Name _____ *Formatted For Two-Sided Printing*

Staple This Side

Digital Design using HDLs
LSU EE 4755
Final Examination

Thursday, 7 December 2023 15:00-17:00 CST

Problem 1	_____	(28 pts)
Problem 2	_____	(25 pts)
Problem 3	_____	(27 pts)
Problem 4	_____	(20 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Staple This Side

Problem 1: [28 pts] Appearing below is the solution to Homework 5.

(*a*) On the facing page show the inferred hardware for an instantiation with `n=4`.

(*b*) Explain why the cost of the hardware corresponding to the line `n_match += match` is much lower than one would expect for hardware performing `wc`-bit addition.

☐ The `n_match += match` is much less expensive because:

```
module uniq_vector_seq
  #( int we = 10, n = 4, wc = $clog2(n+1) )
   ( output logic [n-1:0] uniq_bvec,     output logic [wc-1:0] n_match,
     input uwire [we-1:0] element,       input uwire start, clk );

   logic [we-1:0] elements [n-1:0];
   logic [n-1:0] occ_bvec;
   logic [wc-1:0] uniq_at [n-1:0];

   always_ff @( posedge clk ) begin

      automatic logic [wc-1:0] match_pos = n;
      n_match = 1;

      for ( int i=n-1; i>=1; i-- ) begin

         automatic logic next_occ_bvec = !start && occ_bvec[i-1];
         automatic logic match = next_occ_bvec && element == elements[i-1];

         n_match += match;

         if ( match ) match_pos = i;

         elements[i] <= elements[i-1];
         occ_bvec[i] <= next_occ_bvec;

         uniq_at[i] <= match ? n : uniq_at[i-1];
         uniq_bvec[i] <= !next_occ_bvec || !match && i >= uniq_at[i-1];

      end

      elements[0] <= element;
      occ_bvec[0] <= 1;
      uniq_at[0] <= n - match_pos;
      uniq_bvec[0] <= match_pos == n;
   end

endmodule
```


Staple This Side

☐ Show inferred hardware for `n=4`.

☐ Do not confuse ports with parameters. ☐ Do not confuse elaboration-time computation with computation hardware.

