ECE 5760 Lab 0

Submitted 09/11/2009

1. Introduction

The purpose of Lab 0 was to develop familiarity with the lab equipment, Altera hardware, and design software by creating a simple wave generator. The wave generator was designed to use Direct Digital Synthesis (DDS) to produce either a sinusoid or a triangle wave, with wave type and fundamental frequency selected using switches and pushbuttons on the development board. The generated wave output was used to drive the "red" pin of the DE2 board VGA interface. Some of the limitations of DDS as well as the Altera hardware itself were explored by examining the output signal when system clock frequency and wave frequency were pushed to extreme values. A roll-off in signal amplitude was demonstrated as the wave frequency approached the Nyquist rate, and breakdown in the output signal quality was observed when the system clock frequency was driven faster than about 275 MHz.

2. Design and Testing Methods

The design involved in Lab 0 was minimal, due to the limited required functionality of the wave generator. The key points of the design were the DDS unit, the user interface, and the VGA output. Testing was carried out using the oscilloscope to verify correctness of frequency and waveform shape, as well as to examine the effects of increasing wave frequency and system clock frequency.

2.1. Direct Digital Synthesis (DDS)

DDS is a commonly used method for periodic waveform synthesis using a given reference frequency, a lookup table, an M-bit adder and associated accumulator, and an N-bit phase increment value. The top O bits of the accumulator are used as an index to the lookup table, which stores pre-computed samples of a single period of the desired output waveform (a sinusoid, sawtooth wave, etc). Adjusting the phase increment value—the amount added to the accumulator value on each cycle of the reference clock—allows us to adjust the fundamental frequency of the output waveform. The output frequency can be determined by the phase increment using the following expression:

$$f_{out} = \frac{\Delta\phi}{2^M} f_{ref}$$

Additionally, the phase increment for a desired frequency is

$$\Delta\phi \cong \frac{2^M f}{f_{ref}}$$

where exact equality cannot be assured due to the fact that $\Delta\phi$ is an integer value and therefore cannot exactly replicate all frequency values in $\mathbb{R}$.

Our design uses a 32-bit accumulator, a 31-bit increment, and an 8-bit accumulator output. The accumulator output indexes into two 256-entry lookup tables—one for sine wave lookup, the other for triangle wave lookup. The 10-bit output of the currently selected waveform type is selected by multiplexor and connected to the VGA_R output of the DE2 board. The value of the phase increment $\Delta\phi$ is determined indirectly by the user, as detailed below.

2.2. User Interface

In order to adjust the frequency of the DDS output signal, the 18 switches on the DE2 board are used to specify f_{out} as an 18-bit binary number {SW[17] … SW[0]}. This allows users to select frequencies from the range 1Hz $\leq f_{out} \leq$ 262,143 Hz. Additionally, the pushbutton KEY3 was used as a frequency multiplier. When KEY3 is released, $f_{out} =$ {SW[17] … SW[0]}; when pressed, $f_{out} = 256$ x {SW[17] … SW[0]}. This increases the range of possible output frequencies to 1Hz $\leq f_{out} \leq$ 67,108,608 Hz. However, using the original system clock of 50 MHz, Shannon's sampling theorem holds that synthesis of a sinusoid with frequency $\geq$ 25 MHz is impossible. Therefore, the frequency output range is in general 1Hz $\leq f_{out} \leq \min$ (CLK/2, 67,108,608 Hz).

To correctly map a user-specified frequency to phase increment, the value $\frac{2^{32}}{CLK}$ is pre-computed. For a 50 MHz clock, this works out to approximately 85.899. The value is then left shifted by 10, multiplied in fixed point by the frequency value specified by {SW[17]…SW[0]}, and then right-shifted by either 10 or 2, depending on whether the frequency multiplier key is asserted.

The second component of the user interface is the waveform selection, implemented using KEY[2]. When KEY[2] is not pressed, the output is a sine wave at the specified frequency. When it is pressed, the output is a triangle wave with fundamental frequency equal to the specified frequency.

2.3. VGA Output

Because the wave generator outputs pure voltage values through the VGA interface, without the need for any sort of framing or timing, the code required to configure the VGA module is very simple. We merely connect the output of the DDS lookup table mux to the VGA_R pin, drive VGA_BLANK and VGA_SYNC to 1, and assign the main system clock to VGA_CLK.

