Simulating Javascript Models

William_Kaupinis@eightolives.com
Mar 29, 2010

Abstract

Hardware models described by ECMA Javascript provide an easy, no-compile way to simulate a design using eightolives' Workspaces Desktop tools.

A ScriptableArchitecture using an external Javascript file is developed for a D flip flop and simulated.

Why Javascript?

- ECMA Javascript is an established object-oriented language which does not require a compiler to execute
- It interfaces to the eightolives Workspaces Desktop toolset and designs
  1) Define the interface
  2) Create a Javascript template using the DesignTool
  3) Add the functionality in the Editor tool
  4) Import the Architecture
  5) Simulate the design in the WaveformViewer tool

How it works

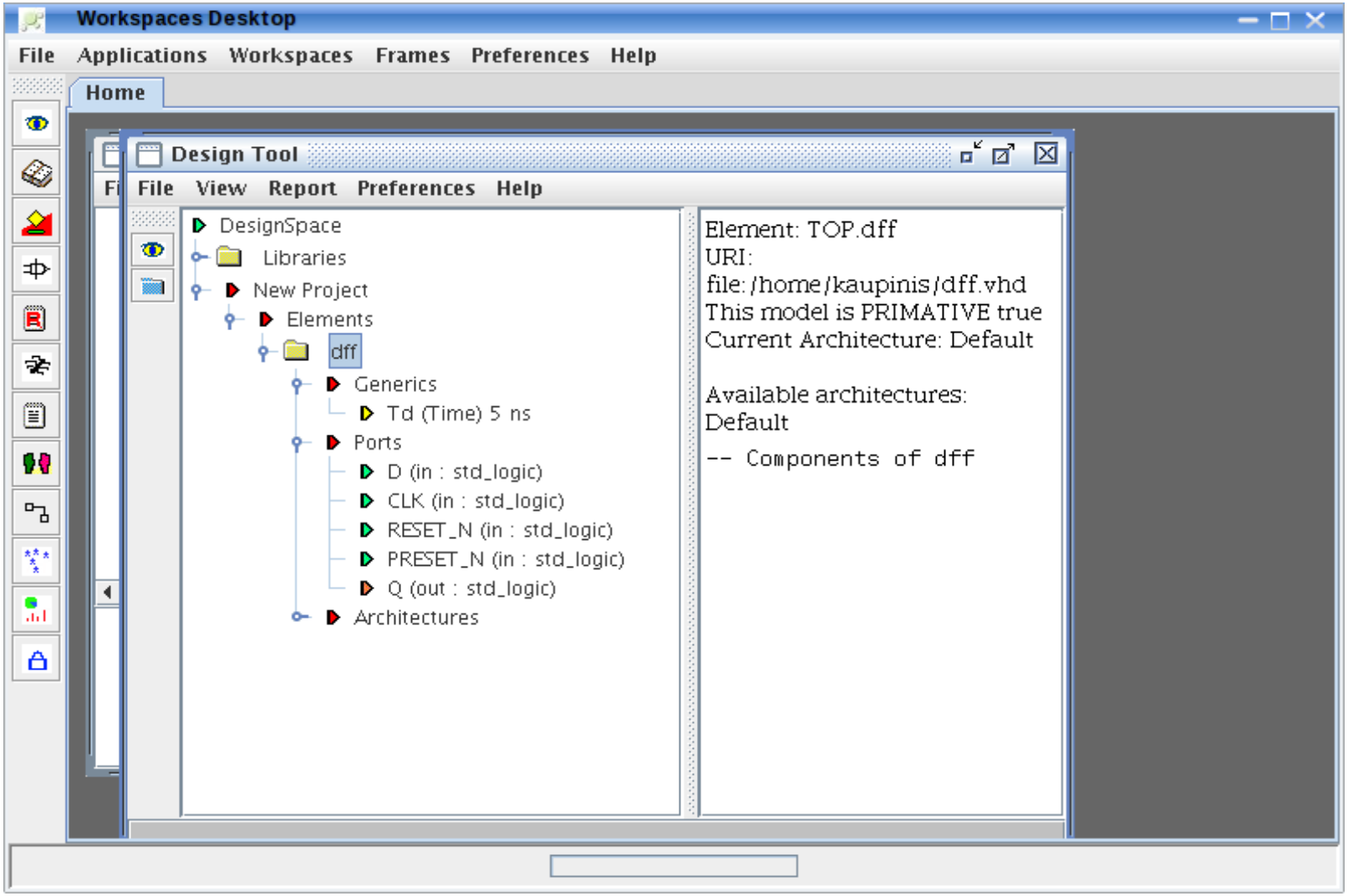
- Hardware is modeled using the Java package com.eightolives.Hardware API
- The API's ScriptableArchitecture class calls three functions in a Javascript file
  - initialize() – defines the relevant Signals and variables
  - uninitialize() - initializes signals and variables prior to a simulation run (set things to 'U')
  - execute() – called to modify state and outputs when an input changes (this is where the functionality lies)

