

Massachusetts Institute of Technology
 Department of Electrical Engineering and Computer Science
 6.111 – Introductory Digital Systems Laboratory

Problem Set 4

Issued: March 5, 2003

Due: March 12, 2003

NOTE: Some files for this problem set are provided in `/mit/6.111/www/s2003/problemsets/ps4`.

As you are now in the middle of taking 6.111, you realize that you are picking up skills that are useful in the real world. You come up with a brilliant idea, and get some of your friends to help you out. Unfortunately, with the economy in its present state, you are unable to get any venture capital funding. Your master plan is almost finished, but you need a way to calculate sales tax (of course, you don't divulge your secrets in this text because you are worried that someone may steal your idea).

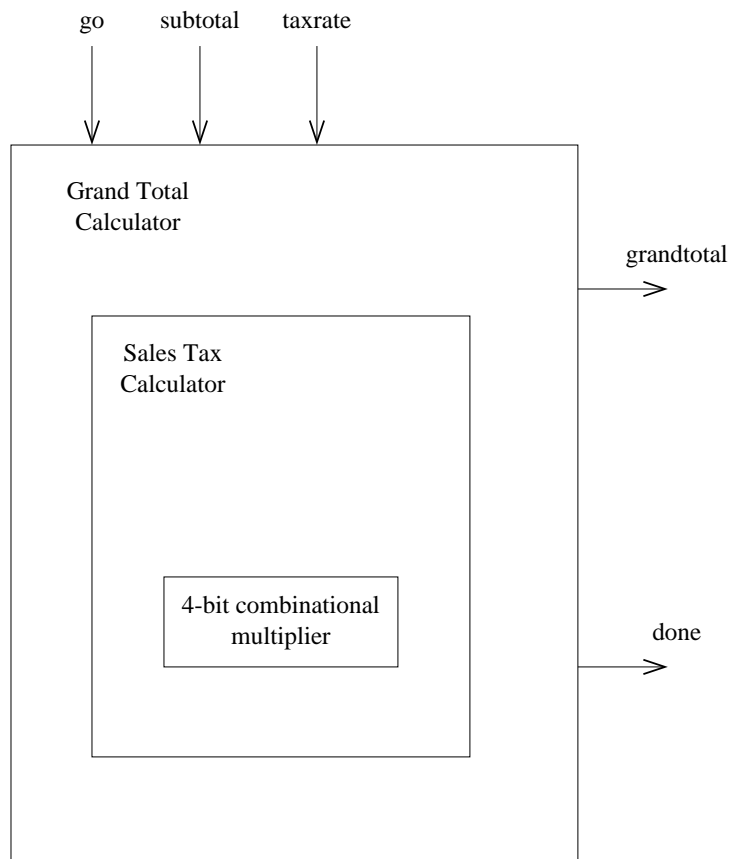
As you are brainstorming an idea for your sales tax problem, you type 'inc' in your personal athena workstation in your dorm room, and are very surprised to see the following subject in your new mail:

6111 03/05 reuse@mit

Cash Register available outside 38-600.

You quickly run to lab, since this is the answer to your problem. However, you find that the cash register is broken and cannot calculate sales tax. You are dumbfounded as to why there would be a broken cash register outside lab. Clearly, the previous owner is not a 6.111 student; any 6.111 student can fix a silly problem like this.

You pop open the back of the cash register, and start to smile, seeing that the problem is indeed easily fixable. You say to yourself, "This is just the type of problem the 6.111 staff would give me for a fourth problem set." You draw the following diagram below.



Luckily, not all of the cash register is broken. Using your mad debug skills, you find that the “Sales Tax Calculator” block is working, and the other two blocks are non-functional. You list the following characteristics about the system:

- The cash register provides `go`, a synchronized pulse that begins the sales tax/grand total computation.
- `subtotal` is an 8-bit number that represents the subtotal for an order.
- `grandtotal` is also an 8-bit number that represents the final total (with sales tax) for an order.
- `taxrate` is a 4-bit number that represents the sales tax percentage for an order.
- `done` is a signal that tells that `grandtotal` is valid for the given `subtotal`. You may assume that `subtotal` and `taxrate` are valid for the entire computation.

