LSU EE 4755 Homework 3 Due: 17 Oct 2022, 11:30 CDT

Resources

To help with this assignment review the simple cost model slides and the material in generate statement demo code.

The following problems ask for both inferred hardware and a cost/performance analysis: 2019 Midterm Exam Problem 3c (equality module with shifted inputs), 2021 Midterm Exam Problem 2 (a concentrator for neural network hardware reading sparse weights).

The following are good cost and performance analysis questions (these are the same ones mentioned in the simple model slides): The "find oldest" (big mux) problem covered in class can be found in 2017 Final Exam Problem 3, the knapsack problem hardware covered in class can be found in 2016 Final Exam Problem 2 and 4.

The following are good inferred hardware and optimization problems. Start with 2019 Midterm Exam Problem 1 (a recursively described clz [count leading zeros] module). A problem combining both recursive and iterative generate statements can be found in 202 Midterm Exam Problem 4.

A sequential version of the ASCII-to-value hardware was also assigned in this course. The hardware was described by procedural code and it operated sequentially, so I don't suggest that it specifically be studied for clues on how to solve this assignment.

Problem 1: Compute the cost and delay, using the simple model, of the `atoi1` module (from the solution to Homework 1) instantiated with `r=12`. Base this on a module with reasonable optimizations applied and be sure to account for constants when computing cost and delay.

- Base your analysis of ripple implementations of the adder and magnitude comparison units.
- Show cost.
- Show delay of each output and identify the critical path.
- **Account for constants** when computing cost and delay.

```
module atoi1
  #( int r = 32, w = $clog2(r) )
   ( output logic [w-1:0] val,    output logic is_digit,
     input uwire [7:0] char );

   logic [w-1:0] val_09, val_az, val_n;
   logic is_09, is_az;

   digit_valid_09 #(r,w) v09( is_09, val_09, char );
   uwire [7:0] char_uc;
   char_to_uc tuc(char_uc,char);
   digit_valid_az #(r,w) vaz( is_az, val_az, char_uc );

   uwire [w-1:0] z = 0;
   mux2 #(w) mval(val_n,is_09,val_az,val_09);
   mux2 #(w) mval0(val,is_digit,z,val_n);
```

```
   assign is_digit = is_09 || is_az;
endmodule

typedef enum
  { Char_0 = 48, Char_9 = 57, Char_A = 65, Char_Z = 90, Char_a = 97, Char_z = 122 }
  Chars_Special;

module digit_valid_09
  #( int r = 9, vw = $clog2(r) )
   ( output uwire valid, output uwire [vw-1:0] val, input uwire [7:0] char );
   assign val = char - Char_0;
   assign valid = char >= Char_0 && char <= Char_9 && char < Char_0 + r;
endmodule

module char_to_uc( output uwire [7:0] uc, input uwire [7:0] c );
   uwire is_lc = c >= Char_a && c <= Char_z;
   uwire [7:0] uc_if_lc = c - Char_a + Char_A;
   mux2 #(8) m( uc, is_lc, c, uc_if_lc );
endmodule

module digit_valid_az
  #( int r = 11, vw = $clog2(r) )
   ( output uwire valid, output uwire [vw-1:0] val, input uwire [7:0] char );
   assign val = 10 + char - Char_A;
   assign valid = char >= Char_A && char < Char_A + r - 10;
endmodule

module mux2
  #( int w = 3 )
   ( output uwire [w-1:0] x,
     input uwire s,    input uwire [w-1:0] a0, a1 );
   assign x = s ? a1 : a0;
endmodule
```

Problem 2: Appearing further below is the `atoi_it` from the solution to Homework 2.

(*a*) Show the hardware inferred for an `atoi_it` module instantiated with `r=14` (yes, radix 14) and `n=3`.

- Show `atoi1`, `mult_by_c`, and `add` instances as modules, do not show what is inside.
- Show the hardware inferred for the operators, such as `&&` and `?:`.
- Do not confuse parameters and ports.
- Omit hardware that does not belong, such as "hardware" to compute values needed at elaboration time.
- Be sure to show the inferred logic. Remember that generate statements describe what happens at elaboration time, not what happens at simulation time nor does it describe operations performed by the hardware.

