LSU EE 4755 Homework 2 Due: 22 September 2016

Problem 0: First, follow the instructions for account setup and homework workflow on the course procedures page, `http://www.ece.lsu.edu/koppel/v/proc.html`.

Look through the code in `hw02.v`. It contains partially completed modules for an ASCII-coded radix-R adder. An overview of ASCII-coded adders and the contents of `hw02.v` is given here in Problem 0, where there is nothing to answer. The problem problems start at Problem 1.

Consider first a hypothetical ASCII-coded radix-10 (decimal) adder with two 5-character ($5 \times 8 = 40$ bit) inputs, and a 5-character output. If the strings `___10` and `__418` appeared at the inputs, the string `__428` should appear at the output (the underscores are supposed to be blanks). The adder could be constructed from 5 *ASCII full adders*, which operate analogously to binary full adders. Each ASCII full adder has two 8-bit inputs, an 8-bit output, and a one-bit carry in and carry out. The value output is the sum of the values at the two inputs plus the carry in. The ASCII full adders should be connected to each other in the same way that binary full adders are connected to make a ripple adder.

BCD and ASCII adders have the following design detail that needs to be decided upon: what to do about input that's not a valid digit. For example, what should the radix-10, ASCII adder do for `__x10` + `__234`? For the adders in this assignment, the decision is to end the number at the first non-digit character starting from the right. So `__x10` would be 10 however `__10_` and `_10y` would both be treated as zero because there first digit character is after non-digit characters (starting from the right).

The modules in this assignment try to use inputs `is_dig_in` and `is_dig_out` to indicate whether there is still a run of digits. (There is a small problem in the implementation, a topic for a future assignment. Anyway, the testbench doesn't test for that.)

Here is a summary of the modules in `hw02.v`:

`aa_decimal_digit_val`: Is complete, don't touch. Determines the binary value and validity of an ASCII decimal digit.

`aa_digit_val`: Incomplete, see Problem 1. Should determine the binary value and validity of a radix-R digit. Tested by the testbench.

`aa_full_adder`: Incomplete, see Problem 2. Should add two radix-R ASCII digits.

`aa_width2`: Is complete, don't touch. A two-digit ASCII-coded, radix-R adder. Instantiates two `aa_full_adder` modules. Tested by the testbench. Will not work correctly when `aa_digit_val` or `aa_full_adder` don't work correctly (which is the initial state of the file).

`reference_adder`: Complete, don't touch. A binary adder with the same range as a 2-digit, radix-R adder. It's purpose is to compare the cost and performance of synthesized hardware.

The modules below are used to implement the testbench. Only modify these to help debug your code.

`radtos`: Convert an integer into a radix-R ASCII string.

`aa_test`: Top-level module for the testbench. It instantiates testbenches for `aa_digit_val` and `aa_full_adder` at each radix from 2 to 16.

`aa_test_digit_val`: Test `aa_digit_val` using every possible input.

`aa_test_width2`: Test `aa_width2` using 100 randomly chosen numbers. These numbers only contain digits.

Run the testbench on the unmodified file. It should report errors for `aa_digit_val` and for `aa_width2`.

Note: There are no points for this problem.

Problem 1: Module aa_decimal_digit_val, below, has an 8-bit input char and two outputs. Output is_dig is 1 iff char (an ASCII character) is considered a decimal digit. Output val is the value of that digit (in binary), or zero if it's not a digit.

```
module aa_decimal_digit_val
 ( output wire [3:0] val, output wire is_dig, input wire [7:0] char );
   assign       is_dig = char >= "0" && char <= "9";
   assign       val = is_dig ? char - "0" : 0;
endmodule
```

Originally module aa_digit_val (see hw02.v) is the same as aa_decimal_digit_val. Modify aa_digit_val so that it honors the value of its radix parameter. That is, modify it so that is_dig is 1 iff char (an ASCII character) is considered a digit in radix radix and so that val is the value (in binary) of that digit. The module should work correctly for all radices from 2 to 16. For radices ≥ 10 only use lower-case letters for alphabetic digits. Please don't change the width of val.

Run the testbench (press F9) to check whether aa_digit_val is running correctly and make sure that it is synthesizable.

To check for synthesizability of a module follow the Verilog Synthesis steps given on the procedures page up to and including the elaborate command. There should be no warnings. The synthesis script can be run with the command `rc -files syn.tcl`, it's purpose will be described in the next homework.

Problem 2: When completed module aa_full_adder is supposed to add together two digits of a radix-R number represented in ASCII plus a carry in. Output sum of the module is the ASCII digit of the sum, and output carry_out is 1 iff there is a carry.

Complete module aa_full_adder so that it operates as described. The module should instantiate two aa_digit_val modules and use them to generate the sum digit. The module must be synthesizable, it can be written using implicit structural or behavioral code.

Run the testbench to verify correct functioning.

To check for synthesizability of a module follow the Verilog Synthesis steps given on the procedures page up to and including the elaborate command. There should be no warnings. The synthesis script can be run with the command `rc -files syn.tcl`, its purpose will be described in the next homework.