Step 1 Define the Interface

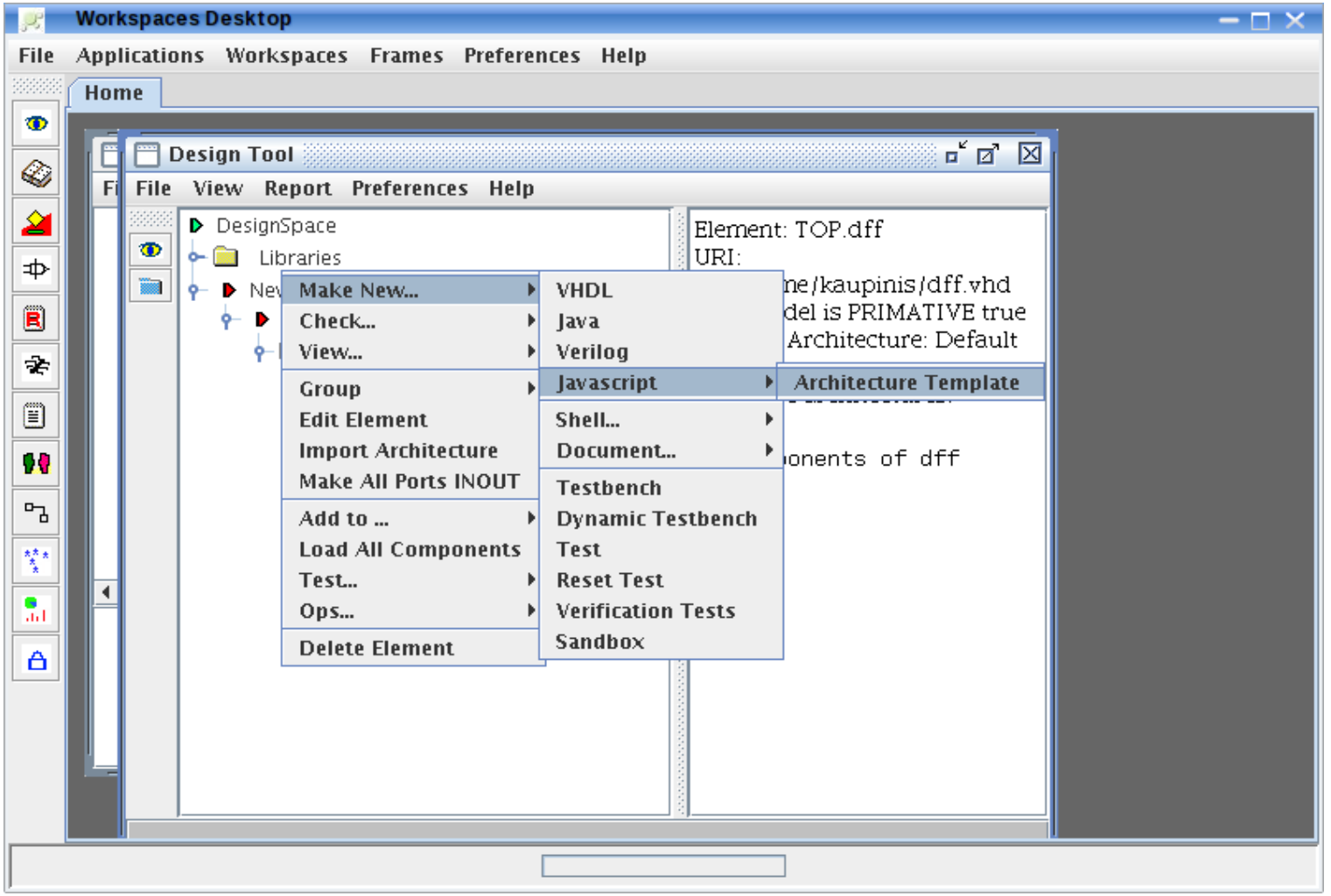
- Create an Element in the DesignTool by reading in a VHDL file or just use the menus to create the Element and add I/O ports

