IBIS–X?

Status
• Prototype begun. November public review.
• Proto can scan complete description of 3.2.
• Committee review, have covered half.
• Getting bogged down on small stuff.
• I have some questions.

Syntax
• LL(1) grammar with LR(1) expressions.
• Spice vs. equations vs. both.

Hot debate
How Spice like?
• [ ] As close as possible.
• [ ] Keep it netlist oriented, but "fix" the syntax.
• [ ] No need to make it Spice like at all.
• [ ] Who needs resistors, we got equations.
Hot debate
What simulators should support IBIS?
- [ ] All of them. There is no excuse for not supporting IBIS.
- [ ] Keep the status quo.
- [ ] Only transmission line simulators.
- [ ] Only Spice–type simulators.
- [ ] Only the most sophisticated.

Hot debate
What skill level should it take to make an IBIS model.
- [ ] A child could do it.
- [ ] A technician, ordinarily skilled.
- [ ] An undergraduate.
- [ ] A circuit designer.
- [ ] An expert.

Hot debate
How rich should it be?
- [ ] Subset of SPICE.
- [ ] About what SPICE does.
- [ ] More than SPICE.

Hot debate
Staged release – when to make the first?
- [ ] As soon as it supports IBIS 3.2.
- [ ] Wait til we have full SPICE.
- [ ] Wait til all the nuances are there.
Software architecture

- A "compile" pass that translates to the simulators format.
- Then run the simulator.
- A good simulator will do this automatically.

Hot debate
Expression syntax.

- [ ] Make it so it can be translated to PWL’s and the like, that most simulators already have.
- [ ] Require a general expression element, beyond what SPICE has.

Elements needed by the simulator.
Minimum.

- The basic ones, with PWL.
- 2 dimensional PWL – time and signal.
- A "trigger" to shift time tables.

Elements needed by simulator.
Arpad’s version.

- 4 generalized expression elements:
  - — voltage
  - — current
  - — charge
  - — flux
- Traditional elements (R, C, ...) will be omitted.
Separation of Data and Structure

Structure

- "Structure" = A spice-like description, with expressions.
- "Data" = A "data sheet" set of tables and attributes, like IBIS has always been.

IBIS is, and should remain, primarily a data sheet language.

<table>
<thead>
<tr>
<th>Conditionals: select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define Model</td>
</tr>
<tr>
<td>select (control)</td>
</tr>
<tr>
<td>inherit Series</td>
</tr>
<tr>
<td>case &quot;[On]&quot;</td>
</tr>
<tr>
<td>end case</td>
</tr>
<tr>
<td>end select</td>
</tr>
<tr>
<td>End Define Model</td>
</tr>
</tbody>
</table>

Data

<table>
<thead>
<tr>
<th>Model</th>
<th>abcde</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Define Model]</td>
<td>Series [pin1 pin2]</td>
</tr>
<tr>
<td>R Series</td>
<td>6 ohm 12 ohm</td>
</tr>
<tr>
<td>C Series</td>
<td>4 pF 5 pF</td>
</tr>
<tr>
<td>[End Define Model]</td>
<td>abcde</td>
</tr>
</tbody>
</table>

Separation of data and structure
**Conditionals: if**

```
[Define Model] Series_switch (pin1 pin2 control)
  .select (control)
    .case "[Off]"
      .inherit Series
    .end case
    .case "[On]"
      .inherit Series
    .end case
  .end select
[End Define Model]
```

**Time dependent tables, triggers**

```
[Define Model] Simple_driver (pin gnd control)
  vsource Vd (pin gnd) v = [Rise](T−TR) || [Fall](T−TF)
  trigger TR (v(control) > v_high)
  trigger TF (v(control) < v_low)
[End Define Model]

[Model] out1
  Model_type Simple_driver
  v_high = 3
  v_low = 2
  [Rise]
  0 0
  2n 5
  [Fall]
  0 5
  2n 0
  [End Model] out1
```

**Compatible driver element**

```
Driver Udrv (pin gnd pullup_ref pulldown_ref
  trig_10 trig_01 trig_z xtrig_z) {
  states = [1 0 z],
  S0 = [Pullup](-V),
  S1 = [Pullup](+V),
  S2 = [Pulldown](V),
  T10 = [Falling_Waveform](T−TF,*)
  T01 = [Rising_Waveform](T−TR,*)
  R10 = [Ramp]dv/dt_f,
  R01 = [Ramp]dv/dt_r,
  V1 = [Pullup_Reference] || [Voltage_Range],
  V0 = [Pulldown_Reference] || 0,
  mask = [Power Clamp](V(pullup_ref)−V)
    + [Gnd Clamp](V−V(pulldown_ref) )
```

**Array**

```
.array [Add_Submodel]
  .if ($1 == "Non_Driving")
    .node sm_enable
      inverter U1 (sm_enable gnd en)
    .else if ($1 == "All")
      .node sm_enable
dsource U2 (sm_enable gnd) 1
    .else
      .assert ($1 == "Driving")
      .define sm_enable en
    .end if
  subckt X$0 (en=sm_enable ...) $0
  .end array
```

```
[Add Submodel]
  Submodel_name Mode
  Bus_Hold_1 Non-Driving
  Dynamic_clamp_1 All
```
Alarm

```plaintext
alarm_failure {
  V(pin) > [Model_Spec]D_overshoot_high ||
  V(pin) < [Model_Spec]D_overshoot_low  ||
  V(pin) > [Model_Spec]S_overshoot_high
  for [Model_Spec]D_overshoot_time ||
  V(pin) < [Model_Spec]S_overshoot_low
  for [Model_Spec]D_overshoot_time ||
  never) "signal exceeds safe limits"
```

Value expression

```plaintext
capacitor Cttwr {pin power_clamp_ref}
  C = [TTpower] * [POWER_Clamp]|(-V) / VT || open
```

2d tables

```plaintext
Vccs Mosfet (pin1 pin2 0 pin2) I = {
  [Series_MOSFET](table=V(0 pin2),Vds=V(pin1 pin2)) ||
  open)

[Series_MOSFET]
Vds = 1.0
  5.0V   257.9m   153.3m   399.5m
  4.0V   203.0m   119.4m   317.3m
  3.0V   129.8m   74.7m    205.6m
  2.0V   31.2m    16.6m    51.0m
  1.0V   52.7p    46.7p    56.7p
  0.0V   0.0      0.0      0.0
Vds = 2.0
  5.0V   457.9m   353.3m   599.5m
  4.0V   403.0m   319.4m   517.3m
  0.0V   0.0      0.0      0.0
[End Series MOSFET]
```