2.4. Testing Clock Failure using PLLs

By driving the DDS unit from a clock generated by one of the on-chip PLLs, we were able to locate the frequency at which the circuit broke down and determine the cause of the breakdown. Because the design depends not only on the FPGA, but also on the VGA DAC, there are two possible points of failure. The failure point of the DAC is chip-dependent (fabrication-dependent) and cannot be controlled. The

failure point of the FPGA is due to signal delay and path lengths and can be tuned, therefore, by modifying the design and layout as necessary.

Since the design is rather small, we expected the FPGA to be fast enough to out-perform the DAC. When pushed in clock speed, the output signal began to breakdown at approximately 275MHz. The resulting signal showed frequent voltage spikes due to the inability of the DAC to keep up with the changing input signals. The waveform was otherwise-clean, indicating that the FPGA was performing adequately. In fact, according to the timing analysis performed during compilation, the design should be capable of performing up to approximately 340MHz.

2.5. Testing Amplitude Roll-off as Signal Frequency Approaches Nyquist Rate

By increasing the target frequency to within the Nyquist limit of $f_{clk}/2$, we were able to show the effects of signal recovery in discrete time. As the target frequency approaches the limit, the average magnitude of the signal decreases. Presently, we have two theories as to why this occurs. It is possible that this occurs because as we decrease the number of samples per signal period, the probability of having a near-peak value on the output reduces. The resulting sample values, therefore, average to lower-than-peak values, and because the scope is unable to interpolate without additional information, the output waveform appears to have a lower peak-to-peak magnitude. However, it seems that as the number of samples per period decreases, it would be the shape of the sinusoid which decreases, not the peak-to-peak voltage. As Fig. 2 (below) shows, the amplitude of the signal decreased by more than 50% of the original peak-to-peak value. If this amplitude roll-off were to occur due to the low sampling rate, then the 50% loss implies that no samples were taken from points in the wave where voltage reached half of peak value. If this were the case, the resulting sample-and-hold waveform produced by the DDS unit would be a very poor approximation of a sine wave. Using the oscilloscope, however, we verified that the DDS output waveform retains a sine-like shape even as amplitude decreases. Therefore, the low sampling rate is not likely the reason for the amplitude roll-off.

A more plausible explanation for this effect is the frequency shaping imposed by the VGA interface itself. When the output frequency is low, the voltage change between samples is small. However, as the output frequency increases, the voltage change between successive samples increases as well. At 10 MHz output frequency, only 5 samples are used per signal period, and so the voltage steps between samples become significantly large. When these large voltage steps are performed at 50MHz, the transition time becomes larger than the clock period, and so the amplitude of each transition becomes scaled by a roll-off factor. This explains the fact that while the amplitude is scaled, the shape of the waveform remains intact.

3. Documentation

Due to the small design involved in Lab 0, the necessary documentation is minimal. The following RTL diagram shows the complete structure of our design, which can also be found in the Verilog code in Appendices A and B.

NOTE: Although we are submitting both partners' code, the structure of each design is similar enough that the following figure applies to both designs. The differences to the two design files are cosmetic only.

