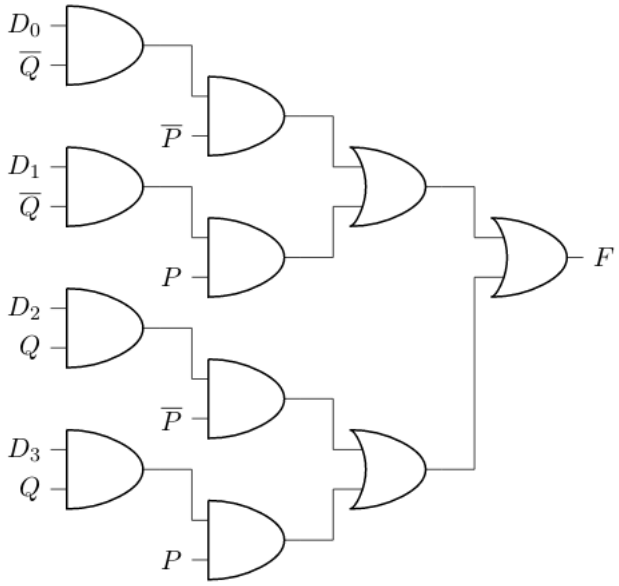
CS-UY 2214 — Recitation 3

Introduction

Complete the following exercises. Unless otherwise specified, put your answers in a plain text file named `recitation3.txt`. Number your solution to each question. When you finish, submit your file on Gradescope. Then, in order to receive credit, you must ask your TA to check your work. Your work should be completed and checked during the recitation session.

Problems

1. Consider the following circuit, having inputs P, Q, D0, D1, D2, and D3; and output F. Note that some of the inputs are NOTted, indicated by a horizontal bar.



   What must be the value of D0, D1, D2, and D3, so that $F = P \bigoplus Q$ (where $\bigoplus$ means XOR)?

2. In the C programming language, the `&` operator denotes bitwise AND, `|` denotes bitwise OR, `^` denotes bitwise XOR, and $\sim$ denotes bitwise NOT. (Distinguish these bitwise operators from their logical counterparts: `&&` for logical AND; `||` for logical OR; and `!` for logical NOT. There is no explicit logical XOR operator in C, although the `!=` operator does the same thing.)

   Calculate the result of the following bitwise operations on 4-bit unsigned decimal numbers. (These are not 2's-complement numbers!) Give your answer in decimal.

   For example, let's calculate `14 & 3`. First, let's convert both numbers to 4-bit binary:

$$\begin{array}{cccccc} & 1 & 1 & 1 & 0 & = 14_{10} \\ \& & 0 & 0 & 1 & 1 & = 3_{10} \\ \hline & ? & ? & ? & ? & \end{array}$$

Now, perform AND on each column:

$$\begin{array}{cccccc} & 1 & 1 & 1 & 0 & = 14_{10} \\ \& & 0 & 0 & 1 & 1 & = 3_{10} \\ \hline & 0 & 0 & 1 & 0 & \end{array}$$

Finally, convert the result back to decimal:

$$\begin{array}{cccccc} & 1 & 1 & 1 & 0 & = 14_{10} \\ \& & 0 & 0 & 1 & 1 & = 3_{10} \\ \hline & 0 & 0 & 1 & 0 & = 2_{10} \end{array}$$

(a) `9 | 3`

(b) `8 ^ 8`

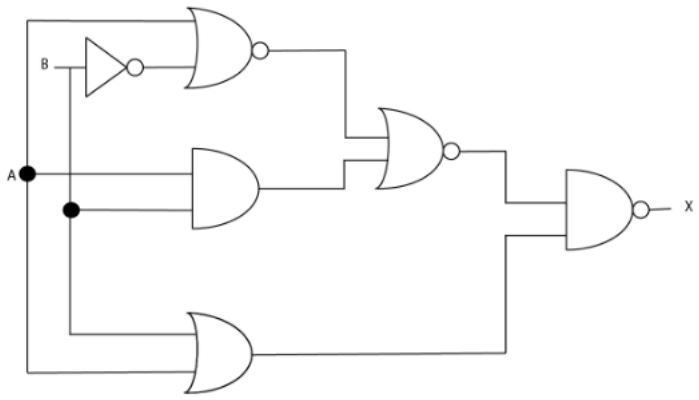
(c) `~6`

Hint: for this one, convert 6 to binary, flip all the bits, and then convert back to decimal. Remember that these are unsigned numbers, not 2's complement, so a 1 in the most significant bit does not signify a negative number.

3. If you haven't yet done so, complete the Verilog tutorial, which you can find on Brightspace. You don't have to submit anything: this is just to familiarize you with the tools.

   You may want to look at the Verilog cheat sheet and reference manual, also available on Brightspace.

4. Consider the following circuit, having inputs A and B, and output X.



   Create a Verilog module named `recitation3a` in a file named `recitation3a.v`. Your Verilog module should implement the above circuit.

   Do not simplify the equation.

   Your module will start like this:

```
module recitation3a(A, B, X);
```

Be sure to declare your ports as `input` or `output`; declare any additional `wire`s you may need; and use `assign` statements to connect ports to gates.

5. Write a combinatorial Verilog module `majority`. Its input, A, is a 3-bit value. Its output, Y, is a 1-bit value, which is set to 1 if and only if the majority of the bits in A are set 1, and to 0 otherwise. Use only Verilog's low-level logical gates (operators `|`, `&`, `^`, $\sim$) in your answer.

   Your module will start like this:

   ```
   module majority(A, Y);
   ```

   Be sure to declare your ports as `input` or `output`; declare any additional `wire`s you may need; and use `assign` statements to connect ports to gates.

   Put your code in a file named `majority.v`. Test your code using the `majority_tb.v` test bench file provided in `recitation3a.zip`. Note that the test bench does not automatically verify the test cases; you'll need to look at the waveform in a waveform viewer in order to determine if your output is correct.

   There are two options for waveform viewers. If you're using Vital or your own Linux machine, you can use GTKWave to view the waveform. If you're using Anubis, use this web site.

   Make sure that your waveform doesn't contain any red (unknown) values. Check with your TA if you aren't sure if your waveform is correct.

   To run the test bench, compile `majority_tb.v` and run the resulting binary in the simulator. View the resulting waveform in a waveform viewer. The following commands will do it:

   ```
   iverilog -o majority_tb.vvp majority_tb.v
   vvp majority_tb.vvp
   ```

   The resulting file `majority.vcd` is your waveform file.

   Submit your module in a file named `majority.v`. Submit a screenshot of your waveform, as displayed in your waveform viewer, in a file named `majority.png` or `majority.jpg`.

6. Complete the following C function, which will return a boolean indicating whether the least significant bit of its parameter is set. That is, `queryLSB(90)` should return `false`, and `queryLSB(45)` should return `true`.

   ```
   bool queryLSB(unsigned int x) {
       return YOUR CODE HERE;
   }
   ```

   Your solution should be one line. Use the C logical operators (`&`, `|`, `^`, $\sim$).

7. Complete the following C function, which will return a value equal to its parameter, with the third least significant bit flipped *on*. That is, `flipOnBit3(90)` should return 94, but `flipOnBit3(45)` should return 45, since the third least significant bit is already on in 45.

   ```
   unsigned int flipOnBit3(unsigned int x) {
       return YOUR CODE HERE;
   }
   ```

   Your solution should be one line. Use the C logical operators (`&`, `|`, `^`, $\sim$).