Simulating Sub-Circuits

Background

The first installment of the project dealt with the capability to simulate the actions of four very simple gate types, a INV gate, an AND gate, an OR gate and an XOR (exclusive or) gate. The second installment extended that capability by describing the simple logic gates using Verilog, but the Verilog was restricted to descriptions of circuits which only contained a single gate. The third installment allowed Verilog circuits to contain multiple gates. The final installment of the project allows a Verilog circuit to make an instance of another Verilog circuit. For instance, in installment 3, we simulated a two-input multiplexor in MUX2.v that contained multiple gates. It is possible to create a 4 input multiplexor by making three instances of the MUX2 circuit, and wiring them together correctly.

Verilog Infrastructure

There are no changes in the usage of Verilog infrastructure provided to you between the third installment of the project, and the fourth installment, but there have been many internal changes to make the code more efficient, so the name of the support file has changed to verilog3.c – just to ensure you use the latest version. Since the usage of the functions has not changed, the documentation hasn't changed either, but I'm repeating that documentation in the reference section that describes what each function does at the end of this document.

As in installment 3, I have also included a C program called "printCircuit.c" which is an example of how to use the Verilog infrastructure. Look at this code to get some idea of how to use the infrastructure.

You may also look through the code that implements the Verilog infrastructure, but there are some concepts used in that code that we haven't discussed yet. We will talk more about the code when we discuss these concepts.

Project Description

Create a C program called simModule.c. This program is very similar to the simVerilog.c program you created for installment 3. You may start with the code you used for installment 3, and modify it to accommodate the new installment 4 requirements.

The simModule.c program needs to:

1. Check to make sure there are at least 3 command line inputs specified. If not, print a message that indicates what the command line should be, and exit with a non-zero return code.
2. Invoke the Verilog openModule statement passing argv[1] in as the argument, and keeping track of the module number returned. (If there is a problem reading the Verilog file, openModule will exit the program for you, so there is no need to check to see if it worked.)

3. Loop through the remaining command line inputs. Invoke the "atoi()" library function to get the integer value represented by each command line argument, and use that value to set each module input pin in the Verilog circuit to that value, using the order the module input pins appear in the Verilog module header. If there are not enough command line values, print the names of the input module pins which were not set. If there were too many command line values specified, print the values not used in a message.
4. Loop through the circuit as long as some activity occurs, where activity is either setting a pin in the circuit to a known value, or setting a net in the circuit to a known value. Inside this loop you will net to:
    a. Propagate all source pin values to nets.
    b. Propagate all net values to net sinks.
    c. Evaluate all instances and set the instance output pin based on the input values.
5. Print the result to the user using the printModuleIO function.
6. Invoke the freeModule function for your module.
7. If your program worked correctly, return an exit code of zero.

This is the exact same set of instructions as in installment 3. There is only one subtle difference. In 4.c. when I say "Evaluate all instances", in installment 3, those instances were instances of gates. In installment 4, those instances may still be gates, but they may be instances of lower level Verilog circuits. For instance, the MUX4.v Verilog file has an instance statement of the form:

MUX2 sel01(I0,I1,S0,sel01out);

… This instance statement says that instance with instance name "sel01" is an instance of the MUX2 circuit, represented by the file MUX2.v. The I0 net connects to the first module pin of MUX2, the I1 connects with the second module pin of MUX2, the S0 pin connects with the third instance pin of MUX2, and the sel01out net connects with the fourth instance pin of MUX2. You will need to find all the input pins in this instance (and the Verilog infrastructure has pre-read MUX2.v so it knows which pins are inputs and which are outputs), find the values in the MUX4 circuit for those input pins, make a new version of MUX2 by running the openModule function for MUX2, and set the module input pins for MUX2. Then you will need to simulate MUX2. Once that is done, you can then look at the module output pins for MUX2, and use their value to set the instance pin values for the output instance pins in the MUX4 circuit.

This sounds hard, but you already know how to simulate the MUX2 circuit. We did that in installment 3. You just need to figure out how to make the simulation code into a function and invoke that function recursively when there is an instance of a Verilog circuit.

There are some rules about the instances of lower level Verilog circuits, as follows:

1. The instance name in the instance statement should be the name of a Verilog file, except the Verilog file will have a .v suffix.