Part I - Let’s Do Some Multipliyin’

First, you jump up and down because you realize that lecture called “Arithmetic Structures” on March 5th included a slide on combinational multipliers. You quickly scratch down the following entity:

```
entity fourbitmult is port(
    multA      : in std_logic_vector(3 downto 0);
    multB      : in std_logic_vector(3 downto 0);
    multOUT    : out std_logic_vector(7 downto 0));
end fourbitmult;
```

Write the architecture for the 4-bit combinational multiplier (please use the provided file, `fourbitmult.vhd` as a starting point. Please submit your VHDL code for this part.

Part II - Weird Binary

The Sales Tax calculator is now functional because you program your chip and plug the multiplier into the Sales Tax calculator. You are pretty sure you know what you are doing, but you start to scratch some notes on some paper.

$$\begin{array}{cccccccc}
 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} \\
 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1
 \end{array}$$

With this odd numbering system, you realize you can start representing subtotals as accurate as a quarter. You can use the bottom two bits to represent a half-dollar and a quarter. The example above represents \$26.75.

What monetary value do the following binary numbers represent using the above system?

- 00101101
- 10110110
- 10101000

Next, you check to see that you know how the sales tax is represented as a 4-bit number.

$$\begin{array}{cccc}
 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\
 0 & 1 & 0 & 1
 \end{array}$$

The sales tax above represents a 15.625% sales tax.

What percentages do the following 4-bit numbers represent using this system?

- (d) 0011
- (e) 1001
- (f) 1100

The last thing you want to check for yourself is the output from the sales tax calculator. It is a 12-bit output, generated from the 8-bit and 4-bit product. You remember from grade school mathematics that you can count decimal places to find the decimal placement in the product.

(g) Multiply (a) and (d) in binary, and multiply them in decimal. Show your decimal place placement and confirm that you are right.

Part III - Almost Fixed

Yippee! Now you are sure about how the numbering system works, so you can fix the “Grand Total Calculator.” Build this block by including the “Sales Tax Calculator” as a component. The “Sales Tax Calculator” already includes `fourbitmult` as a component. The `subtotal`, `grandtotal`, and `salestax` vinputs all follow the numbering convention stated in the previous part.

Below is an entity for `salestaxcalc`:

```
entity salestaxcalc is port(
    clk          : in std_logic;
    go           : in std_logic;
    taxrate      : in std_logic_vector(3 downto 0);
    subtotal     : in std_logic_vector(7 downto 0);
    tax          : out std_logic_vector(11 downto 0);
    done         : out std_logic);
end salestaxcalc;
```

You realize that this component is easily controlled with the `clk` and `go` signals. When the `go` signal is asserted, `taxrate` and `subtotal` are multiplied. When the component finishes the task, `done` is asserted, and `tax` is valid as long as `go` is not asserted again.

The following is an entity for `grandtotalcalc`:

```
entity grandtotalcalc is port(
    clk          : in std_logic;
    go           : in std_logic;
    taxrate      : in std_logic_vector(3 downto 0);
    subtotal     : in std_logic_vector(7 downto 0);
    grandtotal   : out std_logic_vector(7 downto 0);
    done         : out std_logic);
end grandtotalcalc;
```

Write the VHDL file for `grandtotalcalc` (no code provided). Be sure that you use the `go` and `done` signals in the same way that `salestaxcalc` uses them.

Print out a simulation and confirm that the system is working appropriately. Submit both the VHDL code and the simulation.

Part IV - Stupid Cash Register

Unfortunately, when you turn on the cash register and test it out, but the results are all wrong! After more debugging, you find out that the “Grand Total Calculator” takes in a two’s complement input. The `subtotal` input actually is a negative number, since it represents how much the customer owes.

You breathe a sigh as you find this out. Any 6.111 student would know that interfaces are very important and can easily cause implementation issues. But of course, since this cash register was not developed in the 6.111 lab, the interfaces are vague.

Since you don't want to change the rest of the system, you find a chip that does a conversion for you. However, you want to be sure you understand how the conversion happens.

Draw a block diagram representing how you would convert two's complement to magnitude, and describe the procedure in a few sentences. Be sure to satisfy all inputs (negative and positive numbers), and have a magnitude and a sign output.

Part V - The Easy Question

This entire time you've been slaving away at this cash register. Now, just what is your brilliant idea? (Oh, and the staff isn't just trying to steal your idea)

Honest, we aren't! :P