Quick Start Guide

the SystemBuilder command line interface

System Builder Team

document edition 1

ASTG Technical Memo
Programming Solutions Group
Xilinx
December 6, 2007

Contents

Contents 2

1 Executing dataflow models 3
- 1.1 Command line interface 3
  - 1.1.1 Tool Environment 3
  - 1.1.2 General Features 4
  - 1.1.3 Design capture 4
  - 1.1.4 Invoking the simulator 5
  - 1.1.5 Invoking Compilation 7
- 1.2 Actor Language tool support 8

Chapter 1

Executing dataflow models

1.1 Command line interface

The SystemBuilder tool suite encompases three phases of system development – design capture, simulation, and compilation. The tool suite uses a front-end executable script named `sb` for simulation and compilation and the Eclipse (http://www.eclipse.org) IDE for design capture.

Invocation	*Description*	*Detail*
`sb help`	Display help message and version information	Section 1.1.2
`sb sim` *target*	Simulate the specified simulatable entity	Section 1.1.4
`sb` *network*`.vhd`	Generate HDL for the network named *network*	Section 1.1.5

Table 1.1: Command Line Interface Summary

1.1.1 Tool Environment

The command line interface is designed to operate within any Unix style environment including cygwin. In addition to those elements supplied with SystemBuilder, the execution environment must supply the following:

java Java Runtime Environment version 1.5.0 or higher. Available from http://java.sun.com

make Standard make utility. Available under cygwin from the "Devel" collection.

shelltools Standard shell tools including `sh`, `basename`, `dirname`, `echo`, `mv`, `pwd`, `rm`, `tr`, `uname`, `diff`, `grep`, and `sed`. All are available in the cygwin "Base" collection and should be standard install for all other Unix style environments.

1.1.2 General Features

version `sb help` – prints a general help message including version information.

verbosity `sb.logging` – a file (peer to sb) containing the verbosity level settings for the tools. Changing the xcal.user.level value affects verbosity of the tools. Valid levels are (from least to most verbose): `OFF`, `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, `FINEST`, `ALL`.

1.1.3 Design capture

Eclipse

The preferred development environment is to use Eclipse IDE (http://eclipse.org) build 3.2.1 or newer. A plugin for eclipse is available that provides syntax highlighting and source code syntax error checking for all Network language (NL) and CAL sources. This plugin is pre-installed in the Eclipse images contained on the release CD. If you need to install this plugin into your copy of Eclipse, the process is:

- Access the software updates feature of Eclipse via the menu chain:

  ```
  Help->Software  Updates->Find and Install...
  ```

- Choose `Search for new features to install` and click `Next>`
- Select `New Remote Site...`
- For "Name:" use "OpenDF Site"
- For "URL:" use "http://opendf.sourceforge.net/eclipse"
- click `OK`
- make sure the checkbox next to "OpenDF Site" is selected and click `Finish`
- When the Updates window re-appears choose the Open Dataflow Plugin *x.x.x* release and click `Next`
- Accept the license and click `Finish`

Following the above steps will associate CAL source files (*.cal) with the CAL editor and Network Langauge files (*.nl) with the NL editor.

1.1.4 Invoking the simulator

The SystemBuilder tool includes a command line simulator for networks of Actors or single Actor instances. The simulator may be invoked as:

```
sb sim [options] <simulatable-entity>
  -ea                   enables assertion checking during simulation
  -n <##>               defines an upper bound for number of
                          simulation steps
  -t <##>               defines an upper bound for the simulation time
  -t <##>               defines an upper bound for the simulation time
  --max-errors <##>     defines an upper bound for the maximum number
                          of allowable errors during simulation
  -i <file>             identifies the input stimuli (vector) file
  -o <file>             defines the output vectors -- will be overwritten
                          if existing
  -D <param def>        allows specification of parameter defs
  -q                    run quietly
  -v                    run verbosely
  -bbr                  detect and report output-blocked actors on deadlock
  -bi                   ignore buffer bounds (all buffers are unbounded)
  -bq <##>              produces a warning if an input queue ever becomes
                          bigger than the specified value
  -mp <paths>           specifies the search paths for model files
  -cache <path>         the path to use for caching precompiled models.  If
                          none is specified, caching is turned off.
  --version             Display Version information and quit
```

Where:

any integer. A negative value indicates infinite time/steps. Default is infinite.

file any readable/writable ASCII file following the format listed in 1.1.4

platform class the 'platform' on which the CAL Actors are interpreted. Default is `com.xilinx.systembuilder.evaluator.SystemBuilderPlatform`. Other valid values include: `caltrop.interpreter.util.DefaultPlatform` and `hades.cal.HadesPlatform`.

param def a name/value specification of the form name=value. If name or value contain spaces the tokens must be enclosed in appropriate quotes. eg `-D a=11` passes the value 11 to parameter "a", while `-D b=2=3` passes the value "false" (as a result of evaluating "2=3" to the parameter "b".

