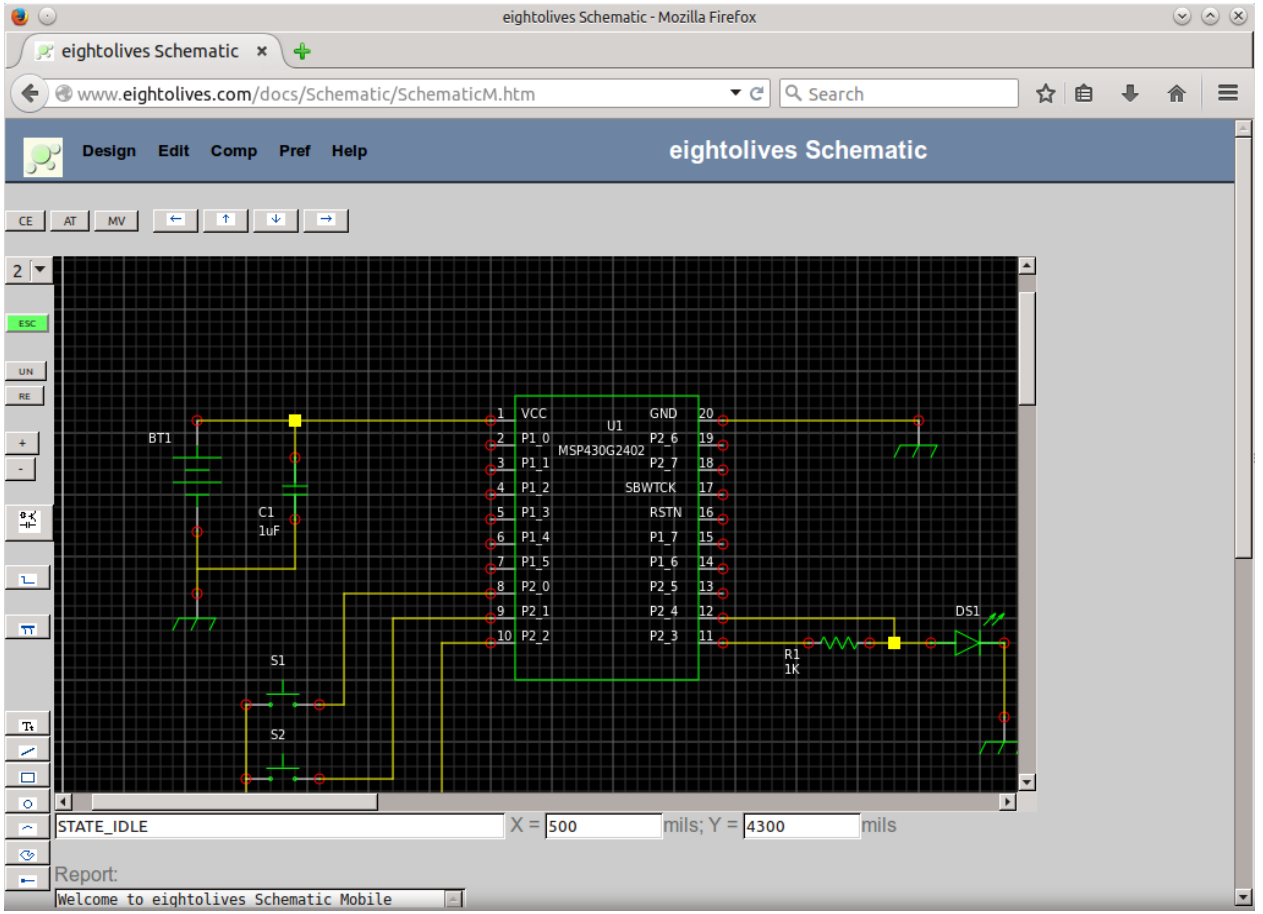
A Look at Eightolives' Hardware Model API



