Programming Project 5 (Digital Circuits, Sequential)
Computer Organization CSI404 Spring 2015

Asg Thursday, Mar 26. The push of versions 1-5 is due Apr.10, 11:59PM

Assignment Summary

The final version will simulate a fixed period kitchen timer or microwave 7-second warmup function. It is very simple: One on/off switch, one 0-to-7 numeric display, and a buzzer. Here is how it should work:

1. When you set the switch to the on position, and keep typing 1 to specify (by your computer input) that the switch should remain turned on, the display counts down from 7 to 0. When the count reaches 0, the buzzer starts and keeps buzzing; and the display stays at zero.

2. Of course, when you run your simulation program with `./a.out`, (it will simulate that) initially, the on/off switch is off, the display is 0, it is not counting, and the buzzer is silent. (That is like powering up a hardware device.)

3. After it starts buzzing, it keeps buzzing as long as you type 1. Typing 1 simulates not touching the switch while it remains in the ON position.

4. To stop the buzzer you must type 0.

5. To time another cycle, you must turn it on by typing 1 again. (You must have stopped it by typing 0 at least once before you can turn it on again.)

6. Furthermore, at any time, you can stop the countdown, stop the buzzing and reset the count to zero by turning the switch to off. (Specify turning it off by typing 0.)

You might now look at an input/output sample at the end of this document.

Learning Goals

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

1. C++ is easy to use as a better kind of C, where you can define and instantiate classes that TIE TOGETHER structured data with functions or methods that operate upon that data. Knowing Java, YOU DON'T HAVE TO KNOW ANYTHING beyond C to do this. You'll just use C's `stdio` library. C++ also checks data types more carefully.

   (Well, you do have to know to name your source file with the `.cc` extension and use the `g++` command instead of `gcc`. Also, we give you the code to allocate our three register instances, to define methods just like in Java, and call them with Java's dot syntax.)

2. A register is a unit of digital electronic hardware that TIES TOGETHER bitwise STORAGE of data with INPUT of new data that will replace the old data it stored when a CLOCK operation occurs.

   You will take our C++ simulation of a 1-bit D-flip-flop kind of register and recode it into a simulation of a 3-bit register.

3. REGISTERS plus a COMBINATIONAL circuit, where the registers change state when the CLOCK ticks, comprise the SEQUENTIAL CIRCUIT design of computers and other digital circuits.

4. Design a simple sequential circuit that, like a CPU, combines control logic with binary arithmetic logic. The final version will simulate our kitchen timer.

   For simplicity and for debugging, your program will wait for you to type each input instead of down-counting itself every second like a real timer.

5. Write basically C code to first SIMULATE and TEST prototypes of simpler circuits that are stages of an incremental development of the timer, and, second complete and test your timer.

6. 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 -a -m "blabla"` (and with informative messages though.)

7. Always use `g++ -Wall ...` so `g++` prints **all warnings** and you can use them to track down mistakes.

You should do AT LEAST 5 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 our new file `seqCirc1.cc` in the `DigCircs` subdirectory; of course go into it with `cd`

1 Verify your compiler and our framework works

Compile the code we gave you and run it, capturing its output in the `typescript` file. (You MUST USE the command `g++ -Wall seqCir1.cc` instead of `gcc` because the code is in C++ instead of C.)
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 -a -m "Framework verified OK"`

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

2 Copy & modify our 1-bit register to make a 3-bit register

Figure out how our C++ code simulates a single D-type master-servant flip-flop which was covered in the course. Note that the TWO operations are separate: (1) A method to set what the new state will be, by storing the new value into the variable that represents the master latch. (2) The clock operation that simulates transferring the value from the master latch to the servant latch, which drives the output.
Your job is to copy and then modify all this code to make a simulated 3-bit register. For software efficiency sake, it should use the bottom 3 bits of single C++ `unsigned int` variables to simulate 3 flip-flops at a time.
You're not finished until you modify the rest of the program and actually use it to test that your 3-bit register works correctly. Do it like our code tested our 1-bit register.
As before, do your final test under the `script` command and make sure the `typescript` file containing the record of your testing is included in the commit.

