EEC 281 - Homework/Project #3

Winter 2020

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. Please send me email if something is not clear and I will update the assignment using green font.

Notes:

- **Submit:** (1) all *.v hardware and necessary testing code you wrote (no generated or provided files), and (2) other requested items such as diagrams:

  i. A paper copy of (1) and (2) [instructions], and

  ii. An electronic copy of (1) uploaded to Canvas (under "Assignments") in a tar or zip file. Label directories or files so it is clear to which problem they belong. For example, 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 your diagram and explanation, and a reasonable knowledge of what simple blocks do. A satisfactory diagram may require multiple pages of paper taped together into a single large sheet.

- **Verilog.** If a problem requires a verilog design, turn in paper 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 reference circuit (using high-level functions such as "+")—no third circuit. Include two copy & paste sections of text from your simulation's output (one for pass, and one for fail where you purposely make a *very small* change to either your designed hardware circuit or your reference circuit to force the comparison to fail) that look something like this:

```
input=0101, out_hw=11110000, out_ref=11110000, 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.

- **Synthesis.** If a problem requires synthesis, turn in paper copies of the following. Print in a way that results are easy to understand but conserves paper (multiple files per page, 8 or 9 point font, multiple columns). Delete sections of many repeated lines with a few copies of the line plus the comment: `<many lines removed>` .

  1. `dc_compile` (or equivalent)
  2. `*.area` file
  3. `*.log` file; 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 but keep an eye out for any situations where Design Compiler may not be compatible with it.

  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, you must write by hand 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. Here 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.

```
  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 readable, understandable, and well commented.

---

1. [20 pts] Using matlab, write a function which calculates the minimum number of partial products needed to calculate the product of a number multiplied by a fixed number. Both *+multiplicand* and *–multiplicand* partial products are allowed. Consider positive and negative numbers with a resolution of 0.5 (e.g., 0, 0.5, 1.0, ...). For example, a multiplicand of +5.5 should result in 3 partial products (either +4, +2, –0.5; or +4, +1, +0.5).

   Use your own algorithm, or try an algorithm that works like this:

   - For this discussion, "power of two" means both +2^k and –2^k where k is some integer.
   - Assuming the number is not a power of two (trivial solution), start with the powers of two just smaller and just larger than the input number. For example, if the input is 56, consider 32 and 64. Note the following matlab commands and results:
     ```
     log2(56) → 5.8074
     ceil(log2(56)) → 6
     floor(log2(56)) → 5
     abs(-56) → 56
     ```
   - Choose whichever number is closer. In this example, choose 64.
   - Subtract that power of two and repeat the procedure with the new number until the remainder is reduced to zero. For example, choose 56 – 64 = –8 so now use –8.

   a) [15 pts] Write the described function in matlab.

   b) [5 pts] Assuming your function is called `numppterms()`, run the following bit of matlab code (a few points need fixing), submit the figure, and report the Total Sum for all numbers 0.5 – 100.00 .

   ```
   StepSize = 0.5;
   NumTermsArrayPos = zeros(1, 100/StepSize);    % small speedup if init first
   for k = StepSize : StepSize : 100,
      NumTermsArrayPos(k/StepSize) = numppterms(k);
   end

   fprintf('Total sum for +0.5 - +100.00 = %i\n', sum(NumTermsArrayPos));

   figure(1); clf;
   plot(StepSize:StepSize:100, NumTermsArrayPos, 'x');
   axis([0 101 0 1.1* max(NumTermsArrayPos)]);
   xlabel('Input number');
   ylabel('Number of partial product terms');
   ```

---

2. [30 pts] An FIR filter has the coefficients,

   ```
   coeff = [26   -43   317   696   317   -43   26];
   ```

   Assume the coefficients cannot be scaled larger, but they can be scaled smaller (up to 50% smaller) without a change to the rest of the system. This implementation works with integers only, so round() all scaled coefficients.

