An Introduction to Logic-Sim

1 Logic Simulator

The logic simulator program is named `logic-sim` and it is *only* available on the systems in the MathLab (not on Dooley). If you are login to an ITD system, you need to rlogin to one of the MathLab machines. `logic-sim` reads a circuit from a file and builds the digital circuit. It creates a window and displays the circuit in that window. You can then interact with the circuit by toggling switches in the circuit.

Begin to familiarize yourself by performing the following experiment by getting the file "allgates" in the `demo` directory from the class account and run `logic-sim` on it using:

```
logic-sim allgates
```

(You can re-direct its output/error messages to a file or to /dev/null if you want to discard the messages. If you did redirect the messages to an output file, you can search this file for errors with your favorite editor, looking for errors this way is much easier !)

The command will cause a window to pop up and in the window you will see a circuit with 2 switches in the left column and 7 basic components in the center column: and, or, not, nand, nor, xor (exclusive or) and xnor (exclusive nor, which is the same as not-exclusive or). We did not discuss xor and xnor, so ignore those. The rightmost column contains a set of 7 'probes', which are bascially lights that goes on (filled) if the signal probed is '1' and off (not filled) if it is '0'. You can now interact with the circuit by toggling the switches in the circuit (if you have some switching in the circuit). By the way, the *only* kind of interaction allowed is toggling of switches.

A switch is created with the command "Switch", look for these commands in the file "allgates". You will see these:

```
Switch 1a sw0 ZERO;
Switch 1a sw1 ONE;
```

Each switch is associated with a key. The first switch you create corresponds to the zero ('0') key, the next switch to '1', and so on. You can toggle the switch by hitting the corresponding key on your keyboard, but make sure your mouse cursor is in the circuit window when you do so. Try pressing the ‘0’ and ‘1’ keys and see the effects on the circuit.

The input for the logic simulator follows a simple syntax:

- <component> <coordinates> <in1> <in2> …<out>

  where

  - <component> is the name of the component
  - <coordinates> is the coordinates that the component is placed
  - <in1> <in2> …are names for the input signals (each signal is assigned an unique name)
  - <out> is the name of the output signal

Each command is terminated by a semi-colon. Most components have one or more inputs and one or more outputs. Each input and output must have an **unique** name. One component does not have any outputs: the probe. A probe is a light that will be lit when its input is ONE and will be off when its input is ZERO. One component does not have any inputs: the switch. A switch takes its input from the keyboard as explained above. We will learn more about the different components later in this handout and in project descriptions.

You can have C-type comments embedded in the file. For example, the ‘allgates’ circuit file contains the following:

```
/*  allgates netlist - NOTE: comment */

Switch aa sw0 ZERO;
Switch aa sw1 ONE;

And ab sw0 sw1 and;
Probe ac and;

Or ab sw0 sw1 or;
Probe ac or;

Not ab sw0  not;
Probe ac not;
```

```
Nand ab sw0 sw1 nand;
Probe ac nand   ;

Nor ab sw0 sw1 nor;
Probe ac nor;

Xor ab sw0 sw1 xor;
Probe ac xor;

Xnor ab sw0 sw1 xnor;
Probe ac xnor;
```

The coordinate is given as (row, column) and you can either use letters or digits to indicate a row and a column. So the coordinate grid consists of 26x26 squares if you use latters, and I recommend you use letters (but you will find some examples in the demo directory that use digits). The upper left coordinate is 'aa', the one at it's right is 'ab', and the one below it is 'ba', and so on. Each coordinate identifies a "square" that you can use to place components. For example:

```
Switch aa sw0 ZERO;
```

This command will create a "switch" in the square 'aa'. You can put as many components in a square as you wish:

```
Switch aa sw0 ZERO;
Switch aa sw1 ONE;
```

will stack two switches in the square 'aa'.

If you like a different kind of arrangement other than stacking, then you would need to place the component in a specific coordinate in a square. Yes, within a square (e.g., square 'aa'), you have 26x26 subsquares ! E.g., the coordinate 'aa.aa' refers to the top-left subsquare in the top-left square of the grid.