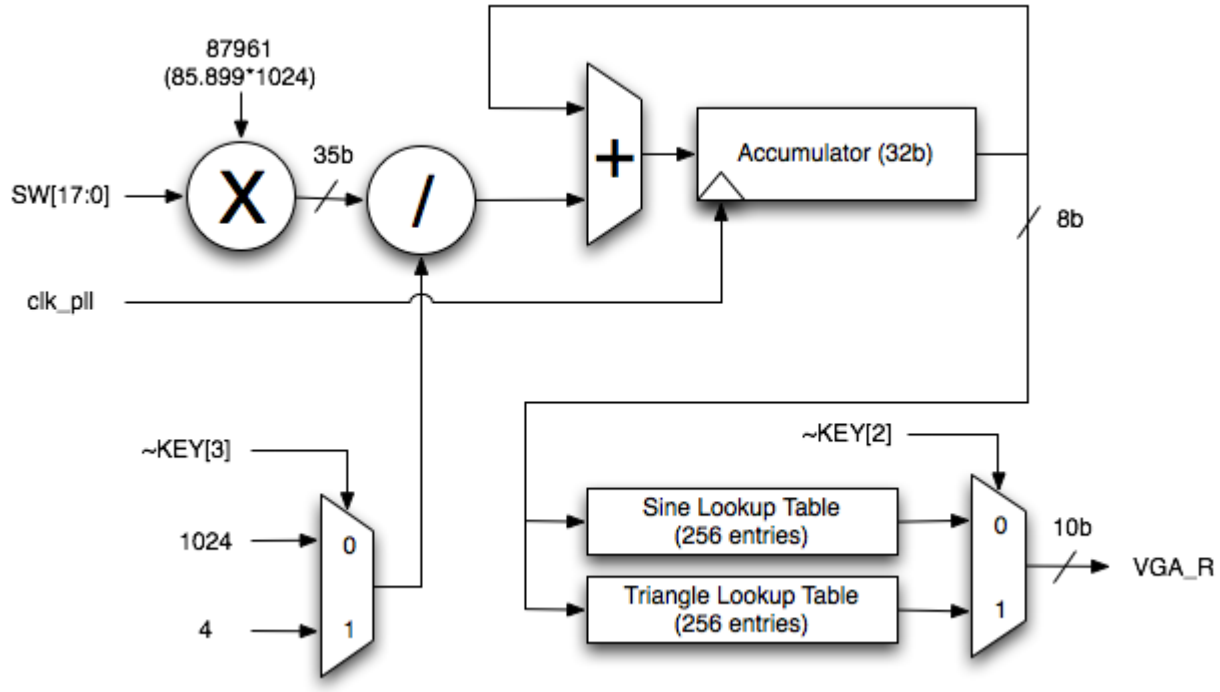

Figure 1: RTL diagram for Lab 0 design

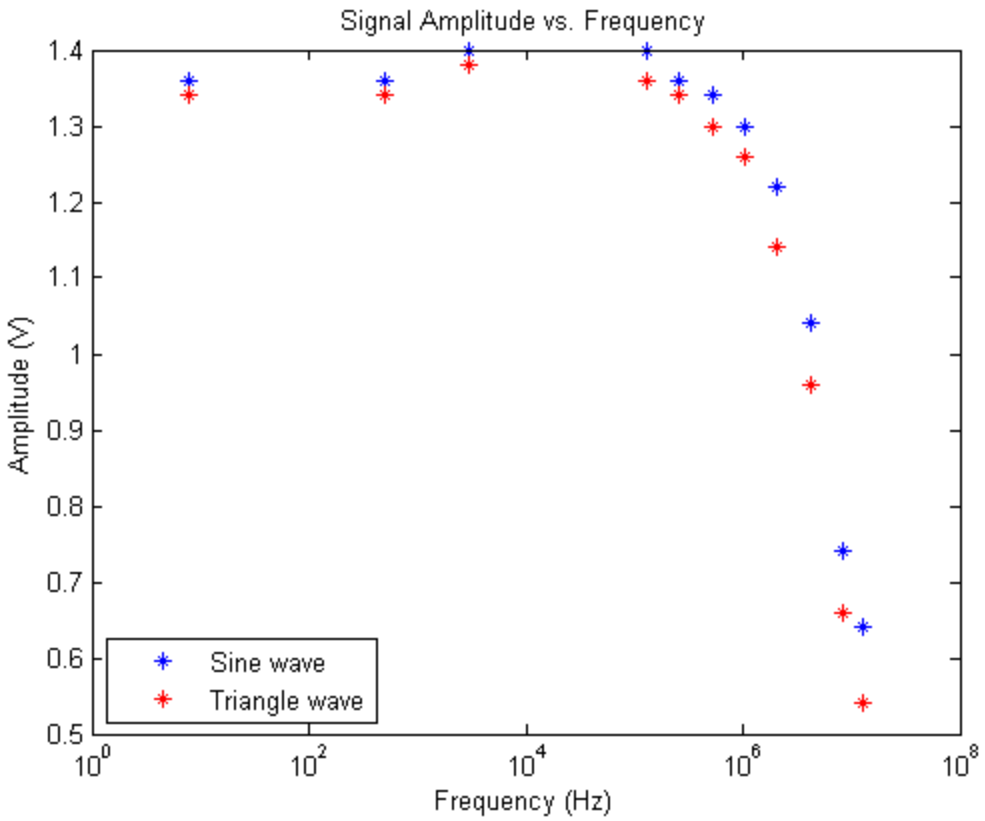

Figure 2: Signal Amplitude vs. Frequency Plot

4 Notes

We each wrote and tested our own separate code for this project. We combined our circuits' analysis results in order to create a single design analysis and lab report. By comparing the two designs we found good similarity in circuit breakdown points; however, for an unknown reason, Tom's design did not exhibit amplitude rolloff at higher target frequencies.