2. It is invalid to create Verilog with circular references. In other words, if the MUX4.v file instances MUX2, then the MUX2.v file may not reference MUX4.
3. Because of rule 2, eventually you must get to Verilog circuits which contain only gates.
4. If there is an instance of a lower level circuit, but no file exists in the current directory with that name, the results should be unknown. (You may issue an error message and quit if you want to, but I found it easier to continue simulation with the output pin values unknown.)

The following is an example of a correctly coded program:

```
>./simModule AND_2 1 0
AND_2 Inputs: A0=1, A1=0; Outputs: Z=0;
>./simModule MUX2 1 0 1
MUX2 Inputs: I0=1, I1=0, SEL=1; Outputs: Z=0;
>./simModule MUX4 1 0 1 0 0 0
MUX4 Inputs: I0=1, I1=0, I2=1, I3=0, S1=0, S0=0; Outputs: Z=1;
```

Hints and Suggestions

- Don't forget to include "verilog.h" in your program, and to compile using the command:
  ```
  gcc -g -Wall -o simModule simModule.c verilog3.c
  ```
- Use the printModuleState function to see what is set and what is not set to debug your code.
- Use the gdb debugger to help find specific problems in your code.

Project Submission

Upload your simModule.c file in the Project Installment 4 submission area on MyCourses in the Content area under “Project Submissions”.

Project Grading

After the due date, your submission will be graded as follows. The project is worth 100 points. Your code will be compared to all other student’s code using an automated code plagiarism checker that can detect copied code even if you try to “fix” it after copying. Then, your code will be compiled on a BU Linux machine. If there are compiler errors, the professor will attempt to fix your code. If your code can be fixed, the rest of the grading will be performed on that fixed code. Once compiled, your code will be tested with various Verilog circuits that contain one or more gates with various inputs. The results will be compared to correct results, based on the requirements above. Points will be deducted for the following reasons:

Problem	Deduction
Bad submission (e.g. name not simModule.c)	-5 points
Compiler or logic error that can be fixed	-15 points per error
Unfixable error, or >4 errors	-60 points
Compiler warning messages	-5 points per type of warning message
Illegible or Poorly formatted code	Up to -10 points
Incorrect results	-10 points per invocation (up to -40 points)
Late submission	-10 points per every 24 hours late
Code matches another student’s code	-100 points

Verilog Infrastructure Reference

The Verilog Infrastructure consists of a series of functions provided to you. These functions will read a Verilog file, and return information to you about the contents of that file. The functions also provide the capability to specify and retrieve Boolean values on pins and nets in the Verilog circuit defined within the Verilog file.

The infrastructure keeps track of entities in the Verilog circuit, such as modules, pins, nets, and instances using arbitrary numbers which can be thought of a module numbers, pin numbers, net numbers, and instance numbers. I will use variable names "mn", "pn", "nn", and "in" to keep track of the module number, pin number, net number, and instance number in this reference. When there are lists of entities, such as the list of nets in a module, there will be a function that returns the size of that list (e.g. moduleNumNets(mn)). You can then specify an index into that list – any number between 0 and the size of the list – 1 as the "index" into the list. For each list, there is a function which takes an argument that is the index into the list, and returns the number associated with the entity in the list. I will use variable names "pi", "ni", "ii", and "si" to reference the index of a pin in a pin list, the index of a net in a net list, the index of an instance in an instance list, and the index of a sink in a sink list respectively in the following documentation. For instance, to loop through all the nets in the module with module number mn, you can use the following code:

```
for(int ni=0;ni<moduleNumNets(mn);ni++) {
        int nn=moduleNet(mn,ni);
        // Handle net with net number nn
}
```

Top Level Functions

int openModule(moduleName)

- Argument: char * moduleName – the name of a Verilog module.
- Function: Reads file moduleName.v (where moduleName is the argument) and collects all the information about the circuit described in that file in internal tables. If there are any errors, such as file not found or incorrect Verilog data, a message will be printed, and the program will abort. If everything is successful, the module number will be returned for later use.
- Return Value: The module number (mn) to be used in other calls.

char * moduleName(mn)

- Argument: int mn – the module number of a Verilog circuit.
- Function: retrieves the module name of the function
- Return Value: A string (character array) that contains the module name

void printModuleState(mn)

- Argument: int mn – the module number of a Verilog circuit.
- Function: Prints out all known information about the Verilog module mn. To be used to debug – to see all the nets, pins, and instances and all the known information about those entities.

void printModuleIO(mn)

- Argument: int mn – the module number of a Verilog circuit.
- Function: Prints the module name, and the module pins and their values. This is the expected form of output when simulating a gate or a circuit in this project.

void freeModule(mn)

- Argument: int mn – the module number of a Verilog circuit.
- Function: Releases all the information associated with module mn (and recovers space used by that module). After freeing a module, the module number should not be used as arguments to subsequent calls to other infrastructure functions.