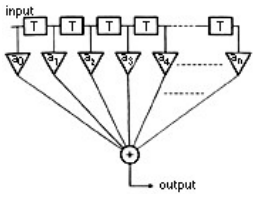
   a) [5 pts] How many partial products are necessary to implement the FIR filter with the given coefficients?

   b) [5 pts] Find the scaling for the coefficients that yields the minimum number of partial products.

   c) [5 pts] Turn in a plot of the number of required partial products vs. the scaling factor. The plot should look something like the bogus results this matlab code generates.

   ```
   figure(1); clf;
   plot(0.5:0.001:1.0, round(5* rand(1,501)+1), 'x');
   axis([0.48 1.02 0 6.5]);  grid on;
   xlabel('Scaling factor');
   ylabel('Number of partial product terms');
   title('EEC 281, Hwk/proj 3, Problem 2, Plot of bogus results');
   ```

   d) [15 pts] Draw dot diagrams showing how the partial products would be added (include sign extension) for the optimized coefficients you found in (b), using the FIR architecture shown below and an 8-bit 2's complement input word. Use 4:2, 3:2, and half adders as necessary and no need to design the final stage carry-propagate adder.

---

3. [55 pts] Design of an area-efficient low-pass FIR filter. The filter must meet the following specifications when its sample rate is 100 MHz.

   - Passband below 8.0 MHz: no more than 3dB ripple from minimum to maximum levels
   - Passband below 12.0 MHz: no more than 4dB attenuation below gain level at DC
   - Stopband above 18.0 MHz: at least 21dB attenuation below gain level at DC
   - Stopband above 25.0 MHz: at least 25dB attenuation below gain level at DC

a) [10 pts] Write a matlab function `lpfirstats(H)` that takes a frequency response vector H (or a vector of filter coefficients) as an input and returns the four critical values listed above (passband ripple, etc.).

b) [10 pts] Either by hand or with a matlab function, repeatedly call `lpfirstats(H)` to find a reasonable small area filter. There is no need to write a sophisticated optimization algorithm, just something reasonable that does more than simple coefficient scaling. For example, making small perturbations to the frequency and amplitude values that remez() uses such as using 0.01 and other small values instead of 0.00 in the stopband.

It may be helpful to use the following matlab code. Remember that matlab vectors start at index=1 so H(1) is the magnitude at frequency=0.

```
coeffs1   = remez(numtaps-1, freqs, amps);
coeffs2   = coeffs1*scale;
coeffs    = round(coeffs2);
[H,W]     = freqz(coeffs);
H_norm    = abs(H) ./ abs(H(1));
[ripple, minpass, maxstoplo, maxstophi] = lpfirstats(H_norm);
```

Assume area is: `Total_num_partial_products + 2*Num_filter_taps`

Examples from a previous year of the difference between good optimizations and weaker ones--these are class results for ten students for a different filter than the one assigned here:

```
109 area, 31 taps,  47 PPs
109 area, 31 taps,  47 PPs
113 area, 33 taps,  47 PPs
114 area, 33 taps,  48 PPs
123 area, 33 taps,  57 PPs
128 area, 34 taps,  60 PPs
182 area, 55 taps,  72 PPs
221 area, 59 taps, 103 PPs
221 area, 59 taps, 103 PPs
250 area, 61 taps, 128 PPs
```

c) Provide the following for your smallest-area filter in your paper submission.

i) [5 pts] Filter coefficients

ii) [10 pts] The number of taps, number of required partial products, area estimate, and the attained values for the four filter criteria in dB.

iii) [10 pts] A plot made by: `plot_one_lpfir.m` (that *requires* updating) to show the filter's frequency response.

iv) [5 pts] A `stem()` plot of the filter's coefficients.

d) [5 pts] Upload to canvas: i) Your modified version of `plot_one_lpfir.m`, and
ii) Filter coefficients for your smallest-area filter in a matlab-readable vector in a text file called `coeffs.m`

```
% coeffs.m
coeffs = [1 4 -8 25 -8 4 1];
```

EEC 281 | B. Baas | ECE Dept. | UC Davis

Updates:
2020/02/18 Posted