CS 211: Computer Architecture, Spring 2022
Programming Assignment 4: Circuit Simulator (100 points)

Instructor: David Pham

Due: August 15, 2022 at 11:55pm.

1 Assignment 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.

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 = 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

- **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 $\wedge$ IN2) is as follows:

  XOR 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. Generating gray code should be done using Direct Recursive Gray Code ordering.

- **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. Generating gray code should be done using Direct Recursive Gray Code ordering. 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 begins with the number of directives that describe the circuit. It then contains the description of a circuit using the directives described above. Lastly, it contains numbers to be run through the circuit. The numbers represent a binary vector that assign 1's and 0's to your input variables. If the input value contains more digits than there are input variables, you must truncate the input value to match the input variables. Namely, if there are 3 input variables and the input value is 0b1001, you will truncate it to 3 digits to 0b001 and then assign the values of the indices to the variables. The first input variable will always receive the most significant bit (MSB) while the last input variable will receive the least significant bit (LSB). In this example, the input variables would receive the values 0, 0, 1 respectively. For example, say that the circuit description file contains the following:

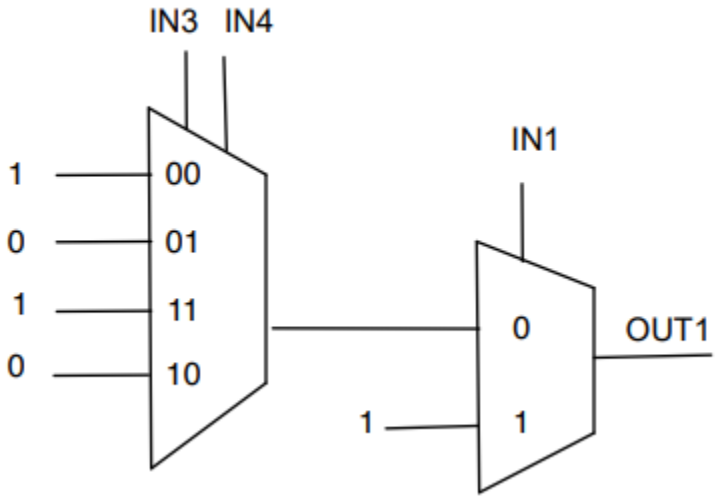
```
5
INPUTVAR 3 IN1 IN2 IN3
OUTPUTVAR 1 OUT1
AND IN1 IN2 temp1
AND IN1 IN3 temp2
OR temp1 temp2 OUT1
2
6
```

The 5 states that their are 5 directives describing the circuit. The next 5 lines will contain the directives in sorted order. Lastly, the following numbers are to be run through the circuit to find the corresponding output.

First

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. The output of your program is the values of the output variables, each seperated by a tab, for each input value.

Figure 1: Figure 1: Example of MUX circuit



Example Execution 1

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

```
5
INPUTVAR 3 IN1 IN2 IN3
OUTPUTVAR 1 OUT1
AND IN1 IN2 temp1
AND IN1 IN3 temp2
OR temp1 temp2 OUT1
2
6
```

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

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

The output of the first three columns are INPUTVAR IN1, IN2, and IN3 respectively for the given input, 2 and 6 respectively. And the last column denotes the value of 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:

```
4
```

```
INPUTVAR 3 IN1 IN3 IN4
OUTPUTVAR 1 OUT1
MULTIPLEXER 4 1 0 1 0 IN3 IN4 temp1
MULTIPLEXER 2 temp1 1 IN1 OUT1
0
7
3
2
```

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

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

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. Also note that the MUX is using gray code ordering.

5 Instruction Summary

- You have to a write a C program that takes a file name as command line arguments.
- The file name will be the circuit and input 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. Generating gray code should be done using Direct Recursive Gray Code ordering.
- 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 input, the output should be on a new line

Structure of your submission folder

All files must be included in the `pa4` folder. The `pa4` directory in your tar file must contain 1 subdirectory. The name of the directoriy should be named first (in lower case). The directory should contain a c source file, a header file (if you use it) and a Makefile. For example, the subdirectory first will contain, first.c, first.h (if you create one) and Makefile (the names are case sensitive).

```
pa4
|- first
   |-- first.c
   |-- first.h (if used)
   |-- Makefile
```

Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named `pa4.tar`. To create this file, put everything that you are submitting into a directory (folder) named `pa4`. Then, `cd` into the directory containing `pa4` (that is, `pa4`'s parent directory) and run the following command:

```
tar cvf pa4.tar pa4
```

To check that you have correctly created the tar file, you should copy it (`pa4.tar`) into an empty directory and run the following command:

```
tar xvf pa4.tar
```

This should create a directory named `pa4` in the (previously) empty directory.

The `pa4` directory in your tar file must contain 1 subdirectory, one each for each of the parts. The name of the directoriy should be named first. Each directory should contain a c source file, a header file (if necessary) and a make file. For example, the subdirectory first will contain, first.c, first.h and Makefile (the names are case sensitive).

AutoGrader

We provide the AutoGrader to test your assignment. AutoGrader is provided as autograder.tar. Executing the following command will create the autograder folder.

```
$tar xvf autograder.tar
```

There are two modes available for testing your assignment with the AutoGrader.

First mode

Testing when you are writing code with a `pa4` folder

(1) Lets say you have a `pa4` folder with the directory structure as described in the assignment.

(2) Copy the folder to the directory of the autograder

(3) Run the autograder with the following command

$python auto_grader.py

It will run your programs and print your scores.

Second mode

This mode is to test your final submission (i.e, pa4.tar)

(1) Copy pa4.tar to the auto_grader directory

(2) Run the auto_grader with pa4.tar as the argument.

The command line is

$python auto_grader.py pa4.tar

Scoring

The autograder will print out information about the compilation and the testing process. At the end, if your assignment is completely correct, the score will something similar to what is given below.

```
You scored
50.0  in  first
Your TOTAL SCORE =  50.0 /50
Your assignment will be graded for another 50 points with test cases not given to you
```

Grading Guidelines

This is a large class so that necessarily the most significant part of your grade will be based on programmatic checking of your program. That is, we will build the binary using the Makefile and source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- **You should not see or use your friend's code either partially or fully. We will run state of the art plagiarism detectors. We will report everything caught by the tool to Office of Student Conduct**.
- You should make sure that we can build your program by just running `make`.
- You should test your code as thoroughly as you can. For example, programs should *not* crash with memory errors.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result **in up to 100% penalty**. Be especially careful to not add extra whitespace or newlines. That means you will probably not get any credit if you forgot to comment out some debugging message.

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