Table of Contents

Introduction

Bliss is an acronym for BLIF to SPICE Synthesis. It is software for converting (synthesizing) files in BLIF (Berkeley Logic Interchange Format) to SPICE netlist.

Bliss is intended for use in academic research. Originally developed by Noriel “Eli” Mallari, Bliss was used together with Odin II for a Verilog to transistor-level SPICE synthesis package. Odin II is developed by Dr. Peter A. Jamieson, a front-end for converting Verilog to BLIF.

SPICE is simulation software for analog circuits. Using Bliss with front-ends for converting Verilog to BLIF like Odin II, digital circuits written in Verilog can be simulated in a mixed-signal environment. This is useful for extracting analog parameters of digital circuits, like timing, power consumption, etc.

Requirements and Installation

Bliss

- Python 2.6.x or Python 2.7.x. Python can be downloaded freely from the Internet. Bliss is not tested on Python 3.x, but technically it should run.
- Windows Users: For ease of use, the path of Python should be added to the System Path, $Path. Python is usually installed in C:\Python26 or C:\Python27.
- There is no installation needed for Bliss. Just get **bliss.py** from the SourceForge project page, http://sourceforge.net/projects/blissvlsi, place it in a folder, and run using Python: python bliss.py -?. This shows available command-line options.
- Default Bliss Libraries are in **blisslib.zip**. You can also download blisslib.zip from the SourceForge project page. Unzip the blisslib folder in the path where you placed bliss.py. See Hello Bliss example on how the directory structure should look like.

Odin II

- Odin II is required for complete Verilog to SPICE synthesis. Odin II can be downloaded from Dr. Peter A. Jamieson's website: http://www.users.muohio.edu/jamiespa/odin_II.html. Guides on how to compile Odin II in Windows and GNU/Linux are included in the download files. Please do include his work as a reference in academic research.
- A tip for compiling Odin II: compile libvpr_x first. Don't forget to uncomment #define WIN32 in ezxml.c when compiling in GNU/Linux.
- Cygwin is required for Windows users. You can download the latest version of Cygwin from http://www.cygwin.com. Cygwin can be downloaded freely from the Internet.
- For ease of use, I suggest putting the Odin II executable (usually **odin_II.exe**) in the same folder as bliss.py.
- You can view Odin II available command-line options by runnning odin_II.exe: odin_II.exe -h.

Bliss Files

The following are files used by Bliss. Note that all files used by Bliss are in ASCII text format, and can be opened using any text editor.

BLIF Source File (.blif)

Bliss accepts a BLIF file, .blif, as input. Currently Bliss only supports BLIF files outputted by Odin II, or any BLIF file with logic blocks similar to outputs of Odin II.

Support for more BLIF files will be added as Bliss is continually developed.

Output Circuit File (.cir)

The output circuit file is the transistor-level SPICE netlist produced by Bliss.

The raw output circuit file only contains the synthesized SPICE netlist, and does not contain technology declarations or stimuli / test signals. See the Bliss Header File (.bh) and Bliss Footer File (.bf) for details on how to add these to the circuit.

Bliss Library Entry (.bl)

Bliss makes use of library files with extension .bl to synthesize BLIF blocks to transistor-level SPICE netlist. By default, .bl files are stored in directory .\blisslib, and are loaded prior to synthesis.

Bliss comes with a default CMOS library, but it you can add more libraries as needed. See section Bliss Library Entries for details.

You can set a different source directory for libraries in Bliss command-line options.

Bliss Header File (.bh)

Bliss header files contain text and declarations placed at the beginning of the SPICE netlist. You can put technology declarations inside this file.

The use of a .bh file is optional.

Bliss Footer File (.bf)

Bliss footer files are appended to the end of the synthesized SPICE netlist. Usually test signals and stimuli are placed in this file.

The use of a .bf file is optional.

Using Bliss

Available command-line options can be viewed by running:

```
python bliss.py -?
```

-or-

```
python bliss.py --help
```

(the command above makes use of double dash!)

Synthesizing BLIF Files

By default, Bliss loads the file default_out.blif as BLIF source file. To synthesize other BLIF files, use the command:

```
python bliss.py -b source.blif
```

Where source.blif is the name of your .blif source file.

Using Header Files and Footer Files

By default, Bliss makes use of the header file default_out.bh, and simply ignores it if it does not exist.

To make use of a header file:

```
python bliss.py -h header.bh
```

You can use any other header file name aside from header.bh.

By default, Bliss makes use of the footer file default_out.bf, and simply ignores it if it does not exist.