Module to Pin Functions

Each module contains both module pins and internal pins. Both are kept in a single list, but the module pins occur first in that list.

int moduleNumModulePins(mn)

- Argument: int mn – the module number of a Verilog circuit.
- Function: Retrieve the number of module pins contained in module mn, so that you can calculate the pin indexes (pi) from 0 to this number to be used in the modulePin function.
- Return Value: The number of module pins in module mn.

int moduleNumPins(mn)

- Argument: int mn – the module number of a Verilog circuit.
- Function: Retrieve the number of pins (both module pins and internal pins) contained in module mn, so that you can calculate the pin indexes (pi) from 0 to this number to be used in the modulePin function.
- Return Value: The number of pins in module mn.

int modulePin(mn,pi)

- Arguments: int mn – the module number of a Verilog circuit.
  int pi – The pin index of a pin in module mn
- Function: Asserts that pi<moduleNumPin(mn), and retrieves the pin number associated with pin index pi in module mn. Note that for very simple Verilog circuits, the pin numbers may match the pin indexes, but in general this will not be true.
- Return Value: The pin number (pn) of the pith pin in module mn.

Pin Functions

The following functions assume you know the pin number (pn).

char * pinName(pn)

- Argument: int pn – the pin number of a Verilog pin.
- Function: asserts that pn is a valid pin number, then retrieves the name of the pin. For module pins, this is the name of the pin itself. For instance pins, this is the instance name, followed by a dot, followed by the name of the pin in the instance. For gate instances, pin names are arbitrarily A0, A1, … for input pins, and Z for the output pin.
- Return Value: A string (character array) that contains the pin name of pin pn.

char pinDirection(pn)

- Argument: int pn – the pin number of a Verilog pin.
- Function: asserts that pn is a valid pin number, then retrieves the direction of the pin.
- Return Value: 'I' if pn is an input pin, 'O' if pn is an output pin.

int pinInstance(pn)

- Argument: int pn – the pin number of a Verilog pin.
- Function: asserts that pn is a valid pin number, then retrieves the instance number of the instance associated with the pin. If the pin is a module pin, it does not have an instance, so a -1 is returned.
- Return value: The instance number (in) of the instance associated with pin pn.

int pinNet(pn)

- Argument: int pn – the pin number of a Verilog pin.
- Function: asserts that pn is a valid pin number, then retrieves the net associated with pin pn. Since all pins in Verilog are connected to a single net, this will return a single result.
- Return Value: The net number (nn) of the net connect to pin pn.

int pinValue(pn)

- Argument: int pn – the pin number of a Verilog pin.
- Function: asserts that pn is a valid pin number, then retrieves the Boolean value of the pin. The value is initialized to -1 (unknown) but can be set by the setPinValue function.
- Return Value: The Boolean value associated with pin pn.

int setPinValue(pn,val)

- Argument: int pn – the pin number of a Verilog pin.
  int val – The Boolean value (should be 0 or 1) to be put on pin pn.

- Function: asserts that pn is a valid pin number. If the pin value matches the val argument, returns zero. Asserts that the pin value is -1 (uknown) because it is an error to set the same pin to a different value. Sets pin pn's value to val, and returns a 1.
- Return Value: A zero if the pin value did not change, or a 1 if the pin value was set to the specified val.

Module to Net Functions

int moduleNumNets(mn)

- Argument: int mn – the module number of a Verilog circuit.
- Function: Retrieve the number of nets contained in module mn, so that you can calculate the net indexes (ni) from 0 to this number to be used in the moduleNet function.
- Return Value: The number of nets in module mn.

int moduleNet(mn,ni)

- Arguments: int mn – the module number of a Verilog circuit.
  int ni – The net index of a net in module mn
- Function: Asserts that ni<moduleNumNets(mn), and retrieves the net number associated with net index ni in module mn. Note that for very simple Verilog circuits, the net numbers may match the net indexes, but in general this will not be true.
- Return Value: The net number (nn) of the nith net in module mn.