Staple This Side

Staple This Side

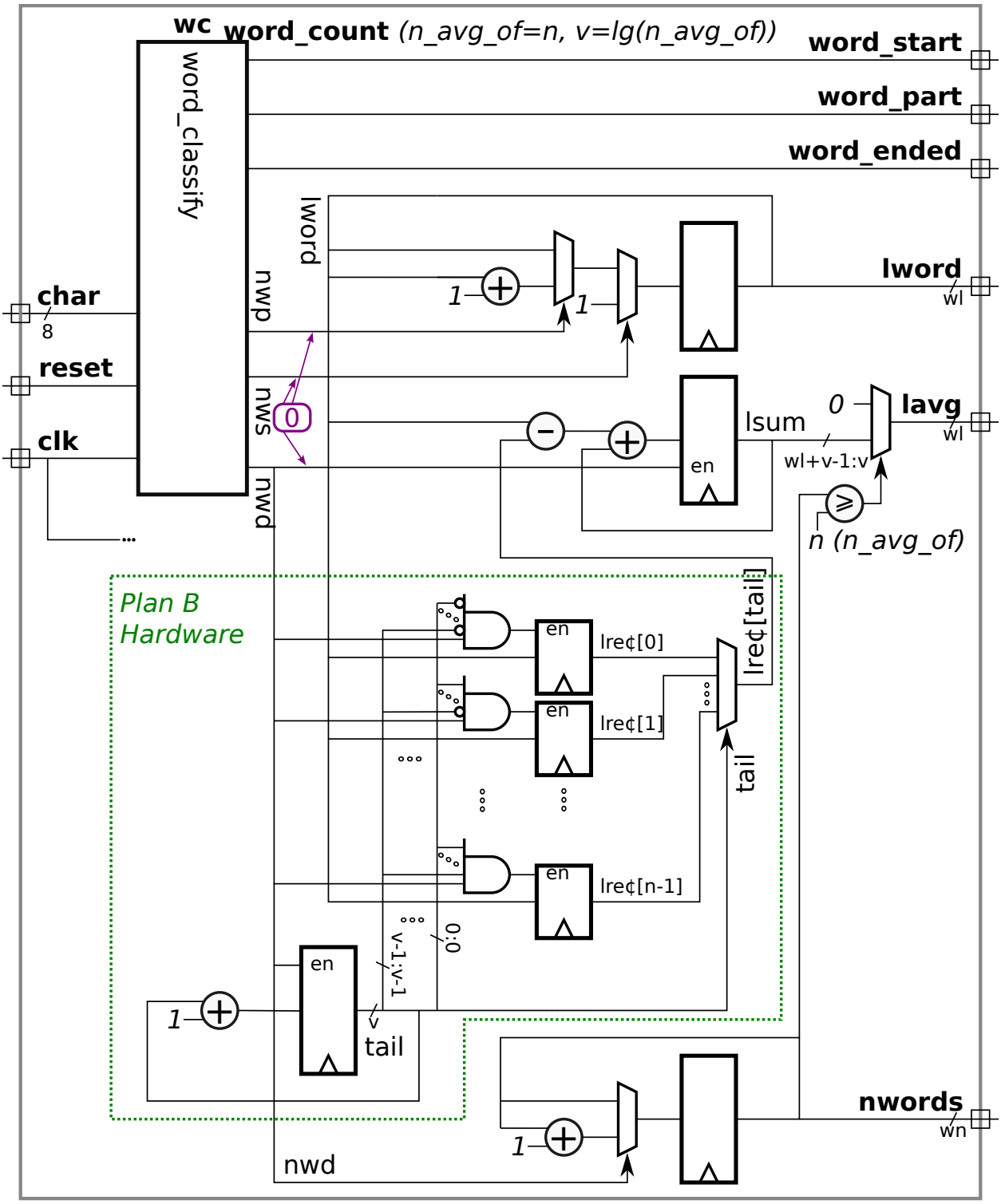
Problem 2: [25 pts] Illustrated on the facing page is a diagram showing inferred hardware similar to the `word_count` module from last year's final exam. An important difference is that it is shown for `n_avg_of=n`, not the specific value of 4. Assume that `n` is a power of 2.

☐ In terms of `n`, `wl`, `wn`, and `v` show simple-model arrival times at each wire and ☐ show a critical path.

☐ Account for cascaded ripple units ☐ constant inputs, and ☐ remember that `n` can be any power of 2, not neccesarily 4.

☐ In terms of `n`, `wl`, `wn`, and `v` compute the simple-model cost of the Plan B hardware, assuming `n` is a power of 2. ☐ Account for constant inputs.

Staple This Side



Problem 3: [27 pts] The two modules below look for a match of input `target` in an `n`-element array `elts` but only check elements 0 to `i_limit-1`. Output `n_match` is the number of matching elements and `match_i` is the lowest `i` for which `elts[i]==target` and `i<i_limit`, or `n` if there is no match. (These modules could be used in the `uniq_vector` module.) Module `fmatch_comb` is complete and works correctly.

(*a*) Module `fmatch_rec` has some code for a recursive implementation. Complete it so that it performs the same calculation as `fmatch_comb`.

☐ Complete `fmatch_rec` so that it computes the same values as `fmatch_comb`.