To make use of a footer file:

```
python bliss.py -b source.blif -f footer.bf
```

You can use any other fooer file name aside from footer.bf.

To combine the use of both header and footer file, and with source file as source.blif:

```
python bliss.py -b source.blif -h header.bh -f footer.bf
```

Setting Output Circuit File Name

By default, Bliss outputs the file as default_out.cir. To output the file using a different file name:

```
python bliss.py -o output.cir
```

You can use any other output file name aside from output.cir.

Using -a Command

It is convenient to make use of just one command to load .blif, .bh, .bf, and output a .cir file. To do this, use the -a command:

```
python bliss.py -a source
```

This will load source.blif, source.bh, source.bf, and will output source.cir.

Using Odin II

Odin II is required for complete Verilog to transistor-level SPICE synthesis.To use Odin II:

```
odin_II.exe -V source.v -o output.blif
```

Where source.v is the Verilog source file, and output.blif is the output BLIF file.

You can use any file name for Verilog source files and output.blif. Without -o option, odin_II.exe outputs files as default_out.blif.

Example

Download the hellobliss.zip file to run these examples. These examples were tested on Windows, but should also work on GNU/Linux systems. (Don't forget to replace the backslashes with forward-slashes. :))

Hello Bliss

The hellobliss folder contains hello.v, hello.blif, hello.bh, hello.bf, and hello.cir. Let's first start with unzipping the contents.

Place the hellobliss folder inside your Bliss directory. You can delete hello.blif and hello.cir to see if Odin II and Bliss are producing outputs properly.

By now your directory structure should look like:

```
.\blisslib

.\hellobliss

.\bliss.py

.\odin_II.exe
```

Run Odin II:

```
odin_II.exe -V .\hellobliss\hello.v -o .\hellobliss\hello.blif
```

Odin II should inform you that your synthesis was successful. Check if hello.blif exists in .\hellobliss

Run Bliss. Since .blif, .bh, and .bf files all have the same name, hello, we can use the -a command-line option:

```
python bliss.py -a .\hellobliss\hello
```

Bliss should inform you that You should have hello.cir in .\hellobliss. Congratulations, you just synthesized your first Verilog file!

Customizing Headers and Footers

Note that the example hello.cir will **NOT RUN IN SPICE SIMULATION**. This example does not define any model files for the transistors used in Bliss library. Don't forget to add model declarations in the header file, stimulus and test signals to the footer file when synthesizing your own circuits.

Default CMOS libraries make use of 'N' for NMOS and 'P' for PMOS transistor declarations. See Bliss Library Entries section for details.

Bliss Library Entries

Bliss makes use of an extensible architecture for library entries. You can use the existing default library with technology CMOS_DEFAULT, or you can make your own.

The following sections discuss the format of a Bliss library entry. Understanding the format will allow you to edit the existing libraries or create new library entries to suit your needs.

Library Entry Format

Bliss Library Entries are saved using Python's ConfigParser format, which is similar to Windows .INI files. The general format for setting values is <keyword> = <value>. For multi-line values, make sure that succeeding lines start with a whitespace, like a tab or space. Additional resources about this file format can be found in the Internet, particularly from Python discussions about the ConfigParser module.

The following is an example Bliss library entry for a 2-input AND gate (as seen from AND2.bl):

```
[BlissLibrary]

name = AND2

tech = CMOS_DEFAULT

desc : 2-Input AND-Gate

blifhead = .names in#0 in#1 out#0

blifbody = 11 1

template = vdd gnd in#0 in#1 out#0

circuit = .SUBCKT AND2 vdd gnd vin0 vin1 vout

      MPMOS0 outsig vin0 vdd vdd P L=1u W=2u
      MPMOS1 outsig vin1 vdd vdd P L=1u W=2u

      MNMOS0 outsig vin0 nsig0 gnd N L=1u W=2u
      MNMOS1 nsig0 vin1 gnd gnd N L=1u W=2u

      MPMOSINV vout outsig vdd vdd P L=1u W=2u
      MNMOSINV vout outsig gnd gnd N L=1u W=1u

      .ENDS
```

Let's discuss each line in the file.

