Open Dataflow Tools

Engineering CLI Guide

Ian Miller

document edition 1

ASTG Technical Memo
Programming Solutions Group
Xilinx
February 8, 2008

Contents

Contents 2

1 Opendf Tools 3
1.1 Overview 3
1.2 Eclipse Front End 4
1.3 Intermediate Formats 4
1.3.1 CALML 5
1.3.2 XLIM 6
1.3.3 Verilog/VHDL 6

2 Additional Resources 7

Chapter 1

Opendf Tools

1.1 Overview

The Open Dataflow tools, including Xilinx proprietary technology, are structured as a sequence of relatively small transformations on XML documents which describe the functionality of a dataflow system. Through the course of transformation from user source code (starting with a source to XML translation) to HDL, simulation model, or other compiler output, there are a number of these XML documents that may be useful representations for integration with other tool flows or for export to other compiler back-ends. Some of these extension points are highlighted in green in figure 1.1.

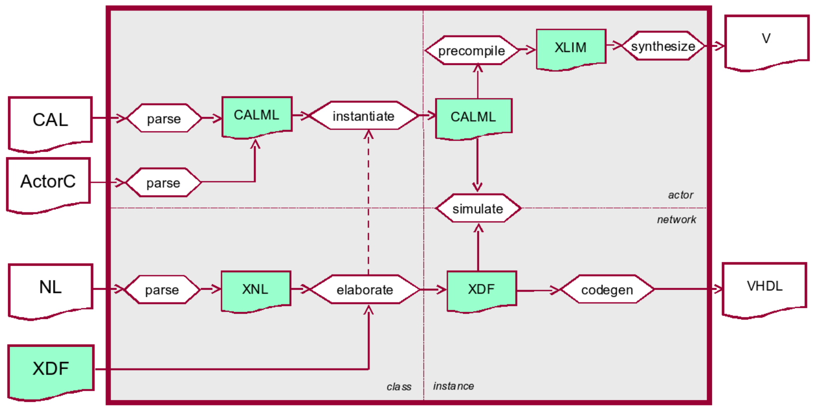

Figure 1.1: Open Dataflow Tool Structure

As a practical matter, the tools are implemented as a combination of Java source code and XSLT transformations. The Java source is used to provide simple interfaces to logical groupings of the XSLT transforms, as well as user interaction facilities for configuration, display and actual GUI (via Eclipse plugins).

This document gives a brief overview of some intermediate formats generated by the Open Dataflow tools as well as the mechanisms for generating those formats from CAL and/or NL source. For further information please see the resources listed in section 2.

1.2 Eclipse Front End

The typical user interface for the Open Dataflow tools is via the Eclipse IDE platform. Various plugins have been developed and deployed which provide access to the various development, simulation and compilation features that have been developed. These plugins are available for download via the OpenDF project eclipse update site at: http://opendf.sourceforge.net/eclipse

1.3 Intermediate Formats

This section provides a high level view of each intermediate format and the command line invocation used to generate that file. Except where noted the input file is CAL source code (example.cal). It should be noted that each command line interface has multiple options that may be used to configure the run. Those options which must be specified are shown in the invocation, however others may be used when necessary. Notably:

- -vv, -v, -q, -qq for controlling verbosity
- -D <param>=<value> to specify parameter bindings for parameterized actors.
- -mp <path> to specify a model search path other than the current directory. This argument may be specified multiple times.

NOTE.
During compilation a spurious error message will frequently be generated. Messages of the following form may be safely ignored:

```
Exceptional event: Cannot ____ types: <<ANY : : >>, <<int : : size=XX>>.
```

1.3.1 CALML

CALML is an XML representation of the Abstract Syntax Tree (AST) for a given piece of CAL source code. Through the course of compilation the CALML document is modified and annotated with additional data including the results of various analyses. Access to the purest AST version of CALML is provided in a file with the '.calml' file name extension. The annotated and slightly transformed CALML is available in a file with the '.pcalml' file name extension. In either case the document generally contains the following elements:

- Expr - expression elements. The type of expression is determined by the 'kind' attribute.
- Decl - declaration elements.
- Type - type elements specify the data type
- Port - port elements identify the input/output ports and their type (if specified)
- Op - operators as defined in the Cal Language Reference 2
- Args - arguments to functions.
- Note - notes with results of various analyses

CALML

Compilation of example.cal to CALML:

```
java -classpath XilinxOpendf.jar net.sf.opendf.cal.main.Cal2CalML
example.cal
```

PCALML

Compilation of example.cal to PCALML (example.pcalml):

```
java -classpath XilinxOpendf.jar net.sf.opendf.cli.SSAGenerator
--preserve example.cal
```

Structurally PCALML contains the same types of elements as the original CALML. One notable difference in the structure is that all the OpSeq elements used for describing operations have been replaced with function applications (<Expr @kind='Application'> elements). The following details some of the additional information and structural changes.

- Function applications are inlined (unless -no-inline is specified in the above Java invocation)

- Operations (BinOpSeq, etc) are converted to function applications (not inlined as user functions are).
- Free variable annotations will be made. Note @kind=freeVar
- Some variable dependencies will be annotated. Note @kind=dependency
- Some constant evaluation/propagation has been performed.
- Some dead code will be eliminated.

1.3.2 XLIM

Compilation of example.cal to XLIM (example.xlim):

```
java -classpath XilinxOpendf.jar net.sf.opendf.cli.SSAGenerator
--preserve example.cal
```

XLIM is the Static Single Assignment form of the CAL actor. Each action within the Actor is a module element within the document. Each actor variable (those which maintain state between action firings) are declared and have initial values specified. In addition to the module elements for each action there is one declared for the action-scheduler. The action-scheduler module represents the logic associated with arbitrating and enabling the action modules.

In addition to utilizing and representing the data contained in the PCALML the XLIM has the following characteristics:

- SSA format. Each operation consumes data from a definitive (non symbolic) source. This may be a prior operation, state variable, or port.
- Fully typed. This is not a requirement for XLIM documents, however it is a characteristic of our generated XLIM.
- All actor states have been enumerated and a total priority order established. This is manifested in the structure of the action-scheduler.

1.3.3 Verilog/VHDL

Compilation of example.cal to Verilog (example.v). NOTE THAT THIS INVOCATION USES THE XLIM AS INPUT:

```
java -classpath XilinxOpendf.jar
com.xilinx.systembuilder.cli_private.HDLCodeGenerator
example.xlim
```

It is required for compilation to HDL output that the input CAL source be fully typed, where each declaration includes a *size* attribute (eg int(size=14) foo). This is only a requirement for HDL compilation. The Verilog HDL is the final compiled version of the CAL source.

Chapter 2

Additional Resources

The following resources may be useful in working with the Opendf Tools.

- CAL Language Reference - http://opendf.svn.sourceforge.net/viewvc/opendf/doc/CLR1/
- XLIM Document Reference - http://opendf.svn.sourceforge.net/viewvc/opendf/doc/Xlim/
- Network Language (NL) Reference - http://opendf.svn.sourceforge.net/viewvc/opendf/doc/NL1/
- Gentle Introduction to Dataflow Programming with Open Dataflow - http://opendf.svn.sourceforge.net/viewvc/opendf/doc/GentleIntro/