☐ Don't forget to show the bit ranges of `elts` in the connections to the recursive instantiations.

```
module fmatch_comb
  #( int n = 22, w = 12, wn = $clog2(n+1) )
   ( output logic [wn-1:0] n_match, match_i,
     input uwire  [w-1:0] elts[n-1:0], target,   input uwire [wn-1:0] i_limit );

   // Do not modify this module. It is correct.
   always_comb begin

      n_match = 0;
      match_i = n;

      for ( int i=n-1; i>=0; i-- ) if (  i < i_limit  &&  elts[i] == target  ) begin
         n_match++;
         match_i = i;
      end

   end

endmodule
```


Staple This Side

```
module fmatch_rec
  #( int n = 22, w = 12, wn = $clog2(n+1) )
   ( output uwire [wn-1:0] n_match, match_i,
     input uwire  [w-1:0] elts[n-1:0], target,   input uwire [wn-1:0] i_limit );

   if ( n == 1 ) begin

      // Do not modify the n==1 code, it works.
      uwire match = i_limit != 0 && elts[0] == target;
      assign n_match = match;
      assign match_i = match ? 0 : 1;

   end else begin

      localparam int nlo =

      localparam int nhi =

      localparam int wnr = $clog2(nhi);
      uwire [wnr-1:0] nm_lo, nm_hi, mi_lo, mi_hi;

      uwire [wnr-1:0] il_lo =


      uwire [wnr-1:0] il_hi =


      fmatch_rec #(nlo,w,wnr) ilo( nm_lo, mi_lo, elts[        ], target, il_lo );

      //                  ☐ Show elts' bit ranges ↑↑↓↓

      fmatch_rec #(nhi,w,wnr) ihi( nm_hi, mi_hi, elts[        ], target, il_hi )


      assign n_match =



      assign match_i =



   end

endmodule
```

Problem 4: [20 pts] Answer each question below.

(*a*) Consider two technology targets, FabFab A1000, an ASIC, and LÜTeq FXL9000, an FPGA. Floating-point multipliers are available on the A1000 and the FXL9000 targets.

☐ On one of these targets a design can have as many multipliers as will fit on the chip. Which target is it? ☐ Explain.

☐ On the other target there is a fixed number of FP multipliers, say 5. Does that mean a design that needs 7 FP multipliers can't use the target? ☐ Explain. ☐ The number of needed multipliers can't be reduced.

(*b*) The output of the module below will be `lt=1` for inputs `a=100, b=40, amt=20`, indicating that $100+40 < 20$, which is wrong of course. It works correctly for `a=100, b=40, amt=5`, meaning the output is `lt=0`.

```
module less_than( output uwire lt, input uwire [6:0] a, b, amt );
   assign lt = a + b < amt;
endmodule
```

☐ Why is the output wrong?

☐ What is the largest value of `amt` for which the module output is correct when the other inputs are `a=100, b=40`?

Staple This Side

Staple This Side

(*c*) The `hw` output of the module below is supposed to be set to the number of `1`s in input `vec` at the positive edge of the clock. Due to a beginner's Verilog error it does not work.

```
module pop #( int n = 5, wn = $clog2(n+1) )
   ( output logic [wn-1:0] hw,  input uwire [n-1:0] vec,  input uwire clk );

   always_ff @( posedge clk ) begin

      hw <= 0;
      for ( int i=0; i<n; i++ ) hw <= hw + vec[i];

   end
endmodule
```

☐ Describe the problem. ☐ Describe how it's possible that `hw` can be greater than `n` with this error. ☐ Fix the problem.

(*d*) Consider the population module below.

```
module pop_comb #( int n = 5, wn = $clog2(n+1) )
   ( output logic [wn-1:0] hw,  input uwire [n-1:0] vec );
   always_comb begin
      hw = 0;
      for ( int i=0; i<n; i++ ) hw = hw + vec[i];
   end
endmodule
```

☐ The loop above is procedural. Re-write the module below so that it is a generate loop. The array `s` should come in handy.

```
module pop_comb #( int n = 5, wn = $clog2(n+1) )
   ( output uwire [wn-1:0] hw,  input uwire [n-1:0] vec );

   uwire [wn-1:0] s [n-1:0];









endmodule
```

Staple This Side