Appendix A: [redacted]'s Code

Excerpt from DE2_TOP.v:

```
//Generate DDS reference clock.
wire clk_dds;
assign clk_dds = CLOCK_50;

//Compute scaled fixed-point phase increment value
//from input target frequency in Hz.
//dphi=(2^32)*f/fref -> 2^32/fref~=85.899
wire [34:0] phase_inc_scaled;
assign phase_inc_scaled = (SW[17:0]*87961); //87961=85.899*1024

//Choose appropriately re-scaled increment value
//depending on input selection. When KEY[3] is pressed,
//rate is scaled by 256.
wire [31:0] phase_inc;
assign phase_inc = ~KEY[3] ?
                   phase_inc_scaled[33:2] :
                   {7'h0,phase_inc_scaled[34:10]};

//Generate frequency with DDS unit.
wire [7:0] dds_output;
dds #(.ACC_WIDTH(32),
      .PHASE_INC_WIDTH(32),
      .OUTPUT_WIDTH(8))
  dds_unit(.clk(clk_dds),
           .reset(1'b0),
           .enable(1'b1),
           .inc(phase_inc),
           .out(dds_output));

//Lookup output value for sine and triangle waves.
wire [9:0] sin_value;
sin_table sine(.in(dds_output),
               .out(sin_value));

wire [9:0] tri_value;
tri_table triangle(.in(dds_output),
                   .out(tri_value));

//Output selected wave type.
assign VGA_CLK = clk_dds;
assign VGA_R = KEY[2] ? sin_value : tri_value;
assign VGA_G = 10'h0;
assign VGA_B = 10'h0;
assign VGA_SYNC = 1'b1;
assign VGA_BLANK = 1'b1;
assign VGA_HS = 1'b0;
assign VGA_VS = 1'b0;

//Display selected frequency (in Hz).
assign LEDR = SW;
hex_driver hex7(.value(4'h0),
                .enable(1'b0),
```

```
                  .display(HEX7));
   hex_driver hex6(.value(4'h0),
                  .enable(1'b0),
                  .display(HEX6));
   hex_driver hex5(.value(4'h0),
                  .enable(1'b0),
                  .display(HEX5));
   hex_driver hex4(.value({2'h0,SW[17:16]}),
                  .enable(1'b1),
                  .display(HEX4));
   hex_driver hex3(.value(SW[15:12]),
                  .enable(1'b1),
                  .display(HEX3));
   hex_driver hex2(.value(SW[11:8]),
                  .enable(1'b1),
                  .display(HEX2));
   hex_driver hex1(.value(SW[7:4]),
                  .enable(1'b1),
                  .display(HEX1));
   hex_driver hex0(.value(SW[3:0]),
                  .enable(1'b1),
                  .display(HEX0));



module hex_driver(
    input [3:0]      value,
    input            enable,
    output reg [6:0] display);

    always @(value or enable) begin
       if(!enable)display<=7'h7F;
       else begin
        case(value)
        4'h0: display<=7'b1000000;
        4'h1: display<=7'b1111001;
        4'h2: display<=7'b0100100;
        4'h3: display<=7'b0110000;
        4'h4: display<=7'b0011001;
        4'h5: display<=7'b0010010;
        4'h6: display<=7'b0000010;
        4'h7: display<=7'b1111000;
        4'h8: display<=7'b0000000;
        4'h9: display<=7'b0010000;
        4'hA: display<=7'b0001000;
        4'hB: display<=7'b0000011;
        4'hC: display<=7'b1000110;
        4'hD: display<=7'b0100001;
        4'hE: display<=7'b0000110;
        4'hF: display<=7'b0001110;
        endcase // case (value)
       end
    end
endmodule
```

