LSU EE 4755 **Homework 2** **Due: 7 October 2022**

For instructions visit `https://www.ece.lsu.edu/koppel/v/proc.html`. *For the complete Verilog for this assignment without visiting the lab follow* `https://www.ece.lsu.edu/koppel/v/2022/hw02.v.html`.

Problem 0: Following instructions at `https://www.ece.lsu.edu/koppel/v/proc.html`, set up your class account, copy the assignment, and run the Verilog simulator and synthesis program on the unmodified homework file, `hw02.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Homework Background

This assignment is a follow-on to Homework 1, in which the `atoi1` modules will be used to convert an ASCII string holding a number into a value. For example, to convert `"12"` (equivalently `'h3132`) into 12 (equivalently `'b1100` or `'d12` or `'hc`). An ASCII string is a sequence of bytes, in this assignment there can be one or more bytes.

The string is on module input `str` and it is declared so that `str[0]` is the rightmost (least-significant) character of the ASCII string. For example, if the string were `" 987"` then `str[0]` would be the 7 (ASCII value $48+7=55$), `str[1]` would be the 8, `str[2]` the 9, and `str[3]` the space (ASCII value 32).

Let n denote the number of characters in the string. The ASCII number may take up n or fewer characters. For example, for $n=4$ the number 1 would only need one character. The remaining characters can be any non-digit character. For example for the 1 in radix 10 the string can be `" 1"`, or `"abc1"`, but not `"ab21"` since that would be the number 21.

The input to the modules for this assignment, `atoi_it` and `atoi_tr`, is the string. The modules have two outputs, the value, `val`, and the number of digits in the number, `nd`. For example, for input `"9 43"` in radix 10 the value is 43 and the number of digits is 2. The `9` does not count because it is separated by a non-digit character from the `43`. For radix 16 and input `" a12"` the value is 2578 and the number of digits is 3. If the radix is 3 and the string is `"32"` then the value is 2 and the number of digits is 1. The 3 is not a valid digit in trianary, and so it ends the number.

For `r=16` the valid characters are 0 to 9, A to F, and a to f, with a and A, b and B, ... treated equivalently. The module should work for any `r` up to 36.

As of this writing the testbench evaluates radices 10 and 16 and a variety of string lengths. Feel free to modify the testbench to try different radices. (Search for `testbench` and figure out the code.)

Reference Module

To help you get started, there is a reference module, `atoi_pr`, that correctly computes the value of a string. This module would not be a correct solution to either problem.

```
module atoi_pr
  #( 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] );

   always_comb begin
      val = 0; nd = 0;
      for ( int i=0; i<n; i++ ) begin
         // Get val of current char. If val is < 0 then char is not a digit.
         automatic int dval = atoi1_func(str[i],r);
```

```
        if ( dval < 0 ) break;
        val += dval * r**i;
        nd++;
      end
    end
endmodule
```

Testbench

To compile your code and run the testbench press F9 in an Emacs buffer in a properly set up account. In an unmodified assignment the testbench will generate output that includes the following near the end:

```
Total errors for radix 10: 14000 len, 14140 val
Total errors for radix 16: 14000 len, 14224 val
Total errors for string length  1:  4000 len,  4052 val
Total errors for string length  2:  4000 len,  4052 val
Total errors for string length  3:  4000 len,  4052 val
Total errors for string length  4:  4000 len,  4052 val
Total errors for string length  7:  4000 len,  4052 val
Total errors for string length  8:  4000 len,  4052 val
Total errors for string length  9:  4000 len,  4052 val
Total errors for mod atoi_it: 14000 len, 14182 val
Total errors for mod atoi_tr: 14000 len, 14182 val
```

The errors are tallied above three ways: by radix, by string length, and by module (`atoi_it` and `atoi_tr`). In the output above both modules have errors, and their are errors at each radix and length. In the output below module `atoi_it` has zero errors, and errors only occur at lengths 3, 7, 9. The errors would have to be due to `atoi_tr`:

```
Total errors for radix 10:  1201 len,  1201 val
Total errors for radix 16:  1144 len,  1036 val
Total errors for string length  1:     0 len,     0 val
Total errors for string length  2:     0 len,     0 val
Total errors for string length  3:   687 len,   687 val
Total errors for string length  4:     0 len,     0 val
Total errors for string length  7:  1434 len,  1434 val
Total errors for string length  8:     0 len,     0 val
Total errors for string length  9:   224 len,   116 val
Total errors for mod atoi_it:     0 len,     0 val
Total errors for mod atoi_tr:  2345 len,  2237 val
Total number of errors: 4582
```

