CS 211: Computer Architecture
Programming Assignment 5: Circuit Simulator in C

Instructor: Prof. Santosh Nagarakatte

Due: December 10, 2019, 11:55pm

1 Introduction

This assignment is designed to give you some experience in C programming while also increasing your understanding of circuits. You will be writing a C program to simulate the output of combinational circuits.

2 Circuit Description Directives

One of the inputs to your program will be a circuit description file that will describe a circuit using various directives. We will now describe the various directives.

The input variables used in the circuit are provided using the **INPUTVAR** directive. The **INPUTVAR** directive is followed by the number of input variables and the names of the input variables. All the input variables will be named with capitalized identifiers. **An identifier consists of at least one character (A-Z) followed by a series of zero or many characters (A-Z) or digits (0-9)**. For example, some identifiers are IN1, IN2, and IN3. An example specification of the inputs for a circuit with three input variables: IN1, IN2, IN3 is as follows:

```
INPUTVAR 3 IN1 IN2 IN3
```

The outputs produced by the circuit is specified using the **OUTPUTVAR** directive. The **OUTPUTVAR** directive is followed by the number of outputs and the names of the outputs.

An example specification of the circuit with output $OUT1$ is as follows:

```
OUTPUTVAR 1 OUT1
```

The circuits used in this assignment will be built using the following building blocks: **NOT**, **AND**, **OR**, **NAND**, **NOR**, **XOR**, **XNOR**, **DECODER**, and **MULTIPLEXER**.

The building blocks can produce temporary variables as outputs. Further, these building blocks can use either the input variables, temporary variables, a boolean '1' or a '0' as input.

Note: Output variables will never be used as inputs in a building block.

All the temporary variables will also be named with lower case identifiers (i.e., temp1, temp2, temp3, ...).

The specification of each building block is as follows:

- **NOT**: This directive represents the *not* gate in logic design. The directive is followed by the name of an input and the name of an output.

  An example circuit for a NOT gate ($OUT1 = \overline{IN1}$) is as follows.

  ```
  NOT IN1 OUT1
  ```

- **AND**: This directive represents the *and* gate in logic design. The directive is followed by the names of the two inputs and the name of the output.

  An example circuit for an AND gate ($OUT1 = IN1.IN2$) is as follows:

  ```
  AND IN1 IN2 OUT1
  ```

- **OR**: This directive represents the *or* gate in logic design. The directive is followed by the names of the two inputs and the name of the output.

  An example circuit for an OR gate ($OUT1 = IN1 + IN2$) is as follows:

  ```
  OR IN1 IN2 OUT1
  ```

- **NAND**: This directive represents the *nand* gate in logic design. The directive is followed by the names of the two inputs and the name of the output.

  An example circuit for an NAND gate ($OUT1 = \overline{IN1.IN2}$) is as follows:

  ```
  NAND IN1 IN2 OUT1
  ```

- **NOR**: This directive represents the *nor* gate in logic design. The directive is followed by the names of the two inputs and the name of the output.

  An example circuit for an NOR gate ($OUT1 = \overline{IN1 + IN2}$) is as follows:

  ```
  NOR IN1 IN2 OUT1
  ```

- **XOR**: This directive represents the *xor* gate in logic design. The directive is followed by the names of the two inputs and the name of the output.

  An example circuit for an XOR gate ($OUT1 = IN1 \oplus IN2$) is as follows:

  ```
  XOR IN1 IN2 OUT1
  ```

- **XNOR**: This directive represents the *xnor* gate in logic design. The directive is followed by the names of the two inputs and the name of the output.

  An example circuit for an XNOR gate ($OUT1 = \overline{IN1 \oplus IN2}$) is as follows:

  ```
  XNOR IN1 IN2 OUT1
  ```

- **DECODER**: This directive represents the *decoder* in logic design. The directive is followed by the number of inputs, names of the inputs, and the names of the outputs. The output are ordered in **gray code** sequence.

  An example decoder with two inputs $IN1$ and $IN2$ is specified as follows:

  ```
  DECODER 2 IN1 IN2 OUT1 OUT2 OUT3 OUT4
  ```

  $OUT1$ represents the $\overline{IN1}.\overline{IN2}$ output of the decoder, $OUT2$ represents the $\overline{IN1}.IN2$ output of the decoder, $OUT3$ represents the $IN1.IN2$ output of the decoder, $OUT4$ represents the $IN1.\overline{IN2}$ output of the decoder. Note that the outputs of the decoder (i.e., OUT1, OUT2, OUT3, and OUT4) are in gray code sequence.

