

CSE 260M - Homework 7

Due October 18, 2006

1. Write a VHDL description for a full subtractor. Simulate your subtractor to be sure it works.
2. Now use your 1-bit full subtractor to build a 4-bit full subtractor and see that it works for the following cases: A=1000, B=0001; A=0111, B=1000; A=1001, B=0110. Use structural VHDL to instantiate your 1-bit subtractor four times. **Turn in your VHDL code as well as your simulation of the three specific cases, above.**
3. For the following VHDL code, what would be the result of the signal *d*:
 - a.

```
signal a, b, c, d: std_logic;  
a <= '1';  
b <= '0';  
c <= '1';  
d <= (a xor b) and c;
```
 - b.

```
signal a, b, c, d: std_logic;  
c <= '0';  
b <= not(a);  
a <= c;  
d <= not(b);
```
 - c.

```
signal a, b, c, d: std_logic;  
a <= '1';  
b <= '0';  
c <= a xor b;  
process (a,b,c) begin  
    if (a =b) then  
        d <= c;  
    elsif (a = c) then  
        d <= not(b);  
    else  
        d <= not(a);  
    end process;
```
4. Without using a process, write a VHDL model for a 10-to-1 MUX. Be sure to test it to see that it works!

5. Write a VHDL model for a circuit that has two outputs G and F such that G is high when the input BCD digit is prime and F is high when the input BCD digit is divisible by 3. There should be a third output, E , that is high when the input is not a valid BCD digit (and the G and F output values don't matter in this case). Think about writing a model that, when synthesized, would yield a small circuit.

6. Using VHDL, design a multiplier that accepts two 8-bit inputs and produces a correct output. Use whatever techniques you'd like (e.g. think about the shift-and-add algorithm discussed in class) but do not try to do something like $a = b * c$, directly in VHDL. You need to design a circuit to do it. Simulate your circuit for some specific cases of your choosing. Think about what cases are useful and which ones test portions of your circuit that are *edge* cases (such as multiplying by zero, one, having a lot of carries from one bit to the next, etc.). **Turn in your VHDL code, the output from your simulation and a brief discussion of why you chose the cases you did and why you feel this is sufficient.** Clearly document your code so it's clear what your algorithm is and name your inputs and outputs in a consistent and logical way (i.e. don't call the sixteen inputs $a-p$... use a `std_logic_vector`).