The messages above are tallies printed near the end. Detailed messages are printed for the first few errors. Here are two error messages (of many from the same run as above:

```
Mod-atoi_tr R-10 n- 7 Ty-SP  Error val 1 != 2011 (correct) for string "   2011"
Mod-atoi_tr R-10 n- 7 Ty-SP  Error len 1 != 4 (correct) for string "   2011"
```

Each of the two lines indicates that the error was with module `atoi_tr` instantiated at `r=10` (radix 10) and string length of $n = 7$. (Don't confuse string length with the length of the number in the string.) `Ty-SP` indicates the type of test, in this case a number padded with spaces. The first line indicates that the value should have been 2011 but the module output was 1. The second line informs us that the length should have been 4, but the module `nd` output was 1.

There are three types of tests: `Ty-SC`, `Ty-SP`, and `Ty-GE`. For `Ty-SC` tests the number is always

one digit (regardless of the string length). For `Ty-SP` tests the number is followed spaces. For `Ty-GE` the number is followed by any non-digit character.

The testbench only shows details for the first 4 errors of each type at each radix. If you want to see more errors feel free to edit the testbench. Search for `err < 5`. Feel free to edit the testbench in other ways to facilitate debugging. The TA-bot will run your code using its own testbench, so don't worry about being accused of cheating by modifying the testbench.

Similar Problems

See the l025-gen-elab.v demo code for examples of how to use generate statements iteratively (needed for Problem 1) and recursively (needed for Problem 2). An easy example is `ripple_w` from that set. Pay attention to how the carry signals are connected from one BFA to the other:

```
module ripple_w
  #( int w = 4 )
   ( output uwire [w-1:0] sum,   output uwire cout,
     input uwire [w-1:0] a, b,   input uwire cin);

   uwire         c[w-1:-1];
   assign        c[-1] = cin;
   assign        cout = c[w-1];

   for ( genvar i = 0; i<w; i++ )
     bfa bfai( sum[i], c[i], a[i], b[i], c[i-1] );
endmodule
```

A simple recursive module is `min_t` which finds the minimum of its `n` inputs:

```
module min_t
  #( int w = 4,  n = 8 )
   ( output uwire [w-1:0] e_min,   input uwire [w-1:0] e [ n-1:0 ] );

   if ( n == 1 ) begin

      assign e_min = e[0];

   end else begin

      localparam int n_lo = n / 2;
      localparam int n_hi = n - n_lo;

      uwire [w-1:0] m_lo, m_hi;

      min_t #(w,n_lo) mlo( m_lo, e[n_lo-1:0] );
      min_t #(w,n_hi) mhi( m_hi, e[n-1:n_lo] );

      min_2 #(w) m2( e_min, m_lo, m_hi);
   end
endmodule
```

See the count-leading-zeros assignment from 2019 Homework 2 for an example of how to recursively instantiate a module and combine results.

Problem 1: Module `atoi_it` has an `n`-character input `str`, and outputs `val` (value) and `nd` (number of digits), as well as parameters `r` (radix) and `n` (number of characters in string). Following the rules further below, complete module `atoi_it` so that `val` is the value of the radix-`r` ASCII representation of a number in `str` and `nd` is set to the number of digits in the number (not to be confused with the number of characters in the string). Further details are described in the background section above.

Module `atoi_it` must use instantiations of module `atoi1` to convert characters to their values and it must use instantiations of `mult_by_c` to do multiplication by a constant. The module may also instantiate `add` and `mux2` modules, but it doesn't have to. A selection of modules is defined under the Problem 0 section of `hw02.v`.

Module `atoi_it` must not instantiate itself (that's Problem 2). Instead, use a generate loop to instantiate the `atoi1` and `mult_by_c` modules.

To help you get started, module `atoi_it` includes an instantiation of `atoi1` and `mult_by_c`. But, those are not in a generate loop and won't work. They are only there to show you how to instantiate something correctly.

Make sure that your module is synthesizable by running the synthesis script. The command is `genus -files syn.tcl`.

Problem 2: Module `atoi_tr` has the same ports and parameters as `atoi_it` and should produce the same outputs. Complete `atoi_tr` so that it does so by recursively instantiating two instances of itself, with each instance operating on about half of the string. As with `atoi_it`, it must use instantiations of `atoi1` to convert characters and `mult_by_c` to perform multiplication. Make sure that the module is synthesizable.

Some may have realized (or will come to realize) that for certain radices neither multiplication nor addition (at least for values) is needed. Don't worry about that, it's okay to use `mult_by_c` even when not needed.

The module must be synthesizable. See the comments in the code for other requirements and things to look out for.