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.std_logic_arith.ALL;
LIBRARY WORK;
ENTITY dff IS    -- model a D flip flop
GENERIC(
    Td    : Time := 5 ns     -- specify delay time
    );
PORT(
    D    : IN std_logic;
    CLK    : IN std_logic;
    RESET_N    : IN std_logic;
    PRESET_N    : IN std_logic;
    Q    : OUT std_logic
    );
END dff;
ARCHITECTURE default OF dff IS
BEGIN
END default;
```

Step 2 Right Click the Element



Select Make New > Javascript > Architecture Template



The Created Template (1 of 4)

```
// dff_architecture.js                                    2 forward slashes // define start of a comment

// This is a template for a Javascript part of a ScriptableArchitecture

// let a be the Architecture for which this script implements functionality
a = null;                                                 //  we define “a”  as the Architecture object

// let e be the the parent Element
e = null;                                                 // we define “e” as the parent Element object

// define internal variables
LOGIC_0 = 48; // ASCII 0 decimal version        // Javascript quirk expresses characters as ASCII codes
LOGIC_1 = 49; // ASCII 1 decimal version       // use these variables when making comparisons
lastclk = 'U';
Td = new com.eightolives.Hardware.Time("Td", "1 ns");  // defines a delay time object

iQ = 'U';                                                 // variables of all outputs are made with an”i” prefix
```

The Created Template (2 of 4)

```
// initialization routine
function initialize(architecture)          // the initialize function initializes and defines all variables
{
a = architecture;
e = a.getParent();
if(e.getLabel() != null) cpf.print("This model is for " + e.getLabel() + ": " + e.getFullName());  // cpf.print prints to the CommandProcessor window
else  cpf.print("This model is for " + e.getFullName());
att = new com.eightolives.Hardware.Attribute("PRIMITIVE", "true");  // the PRIMITIVE attribute must be set true to indicate its simulatable
e.setAttribute(att);

// initialize all varaibles used in the script
D = e.getSignal("D");
CLK = e.getSignal("CLK");
RESET_N = e.getSignal("RESET_N");
PRESET_N = e.getSignal("PRESET_N");
Q = e.getSignal("Q");
Td = e.getGeneric("Td").getInitialValueAsString();
lastclk = 'U';
iQ = 'U';
} // end initialize
```

The Created Template (3 of 4)

```
// uninitialize is used to define all internal variables prior to start of simulation

function uninitialize(simqueue)

{

sim = simqueue;  // sim is a reference to the simulator

cpf.print("uninitialize()");

// initialize all internal variables

lastclk = 'U';

iQ = 'U';

} // end uninitialize
```

The Created Template (4 of 4)

```
// execute is called during simulation whenever a related Signal changes
function execute()      // this is where the functionality is modeled
{
// template synchronous code    // templates examples of code are included but commented out
// if(RESET_N.isLow())
//   {
//   }
// else if((lastclk == LOGIC_0) && CLK.isHigh())  // detect rising edge
//   {
//   }
// template asynchronous code
// iQ = DIN.getCvalue();
// iY = A.and(B.or(C));
// assign outputs with delay
// Q.setTo(iQ, Td);
// save clock values for next cycle edge detection
// lastclk = CLK.getCvalue();
} // end execute
//end of Javascript part of ScriptableArchitecture
```

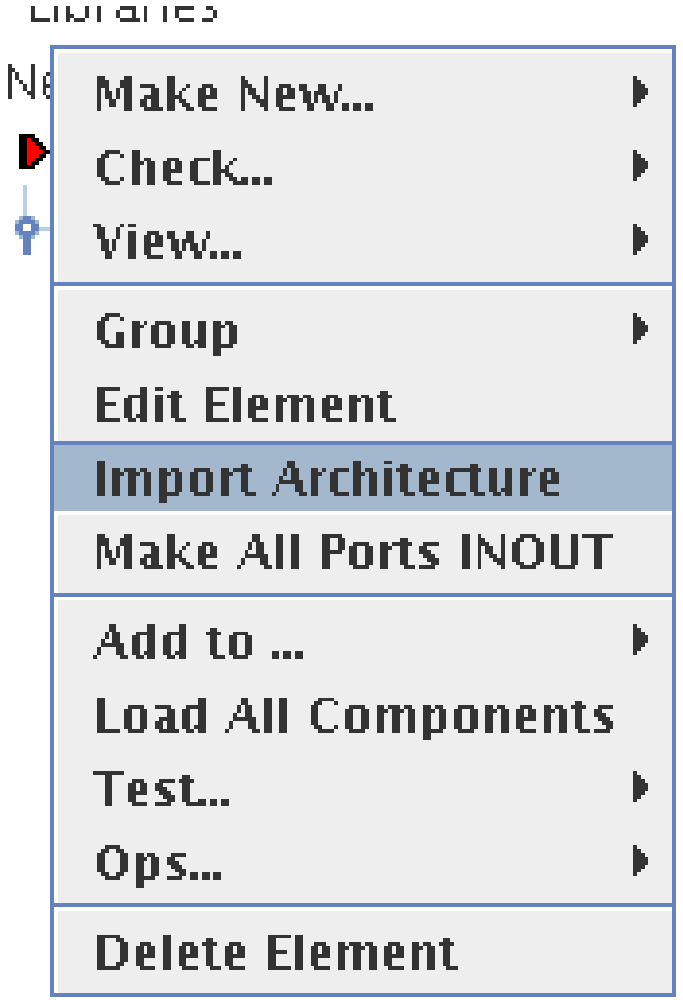
Step 3 Edit the execute() function

