Digital Design using HDLs

EE 4755

Final Examination

Wednesday, 5 December 2018  15:00-17:00 CST

Problem 1  ________ (20 pts)
Problem 2  ________ (25 pts)
Problem 3  ________ (20 pts)
Problem 4  ________ (10 pts)
Problem 5  ________ (25 pts)

Exam Total  ________ (100 pts)

Good Luck!
Problem 1: [20 pts] Appearing to the right is the hardware inferred for the Homework 7 Problem 2 module, the fast sequential multiplier which skipped over zeros in the multiplicand.

(a) Notice that some hardware circled in blue. Optimize that hardware and show the cost of the optimized hardware. The optimized hardware should generate signals sv_pro and oa_new. If possible, replace the multiplexors with simpler gates.

☐ Show optimized hardware.

☐ Cost of optimized hardware:
(b) In the version of the module appearing below the \( \geq \) units have been replaced by one module, gt, the changed hardware appears in blue. As can be inferred from the diagram bit \( i \) of the output of gt, gt\(_{TV} \), is 1 iff \( i \geq \text{iter} \). In the Verilog code below gt is instantiated but it is not being used. Modify the Verilog code so that the existing for loop uses the output of gt instead of the \( \geq \) operators. Pay attention to the version of iter used by gt.

- Use gt output in existing for loop.
- Make sure that gt uses correct iter version.

```verilog
module mult_seq_d_prob_2
  #( int w = 16, int m = 2 )
  ( output logic [2*w-1:0] prod,
    output logic out_avail,
    input uwire clk, in_valid,
    input uwire [w-1:0] plier, cand );

  localparam int n = ( w + m - 1 ) / m;
  localparam int iter_lg = $clog2(n);
  uwire [n-1:0][m-1:0] cand_2d = cand;
  bit [iter_lg-1:0] iter, next_iter;
  logic [2*w-1:0] accum;

  uwire [n-1:0] gtv;

  uwire [iter_lg-1:0] gt_iter = (in_valid ? 0 : iter); // √ FILL IN

gt #(n,iter_lg) gti( gtv, gt_iter );

always_ff @( posedge clk ) begin

  if ( in_valid ) begin
    iter = 0; accum = 0; out_avail = 0;
  end else if ( !out_avail && iter == 0 ) begin
    prod = accum; out_avail = 1;
  end

  accum += plier * cand_2d[iter] << ( iter * m );

  next_iter = 0;

  // for ( int i=n-1; i>0; i-- ) if ( i>iter && cand_2d[i] ) next_iter = i;
  for ( int i=n-1; i>0; i-- ) if ( gtv[i] && cand_2d[i] ) next_iter = i;

  iter = next_iter;
end
endmodule
```
Problem 2: [25 pts] The point of the `gt` module in the previous problem was to reduce cost, just in case the synthesis program didn’t notice that the cost of computing each of \(n-1\times iter, n-2\times iter, \ldots, 2\times iter, 1\times iter\), would be less than \(n - 1\) times the cost of computing one of them. The recursive module below computes these quantities and can be used for the `gt` module from the previous problem.

```verilog
module gtd_rec #( int n = 16, int lgn = $clog2(n) )
    ( output logic [n-1:0] gt, input uwire [lgn-1:0] iter );
localparam int nh = n / 2; // Note: n must be a power of 2.
if ( n == 2 ) begin
    assign gt[0] = 0;
    assign gt[1] = !iter[0];
end else begin
    uwire [nh-1:0] gtlo;
    gtd_rec #(nh) glo( gtlo, iter[lgn-2:0] );
    localparam logic [nh-1:0] zeros = 0, ones = -1;
    assign gt = iter[lgn-1] ? { gtlo, zeros } : { ones, gtlo };
end
endmodule
```

(a) Show the hardware that will be inferred for this module for an arbitrary value of \(n\). In this case, do not show what is inside the recursively instantiated module.

☑ Show hardware for arbitrary \(n > 2\). (Don’t show recursive module contents.)

Solution appears below.
(b) There should be a significant optimization opportunity in the hardware above. Show it.

Show how the hardware will be optimized. The result should be AND, OR, and other basic logic gates.

Solution appears below. From the previous solution notice that the $n/2$ LSB of the lower mux input are all zeros. Therefore we can optimize the three gates per bit, into just an AND gate using the inverted select signal. Similarly, the $n/2$ MSB of the upper mux input are all 1's, so we can optimize those bits into just an OR gate.
(c) Show the hardware that will be inferred for \( n = 8 \) after elaboration. That is, show the hardware inside all of the recursive instantiations.

\[ \checkmark \] Show hardware for \( n = 8 \). Show the contents of all recursively instantiated modules.

The solution appears below.

\[ \text{gtd\_rec, } n=8, \ lgn=3 \]

\[ \text{iter} \]

\[ \text{msb} \]

\[ \text{lsb} \]

\[ \text{gt} \]

\[ \text{msb} \]

\[ \text{gt\_to[0]} \]

\[ \text{gt\_to[1]} \]

\[ \text{gt\_to[2]} \]

\[ \text{gt\_to[3]} \]

\[ \text{1'b0} \]

\[ (d) \] Compute the cost and delay using the simple model. Show these in terms of \( n \) assuming that \( n \) is a power of 2.

\[ \checkmark \] Cost and \[ \checkmark \] delay in terms of \( n \).

The cost of the hardware for \( n = 2 \) is 0 (because with the simple model NOT gates are free!). The cost of the hardware for size \( n = 2^{\eta}, \eta > 1 \) is \( n \) gates plus the cost of a size \( n/2 \) module. The total cost for a module of size \( n = 2^{\eta}, \eta > 1 \) is

\[ \sum_{l=2}^{\eta} 2^l = 2^{\eta+1} - 4 = (2n - 4) u_c. \]

Since the critical path through each level is 1, the total delay is

\[ \sum_{l=2}^{\eta} 1 u_t = (\eta - 1) u_t = (\lg n - 1) u_t. \]
The solution to additional problems will be posted before the 2019 final exam.