(*b*) Show the hardware inferred for an `atoi_it` module instantiated with `r=16` (hexadecimal this time) and `n=3`, and show the hardware after optimization. Consider the impact of optimization on the `mult_by_c` and `add` modules, which should be considerable since `r` is a power of 2.

```
module atoi_it
  #( int r = 11, n = 5, wv = $clog2( r**n ), wd = $clog2(n+1) )
   ( output logic [wv-1:0] val,
     output logic [wd-1:0] nd,
     input uwire [7:0] str [n-1:0] );

   uwire [wv-1:0] vali[n-1:-1];
   uwire is_valid[n-1:-1];
   uwire [wd-1:0] ndi[n-1:-1];
   assign is_valid[-1] = 1;
   assign ndi[-1] = 0;
   assign vali[-1] = 0;
   assign nd = ndi[n-1];
   assign val = vali[n-1];

   localparam int wcv = $clog2(r);

   for ( genvar i=0; i<n; i++ ) begin

      // Find Value of Digit i
      //
      uwire [wcv-1:0] valdr;
      uwire is_digit;
      atoi1 #(r,wcv) a( valdr, is_digit, str[i] );

      // Determine if this digit continues a sequence of valid digits
      // starting at str[0].
      //
      assign is_valid[i] = is_digit && is_valid[i-1];

      // Replace value with zero if str[i] is not a digit, or if the
      // string of valid digits has already ended.
      //
      uwire [wcv-1:0] vald = is_valid[i] ? valdr : 0;

      // Multiply (scale) the digit value based on its position in the number.
      //
      uwire [wv-1:0] vals;
      mult_by_c #( .w_in(wcv), .c(r**i), .w_out(wv) ) mc( vals, vald );

      // Add the scaled digit to the value accumulated so far.
      //
      add #(wv) a1( vali[i], vali[i-1], vals );

      // Update the number of digits so far.
      //
      assign ndi[i] = is_valid[i] ? i+1 : ndi[i-1];
   end
endmodule
```

Problem 3: Appearing further below is the `atoi_tr` from the solution to Homework 2. Show the inferred logic for an instantiation with `r=10` and `n=9`.

- Show the logic for one level. That is, show the two instantiations of `atoi_tr`, `alo` and `ahi`, but don't show what is inside of `alo` nor `ahi`.
- Show the `mult_by_c` instantiations as modules, do not show what is inside.
- Show the hardware inferred for the operators, such as `&&` and `?:`.
- Omit hardware that does not belong, such as "hardware" to compute values needed at elaboration time.
- Do not confuse parameters and ports.
- Be sure to show the inferred logic. Remember that generate statements describe what happens at elaboration time, not what happens at simulation time nor does it describe activities performed by the hardware.

```
module atoi_tr
  #( int r = 11, n = 5, wv = $clog2( r**n ), wd = $clog2(n+1) )
   ( output uwire [wv-1:0] val,   output var logic [wd-1:0] nd,
     input uwire [7:0] str [n-1:0] );

   if ( n == 1 ) begin

      uwire is_dd;
      uwire [wv-1:0] valr;
      atoi1 #(r,wv) a( valr, is_dd, str[0] );
      assign val = is_dd ? valr : 0;
      assign nd = is_dd; // Note: nd may be more than one bit.

   end else begin

      // Prepare to split the input string into two halves. Note that
      // the hi half may be larger, and so we use nhi to compute the
      // number of bits needed in the value output (vwh) and the
      // number of digits output (dwh).
      //
      localparam int nlo = n/2;
      localparam int nhi = n - nlo;
      localparam int vwh = $clog2( r**nhi );
      localparam int dwh = $clog2( nhi+1 );
      //
      uwire [vwh-1:0] vallo, valhi;
      uwire [dwh-1:0] ndlo, ndhi;

      // Split input string between two recursive instantiations
      //
      atoi_tr #(r,nlo,vwh,dwh) alo( vallo, ndlo, str[nlo-1:0] );
      atoi_tr #(r,nhi,vwh,dwh) ahi( valhi, ndhi, str[n-1:nlo] );

      // Determine whether the hi half of the string may be part
      // of the number.
      //
      uwire hitoo = ndlo == nlo;
      uwire [vwh-1:0] valhid = hitoo ? valhi : 0;

      // Scale the upper half.
      //
      uwire [wv-1:0] valhis;  // Value High Scaled
      mult_by_c #(vwh,r**nlo,wv) mc( valhis, valhid );

      assign val = vallo + valhis;
      assign nd = hitoo ? nlo + ndhi : ndlo;
   end
endmodule
```