```
// execute is called during simulation whenever a related Signal changes
function execute()
{
if(PRESET_N.isLow())
        {
        iQ = '1';
        }
else if(RESET_N.isLow())
        {
        iQ = '0';
        }
 else if((lastclk == LOGIC_0) && CLK.isHigh())  // detect rising edge
        {
        iQ = D.getCvalue();
        }

// assign outputs with delay
Q.setTo(iQ, Td);

// save clock values for next cycle edge detection
lastclk = CLK.getCvalue();

} // end execute
```

Save the file as *element_name*_architecture.js

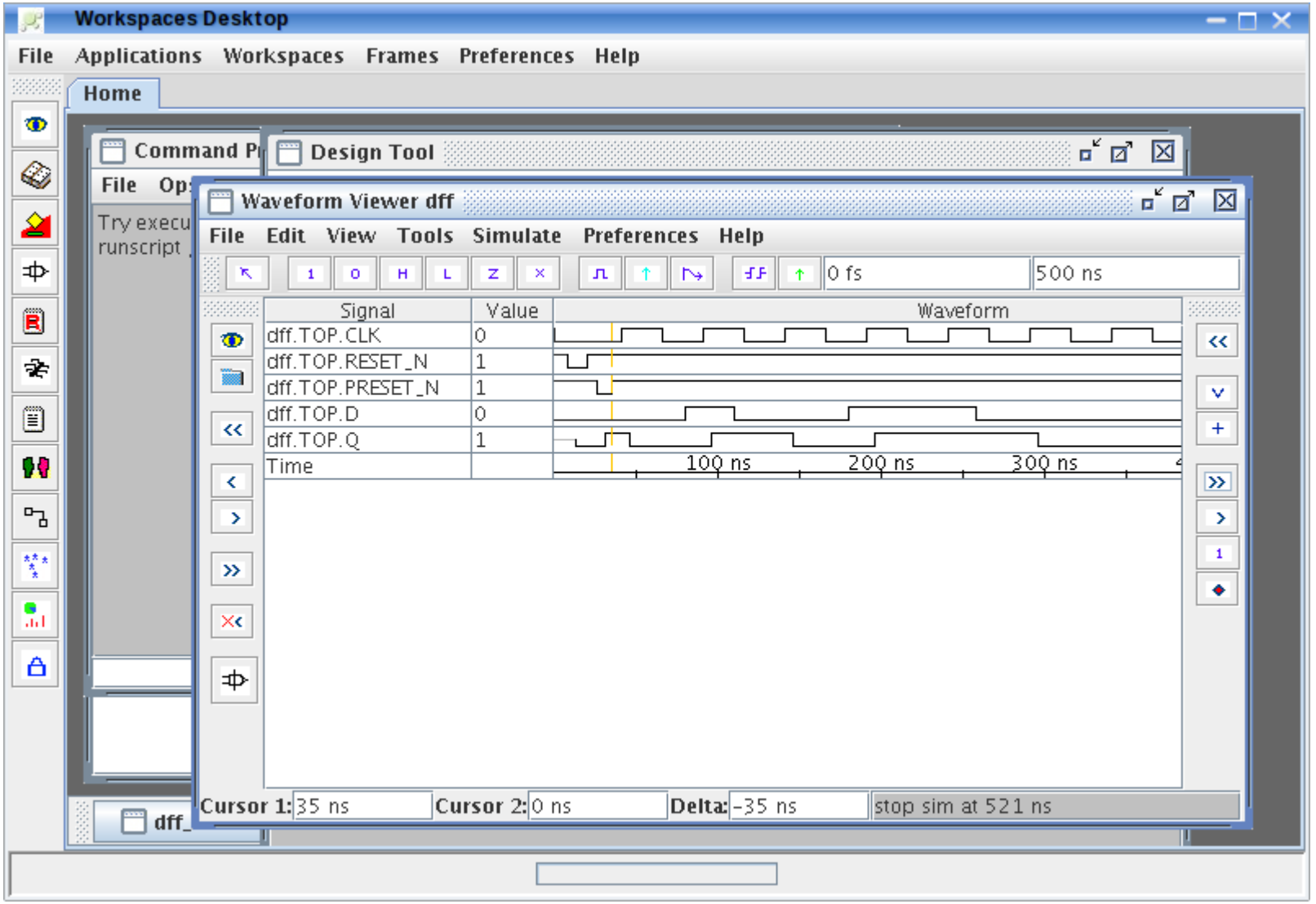
For this example it's dff_architecture.js

Step 4 Import Architecture



- Right click the Element in DesignTool and select Import Architecture and select the .js file saved in step 3
- A Scriptable Architecture will be added to the design
- A CommandProcessor window will open and execute the script

Step 5 Simulate



Right click the Element View > Waveforms

Signal Objects

- Signal objects i.e. A = e.getSignal("A"); use methods defined by com.eightolives.Hardware.Signal
  - A.getCvalue() returns the character value of the Signal
  - A.nand(B) returns the character result of (A nand B)

Example Signal Ops:

```
iQ = A.nand(B.or(C));

iY = D.xor(A);

if(A.getCvalue() == Logic_0)
        {
        iZ = D.getCvalue().nand(iQ);
        }

iY = A.not();

IQ = B.nor(A.and(C).or(B.and(D)));
```

Variables

- Use Javascript variables for storing computation results and state information
  - lastclk = CLK.getCvalue();
- Javascript variables follow Javascript rules
  - iTemp = A.nor(B.nand(iJ));
- Assign variables to output signals using the Signal's setTo method
  - Q.setTo(iTemp, “250 ps”);

Hints

- Error messages appear in the Java Console or in the CommandProcessor window.
- Javascript is case sensitive
- Any variables you create should be included in the initialize and uninitialize functions
- Predefined objects give you access to the tool environment
  - ws – Workspaces ws.getDateStamp()
  - cpf – CommandProcessorFrame cpf.print(“stuff”)
  - dt – DesignTool dt.getElementByName(“
- Use cpf.print(“iQ = ” + iQ); to display messages in the CommandProcessor window

Hints

- You can autoload the .js file from a VHDL entity-only file by adding an ATTRIBUTE SCRIPTABLE = “*pathname* “

```
ATTRIBUTE PRIMITIVE : STRING;

ATTRIBUTE PRIMITIVE OF dff : ENTITY IS "true";

ATTRIBUTE SCRIPTABLE : STRING;

ATTRIBUTE SCRIPTABLE OF dff : ENTITY IS "pathname.js";
```

You can run this demonstration

- From Workspaces Desktop > Bookmarks
  - Select Resources > Examples > Simulating Javascript (this loads the dff.vhd interface definition – VHDL entity)
  - Right click the dff Element in the design tree and Import Architecture selecting dff_architecture.js
  - Right click the dff Element and View > Waveforms
  - File > Load > VCD Project Waveform
    - Select dff.vcd
  - Click the “+” button (Add waveforms to the sim queue)
  - Click the “>>” button to run the sim

For more information

- Check the tutorials at http://www.eightolives.com/tutorials.htm
  - Workspaces Desktop Tool Overview
  - Scripting
  - Modeling Hardware in Java
- Read the Workspaces Desktop Users Manual
- ECMA Javascript info at

  http://www.ecma-international.org/publications/standards/Ecma-262.htm