LSU EE 4755 **Homework 3** **Due: 28 September 2016**

Problem 1: Module `aa_digit_val`, below, is the solution to Homework 2 Problem 1. It has an 8-bit input `char` and two outputs. Output `is_dig` is 1 iff `char` (an ASCII character) is considered a radix-R digit, where $2 \leq R \leq 16$, is the value of parameter `radix`. Output `val` is the value of that digit (in binary), or zero if it's not a digit.

```
module aa_digit_val
  #( int radix = 10 )
   ( output uwire [3:0] val,   output uwire is_dig,   input uwire [7:0] char );

   uwire  is_dig_09 = char >= "0" && char <= "9";
   uwire  is_dig_af = char >= "a" && char <= "f";
   uwire [3:0] val_raw = is_dig_09 ? char - "0" : char - "a" + 10;
   assign     is_dig = ( is_dig_09 || is_dig_af ) && val_raw < radix;
   assign     val = is_dig ? val_raw : 0;

endmodule
```

Provide sketches of what you expect the inferred hardware to look like for `aa_digit_val` as described below. *Hint: Some problems in the EE 4755 2014 Final Exam dealt with numbers in ASCII representation. The optimizations requested below must go beyond those found in the exam solution.*

(*a*) Show a sketch of the inferred hardware before any optimization is done.

(*b*) Show a sketch of the inferred hardware after some optimization has been performed.

- The sketches must show the product of human thought (in particular, the human who's name is on the submission), not a synthesis program.
- When considering the optimizations for the logic generating `is_dig` (including the logic for `is_dig_09` and `is_dig_af`) recall that in general the cost of logic computing `a==b` is less than the cost of logic computing `a>b`.
- When considering the optimizations for the logic generating `val` think about the subtraction operations and what they actually do when `is_dig` is true. If necessary, work out examples of the subtraction by hand in hexadecimal.

There is another problem on the next page!

Problem 2: Module `aa_full_adder` from Homework 2, Problem 2 adds together two digits of a radix-`R` number represented in ASCII plus a carry in. The module description from the solution appears below.

```
module aa_full_adder
  #( int radix = 10 )
   ( output uwire [7:0] sum, output uwire carry_out, output uwire is_dig_out,
     input uwire [7:0] a, b, input uwire carry_in, input uwire is_dig_in);

   uwire [3:0] val_a, val_b;
   uwire       is_dig_a, is_dig_b;

   aa_digit_val #(radix) dva(val_a, is_dig_a, a);
   aa_digit_val #(radix) dvb(val_b, is_dig_b, b);

   assign is_dig_out = is_dig_in && ( carry_in || is_dig_a || is_dig_b );
   uwire [4:0] sum_val = carry_in + val_a + val_b;
   assign     carry_out = sum_val >= radix;
   uwire [3:0] sum_dig_val = carry_out ? sum_val - radix : sum_val;
   assign sum = !is_dig_out ? " " :
          sum_dig_val < 10  ? "0" + sum_dig_val : "a" + sum_dig_val - 10;

endmodule
```

An obvious objection to an ASCII-coded radix-R adder is that it uses 8 bits to represent a digit that can be represented using only $\lceil \lg R \rceil$ bits.

(a) Show the hardware that might be synthesized for the module `aa_full_adder` based on the description above. This should be the inferred hardware with some optimizations applied. Take care to show the number of bits at the inputs and output of units like adders and comparison logic.

(b) Compare the cost of a d-digit ASCII-coded radix-16 adder to a $4d$-bit ripple adder. (Note that both adders can add numbers in the range of 0 to $2^{4d} - 1$.) Do so by estimating the cost in terms of the number of gates, and state any assumptions, such as the number of gates needed for an x-bit comparison unit.