Version 2

William_Kaupinis@eightolives.com
March 23, 2015

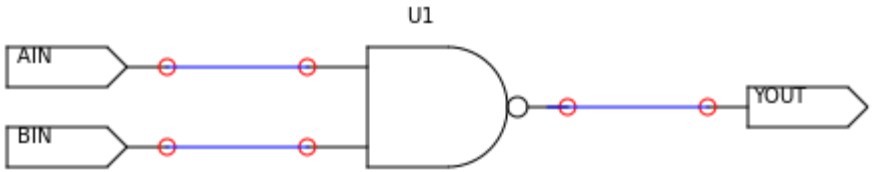
Eightolives' Hardware Models

- The eightolives' Hardware Model API version 2 is a Javascript API that supports an executable representation of schematics and component symbols.
- The model of a schematic can be automatically generated using eightolives' Schematic Mobile
  - Schematic symbols from the eo_simulation library have functional models available.
  - Default (non-functional) template models can be also be made for each component
- Models can be loaded and simulated using eightolives' WaveformViewer

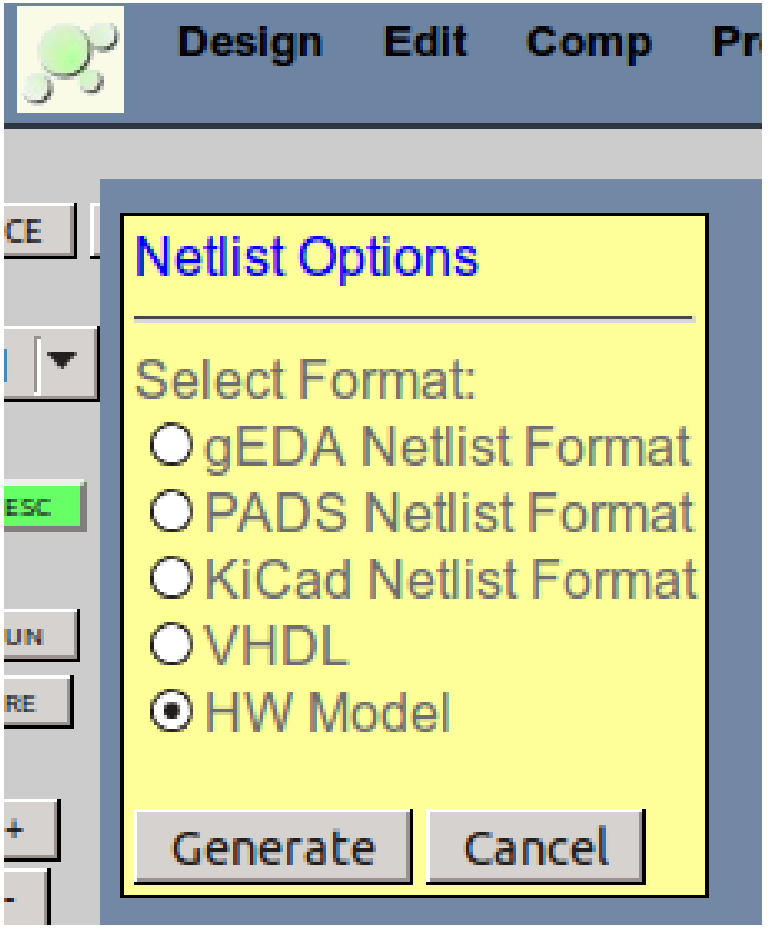
About the Hardware API

- It's written in Javascript so it can execute in a typical web browser environment
- It's structure is somewhat similar to VHDL
  - VHDL entity is like the API's Element
  - Port → Port
  - Generic → Generic
  - Architecture → Architecture
  - Std_Logic → StdLogic
  - Std_logic_Vector → StdLogicVector
  - VHDL Case insensitive → Javascript Case sensitive

Making a Simple Model in Schematic Mobile



- The schematic consists of the built-in 2 input nand gate (gnand2) and Input and Output Port connectors.



