Solutions to Assignment 2

Question 1

The following two testbenches show slightly different approaches.

Each testbench is divided into three processes: one that generates the clock and reset signals; (2) a "checker" process that checks the DUT's outputs and keeps track of the largest error; and (3) a "stimulus" process that applies inputs to the DUT.

The `wait( n == 15)` statement pauses the checker process until the results start coming out of the pipeline after 16 clock cycles.

The first example uses `fread()` to read one test vector at a time:

```
`define XY_BITS    16
`define THETA_BITS 16
`define CORDIC_1 17'd19896        // CORDIC inverse

module cordic_tb ;

   logic clk, rst ;
   logic signed [`XY_BITS:0]    x_i, y_i, x_o, y_o ;
   logic signed [`THETA_BITS:0] theta_i, theta_o ;
   int n = '0 ;

   cordic c0 ( .* ) ;

   // clock and reset
   initial begin
      rst = '1 ;
      clk = '0 ; #500ns ;
      clk = '1 ; #500ns ;
      rst = '0 ;
      forever begin
         clk = '0 ; #500ns ;
         clk = '1 ; #500ns ;
      end
   end

   // checker
   initial begin
      int fd ;
      logic signed [`THETA_BITS:0] angle ;
      logic signed [`XY_BITS:0] sine, diff, maxdiff ;

      fd = $fopen("asg2.tv","r") ;

      wait( n == 15 ) ;          // pipeline delay

      maxdiff='0 ;
     while ( $fscanf(fd,"%x %x",angle,sine) == 2 ) begin
         @(negedge clk) begin
            $display(angle,sine,y_o) ;
            diff = sine > y_o ? sine-y_o : y_o-sine ;
           maxdiff = diff > maxdiff ? diff : maxdiff ;
         end
      end
      $display("maximum error = %d", maxdiff ) ;
      $finish ;
   end


   // stimulus
   initial begin
      int fd ;
      logic signed [`THETA_BITS:0] angle ;
      logic signed [`XY_BITS:0] sine ;

      fd = $fopen("asg2.tv","r") ;

      x_i = `CORDIC_1 ;
      y_i = '0 ;

     while ( $fscanf(fd,"%x %x",angle,sine) == 2 ) begin
         @(negedge clk) theta_i = angle ;
         n++ ;
      end

   end
endmodule
```

The second example uses `$readmemh()` to read all the values into memory. This is simpler (and possibly faster) but requires that all test vectors fit in memory.

```
module cordic_tb2 ;

   logic clk, rst ;

   typedef logic signed [16:0] tb_t ;
   tb_t theta_i, x_i, y_i, theta_o, x_o, y_o ;
   tb_t tv [1000] ;

   int n, N=21 ;

   cordic c0 ( .* ) ;

   // clock and reset
   initial begin
      rst = '1 ;
      clk = '0 ; #500ns ;
      clk = '1 ; #500ns ;
      rst = '0 ;
      forever begin
         clk = '0 ; #500ns ;
         clk = '1 ; #500ns ;
      end
   end

   // checker
   initial begin

      tb_t diff, maxdiff ;

      wait( n == 15 ) ;          // pipeline delay

      maxdiff='0 ;
      for ( int i=0 ; i<N ; i++ )
```

```
        @(negedge clk) begin
           $display(tv[2*i],tv[2*i+1],y_o) ;
           diff = tv[2*i+1] - y_o ;
           diff = diff>0 ? diff : -diff ;
           maxdiff = diff>maxdiff ? diff : maxdiff ;
        end

      $display("maximum error = %d", maxdiff ) ;
      $finish ;
   end

   // stimulus
   initial begin

      $readmemh("asg2.tv",tv) ;

      x_i = `CORDIC_1 ;
      y_i = '0 ;

      for ( n=0 ; n<N ; n++ )
        @(negedge clk) theta_i = tv[2*n];

   end

endmodule
```

The simulation output is:

```
run -all
#       0      0      2
#    2573   2570   2573
#    5147   5126   5124
#    7720   7649   7649
#   10294  10125  10121
#   12867  12539  12541
#   15441  14876  14876
#   18015  17121  17119
#   20588  19260  19256
#   23162  21281  21281
#   25735  23170  23167
#   28309  24916  24912
#   30883  26509  26510
#   33456  27939  27938
#   36030  29196  29193
#   38603  30273  30268
#   41177  31164  31164
#   43751  31862  31859
#   46324  32364  32362
#   48898  32666  32661
#   51471  32768  32765
# maximum error =       5
# ** Note: $finish    : C:/7660/cordic_tb.sv(46)
#    Time: 36 us  Iteration: 1  Instance: /cordic_tb
```