dmem[r_adr];
               @ (we)  {dmem[w_adr] = din;}
          }
     }
}
```

## datapath (2nd order digital IIR filter)

```
PRIMITIVE IIR2 {
     PIN cd { DIRECTION = input}
     PIN cp { DIRECTION = input}
     PIN [15:0] dout  {DIRECTION = output}
     PIN [15:0] din   {DIRECTION = input}
     PIN [1:0]  cntrl {DIRECTION = input}
     PIN [15:0] coeff {DIRECTION = input}
     PIN [15:0] product  {DIRECTION = output; VIEW = none}
     PIN [15:0] accu     {DIRECTION = output; VIEW = none}
     PIN [15:0] state1   {DIRECTION = output; VIEW = none}
     PIN [15:0] state2   {DIRECTION = output; VIEW = none}
     FUNCTION { BEHAVIOR {
               sum =
                    (cntrl==’d0)? din - product :
                    (cntrl==’d1)? accu  - product :
                    (cntrl==’d2 || cntrl==’d3)? accu  + product ;
               @ (!cd)  {
                    product = 16’b0; accu = 16’b0; dout = 16’b0;
                    state1 = 16’b0; state2 = 16’b0;
               }
               : (01 cp){
                    product =
                         (cntrl==’d0 || cntrl==’d2)? coeff * state2 :
                         (cntrl==’d1 || cntrl==’d3)? coeff * state1 ;
                    accu  = sum;
                    dout = (cntrl==’d0) ? accu : dout ;
                    state2 = (cntrl==’d0) ? state1 : state2;
                    state1 = (cntrl==’d0) ? accu : state1;
}    }    }    }
```

## top-level function

```
CELL digital_filter {
     ...
     FUNCTION {
          BEHAVIOR {
               CNT2    u1 {
                    cd    = reset;
                    cp    = clock;
                    count = count;
                    start = data_strobe;
               }
               REG16x4 u2 {
                    we    = write_coeff;
                    din   = coeff_in;
                    dout  = coeff_out;
               }
               IIR2    u3 {
                    cd    = reset;
                    cp    = clock;
                    cntrl = count;
                    din   = data_in;
                    dout  = data_out;
                    coeff = coeff_out;
               }
          }
     }
     ...
}
```

# Outline

Introduction

- Motivation and Goals
- The ALF Story
- Relation between ALF, DCL, OLA
- ALF Flow, Target Applications and Features

ALF Language

- Object Model and Semantic Features
- Meta Syntax and Scoping Rules
- General Purpose ALF Objects

ALF Applications

- Functional Modeling
- Timing and Power Modeling
- Modeling for Synthesis, Test and DRC
- Megacell and Core Modeling

→ Conclusion

# Conclusion

* ALF is designed for generic ASIC libraries in order to
  - facilitate library development for ASIC vendors
  - leverage new EDA tools
  - respond to advanced modeling requirements

* ALF features efficient modeling of ASIC cells and blocks for
  - functionality
  - power and timing
  - physical properties

* ALF Syntax is
  - easy to understand
  - economic in use of keywords and reserved characters
  - versatile and flexible

# Conclusion (cont.)

What ALF is **NOT**

* A programming language

* A delay or power calculation language

* A tool-specific data representation for Simulation, Static timing etc.

* Design data file

* The library seen by the final customer

* Replacement for

  - SDF
  - SPF
  - PDEF
  - Verilog
  - VHDL
  - DCL

# Conclusion (cont.)

What ALF **IS**

* A unified source for consistent library data

* A prefered data entry  from modeling engineers who are not programmers

* A file format forASIC functional, timing and power modeling
  for synthesis, analysis and test

* Library data file

* source, allowing the application to compile for efficiency

* complementary to other standards

  - SDF
  - SPF
  - PDEF
  - Verilog
  - VHDL
  - DCL

# Acknowledgements

## Contributors to ALF 1.0

| | | |
|---|---|---|
| Wolfgang Roethig, PhD | NEC, formerly LSI Logic | ALF chairman |
| Amir Zarkesh, PhD | TDT, formerly Viewlogic | ALF co-chairman |
| Mike Andrews | Mentor Graphics | ALF co-chairman |
| Tim Ayres | Viewlogic/Sunrise | |
| Victor Berman | VI / IEEE | |
| Dennis Brophy | Mentor Graphics / IEEE DPC | |
| Renlin Chang | Cadence | |
| Sanjay Churiwala | Cadworx | |
| Jose De Castro | Mentor Graphics | |
| Timothy Ehrler | VLSI Technology | |
| Paul Foster | Avant! Corporation | |
| Vassilios Gerousis, PhD | Motorola / OVI Board | |
| Kevin Grotjohn | LSI Logic | |
| Johnson Chan Limqueco | Ambit Design Systems | |
| Ta-Yung Liu | Avant! Corporation | |
| Larry Rosenberg | Cadence/VSI | |
| Hamid Rahmanian | Mentor Graphics | |
| Ambar Sarkar, PhD | Viewlogic | |
| Itzhak Shapira | Cadence | |
| Yatin Trivedi | Seva Technologies | Technical Editor |
| Devadas Varma | Ambit Design Systems | |
| David Wallace | Mentor Graphics/Exemplar | |
| Frank Weiler | Avant! Corporation / OVI Board | |

# Focus of Work Groups

ALF work group

Target audience: design & modeling engineers (software background useful)

Explore ALF as modeling language for existing and new applications

datapath compiler
signal integrity
formal verification
...

Propose extensions to ALF for new applications

Promote migration from internal ASIC library formats to ALF

OLA work group

Target audience: software engineers (design & modeling background useful)

help developping the OLA framework

help testing the OLA framework

promote migration of tool-specific interface to OLA

promote development of OLA-based applications

*Our work groups are open - please join us!*

# Contact People


| | | |
|---|---|---|
| ALF work group: | Wolfgang Roethig<br>NEC Electronics, Santa Clara CA<br>Tel. 408 588 5349<br>Email: Wolfgang_Roethig@el.nec.com | chairman |
| | Mike Andrews<br>Mentor Graphics Corp., Wilsonville OR<br>Tel. 503 685 4879<br>Email: mike_andrews@mentorg.com | co-chairman |
| OLA work group: | Jay Abraham<br>Si2 Inc., Austin TX<br>Tel. 512 342 2244 x 52<br>Email:jabraham@si2.org | project coordinator |
| websites: | http://www.ovi.org<br>http://www.si2.org | |