Design > Generate Netlist

Resulting Model Appears in Report Window

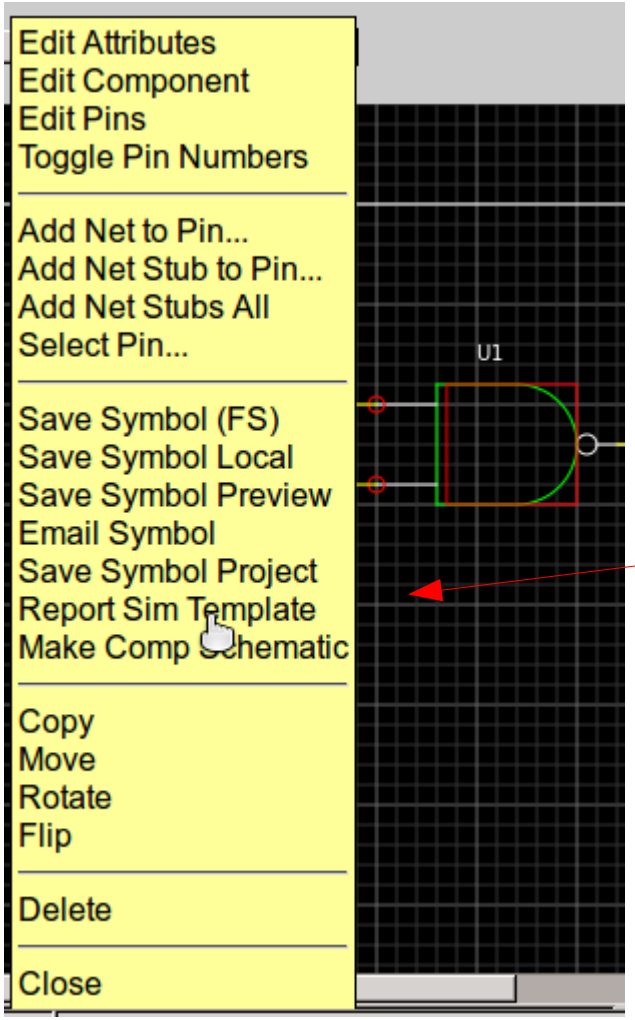

```
// NandGate.js 4/24/15  1429899476925


// this is a hardware model for NandGate
// eightolives hardware model version 2
NandGate.prototype = new Element();
NandGate.prototype.constructor = NandGate;
function NandGate()
{
  this.name = "NandGate";
  this.initialize();
  //define ports
  var AIN = this.addNewStdLogicPort("AIN", 0).signal;
  var BIN = this.addNewStdLogicPort("BIN", 0).signal;
  var YOUT = this.addNewStdLogicPort("YOUT", 1).signal;
  //define architecture
  var a = this.addNewArchitecture("schematic");
  a.initialize();
  //define signals
  //define body
  var U1 = a.addInstance("U1", getInstanceOf("gnand2"));
  if(U1 != null)
  {
  U1.addStringGeneric("device", "gnand2");
  U1.addStringGeneric("refdes", "U1");
  U1.addStringGeneric("logicfunction", "NAND_2");
  U1.addStringGeneric("primitive", "true");
  U1.addStringGeneric("model", "simlib");
  U1.addTimeGeneric("Td", "3 ns");
  U1.connectPortToSignal("Y", YOUT);
  U1.connectPortToSignal("A", AIN);
  U1.connectPortToSignal("B", BIN);
  U1.define();
  }
  /* //schematics normally use the default uninitialize and execute
functions
  a.uninitialize = function()
  {
  }
  a.execute = function()
  {
  }  */
}


addModelDescription(new ModelDescription("NandGate", NandGate,
["gnand2"]));
```

A schematic model primarily defines the interconnection of components.

A schematic model needs no editing.