IMPLEMENTATION NOTE.
Note that, by default, the simulator references all packages and paths relative to the invocation directory. Any package names used in the actor network for Actor or sub-network instances will resolve into a relative directory path according to standard Java package-to-path resolution rules. For example, given a network `testNetwork.nl` containing an Actor `A` with package specification of `cal` where the simulator is run from the `myrepos` directory. The following directory structure is expected:

```
~/myrepos/
         |--> testNetwork.nl
         |--> cal/
                  |--> A.cal
```

Similarly, any external reference (eg parameter to `openFile`) will be relative to the simulator launch directory.
This behavior may be modified by specifying the model-path via the -mp command line option.

Simulatable entities

Simulatable entities include the following:

- Single actor – A single file containing valid CAL source code for a single Actor. The file name must be *name*.cal
- Network Language – An actor network described in the System Builder Network Language (NL). The filename must be *name*.nl
- XDF Actor network – A single file containing valid XDF source for a network of Actors. XDF may contain hierarchical references to other XDF networks or instances of CAL Actors. The file name must be *name*.xdf

Simulation stimuli file format

The command line simulator can use textual files to supply input stimuli data to the simulatable entity and to store generated data from the entity. The input stimuli and output result file formats have the same format as follows:

```
<portname> <time> <token>
```

Where `portname` is the name of a top-level input port, `time` is a 'double' value that defines the time stamp for that input (or output) token, and `token` is the token itself. Token times must be in non-descending order. It is fine for successive token times to be the same. The tokens will be sent to the entity in document order, and with the associated time stamp. If the simulation time is `t`, i.e. the next step in the simulation is scheduled at time `t`, then all tokens up to and including those with time stamp `t` will be sent before that step is executed.

Alternatively, the stimulus may be provided via a stimulus actor within the top level testbench network. Similarly, result data may be verified or appropriately displayed by a sink actor within the testbed network.

1.1.5 Invoking Compilation

CAL source error checking

Syntax errors are best located via the Eclipse CAL source editor which will highlight syntax errors interactively during editing. Installation of the CAL syntax highlighter is accomplished by following the instructions in Section 1.1.3

As an aid in development the SystemBuilder tool suite CLI has an error checker which can be run on any CAL Actor source. The checker will report errors for some of the most common design problems with pertinent context information. Given an Actor `foo.cal` the error check is run by executing (note the target file extension):

```
sb foo.calml
```

Hardware Compilation

Compilation to HDL is possible for individual Actors or entire networks of actors. It should be noted that only those language elements contained in the synthesizable subset of the CAL Actor language may be compiled to HDL. This subset is detailed in section 1.2

Compilation of an NL specified Actor network to HDL is accomplished by generating a `.vhd` target. For example, given an Actor network `topNetwork.nl` the complete HDL for the network (VHDL) plus all Actor instances (Verilog) is generated in response to:

```
sb topNetwork.vhd
```

The HDL for the network will be generated as a peer to the source NL (eg `topNetwork.vhd`) while the HDL for the Actor instances will be contained in a directory named `Actors` which will also be a peer to the source NL.

Compilation of a single Actor to HDL (Verilog) is accomplished by generating a `.v` target. For example, given an Actor “A” specified in `A.cal`, the HDL is generated via:

```
sb A.v
```

IMPLEMENTATION NOTE.
It is important that all parameters must be supplied values when compiling Actor instances or networks of Actors to HDL. Typically this is done by instantiating the top-level (parameterized) Actor or network into an Actor network which contains appropriate and resolvable declarations for all top-level parameters.

1.2 Actor Language tool support

Note: this is not yet an exhaustive list of the constraints placed on CAL source for synthesis.

- list types: All list types must be implemented with the `list` type and not the `List` type.
- lists are the only supported complex data type for synthesis. They may be nested to any depth so long as the data type for the inner most list is of a primitive data type.
- only integer (`int`) and boolean (`bool`) primitive data types are supported for synthesis.
- mult-token writes to an Actor output port are supported only if that Actor port is annotated with `tokenoutputstyle` as `blocking` in the NL.
- multi-token reads are not supported.
- Package names must not contain whitespace.