- **MULTIPLEXER**: This directive represents the *multiplexer* in logic design. The directive is followed by the number of inputs, names of the inputs, names of the selectors, and the name of the output. The inputs are ordered in **gray code** sequence.

  A multiplexer implementing a AND gate ($OUT1 = IN1.IN2$) using a 4:1 multiplexer is specified as follows:

  ```
  MULTIPLEXER 4 0 0 1 0 IN1 IN2 OUT1
  ```

  The above description states that there are 4 inputs to the multiplexer. The four inputs to the multiplexer in gray code sequence are 0 0 1 0 respectively. The two selector input signals are $IN1$ and $IN2$. The name of the output is $OUT1$.

3 Describing Circuits using the Directives

It is possible to describe any combinational circuit using the above set of directives. For example, the circuit $OUT1 = IN1.IN2 + IN1.IN3$ can be described as follows:

```
INPUTVAR 3 IN1 IN2 IN3
OUTPUTVAR 1 OUT1
AND IN1 IN2 temp1
AND IN1 IN3 temp2
OR temp1 temp2 OUT1
```

Note that $OUT1$ is the output variable. $IN1$, $IN2$, and $IN3$ are input variables. $temp1$ and $temp2$ are temporary variables.

Here is another example:

```
INPUTVAR 4 IN1 IN2 IN3 IN4
OUTPUTVAR 1 OUT1
OR IN3 IN4 temp1
AND IN1 IN2 temp2
MULTIPLEXER 4 0 1 0 1 temp2 temp1 OUT1
```

As seen above, a circuit description is a sequence of directives. If every temporary variable occurs as a output variable in the sequence before occurring as an input variable, we say that the circuit description is **sorted**. You can assume that the circuit description files will be sorted.

4 Format of the Input Files

As you will see in the problem statement below, your program will be given one file as input. It contains the description of a circuit using the directives described above.

For example, say that the circuit description file contains the following:

```
INPUTVAR 3 IN1 IN2 IN3
OUTPUTVAR 1 OUT1
AND IN1 IN2 temp1
AND IN1 IN3 temp2
OR temp1 temp2 OUT1
```

5 Common instructions

- You have to a write a C program that takes a file name as command line arguments.
- The file name will be the circuit description file.
- The program should interpret and evaluate the circuit on that assignment, and output the values of the output variables as an order of Gray code.
- The values of the output variables should be space separated and be in the same order as the output variables in the INPUTVAR and OUTPUTVAR directive, e.g., if the circuit description file has the directive INPUTVAR 3 IN1 IN2 IN3, and OUTPUTVAR 3 OUT1 OUT2 OUT3, then the first value should be that of the input variable IN1, IN2, IN3 and output variable OUT1, followed by that of OUT2, and then that of OUT3.
- For every Gray code, the output should be on a new line.

6 The first problem (50 points)

You have to write a program called `first` as described above. You are guaranteed that the circuit descriptions given as input to your program will be sorted. Let's look at an example we have encountered before.

Example Execution 1

Suppose a circuit description file named circuit.txt has the description for the circuit $OUT1 = IN1.IN2 + IN1.IN3$

```
INPUTVAR 3 IN1 IN2 IN3
OUTPUTVAR 1 OUT1
AND IN1 IN2 temp1
AND IN1 IN3 temp2
OR temp1 temp2 OUT1
```