Appendix B: [redacted]'s Code

```
module lab_zero(
    input clk,
    input wave_select_key,
    input freq_mult_key,
    input [17:0] freq,
    output [9:0] out );

    wire [30:0] dds_incr;
    wire [7:0] dds_out;
    wire [9:0] sin_lut_out;
    wire [9:0] tri_lut_out;
    reg [33:0] freq_mult_out;

    //Watch for freq_mult_key presses
    always @ ( freq_mult_key or freq ) begin

        //If KEY3 is pressed, multiply the frequency by 256
        if ( freq_mult_key ) freq_mult_out <= (87961 * freq) >> 2;
        else freq_mult_out <= (87961 * freq) >> 10;

    end

    assign dds_incr = freq_mult_out[30:0]; //Grab the low 26 bits of the
                                           //frequency multiplication output

    // Instantiate DDS unit
    dds #(
          .ACC_WIDTH(32),
          .PHASE_INC_WIDTH(31),
          .OUTPUT_WIDTH(8))
    lab_zero_dds(.clk(clk),
                    .reset(1'b0),
                    .enable(1'b1),
                    .inc(dds_incr),
                    .out(dds_out));

    // Instantiate ROMs
    tri_table tri_lut(.in(dds_out), .out(tri_lut_out));
    sin_table sin_lut(.in(dds_out), .out(sin_lut_out));


    // Mux ROM outputs
      assign out = wave_select_key ? tri_lut_out : sin_lut_out;

endmodule
```

Excerpt from DE2_TOP.v (included just above the "endmodule" statement:

```
wire clk_wire;

   //Create the pll
    pll100 p (
      .inclk0(CLOCK_50),
      .c0(clk_wire));
```

```
// Instantiate the "lab_zero" module
lab_zero lz (
 .clk(clk_wire),  //This will be the PLL-generated clock for the later
                  //part of the lab
 .wave_select_key(~KEY[2]),
 .freq_mult_key(~KEY[3]),
 .freq(SW),
 .out(VGA_R) );

// Configure VGA interface
assign VGA_BLANK = 1'b1;    // VGA BLANK
assign VGA_SYNC = 1'b1;     // VGA SYNC

//Assign the VGA clock.  This will be either the 50 MHz clock or the PLL-
//generated clock
assign VGA_CLK = clk_wire;
```

