INTRODUCTION TO
THE SIM SOFTWARE PACKAGE

Simulation systems are frequently used to design and evaluate digital circuits. In this course, you will use a package named "sim" to study combinational and sequential circuits.

Input to the simulator is C++ source code which describes each component of the circuit being simulated and the connections betweens the components.

Some of the commonly used components are:

```
Not gate -- has a single input and output
And gate -- has a variable number of inputs with a single output
Or gate  -- has a variable number of inputs with a single output
Switch   -- has no inputs but has a single output and an associated key
Probe    -- has only a single input but has no outputs
```

Consider the following example:

```
/*************************************************************************
  Implement the Boolean function F = AB' + BC'D'
*************************************************************************/

#include <Sim.h>                        // Interface to the "sim" package

void simnet()                           // Function must be named "simnet"
{
  Signal a, b, c, d, F;                 // Switch and output objects

  Signal notb, notc, notd;
  Signal and1, and2;                    // Intermediate objects

  // Components and interconnections

  Switch ( SD("a0"), a, 'a' );              // Switch a controlled by 'a' key
  Switch ( SD("b0"), b, 'b' );              // Switch b controlled by 'b' key
  Switch ( SD("c0"), c, 'c' );              // Switch c controlled by 'c' key
  Switch ( SD("d0"), d, 'd' );              // Switch d controlled by 'd' key

  Not ( SD("b1"), b, notb );                // NOT gates
  Not ( SD("c1"), c, notc );
  Not ( SD("d1"), d, notd );

  And ( SD("b2"), (a, notb), and1 );        // AND gates
  And ( SD("c2"), (b, notc, notd), and2 );

  Or ( SD("b3"), (and1, and2), F );         // OR gate

  Probe ( (SD("b4"), "F"), F );             // Probe
}
```

Signals provide the connection between components and must be declared before being used. Some examples:

```
Signal A;                  // 1-bit signal (default)
Signal B (1);              // 1-bit signal
Signal C (4);              // 4-bit signal
Signal D (2, "enable");    // 2-bit signal (labeled as "enable")
```

An optional label may be specified; that label will appear in the X display when the ’F5’ function key is toggled. As a convenience, "sim" supports the "Sig" notation. For example:

```
  Sig (E,8);    // same as "Signal E (8, "E");" -- name and label the same
```

Individual signals within multi-bit signals may be accessed using array-like notation. For example, "C[3]" is a reference to the most significant bit of the 4-bit signal "C", and "C[0]" is a reference to the least significant bit.

Signals may be grouped together into larger units (busses, essentially) using parentheses and commas (and other overloaded operator symbols). Assuming the declarations of "A", "B", "C" and "D" given above:

```
  (A, B, Zero, One, One)  // 5-bit signal <AB011>
  (A, B, C, D)            // 8-bit signal <ABCCCCDD>
```

Note the use of the constant signals "Zero" and "One" in the first example. In the second example, the last two bits of the signal are "D[1]" and "D[0]".

Components must also be declared; the declaration specifies the position of the component in the display, its inputs and its outputs.

The position of a component is based on a coordinate system where the rows are often specified using letters (beginning with a) and the columns are often specified using numbers (beginning with 0). Thus, the position in the upper left corner is "a0". Several components may be placed in the same position.

The position of a component can be specified in several different ways. The most common:

```
  Probe ( SD("c3"), MuxOut );
  Probe ( (SD("c3-d3"), "Mux Output"), MuxOut );
```

The row and column are specified as a character string (two characters only): row c and column 3 of the display, in the first example. In the second example, the declaration specifies the position as a range (rows c and d, column 3), and specifies a label for that component in the display.

Switches are used to generate input values in order to test a function. Each switch is controlled from the keyboard by a specified key. The value produced by a switch (a "0" or a "1") can be toggled by touching the appropriate key. The mouse arrow must be located inside the simulator window for the keys to control the simulator. A switch displays its current value.

Probes are used to check the output values produced by a function. The value displayed by each probe is based on the input values supplied by the switches and the circuit to which the probe is connected.

At every instant, each signal has a value from the set listed below. The meaning of each value is given, along with the way in which a probe will display that value.

```
  UNINITIALIZED:  not initialized ("/")
  ZERO:           de-asserted (background color)
  ONE:            asserted (foreground color)
  XXX:            in transition ("X")
  HIZ:            high impedance state ("-")
```

If a specific value is needed as input to a component, the constant signals "Zero" and "One" should be used, rather than the values listed above.

USING THE SIM PACKAGE

The "sim" package is available only on the CSE Pi array. Thus, you will need to remotely access one of those machines in order to execute "sim".

The circuit simulation package consists of a library of C++ classes and a shell script which compiles and links your source code to the appropriate libraries.

While logged onto one of the machines in the Pi array, the shell script is executed as follows, where "your_file.c" is the name of your source code file:

```
<prompt> /user/cse320/bin/sim your_file.c
```

You may exit the simulator by touching the "F10" function key while the mouse is positioned inside the simulator window. You may also exit by "quitting" on the window. Note that the simulator creates a file named "simex" which is equivalent to "a.out". You may rerun or remove it as you wish.

While the simulator is running, window overlays sometimes erase part of the "sim" display. This can be redrawn by touching the "F12" function key. The simulation can be restarted with initial input values by touching the "F1" function key.

A list of available components may be obtained using:

```
<prompt> /user/cse320/bin/simhelp
```

Information about a specific component may be obtained using:

```
<prompt> /user/cse320/bin/simhelp component_name
```

If you add "/user/cse320/bin" to your search path, the shell scripts can be executed as follows:

```
<prompt> sim your_file.c
<prompt> simhelp
```

One way to update your search path is to copy "/user/cse320/.personal" into the top level directory of your account. Please note that this will replace your existing ".personal" file.

Alternatively, you may edit your existing your ".personal" file:

```
<prompt> ls -al
<prompt> vim .personal
```

Add the following line to your ".personal" file:

```
setenv PATH {$PATH}:/user/cse320/bin
```

If your ".personal" file contains other "setenv PATH" statements, insert this new statement following the existing ones.

To use "sim" during your current interactive session, execute the following command:

```
<prompt> source .personal
```

It will not be necessary to repeat these steps during subsequent sessions.

The course website has additional documentation by the author of the "sim" package:

  http://www.cse.msu.edu/~cse320/Documents/sim_manual.pdf

The "sim" manual lists some keys on the numeric keypad that help control the "sim" display:

  5 -- redraw the screen
  3 -- zoom in one level in the module hierarchy
  9 -- zoom out one level in the module hierarchy

On newer keyboards, those operations are tied to different keys:

  F12   -- redraw the screen
  Page Up   -- zoom in one level in the module hierarchy
  Page Down -- zoom out one level in the module hierarchy

That is, the Function 12 key at the top of the keyboard refreshes the screen, while the "Page" keys on the block of six keys to the left of the numeric keypad control the level of modularization displayed.

The default X window size is often too big for a PC monitor. You can resize the X window using the mouse, or you can re-execute the same program with a different window size. Since "sim" saves the current program in your directory as "simex", you can re-execute the same simulation multiple times. For example:

  <prompt> sim your_file.c   (translate and execute simulation)

  <prompt> simex -g 600x400 (re-execute simulation)

You may need to experiment with the width and height parameters (values for the "-g" option) to get the appropriate size window for your monitor.