The ModelDescription indicates that a model “gnand2” also needs to be loaded

Generating a Component Template



- In Schematic Mobile, select the component and then click to get the pop-up menu
- Select **Report Sim Template**
- The template appears in the Report window where you can copy or save it.

A Template

```
// gnand2.js 4/24/15  1429900261211

// this is a template hardware model for gnand2
// eightolives hardware model version 2

gnand2.prototype = new Element();
gnand2.prototype.constructor = gnand2;

function gnand2()
{
  this.name = "gnand2";
  this.initialize();

  //define ports
  var Y = null;
  var A = null;
  var B = null;

  //define generics
  var device = this.addStringGeneric("device", "gnand2");
  var refdes = this.addStringGeneric("refdes", "U1");
  var logicfunction = this.addStringGeneric("logicfunction", "NAND_2");
  var primitive = this.addStringGeneric("primitive", "true");
  var model = this.addStringGeneric("model", "simlib");
  var Td = this.addTimeGeneric("Td", "3 ns");

  this.define = function()
  {
  //define ports
  clearArray(this.ports);
  Y = this.addNewStdLogicPort("Y", 1).signal;
  A = this.addNewStdLogicPort("A", 0).signal;
  B = this.addNewStdLogicPort("B", 0).signal;
  }
  this.define();

  // set attributes
  setAttributeValue("PRIMITIVE",this, "true");

  //define architecture
  var a = this.addNewArchitecture("simulatable");
  a.initialize();

  a.uninitialize = function()
  {
  Y.set("U");
  }

  a.execute = function()
  {
  }
}
addModelDescription(new ModelDescription("gnand2", gnand2, []));
```


- The template model defines the Element, Ports and Generics
- The primitive attribute indicates this is the lowest level of hierarchy
- The **uninitialize** and **execute** functions need to be refined for functionality to happen.

Basic Definitions

- An **Element** represents a component.
- A **Port** is an in, out or inout pin on an Element.
- An **Architecture** represents an implementation of an Element.
- A signal represents a net within the Element and architecture. Signals may be **StdLogic**, **StdLogicVector**, **Time**, or **StdString**.
- A **generic** is a signal that corresponds to a schematic symbol's attribute.
- Signal values can be 0, 1, H, L, X, Z, -, U, W
  - Strong values are 0, 1, X

Closer look at the component model

```
// gnand2.js 4/24/15  1429900261211

// this is a template hardware model for gnand2
// eightolives hardware model version 2

gnand2.prototype = new Element();
gnand2.prototype.constructor = gnand2;
```

- The first line has the filename (gnand2.js), the creation date and time stamp
- The next line is a comment
- The next line identifies the file as an eightolives hardware model (**required**)
- The next two lines are Javascript code that identifies gnand2 as being a hardware Element

The Constructor function

```
function gnand2()
{
 this.name = "gnand2";
 this.initialize();

 //define ports
 var Y = null;
 var A = null;
 var B = null;

 //define generics
 var device = this.addStringGeneric("device", "gnand2");
 var refdes = this.addStringGeneric("refdes", "U1");
 var logicfunction = this.addStringGeneric("logicfunction", "NAND_2");
 var primitive = this.addStringGeneric("primitive", "true");
 var model = this.addStringGeneric("model", "simlib");
 var Td = this.addTimeGeneric("Td", "3 ns");

 this.define = function()
 {
 //define ports
 clearArray(this.ports);
 Y = this.addNewStdLogicPort("Y", 1).signal;
 A = this.addNewStdLogicPort("A", 0).signal;
 B = this.addNewStdLogicPort("B", 0).signal;
 }
this.define();
```

- The function gnand2 defines the name, defines variables to represent the ports
- Generics are defined based on a symbol's attributes
- A “define” function creates the ports

```
 // set attributes
 setAttributeValue("PRIMITIVE",this, "true");

 //define architecture
 var a = this.addNewArchitecture("simulatable");
 a.initialize();

 a.uninitialize = function()
 {
 Y.set("U");
 }

 a.execute = function()
 {

 }
}

addModelDescription(new
ModelDescription("gnand2", gnand2, []));
```

Constructor (cont)

- The PRIMITIVE attribute is set
- The architecture is named, defined and initialized
- The **uninitialize** and **execute** functions must be defined by the user to add functionality
- **addModelDescription** adds the model to the WaveformViewer's list of models. The arguments are its name, the constructor function, and an array of dependent models needed (an empty array for a PRIMITIVE model).

