ECE120: Introduction to Computing Optional C Exercise for Week 6

Bit-sliced Logic Simulation

Overview:

In this exercise, you will implement code to simulate a bit-sliced N-bit comparator. You should read three numbers from the console: num1, num2, and N. num1 and num2 are two unsigned values to be compared, and N is the number of bits to compare (your comparator should compare the last N bits of num1 and num2).

Your function should print the output bits from each slice of the bit-sliced logic. The bit-patterns used to represent the three possible results ("<", ">", and "=") are different for each student, so please check the specifics of your unique function in "bit_slice_spec.txt", a file in your subversion working directory. To check out the files, please use the "svn update" command in your own ECE120 working directory, and all the files for this exercise will appear in the folder named "OPTIONAL6". Notice that there are two ways to compare two numbers: starting from **the least significant bit** or from **the most significant bit**. The order that your comparator should be implemented is also specified in "bit_slice_spec.txt".

Specification:

You should implement the main function in the file "bit_slice.c" to do the following:

1. Use printf to print a prompt: "Please input two numbers and number of bits to compare\n".
2. Use scanf to read input values for num1, num2, and N. Please use "%u%u%d" or "%u %u %d" as the format. You should type in "3 1 2", for example, compare the last 2 bits of 3 and 1. You can assume that N is never larger than 32.
3. Use the skills that you learned in class to derive the Boolean function for each output bit of the bit-sliced logic using the bit-pattern specified in "bit_slice_spec.txt".
4. You need a for loop to simulate the comparator. The skeleton code of the loop is given in "bit_slice.c". At each iteration of your loop (each bit-sliced logic), use printf to print the output of that bit-sliced logic. Given the inputs A, B, C1, and C0 (see **Fig. 1** on the next page), you should apply the logic that you derived in step 3 to calculate the outputs P and Q. Then print the output P and Q as a 2-bit unsigned value (PQ) in decimal. A and B are bits to be compared. And C1 and C0 represent the result of the previous bit-sliced logic.
5. To print the output, you should use a printf which only prints out the result (in decimal number) and space. For example, "%u ". You may use as much extra spaces as you like to make your output look nice. Your final output may look like " 1 3 2 2 2\n". Each number is the two-bit output PQ from one bit-slice, treated as an unsigned value and printed in decimal.

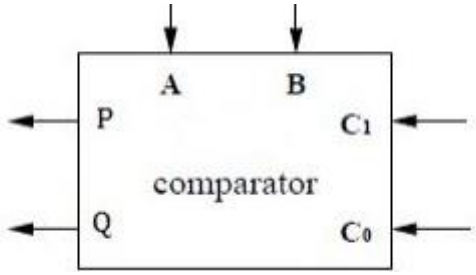

Fig. 1

Based on the bit-patterns specified in bit_slice_spec.txt, you should derive your unique logic of A, B, C1, C0, P and Q. A and B are bits to be compared, and C1 and C0 are from the previous bit slice. Remember to initialize C1 and C0 to appropriate values.

Hints:

- You should first derive the bit-sliced logic on paper and use that logic directly in your code. Remember to use &1 to discard all but the least significant bit after calculating an expression with bitwise operators.
- Keep in mind that the numbers you are comparing are **unsigned** numbers.
- You may find that printing out some variables useful for debugging your code, but please make sure that you remove (or comment out) all extra printfs to get accurate feedback.

Compilation:

Use the command “gcc -Wall -g bit_slice.c -o opt6” to compile your code.
Use “./opt6” to run your code.
Please note that if the code that you submit cannot be compiled, you will receive no feedback.

Checkout and feedback:

Checkout:

Use the "svn update" in your working directory to get the folder “OPTIONAL6” which has all the files. You can check out a new ECE120 working directory using “svn checkout https://subversion.ews.illinois.edu/svn/fa16-ece120/netid ece120”, where netid is your netid.

Commit:

After you commit your code to your svn repository (using the “svn commit” command), our tool will begin to grade your code. The grading should take a couple of minutes.

Feedback:

Please use the "svn update" command to get the feedback after the grading is completed. The feedback will be in separate files named “*.test”. The feedback files are usually one or two sentences indicating your error. Test cases are also available for some errors. For this exercise, we will first test your printf formats, so please make sure that your printfs follow the requirements. For any questions regarding the test cases and this exercise, please email me at beipang2@illinois.edu or to Prof. Lumetta at lumetta@illinois.edu.

If the feedback has only one file “0.test” which says you have passed all the tests, then congratulations, you have completed this exercise.