Table 1 shows a partial list of the available components in the logic simulator (project handouts will explain new components when needed). The component can have one or more inputs and one or more outputs. Each input and output is given an unique name. A name in `logic-sim` begins with a letter, optionally followed by one or more letter, digit, '-' or '_'. The different components are separated by a semi-colon. To connect the output of a component x to the input of another component y, simply use the name of y's output as an input name for component x.

Component	Syntax
Switch	Switch <coord> <output-name> <init-value>
Probe	Probe <coord> <input-name>
Not gate	Not <coord> <input-name> <output-name>
And gate	And <coord> <input-name-1> <input-name-2> ... <output-name>
Or gate	Or <coord> <input-name-1> <input-name-2> ... <output-name>
Nand gate	Nand <coord> <input-name-1> <input-name-2> ... <output-name>
Nor gate	Nor <coord> <input-name-1> <input-name-2> ... <output-name>
D flip-flop	Dff <coord> <set> <D-input> <clock> <clear> <Q-output> <Q'-output>

Table 1: Components and syntax

2 Examples

Example 1:

```
Switch aa sw0 ZERO;
```

defines a switch located at grid coordinate (a,a). The output of this switch is named 'sw0' and the switch is initially off (ZERO). **Only switches can have initial values**. As initial values for a switch, you can use ZERO (off) or ONE (on).

Example 2:

```
Switch aa sw0 ZERO;
Probe "myProbeName" ab sw0;
```

Here, we connect the output of the switch 'sw0' to the input of a probe. **You can label a probe with a name**. This program is found in 'example2' in the demo directory. Try run it and see the effect (remember that the first switch is associated with the 0 key, toggle the 0 key and see the effect).

Note: notice the little '0' at the upper left hand corner in the switch. This indicates that the switch is associated with the key '0'. Now, you can also label a switch with a name, just like we did with the probe. But if you do label a switch, make sure you include the digit that is associated with the switch, because otherwise it would be very difficult to find the key that toggles the labeled switch.

Example 3:

```
Switch aa sw_0 ONE;
     Switch ba sw_1 ONE;
     Switch ca sw_2 ONE;
     And bb sw_0 sw_1 sw_2 out;
     Probe bc out;
```

defines three switches located at grid coordinates (a,a), (b,a) and (c,a). The output of switches one, two and three are named 'sw_0', 'sw_1' and 'sw_2', respectively and they are all initially on. At coordinate (b,b), an 'AND' gate is placed which has three inputs: 'sw_0', 'sw_1' and 'sw_2' (the outputs of the switches). The output of the AND gate is named 'out'. A probe is placed at coordinate (b,c) which use input 'out' (the output of the AND gate). Thus, the probe will display the output of the AND gate. This circuit is found in file 'and3' in the demo directory.

You can play with any digital circuit in the demo directory by copying it over to your directory and run `logic-sim filename`. By hitting the zero ('0') key, you toggle the first switch, '1' the second switch, and so on.

3 Macros

Later in the course, you will need macros to help you build more complex circuits. You may not need it now, just keep these information handy and read about it when you will need it.

You can build more complex components from basic gates and use the complex component in further design using the macros facility in `logic-sim`. The syntax of a macros definition is:

```
Define <comp-name> <input-signals> | <output-signals> ;
   <digital circuit that makes up the component>
Endef;
```

For example, the following is the macros definition for a full adder:

```
Define Full_Adder CarryIn a b | CarryOut Sum;
  Xor aa a b x;
  Xor ab x CarryIn Sum;
  And bb a b y;
  And cb CarryIn x z;
  Or bc-cc y z CarryOut;
Endef;
```

To use new Full_Adder component, use it as:

```
Full_Adder <coordinates> <carry-in> <a> <b> | <sum> <carry-out>
```

Example:

```
Switch aa Cin ZERO;
Switch ba a ZERO;
Switch ca b ZERO;
Full_Adder ab-cd Cin a b | sum Cout;
Probe ae sum;
Probe be Cout;
```

The above macro definition can be found in the file '1-full-adder.a' is the class account's demo directory. The file '4-full-adder' cascades 4 full adders to produce a 4 bit adder circuit (which was shown in class).