Defining Functionality

- The uninitialize function is called when you RESTART a simulation.
  - Signals are set to their values at time 0
- The execute function is called whenever the component's ports change in value.
  - Output values are set with a delay value based on expressions involving input port values and stored variables

Writing Logic Expressions

- Signal variables have a name (eg. CLK.name) and value, (eg. CLK.value)
- Time variables can express setup, hold or delay time
  - var Td = this.addTimeGeneric(“Td”, 3 ns);
- API functions create objects or compute logic functions
  - var C = DIN.and(DATA_ENABLE); // var C is a value
  - var G = nand(A.and(B), C.and(D)); // 4 input NAND
- Output signals are set using setWithDelay
  - Y.setWithDelay(G, Tdelay.value);

Hardware API functions (1/4)

function Attribute(name, value)
function DesignObject()
function addItemToArray(a, arra)
function clearArray(a)
function addAll(thisarray, tothisarray)
function removeItemFromArray(item, arr)
function getAttribute(name, obj)
function getAttributeIndex(name, obj)
function getAttributeValue(name, obj)
function setAttributeValue(name, obj, value)
function ModelDescription(mname, model, prereq)
function isModelLoaded(mname)
function getModelDescription(mname)
function addModelDescription(md)
function checkPrerequisites(md, bfind)
function reportDesign(e, sp)
function getInstanceOf(name)
function risingEdge(lastValue, newValue)
function fallingEdge(lastValue, newValue)

function Element()
Element.prototype.initialize = function()
Element.prototype.define = function()
Element.prototype.getPrimitive = function()
Element.prototype.addNewArchitecture = function(name)
Element.prototype.addNewStdLogicPort = function(name, mode)
Element.prototype.addNewStdLogicVectorPort = function(name, mode, L1, L2)
Element.prototype.getPort = function(name)
Element.prototype.connectPortToSignal = function(pname, sig)
Element.prototype.getSignal = function(name)
Element.prototype.addTimeGeneric = function(name, value)
Element.prototype.addStringGeneric = function(name, value)
Element.prototype.addGeneric = function(sig)
Element.prototype.uninitialize = function()
Element.prototype.execute = function()
Element.prototype.clearSim = function()
Element.prototype.getFullName = function()

Blue items are frequently used in models

Hardware API functions (2/4)

function Architecture()

Architecture.prototype.initialize = function()

Architecture.prototype.addInstance = function(label, e)

Architecture.prototype.uninitialize = function()

Architecture.prototype.execute = function()

Architecture.prototype.clearSim = function()

Architecture.prototype.getSignal = function(name)

Architecture.prototype.addSignal = function(sig)

Architecture.prototype.addNewStdLogicSignal = function(signame)

Architecture.prototype.addNewStdLogicVectorSignal = function(signame, L1, L2)

function Signal()

Signal.prototype.getFullName = function()

Signal.prototype.uninitialize = function()

Signal.prototype.endSimTick = function()

Signal.prototype.set = function(value)

Signal.prototype.setTo = function(value, t)

Signal.prototype.setWithDelay = function(value, t)

Signal.prototype.notifyListeners = function()

Signal.prototype.addConnection = function(p)

function Port()

Port.prototype.notifyListeners = function()

Port.prototype.evaluate = function(sig)

Port.prototype.getPortsAssertedInternalSignalCValue = function()

Port.prototype.getPortsAssertedExternalSignalCValue = function()

