Programming Project 4 (Digital Circuits, Combinational)

Computer Organization CSI404 Spring 2015

Asg Thu, Feb. 26. It is wise to do Versions 1-4 between now and Tue, Mar. 3 so problems and questions can be addressed in class. A push of all versions, 1-8, is due Tue, Mar. 12, 11:59PM.

Learning Goals and Assignment

How to achieve the goals is detailed after we state them.

1. Demonstrate the meaning of logic gates and combinational circuits (or compound logic gates) by simulating them in C, calling given truth table printing functions, and observing each circuit's truth table.
2. Demonstrate programming C functions with `int` argument variables for values that go into a simulation. Also demonstrate programming an `int *` argument that points to some variable (outside the function) to store the result of the simulation.
3. Practice coding bitwise mask and shift operations in C to bundle several boolean values into one `int` variable.
4. Continue to follow practices of testing while developing, and maintaining your revision history (with GIT) for review and credit. Make GIT track a new file by commanding `git add` *name of new file.*
5. Systematic testing and incremental development **save time in the end** because they reduce the risk of time-wasting bug finding.

   Your sub-tasks are specified with numbered versions like in the previous projects. For full credit, the completion AND TESTING RESULTS of each version must be added with `git add` and put in your repository with `git commit`.
6. Begin to work with C function pointer parameters by reading and revising examples.
7. For versions 1-5, you write C functions that use arguments exactly in ways you are told and test them with functions we wrote for you.

   For versions 6-8, you do some original C function interface design. You will choose yourself how many arguments and how they will be used, and must write yourself functions that are consistent with your choices.

   For versions 6-8, you will also practice a common programming skill: Copy and then revise some given code so it prints nicely formatted output of new but similar kinds of information.
8. Always use `gcc -Wall ...` so `gcc` prints **all warnings** and you can use them to track down mistakes.

You should do AT LEAST 8 commits. Here we go:

(See the FAQ at the end if this doesn't work.) Go into your `COProjects` directory and "pull" this assignment's updates:

```
git pull git404@git404.cs.albany.edu:COProjects
```

You will see the `DigCircs` subdirectory; of course go into it with `cd`

1 Version to verify your compiler and our framework works

Compile the code we gave you and run it, capturing its output in the `typescript` file. (Reminders: Use the `script` command; command `exit` to stop recording; and command `less typescript` to review the output. DON'T FORGET the `git add typescript` and `git commit`.

Now study our code in order to easily do the next version.

2 2-input OR gate plus a 2-input NOR gate built from your OR gate and one of our NOT gates

Based on our `print_TT_`*etc.* and `and` and `nand` examples, code and test your own 2-input `or` and `nor` functions.

IMPORTANT: Maintain `main` so that it ALL THE DIFFERENT truth tables from all the previous versions are also printed. (Grading will begin with your final version, so you will loose points if your final version doesn't print all the truth tables.) RETAIN the gate simulation functions for future use and the testing functions for possible future debugging.

For full credit in this AND SUBSEQUENT, follow our design example and code each of your truth table printouts in a SEPARATE function called by `main`, like our `print_TT_`*etc.* examples.

3 2-input Exclusive OR gate built from NOT, AND and OR gates

Code and test your own `eor` function. All our instructions for the previous version apply here and to subsequent versions. (One way uses two ANDs and one OR and some NOTs, the other uses two ORs and one AND and some NOTs; either is acceptable.)

4 3-to-8 binary decoder

Referring to course material and the `print_TT_3to8` function, code and save the `typescript` of your testing of a simulation of a 3-to-8 binary decoder. Instead of

```
Truth table for a 3 input 8 output ALL ONES gate:
```

your program must print:

```
Truth table for a 3 input 8 output 3-to-8 DECODER gate:
```

and it must actually simulate that decoder gate print the correct truth table.

(ZERO credit for simply hard-coding the answer in the printing function, in this or any other version. In other words, don't fake it.)

5 4-to-1 multiplexor

Referring to course material and the `print_TT_2_3to1` function, code and save the `typescript` of your testing of a simulation of a 4-to-1 multiplexor. Instead of

```
Truth table for a 2+4 input 1-bit output ONE gate:
```

your program must print:

```
Truth table for a 4-to-1 1-bit MULTIPLEXOR gate:
```

and it must actually simulate that multiplexor gate.

6 Half-adder

Do similar things for the half-adder gate. Recall that it has two single bit inputs and two output bits. One output bit is the EOR (or 1 bit sum) of the two input bits. The other output bit is the AND (or carry from adding) of the two input bits.

A half-adder has **TWO** output bits. IN ADDITION TO making your C funtion to simulate a half-adder, you ALSO must make a C function to print a two-input two-output truth table. Do that by coding and calling an additional function like our `print_TT_`*etc.* functions to do the job.

It is YOUR CHOICE whether the two outputs are written (by C's assignment operator in code like `*out =` *what to output*`;`) as multiple bits into one C variable, or single bits into two C variables.

7 Full-adder

Do similar things for the full-adder gate. Your full-adder should be composed of two of your previous half-adders. (Of course, instead of two separate half-adder hardware devices, you will code a C function that calls the half-adder simulator twice.)

Recall that it has three single bit inputs and two output bits. One output bit is the 1 bit sum of the three inputs. The other output bit is the carry bit from that summation. Like for the half-adder, you must also make and call a truth table printer for three input bits and three output bits.

Like for the half-adder, you'll have to decide exactly how the output C code will work, and write (and call) yet another function like our `print_TT_`*etc.*

Tip: The correct truth table will have $2^3 = 8$ rows. The output columns or column will display two bits or one equivalent hex number with two bits (0-3).

8 3-bit adder

Do similar things, using one half-adder and two full-adders from your previous version, to simulate a gate that takes two 3-bit inputs and outputs a total of 4 bits: Three outputs are the sum of when the inputs are considered numbers in binary, and the fourth output is the carry bit from that summatation.

Tip: The correct truth table will have $2^(3+3) = 2^6 = 64$ rows. The output columns or column will display 4 bits or one equivalent hex number with four bits, `0-0xF`

9 Finish!

All your `git commit -a` actions save the latest version of already tracked or added files into **your local repository**. TO GET CREDIT, you MUST push to our server:

```
git push git404@git404.cs.albany.edu:submissions/YourNetID/COProjects   master
```

More Freq. Asked Questions about GIT

- I lost my `COProjects` directory of previous project work, it isn't on the computer I'm using, it is totally messed up. How can I get it back?

  If you did the `git push` as directed, you can get back a copy of the repository you pushed to us! Just (in a directory that doesn't contain `COProjects`) command:

  ```
  git clone git404@git404.cs.albany.edu:submissions/ YourNetID/COProjects
  ```

- Can I get Prof. Chaikens repository where he developed the MIPS cross-compilation demo from the Feb. 19 lecture?

  Yes. In a space NOT UNDER your `COProjects`, do:

  ```
  git clone git404@git404.cs.albany.edu:COExamples
  ```