EEC 281 - Homework/Project #2

Work individually, but I strongly recommend working with someone in the class nearby so you can help each other when you get stuck, with consideration of the Course Collaboration Policy. Send an email to me if something is not clear and I will update the assignment using green font.

- **Submit:** (1) all code you wrote (not generated or provided files) including verilog hardware, verilog testing, matlab, etc. (2) other requested items such as diagrams etc.

   i. Upload a single pdf to https://canvas.ucdavis.edu/.

   ii. Place all of your answers and code into a single pdf file with all problems and material *in order* (i.e., problem 1, problem 2,...).

   iii. Add titles to pages and file names so it is clear to which problem they belong. For example, Problem 1, prob1.v, prob1.vt,...

- **Diagrams.** If a problem requires a diagram, include details such as datapath, memory, control, I/O, pipeline stages, word widths in bits, etc. There must be enough detail so that the exact *functional* operation of the block can be determined by someone with a reasonable knowledge of what simple blocks do. A satisfactory diagram may require an extra-large sheet.

- **Verilog.** If a problem requires a verilog design, turn in copies of both hardware and test verilog code.

  - *** Where three '*'s appear in the description, perform the required test(s) and turn in a printout of either:

    1. a table printed by your verilog testbench module listing all inputs and corresponding outputs,

    2. a simvision waveform plot which shows (labeled and highlighted) corresponding inputs and outputs, or

    3. verilog test code which compares a) your hardware circuit and b) a simple Golden Reference circuit (using high-level functions such as "+"). Include two copy & paste sections of text from your simulation's output (one section showing a large number of passes, and one small section showing where you purposely make a *very small* change to either your designed hardware circuit or your reference circuit to force the comparison to fail). It should look something like this:

       ```
       input=0101, out_hw=11110000, out_ref=11110000, ok
       input=0111, out_hw=11110001, out_ref=11110001, ok
       ...

       input=0101, out_hw=11110000, out_ref=11110001, Error!
       ...
       ```

    For 1 and 3, the output must be copied & pasted directly from the simulator's output without any modifications.

  - In all cases, **Show how you verified** the correctness of your simulation's outputs.

  - Keep "hardware" modules separate from testing code. Instantiate a copy of your processing module(s) in your testing module (the highest level module) and drive the inputs and check the outputs from there.

  - Your verilog must implement *hardware* and be cleanly synthesizable, and follow guidelines in the verilog handouts. For example, having a `for` loop in your "hardware" verilog will result in an automatic **80%** reduction in points since the verilog is therefore not implementing hardware.

- **Synthesis.** If a problem requires synthesis, turn in copies of the following. Print in a way that results are easy to understand. Delete sections of many repeated lines with a few copies of the line plus the comment: `<many lines removed>` .

   1. `dc-<filename>.tcl` (or equivalent)
   2. `*.area` file
   3. `*.logs` file (not command.log) Edit and reduce "Beginning Delay Optimization Phase" and "Beginning Area-Recovery Phase" sections.
   4. `*.tim` file; first (longest) path only

  The "`always @(*)`" verilog construct may be used.

Run all compiles with "medium" effort. Do not modify the synthesis script except for functional purposes (e.g., to specify source file names).

- **Functionality.** For each design problem, clearly state: 1) whether the design is fully functional, and 2) the failing sections if any exist.
- **Point deductions/additions.** `TotalProbPts` is the sum of all points possible.
    - [Up to `TotalProbPts × 50%`] point reduction for not plainly certifying/showing that your circuit is **functionally correct**. This sounds drastic but you should have checked the correctness of your circuits' outputs anyway, it is impractical for the grader to check every result of every submission by eye, and thus an un-certified design will be treated like a marginally-functional design after a cursory glance at the hardware. In the worst case if there is no indication a design works at all, zero points will be given.

      Following is an example of a fine way to certify correctness, if the "Y/N" is written either a) by hand individually for each test or b) automatically with a golden reference checker; but not printed automatically without individual checking.