Blue items are frequently used in models

Hardware API functions (3/4)

function getIndexOf(c)
function resolve(a, b)
function StdLogic()
StdLogic.prototype.getIndexOf = function(c)
StdLogic.prototype.evaluate = function(v1, v2, cmd)
function evaluate(v1, v2, cmd)
StdLogic.prototype.not = function()
StdLogic.prototype.nand = function(sig)
StdLogic.prototype.nor = function(sig)
StdLogic.prototype.and = function(sig)
StdLogic.prototype.or = function(sig)
StdLogic.prototype.xor = function(sig)
StdLogic.prototype.xnor = function(sig)
StdLogic.prototype.isLow = function()
StdLogic.prototype.isHigh = function()
StdLogic.prototype.toStrong = function()

function isHigh(c)
function isLow(c)
function not(c)
function nand(a, b)
function nor(a, b)
function and(a, b)
function or(a, b)
function xor(a, b)
function xnor(a, b)
function toStrong(sg)

Blue items are frequently used in models

Hardware API functions (4/4)

function StdLogicVector()

StdLogicVector.prototype.getSize = function()

StdLogicVector.prototype.forOthersSet = function(c)

StdLogicVector.prototype.toStrong = function()

function Time()

Time.prototype.setTime = function(ts)

Time.prototype.getUnitsAsString = function()

Time.prototype.getUnitsScaleFactor = function(unittype)

Time.prototype.getScaledTime = function(t, z)

function getScaledTime(t, z)

- All functions are defined in the file eo_hardware.js
- See the User Manual for more information on using the API

Blue items are frequently used in models

Adding Function to the NAND gate

```
a.uninitialize = function()
{
Y.set("U");
}

a.execute = function()
{
  var c = A.nand(B);
  Y.setWithDelay(c, Td.value);
}
```

- The template set up the unintialize function
- To the execute function we add the API call A.nand(B) which the computes the NAND of signals A and B
- The output is updated after the propagation delay time Td.value
- That's it.

D Flip Flop

- A clocked D Flip Flop model with preset and reset stores the value lastCLK

```
this.lastCLK = "U";

a.uninitialize = function()

{

Q.set("U");

QN.set("U");

this.lastCLK = "U";

}
```



```
a.execute = function()
 {
  if(PN.isLow())
  {
    Q.setWithDelay("1", Td.value);
    if(RN.isLow())
    {
       QN.setWithDelay("1", Td.value);
    }
    else
    {
       QN.setWithDelay("0", Td.value);
    }
  }
  else if(RN.isLow())
  {
    Q.setWithDelay("0", Td.value);
    QN.setWithDelay("1", Td.value);
  }
  else if(risingEdge(this.lastCLK, CLK.value))
  {
    Q.setWithDelay(D.value, Td.value);
    QN.setWithDelay(D.not(), Td.value);
  }

  this.lastCLK = CLK.value;
 }
```

Slotted and Partitioned Symbols

- For “slotted” components that contain multiples (dual flip flop, hex inverter), partitioned symbols (U1a, U1b) or any symbol you can create a schematic template of the component (hierarchal schematic)
  - Pin names of slotted components have the basic pin name suffixed with an underscore and pin number in Hardware Models to differentiate between sub-parts
- Then put in and connect simlib elements
- Then create a Hardware Model of the schematic

FAQ

- Can you simulate large designs? Not likely. Probably memory or speed limited. See and try some of the available examples.
- Will it simulate very quickly? No. Javascript is an interpretive language not optimized for speed.
- Can I migrate my work to other simulators? The VCD waveform output files are standard. Schematic Mobile can also output VHDL which can be used with other simulators.
- What is version 1 of the API? Version 1 is a similar Java API that works with eightolives Workspaces Desktop

For more information

- Visit http://www.eightolives.com
- See app notes on Using Schematic Mobile
- See app notes on Using WaveformViewer