Appendix 3: Shared Code

```
module dds(
    input                            clk,
    input                            reset,
    input                            enable,
    input [(PHASE_INC_WIDTH-1):0]    inc,
    output wire [(OUTPUT_WIDTH-1):0] out);

   parameter ACC_WIDTH = 1;
   parameter PHASE_INC_WIDTH = 1;
   parameter OUTPUT_WIDTH = 1;
   parameter PIPELINE = 0;

   //Increment by zero to implement disable.
   wire [(PHASE_INC_WIDTH-1):0] inc_value;
   //assign inc_value = enable ? inc : {PHASE_INC_WIDTH{1'b0}};
   assign inc_value = inc & {PHASE_INC_WIDTH{enable}};

   //Zero-extend phase increment to accumulator width.
   wire [(ACC_WIDTH-1):0] inc_extended;
   assign inc_extended = {{ACC_WIDTH-PHASE_INC_WIDTH{1'b0}},inc_value};

   wire [(ACC_WIDTH-1):0] next_value;
   generate
      if(PIPELINE) begin
         wire [(ACC_WIDTH-1):0] inc_km1;
         delay #(.WIDTH(ACC_WIDTH))
           next_delay(.clk(clk),
                      .reset(reset),
                      .in(inc_extended),
                      .out(inc_km1));
         assign next_value = accumulator+inc_km1;
      end
      else begin
         assign next_value = accumulator+inc_extended;
      end
   endgenerate

   reg [(ACC_WIDTH-1):0] accumulator;
   always @(posedge clk) begin
      //accumulator <= reset ? {ACC_WIDTH{1'b0}} :
      //               accumulator+inc_extended;
      accumulator <= next_value & {ACC_WIDTH{~reset}};
   end

   //Output is the top bits of the phase accumulator.
   assign out = accumulator[(ACC_WIDTH-1):(ACC_WIDTH-OUTPUT_WIDTH)];
endmodule

//This file was automatically generated by
//Matlab on 01-Sep-2009 16:17:43.
`include "sin_table.vh"

module sin_table(
    input [`SIN_TABLE_INPUT_RANGE]       in,
```

```
 output reg [`SIN_TABLE_OUTPUT_RANGE] out);

always @(in) begin
  casez(in)
    `SIN_TABLE_INPUT_WIDTH'd0: out <= `SIN_TABLE_OUTPUT_WIDTH'd512;
    `SIN_TABLE_INPUT_WIDTH'd1: out <= `SIN_TABLE_OUTPUT_WIDTH'd524;
    `SIN_TABLE_INPUT_WIDTH'd2: out <= `SIN_TABLE_OUTPUT_WIDTH'd537;
    `SIN_TABLE_INPUT_WIDTH'd3: out <= `SIN_TABLE_OUTPUT_WIDTH'd549;
    `SIN_TABLE_INPUT_WIDTH'd4: out <= `SIN_TABLE_OUTPUT_WIDTH'd562;
    `SIN_TABLE_INPUT_WIDTH'd5: out <= `SIN_TABLE_OUTPUT_WIDTH'd574;
    `SIN_TABLE_INPUT_WIDTH'd6: out <= `SIN_TABLE_OUTPUT_WIDTH'd587;
    `SIN_TABLE_INPUT_WIDTH'd7: out <= `SIN_TABLE_OUTPUT_WIDTH'd599;
    `SIN_TABLE_INPUT_WIDTH'd8: out <= `SIN_TABLE_OUTPUT_WIDTH'd612;
    `SIN_TABLE_INPUT_WIDTH'd9: out <= `SIN_TABLE_OUTPUT_WIDTH'd624;
    `SIN_TABLE_INPUT_WIDTH'd10: out <= `SIN_TABLE_OUTPUT_WIDTH'd636;
    `SIN_TABLE_INPUT_WIDTH'd11: out <= `SIN_TABLE_OUTPUT_WIDTH'd648;
    `SIN_TABLE_INPUT_WIDTH'd12: out <= `SIN_TABLE_OUTPUT_WIDTH'd661;
    `SIN_TABLE_INPUT_WIDTH'd13: out <= `SIN_TABLE_OUTPUT_WIDTH'd673;
    `SIN_TABLE_INPUT_WIDTH'd14: out <= `SIN_TABLE_OUTPUT_WIDTH'd684;
    `SIN_TABLE_INPUT_WIDTH'd15: out <= `SIN_TABLE_OUTPUT_WIDTH'd696;
    `SIN_TABLE_INPUT_WIDTH'd16: out <= `SIN_TABLE_OUTPUT_WIDTH'd708;
