1 - Introduction to SIS

Laboratory of Computers Architectures

Francesco Setti
Department of Computer Science
University of Verona

October 25-26, 2018

1 About SIS

SIS (Sequential Interactive Synthesis) is an interactive tool for the synthesis and optimization of combinational and sequential circuits. It takes in input a circuit description provided in a high-level format as BLIF, PLA, EQN, or KISS and produces an optimized net-list preserving the input-output behavior of the original circuit. It incorporates a set of logic optimization algorithms that can be used at each stage of the synthesis process.

2 Installation

SIS has been originally developed and distributed by U.C. Berkeley. The last official release is version 1.2 and is available here. This version was written back in 1994, raising several compatibility issues when compiled on modern PCs.

An unofficial version (1.3.6) has been released in 2006 and is available here. Nevertheless, this version also presents compatibility issues with modern Linux distributions, and in particular with last GCC compilers.

A patched version of SIS 1.3.6 is available on github (here), and on the moodle page of this course (here).

Please note that SIS only works on Unix-like environments, this means that it can work on Linux (every distribution) and MacOS. The best option for running SIS on a Windows machine is to use the "Windows Subsystem for Linux", which is an advanced feature for PCs running a 64-bit version of Windows 10 Anniversary Update or later. For a complete guide on how to install it, see here. You can then launch a new Ubuntu shell by running `bash` from a command-prompt.

2.1 Download sources

You can download the `sis.tar.gz` archive from the e-learning page of this course here. To access this page you need to have a valid UniVR account and be registered to the course. Once you have downloaded the file, move to the folder where it is, uncompress and enter the new folder.

```
$ tar xvzf sis.tar.gz
$ cd sis
```

Here you will find a `README.txt` file with detailed instructions for installation.

2.2 Linux (and Windows) users

Here you have two options:

1. run the installation script `sis-installer.sh`

   ```
   $ sudo apt install sis_1.3.6-1_amd64.deb
   ```

   or

2. install the `.deb` package (only for debian distributions)

   ```
   $ ./sis-installer.sh
   ```

2.3 MacOS users

Install the `.pkg` precompiled package.

2.4 UniVR labs

In our labs, SIS is already installed. Despite this, the SIS folder is not included in the system path. You can add the path to your user path by appending it in the `~/.bashrc` file in your home folder. To do that, open the file `~/.bashrc` with super-user priviledges

```
user@mypc: ~$ sudo gedit .bashrc
```

than append the line `PATH="/usr/local/sis-1.3.6/bin/:$PATH"` at the end of file and save it. To use SIS you have now to open a new terminal or rerun `.bashrc` script by

```
user@mypc: ~$ source .bashrc
```

3 The BLIF format

The `BLIF` (Berkeley Logic Interchange Format) is a format for describing a logic-level circuit in textual form. A circuit is an arbitrary combinational or sequential network of logic functions. A circuit can be viewed as a directed graph of combinational logic nodes and sequential logic elements. Each node has a two-level, single-output logic function associated with it.

3.1 Models

A model is a flattened hierarchical circuit. A `blif` file can contain many models and references to models described in other `blif` files. The first model encountered in the main `blif` file is the one returned to the user.

A model is declared as follows:

```
.model <model-name>
.inputs <input-list>
.outputs <output-list>
<command>
      .
      .
      .
<command>
.end
```

The meaning of each non-terminal entity is given below:

`<model-name>` is a string giving the name of the model. If not specified, it is assigned the name of the `blif` file being read.

`<input-list>` is a white-space-separated list of strings (terminated by the end of the line) giving the formal input terminals for the model. Multiple `.inputs` lines are allowed, and the lists of inputs are concatenated. If not specified, it can be inferred from the signals which are not the outputs of any other logic block.

`<output-list>` is a white-space-separated list of strings (terminated by the end of the line) giving the formal output terminals for the model being declared. Multiple `.outputs` lines are allowed, and the lists of outputs are concatenated. If not specified, it can be inferred from the signals which are not the inputs to any other logic block.