      ```
        inA       inB     outExp  outMantissa     I Certify Correct
      -------- --------  ------ --------------   -----------------
      10101100 00110101  110010 01100110100101           Y
      00000101 10110101  101010 01010101010101           Y
      01010100 11101010  010100 11010101100101           no    // this indicates I recognize there is an error here
      ```

    - [Up to `TotalProbPts × 10%`] point reduction if parts of different problems are mixed up together (please don't do it; it makes grading much more difficult than you probably realize)
    - [Up to `TotalProbPts × 10%`] extra credit will be given for especially thorough, well-documented, or insightful solutions.
- **Clarity.** For full credit, your submission must be easily understandable and well commented. Print code and CAD reports using a mono-spaced font (e.g., `Courier`) and a small size such as 9 point.

Total: 235 points

For this homework/project, you may use the "`always @(*)`" verilog construct but keep an eye out for any situations where Design Compiler is not compatible with it.

1. [20 pts] The purpose of this problem is to familiarize you with the synthesis process and to give you a rough feeling for the size of a few simple circuits in our standard cell library's technology. Copy the files from the DC tutorial (see link on the main EEC281 page) to get started. Synthesize the following circuits and report their total cell area. Do not include registers (flip-flops). Also, do not declare any wires or registers as "signed", but assume words are all 2's complement unless stated otherwise. No need to simulate, but your verilog must compile correctly (run "make check"). Also, for this problem, do not worry if designs do not meet timing (negative slack time).

   Turn in: 1) source verilog for each circuit, 2) totals in a single table so it can be used as a note sheet in the future. Do not submit any output synthesis reports.

   a) [2 pts] bitwise NOR of two 12-bit numbers (12-bit output)

   b) [2 pts] One 3:2 adder using verilog "`&`" "`|`", "`^`", "`~`". Draw your circuit and the circuit output by DC.

   c) [2 pts] One 3:2 adder using verilog "+".

   d) [3 pts] 12-bit carry-propagate adder (13-bit output). Use "+" in verilog. Write the circuit's maximum delay in your table.

   e) [5 pts] 12-bit carry-save adder that adds three 12-bit words into a single carry-save output. Write the circuit's maximum delay in your table.

   f) [3 pts] 8-bit x 8-bit unsigned multiplier (16-bit output). Use "*" in verilog.

   g) [3 pts] 16-bit x 16-bit unsigned multiplier (32-bit output). Use "*" in verilog.

2. [20 pts] Build a ripple-carry adder with 12-bit inputs and 12-bit output using full adders from part 1(c). Register all inputs and outputs (to make synthesis timing accurate).

a) [10 pts] Write design in verilog, test with at least 15 test cases in one simulation with a unique set of inputs calculated each clock cycle. Verify using method ***(3).

b) [10 pts] Synthesize the design with a high clock frequency to find the maximum clock rate. State the maximum clock rate and corresponding area. Submit *.area and *.tim (longest path only) reports only.

3. [30+10 pts] Repeat Problem 2(a)(b) with a carry-select adder composed of two 6-bit sections.

4. [30+10+10 pts] Repeat Problem 2(a)(b) with a carry-select adder composed of three sections whose widths are chosen to minimize delay.

   c) [10 pts] Justify the partitioning you chose.

5. [50+10 pts] Repeat Problem 2(a)(b) but pipeline the ripple-carry adder into 3 pipeline stages.

6. [5 pts] Write a single table with 1) max clock frequency and 2) area for problems 2–5.

7. [40 pts] Find the maximum possible clock frequency for a pipeline stage with no logic; that is, one register connected directly to another register.

   a) [25 pts] Appropriately synthesize one flip-flop connected to a second flip-flop to find the desired value. In a few sentences, state how you did this.

   b) [15 pts] Report each component of the minimum clock cycle time.