3 Code simulating a combinational circuit that inputs a 3-bit integer and outputs the 3-bit result of subtracting 1

(When the input is 7, the output should be 6. For input 0 the output should be 7.)
C++ problem (1): `and`, `or` and `not` are **keywords** in C++ but not in C. If you used these names for your own functions, you must rename them, to say `AND`, `OR`, `NOT`.
C++ problem (2): C++ is more picky about the distinction between signed and unsigned int's. Just redeclare every `int` variable to be `unsigned int`, because no `int` should be negative in in this project
Use your work from the previous assignment if you can. **Just subtract 1 by adding 7.**
As before, you get full credit ONLY IF the commit has your record of ACTUALLY TESTING your work!

Your testing code should print the 3 input bits (either in binary form 000, 001, ..., 111 or decimal 0, 1, ..., 7; your choice) AND the 3 output bits, in the same form. Make sure to test the boundary cases and some middle cases. Or write a loop to test them all.

4 Combinational circuit to output whether all 3 input bits are 0

Full credit only if your record of testing is included!

5 Complete your simulated kitchen timer

It should print the initial states of the 3-bit counter register, ON/OFF register (1 bit flip-flop) and buzzer register (another 1 bit flip-flop).

AFTER processing EACH input of 0 or 1, it should simulate the clock tick and then print the new states of the counter, ON/OFF register and buzzer register.

You don't need program it to exit; you can stop it by typing Control-`c` which sends the Unix STOP signal.

It's a good idea to sketch and debug A STATE DIAGRAM (as shown in the lectures and textbook readings) before you get lost in the coding.

The idea is to make the stored ON/OFF state value distinguish two situations:

OFF The timer is ready for the user to command that a new timing cycle shall start by typing a 1, to turn the switch on.

ON The timer is either currently counting down or it is continuing to buzz while displaying the count of 0.

There should be a second D-flip-flop that directly controls whether the buzzer is buzzing or is silent. It MUST BE SEPARATE from the ON/OFF state flip-flop because when the ON/OFF state is ON, sometimes the timer is buzzing and other times the buzzer is silent.

At this point it's wise to carefully STUDY and ANALYZE AGAIN our introductory description of the kitchen timer, and relate all its details with the sample run transcript printed below.

Your final version will simulate what your customers see, so the count must be displayed in the everyday 7, 6, ..., 0 digits, not binary! (The 0s and 1s for the ON/OFF and buzzer states should be printed that way because they are debugging output.)

6 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`

Sample run of the final version.

It's all on the last page.

```
$ g++ -Wall SeqCirc.cc
$ ./a.out
The initial count state is 0
The initial On/Off reg state 0
The initial buzzer state reg state 0
You can stop the program at any time with C-c.
Type 1 to start the timer or keep it going.
Type 0 to stop the timer or keep it stopped.
Input:0
The On/Off reg state 0
Count is 0.  Buzzing is 0.
Type 1 to start the timer or keep it going.
Type 0 to stop the timer or keep it stopped.
Input:1
The On/Off reg state 1
Count is 7.  Buzzing is 0.
(I removed the repetitive prompt lines
 to make the assignment document shorter.)
Input:1
The On/Off reg state 1
Count is 6.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 5.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 4.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 3.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 2.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 1.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 0.  Buzzing is 1.
Input:1
The On/Off reg state 1
Count is 0.  Buzzing is 1.
Input:0
The On/Off reg state 0
Count is 0.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 7.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 6.  Buzzing is 0.
Input:0
The On/Off reg state 0
Count is 0.  Buzzing is 0.
Input:0
The On/Off reg state 0
Count is 0.  Buzzing is 0.
Input:1
The On/Off reg state 1
Count is 7.  Buzzing is 0.
Input:0
The On/Off reg state 0
Count is 0.  Buzzing is 0.
Type 1 to start the timer or keep it going.
Type 0 to stop the timer or keep it stopped.
```

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`