`<command>` is one of:

Anywhere in the file a `#` (hash) begins a comment that extends to the end of the current line. Note that the character `#` cannot be used in any signal names. A `\` (backslash) as the last character of a non-comment line indicates concatenation of the subsequent line to the current line. No whitespace should follow the `\`.

NOTE: `blif` sintax is case sensitive(!), which means that keywords like `.model` and `.end` must be lower-case, and you can use upper- and lower-case to name inputs and outputs.

3.2 Logic Gates

A *logic gate* associates a logic function with a signal in the model, which can be used as an input to other logic functions. A logic gate is declared as follows:

```
.names <in-1> <in-2> ... <in-N> <output>
<single-output-cover>
```

`output` is a string giving the name of the gate being defined.

`in-1, in-2, ... in-N` are strings giving the names of the inputs to the logic gate being defined.

`single-output-cover` is, formally, an N-input, 1-output description of the truth table of the logic function implemented by the gate. {0, 1, -} is used in the N-bit wide *input plane* and {0, 1} is used in the 1-bit wide *output plane*. The ON-set is specified with 1's in the *output plane*, and the OFF-set is specified with 0's in the *output plane*.

Example

Consider the `blif` file:

```
.model logfun
.inputs a b
.outputs c
.names a b c
10 1
11 1
.end
```

This model implements the logic function called **logfun** that takes as inputs the signals **a** and **b**, returns the signal **c**, and has the following truth table:

a	**b**	**c**
0	0	0
0	1	0
1	0	1
1	1	1

Note that there are multiple ways to implement the same logic function. The following models are all correct implementations of the above truth table:

```
.model logfun_1
.inputs a b
.outputs c
.names a b c
00 0
01 0
.end
```

```
.model logfun_2
.inputs a b
.outputs c
.names a b c
1- 1
.end
```

```
.model logfun_3
.inputs a b
.outputs c
.names a b c
0- 0
.end
```

```
.model logfun_4
.inputs a b
.names a b c
1- 1
.end
```

```
.model logfun_5
.outputs c
.names a b c
0- 0
.end
```

```
.model logfun_6
.names a b c
1- 1
```

To declare a constant value equal to 0 or 1, you can use the following constructs:

```
.model const0
.outputs out
.names out
.end
```

```
.model const1
.outputs out
.names out
1
.end
```

4 Getting started with SIS

To run SIS on your machine, after following the installation procedure described above or on the UniVR labs, you can simply open a terminal (CTRL+ALT+T) and type sis:

```
user@mypc: ~$ sis
UC Berkeley, SIS 1.3.6 (compiled 2017-10-19 11:32:50)
sis>
```

As usual, you can access the documentation by typing the keywords man or help followed by the command you want to learn about.

```
sis> man <command>
  ...
sis> help <command>
 ...
```

By typing alias, you get the list of all alias of sis commands.

```
sis> alias
1h      state_assign nova -e h
ai      add_inverter
alt     print_altname
asb     resub -a
```

```
c         chng_name
clp       collapse
crit      -m unit-fanout -a -p 2
...
```

To exit from the SIS environment you can type `quit`.

In SIS, arrows and shortcuts does not work. To re-execute the last command you can use `%%`, while the `%chars` re-executes the last command starting with characters *chars*.

A list of useful commands follows:

`read_blif <filename>` reads in a circuit model specified in *filename*.

`write_blif` prints (in terminal) the current circuit model in `blif` format

`write_blif <filename>` writes the current circuit model to file *filename* in `blif` format

`simulate <input-list>` for a given set of values (0 or 1) of the primary inputs in *input-list*, it prints the values produced at each of the primary outputs

`print_stats` prints the current circuit status, which includes the network name, number of primary inputs (pi), number of primary outputs (po), number of nodes (nodes), number of latches (latches), the number of literals in the sum-of-products form (lits(sop)), and the number of states in the state transition graph (#states(STG))

`source <filename>` executes a list of commands contained in a file *filename*