IBIS-X and the IBIS Macro Language

Author: Stephen Peters

AV&T group, IBIS Vice-Chair

Edited by: Arpad Muranyi

Intel Corporation

16-March-2001

Background

- Current IBIS up against limits
  - Description lagging new technologies
  - Limited support for return path modeling
  - Limited support for coupled package modeling
  - No support on horizon for receiver models
  - No support for frequency domain analysis
- IBIS-X Goal: provide the path to a ‘next generation’ I/O buffer description language.

IBIS-X requirements

- Keep the good stuff
  - standardized description
  - protect IP
  - document SI parameters
  - continue support for board level (behavioral) simulators
- Expand format, structure to fix problem areas
  - expandability, flexibility (add nodal connectivity, get away from fixed assumptions and keywords)
  - enable accurate modeling of SSO, power delivery and package effects
  - support behavioral receivers
  - enable generalized equations, S-parameters, etc.

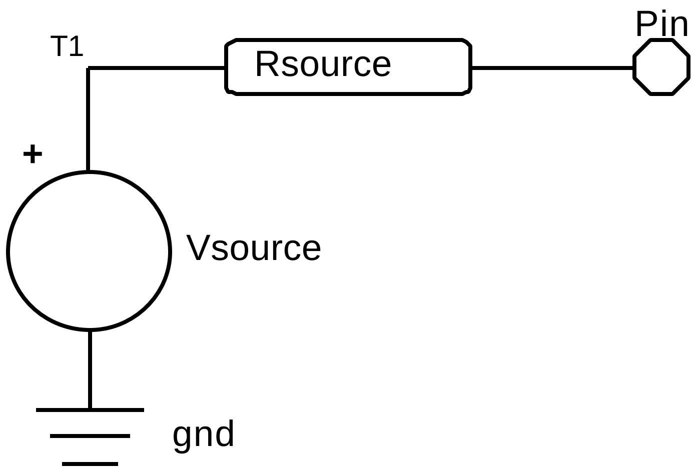
IBIS-X Approach

- Nodal descriptions are a necessity
  - die interconnect (pin - pad - buffer) descriptions
  - power delivery and pin-pin coupling
  - connections for black boxes
- Macro language for creating new model prototypes
  - flexible, extendable, allows existing IBIS models to be reused
  - enables a wide variety of descriptions
- *Bottom Line: user will have the ability to create his/her own custom buffer (model_type), and specify a package model in a standard format.*

Overview of Model Type Creation

- User creates a new model type using the [Define] keyword
- User describes new type using macro language
  - interconnect L/R/C elements and sources, data/values are parameterized
  - programming flow statements/operators available
  - special statements for trigger events, etc.
- User then instantiates model in .ibs file ([Model] statement) and supplies data for that instance, just as before.

Example (define the model)



```
[Define Model] simple_driver (gnd, pin, control)
node T1
Vsource (T1 gnd) V=Rise[T-TR] || Fall[T-TF]
Rsource (pin T1) R=Rsrc[T-TR] || Rsrc[T-TF]
trigger TR (Logic(control) == 1)
trigger TF (Logic(control) == 0)
[End Define Model]
```

Example (supply the data)

```
[Model] driver_1
Model_type simple_driver
[Rise]
0n  0
2n  1
5n  4
7   5

[Fall]
0n  5
2n  4
5n  1
7n  0

[Rsrc]
0n  7
2n  1k
3.5 100k
5n  1k
7n  5
```

Basic Elements

SPICE Compatible

Resistor (R)	(defines v = f(I,t); r=constant or f(t))
Capacitor (C)	(defines q = f(v,t); c=constant or f(t))
Inductor (L)	(defines flux = f(I,t); L= constant or f(t))
VCVS (E)	(defines Vout = f(Vin,t))
VCCS (G)	(defines Iout = f(Vin,t))
Isource (I)	(defines fixed I or I= f(t))
Vsource (V)	(defines fixed V or V= f(t))
Diode (D)	(basic diode, subset of Spice parameters)
Subckt (X)	(used for submodel, driver schedule)

Extended (some, but not all SPICE support)

Voltage controlled Resistance (VCR)	(defines R = f(v))
Voltage controlled Admittance (VCG)	(defines Y = f(v))
Voltage controlled Cap (VCCAP)	(defines C = f(v))
Uncoupled transmission lines	
Coupled transmission lines	

Basic Elements (cont.)

- Syntax for elements should be familiar
  - <type> <instance name> (<nodes>) <value>
  - Example: Resistor R12 (2 4) R=10k
- Values can take on several forms
  - simple value (10k)
  - scalar symbolic value (C_comp)
  - built-in symbolic value (VT)
    - uses the value of an internal variable, such as TIME
  - 1D table symbolic (I=Pullup[V])
  - 2D table symbolic (I=Pullup[V,T] or I=Series_mosfet[Vc,Vo])
  - Expressions can be made of combinations of above.
    - Operators are: +, -, *, || with usual bindings.
- || is evaluated as in Perl, allows for optional data sets / keywords
- Example : Vx (Vcc gnd) V=(Pullup Reference) || (Voltage Range)

Extended Blocks Beyond Spice

Needed now for Supporting drivers & receiver blocks

Driver	(a complex device for backward compatibility)
Reshape	(generates a digital pulse, reshaped from it's input)
Delay	(out = in, but delayed)
Voltage Controlled Delay	

Possible (likely) extensions for future work

- Behavioral voltage source ("B" element)
- Behavioral current source ("B" element)
- Integrator block
- Differentiator block
- Behavioral integrator
- Behavioral differentiator

.Extended (.dot) Elements

Model Creation/Functionality Related

- **trigger** (time value expression) – generates a trigger event when condition is met
- **node** – declares node name to be local
- **inherit** – inherit properties of another model or base structure

Debug/Visibility Related

- **export** – list of local symbols that can be made visible to the user
- **alarm** (time value expression) – notifies user when some event happens, used for error checking and the like
- **assert** (static logic expression) – check for a condition to be true, again for error checking

More Extended (.dot) Elements

Programming Flow

if (static logic expression)

else if (static logic expression)

end if

select

case

end select

foreach (index) **in** (pointer to data structure) – sets up an array of objects tied to a keyword. Used to implement driver schedule and submodel functionality

Equation Based Modeling

- General equations (Berkley B-element) not supported directly in initial release
  - Looking for quick implementation/adoption using existing technology.
  - Equation support planned for future release based on feedback
  - However, no reason one can't turn equations into tables which *are* supported.
- Nothing in IBIS-ML prevents the support of equation based modeling (i.e. support for Berkley B-element) in future release.

Current Status

- IBIS 3.2 has been implemented in IBIS-ML
- Working group meets bi-weekly to work on writing formal specification(s)
  - Overall IBIS-X specification
  - Library guide
  - Programmers Language Reference Manual
- Much work remains on die interconnect section
  - Coordinating with IBIS connector spec
  - Need for additional volunteers