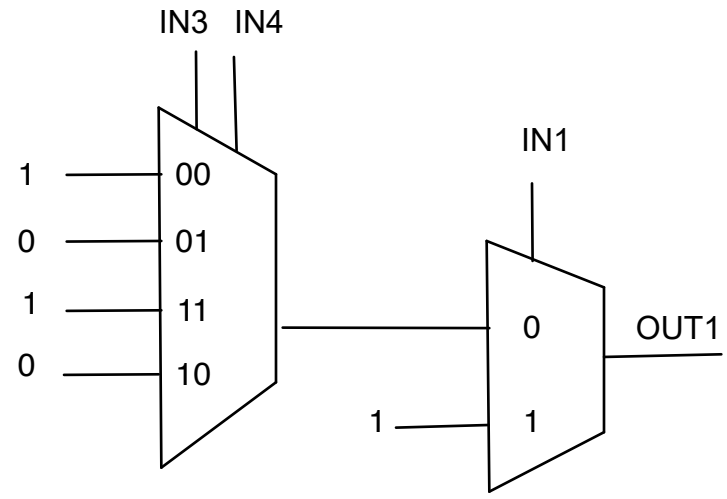

Figure 1: Circuit with Multiplexers from Homework 1

Then, on executing the program with the above circuit description file, your program should produce the following output (one line for each Gray code).

```
./first circuit.txt
0 0 0 0
0 0 1 0
0 1 1 0
0 1 0 0
1 1 0 1
1 1 1 1
1 0 1 1
1 0 0 0
```

The output of the first three columns are INPUTVAR IN1, IN2, and IN3 respectively. And the last column denotes as the OUTPUTVAR OUT1.

Note the use of the temporary variables in the circuit description file to represent intermediate outputs.

Example Execution 2

The circuit description file (circuit.txt) for the circuit in Figure 1 is as follows:

```
INPUTVAR 3 IN1 IN3 IN4
OUTPUTVAR 1 OUT1
MULTIPLEXER 4 1 0 1 0 IN3 IN4 temp1
MULTIPLEXER 2 temp1 1 IN1 OUT1
```

When we execute the program the output should be as follows:

```
./first circuit.txt
```

```
0 0 0 1
0 0 1 0
0 1 1 1
0 1 0 0
1 1 0 1
1 1 1 1
1 0 1 1
1 0 0 1
```

The output of the first three columns are INPUTVAR IN1, IN3, and IN4 respectively. And the last column denotes as the OUTPUTVAR OUT1.

Note the use of the temporary variables in the circuit description file to represent intermediate outputs.

7 The second problem (50 points)

For the second problem, you have to write a program called `second` as described above. For this part, the circuit descriptions given as input to your program **need not be sorted**. Let's take up an example we saw before:

Example Execution 1

Suppose a circuit description file named circuit.txt contains the following:

```
INPUTVAR 4 IN1 IN2 IN3 IN4
OUTPUTVAR 1 OUT1
AND IN1 IN2 temp1
MULTIPLEXER 4 0 1 0 1 temp1 temp2 OUT1
OR IN3 IN4 temp2
```

(Note that the description is **not sorted.**)

Then the execution of the program should result in the following:

```
./second circuit.txt
0 0 0 0 0
0 0 0 1 1
0 0 1 1 1
0 0 1 0 1
0 1 1 0 1
0 1 1 1 1
0 1 0 1 1
0 1 0 0 0
1 1 0 0 1
1 1 0 1 0
1 1 1 1 0
```

```
1 1 1 0 0
1 0 1 0 1
1 0 1 1 1
1 0 0 1 1
1 0 0 0 0
```

The output of the first four columns are INPUTVAR IN1, IN2, IN3, and IN5 respectively. And the last column denotes as the OUTPUTVAR OUT1.

Note the use of the temporary variables in the circuit description file to represent intermediate outputs.

8 Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named `pa5.tar`. To create this file, put everything that you are submitting into a directory (folder) named `pa5`. Then, `cd` into the directory containing `pa5` (that is, `pa5`'s parent directory) and run the following command:

```
tar cvf pa5.tar pa5
```

To check that you have correctly created the tar file, you should copy it (`pa5.tar`) into an empty directory and run the following command:

```
tar xvf pa5.tar
```

This should create a directory named `pa5` in the (previously) empty directory.

The `pa5` directory in your tar file must contain one subdirectory. The subdirectory should be named `first` and `second` (in lower case). Each directory should contain your source files, header files, and a make file. Running the makefile in the `first` folder, should produce the binary `first`, and doing the same in the `second` folder should produce the binary `second`.

Use the autograder to test your submission during development and before submission as you had done with your other assignments.

9 Grading Guidelines

- Your program should work with the provide autograder
- You should make sure that we can build your program by just running `make`.
- You should test your code as thoroughly as you can.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result **up to 100% penalty**. There should be no additional information or newline. That means you will probably not get any grade is you forgot to comment out some debugging message.

Be careful to follow all instructions. If something doesn't seem right, ask.