[BlissLibrary]

This line indicates a section, and that the .bl text file is a BlissLibrary entry.

name

This defines the name of the library entry. This is used when the SPICE sub-circuit is instantiated in the output SPICE circuit.

tech

This defines the technology of the library entry. Currently Bliss does not support this, but the upcoming versions will.

desc

A description about the library entry.

blifhead

The blifhead definition inside the BLIF source file. This is matched by Bliss versus a BLIF block (typically a .names or .latch declaration), and assigns this library entry if there is a match. See Bliss Template Structure for details on how Bliss parses this parameter.

blifbody

The body of the BLIF block. This is matched by Bliss versus a BLIF block (typically a .names declaration), and assigns this library entry if there is a match. Usually this is present in .names declarations and makes use of the BLIF PLA format, and non-existent in .latch declarations.

template

The template structure on how this library entry is instantiated as a sub-circuit in the output circuit file. See Bliss Template Structure for details on how Bliss parses this parameter.

circuit

The equivalent SPICE sub-circuit of the library entry. Take note that all sub-circuit dependencies should be included in this sub-circuit file, as Bliss does not currently support checking the dependencies inside library entries.

The CMOS default library makes use of 'N' for NMOS and 'P' for PMOS. Make sure the transistor models you use define these properly.

Bliss Template Structure

Bliss makes use of a flexible template structure to find and set inputs and outputs of a logic block. Currently two parameters in Bliss Libraries make use of this: blifhead and template.

blifhead Structure

The blifhead parameter is compared to the logic block in the BLIF file to check for a match. If a match is found, Bliss assigns the library entry to the block, and includes the library entry into the output circuit file.

The comparison is a simple x == y check. However, different logic blocks have different nodes, which makes the comparison a bit complicated. For example, the following declaration defines a 2-input AND gate:

```
.names a b c

11 1
```

The following also defines a 2-input AND gate:

```
.names x y z

11 1
```

Bliss can recognize that both declarations above mean the same, a 2-input AND gate, using the concept of **placeholders**. Bliss considers the two .name declaration above as the same 2-input AND gate using the following structure:

```
blifhead = .names in#0 in#1 out#0
```

The keywords in#0, in#1, and out#0 are placeholders for input nodes and output nodes, respectively. To catch a .names declaration with 4-inputs, blifhead would look like this:

```
blifhead = .names in#0 in#1 in#2 in#3 out#0
```

A peculiar case exists for a .latch declaration in BLIF:

```
.latch din qout re sourceclk 0
```

Where din is the input to the latch, qout is the output, and sourceclk is the clock input to the .latch. Essentially this is a D flip-flop in BLIF.

For such declarations, re and 0 are fixed, with only the input/output nodes changing. If this is the case, the blifhead parameter to match a device with this one should be:

```
.latch in#0 out#0 re clk 0
```

Bliss recoginizes the same in#0 and out #0 parameters. Bliss also recognizes the placeholder, clk, to assign the global clock node for this D flip-flop.

The following summarizes the supported placeholders for blifhead:

in#<n> - Placeholder for inputs, where <n> is the input number, with the first input defined as 0.

out#<n> - Placeholder for outputs, where <n> is the output number. Circuits typically have only one output, out#0.

clk – Placeholder for clock input. Bliss uses the global clock node, currently set as “clk”.

template Structure

The template parameter defines how the the Bliss library entry is instantiated as a SPICE sub-circuit. An instantiation in SPICE looks like:

X<N> param0 param1 ... paramN SUBCKT

Where X is the instantiation keyword in SPICE, <n> is a unique identifier for the instantiation, paramN the nodes of the sub-circuit, and SUBCKT the sub-circuit name declaraion.

In Bliss, <n> is a number uniquely defining the the BLIF logic block, and the parameters are defined using the template structure.

The template structure in Bliss allows customization of how a library entry is instantiated. Consider the example AND2.bl example from section Bliss Library Entry. The template structure for the 2-input AND gate is defined as:

```
template = vdd gnd in#0 in#1 out#0
```

If a .names declaration in blifhead looks like:

```
.names a b c
```

Placeholders would replace the items in the template parameter, and the final instantiation of the AND2 library entry would be:

```
X0 vdd gnd a b c AND2
```

The following summarizes the supported placeholders for template:

in#<n> - Placeholder for input, where <n> is the input number with first input as 0. This is mapped to the same input placeholders defined in blifhead.

out#<n> - Placeholder for output, where <n> is the output number. Circuits typically have one output defined as out#0. This is mapped to the same output place holders defined in blifhead.

vdd – Placeholder for global vdd. Default is “vdd” when written in block instantiation.

gnd – Placeholder for global gnd. Default is “0” when written in block instantiation.

res – Placeholder for global reset. Default is “res” when written in block instantiation.

clk – Placeholder for global clock. Default is “clk” when written in block instantiation.