Net Functions

The following functions assume you know the net number (nn).

char * netName(nn)

- Argument: int nn – the net number of a Verilog net.
- Function: asserts that nn is a valid net number, then retrieves the name of the net.
- Return Value: A string (character array) that contains the net name

int netSource(nn)

- Argument: int nn – the net number of a Verilog net.
- Function: asserts that nn is a valid net number, then retrieves the pin number of the source pin of net nn.
- Return Value: The pin number (pn) of the source pin of net nn.

int netNumSinks(nn)

- Argument: int nn – the net number of a Verilog net.
- Function: asserts that nn is a valid net number, then retrieves the number of sink pins so that you can calculate the sink indexes (si) from 0 to this number to be used in the netSink function.

- Return Value: The number of sink pins for net nn.

int netSink(nn,si)

- Arguments: int nn – the net number of a Verilog net.
  int si – The sink index of a sink for net nn
- Function: Asserts that si<netNumSinks(nn), and retrieves the pin number associated with sink index si in net nn.
- Return Value: The pin number (pn) of the si^{th} sink of net nn.

int netValue(nn)

- Argument: int nn – the net number of a Verilog net.
- Function: asserts that nn is a valid net number, then retrieves the Boolean value of the net. The value is initialized to -1 (unknown) but can be set by the setNetValue function.
- Return Value: The Boolean value associated with net nn.

int setNetValue(nn)

- Argument: int nn – the net number of a Verilog net.
- Function: asserts that nn is a valid net number. Then retrieves the source pin number for the source pin of this net, and that source pin's value. If the net value matches the source pin's value, returns zero. Asserts that the net value is -1 (uknown) because it is an error to set the same net to a different value. Sets net nn's value to source pin's value and returns a 1.
- Return Value: A zero if the net value did not change, or a 1 if the net value was set to its source pin's value.

Module to Instance Functions

int moduleNumInstances(mn)

- Argument: int mn – the module number of a Verilog circuit.
- Function: Retrieve the number of instances contained in module mn, so that you can calculate the instance indexes (ii) from 0 to this number to be used in the moduleInstance function.
- Return Value: The number of instances in module mn.

int moduleInstance(mn,ii)

- Arguments: int mn – the module number of a Verilog circuit.
  int ii – The instance index of an instance in module mn
- Function: Asserts that ii<moduleNumInstances(mn), and retrieves the instance number associated with instance index ii in module mn. Note that for very simple Verilog circuits, the instance numbers may match the instance indexes, but in general this will not be true.
- Return Value: The instance number (in) of the ii^{th} instance in module mn.

Instance Functions

The following functions assume you know the instance number (in).

char * instanceName(in)

- Argument: int in – the instance number of a Verilog instance.
- Function: asserts that in is a valid instance number, then retrieves the instance name of the instance. Note that the instance name is a unique name within this Verilog circuit of the instance, and is different from the instance module name (see instanceModule ~~instanceModuleName~~).
- Return Value: A string (character array) that contains the instance name

char * instanceModule(in) ~~instanceModuleName(in)~~

- Argument: int in – the instance number of a Verilog instance.
- Function: asserts that in is a valid net number, then retrieves the name of the gate that is being instanced in this instance, which should be one of “and”, “or”, “xor”, or “not”. Note that the instance module name is different from the instance name (see instanceName).
- Return Value: A string (character array) that contains the name of the gate instanced by instance in

int instanceNumPins(in)

- Argument: int in – the instance number of a Verilog instance.
- Function: asserts that in is a valid instance number, then retrieves the number of pins connected to this instance so that you can calculate the pin indexes (pi) from 0 to this number to be used in the instancePin or instanceNet functions.
- Return Value: The number of pins connected to instance in.

int instancePin(in,pi)

- Arguments: int in – the instance number of a Verilog instance.
  int pi – The index of a pin on instance in
- Function: Asserts that pi<instanceNumPins(in), and retrieves the pin number associated with pin index pi in instance in.
- Return Value: The pin number (pn) of the pith pin of instance in.

int instanceNet(in,pi)

- Arguments: int in – the instance number of a Verilog instance.
  int pi – The index of a pin/connection on instance in
- Function: Asserts that pi<instanceNumPins(in), and retrieves the net number associated with connection index pi in instance in.
- Return Value: The net number (nn) of the pith connection of instance in.