    `SIN_TABLE_INPUT_WIDTH'd17: out <= `SIN_TABLE_OUTPUT_WIDTH'd720;
    `SIN_TABLE_INPUT_WIDTH'd18: out <= `SIN_TABLE_OUTPUT_WIDTH'd731;
    `SIN_TABLE_INPUT_WIDTH'd19: out <= `SIN_TABLE_OUTPUT_WIDTH'd742;
    `SIN_TABLE_INPUT_WIDTH'd20: out <= `SIN_TABLE_OUTPUT_WIDTH'd753;
    `SIN_TABLE_INPUT_WIDTH'd21: out <= `SIN_TABLE_OUTPUT_WIDTH'd765;
    `SIN_TABLE_INPUT_WIDTH'd22: out <= `SIN_TABLE_OUTPUT_WIDTH'd775;
    `SIN_TABLE_INPUT_WIDTH'd23: out <= `SIN_TABLE_OUTPUT_WIDTH'd786;
    `SIN_TABLE_INPUT_WIDTH'd24: out <= `SIN_TABLE_OUTPUT_WIDTH'd797;
    `SIN_TABLE_INPUT_WIDTH'd25: out <= `SIN_TABLE_OUTPUT_WIDTH'd807;
    `SIN_TABLE_INPUT_WIDTH'd26: out <= `SIN_TABLE_OUTPUT_WIDTH'd817;
    `SIN_TABLE_INPUT_WIDTH'd27: out <= `SIN_TABLE_OUTPUT_WIDTH'd827;
    `SIN_TABLE_INPUT_WIDTH'd28: out <= `SIN_TABLE_OUTPUT_WIDTH'd837;
    `SIN_TABLE_INPUT_WIDTH'd29: out <= `SIN_TABLE_OUTPUT_WIDTH'd847;
    `SIN_TABLE_INPUT_WIDTH'd30: out <= `SIN_TABLE_OUTPUT_WIDTH'd856;
    `SIN_TABLE_INPUT_WIDTH'd31: out <= `SIN_TABLE_OUTPUT_WIDTH'd865;
    `SIN_TABLE_INPUT_WIDTH'd32: out <= `SIN_TABLE_OUTPUT_WIDTH'd874;
    `SIN_TABLE_INPUT_WIDTH'd33: out <= `SIN_TABLE_OUTPUT_WIDTH'd883;
    `SIN_TABLE_INPUT_WIDTH'd34: out <= `SIN_TABLE_OUTPUT_WIDTH'd892;
    `SIN_TABLE_INPUT_WIDTH'd35: out <= `SIN_TABLE_OUTPUT_WIDTH'd900;
    `SIN_TABLE_INPUT_WIDTH'd36: out <= `SIN_TABLE_OUTPUT_WIDTH'd908;
    `SIN_TABLE_INPUT_WIDTH'd37: out <= `SIN_TABLE_OUTPUT_WIDTH'd916;
    `SIN_TABLE_INPUT_WIDTH'd38: out <= `SIN_TABLE_OUTPUT_WIDTH'd923;
    `SIN_TABLE_INPUT_WIDTH'd39: out <= `SIN_TABLE_OUTPUT_WIDTH'd931;
    `SIN_TABLE_INPUT_WIDTH'd40: out <= `SIN_TABLE_OUTPUT_WIDTH'd938;
    `SIN_TABLE_INPUT_WIDTH'd41: out <= `SIN_TABLE_OUTPUT_WIDTH'd945;
    `SIN_TABLE_INPUT_WIDTH'd42: out <= `SIN_TABLE_OUTPUT_WIDTH'd951;
    `SIN_TABLE_INPUT_WIDTH'd43: out <= `SIN_TABLE_OUTPUT_WIDTH'd958;
    `SIN_TABLE_INPUT_WIDTH'd44: out <= `SIN_TABLE_OUTPUT_WIDTH'd964;
    `SIN_TABLE_INPUT_WIDTH'd45: out <= `SIN_TABLE_OUTPUT_WIDTH'd969;
    `SIN_TABLE_INPUT_WIDTH'd46: out <= `SIN_TABLE_OUTPUT_WIDTH'd975;
    `SIN_TABLE_INPUT_WIDTH'd47: out <= `SIN_TABLE_OUTPUT_WIDTH'd980;
    `SIN_TABLE_INPUT_WIDTH'd48: out <= `SIN_TABLE_OUTPUT_WIDTH'd985;
    `SIN_TABLE_INPUT_WIDTH'd49: out <= `SIN_TABLE_OUTPUT_WIDTH'd990;
    `SIN_TABLE_INPUT_WIDTH'd50: out <= `SIN_TABLE_OUTPUT_WIDTH'd994;
    `SIN_TABLE_INPUT_WIDTH'd51: out <= `SIN_TABLE_OUTPUT_WIDTH'd998;
    `SIN_TABLE_INPUT_WIDTH'd52: out <= `SIN_TABLE_OUTPUT_WIDTH'd1002;
```

```
`SIN_TABLE_INPUT_WIDTH'd53: out <= `SIN_TABLE_OUTPUT_WIDTH'd1005;
`SIN_TABLE_INPUT_WIDTH'd54: out <= `SIN_TABLE_OUTPUT_WIDTH'd1008;
`SIN_TABLE_INPUT_WIDTH'd55: out <= `SIN_TABLE_OUTPUT_WIDTH'd1011;
`SIN_TABLE_INPUT_WIDTH'd56: out <= `SIN_TABLE_OUTPUT_WIDTH'd1014;
`SIN_TABLE_INPUT_WIDTH'd57: out <= `SIN_TABLE_OUTPUT_WIDTH'd1016;
`SIN_TABLE_INPUT_WIDTH'd58: out <= `SIN_TABLE_OUTPUT_WIDTH'd1018;
`SIN_TABLE_INPUT_WIDTH'd59: out <= `SIN_TABLE_OUTPUT_WIDTH'd1020;
`SIN_TABLE_INPUT_WIDTH'd60: out <= `SIN_TABLE_OUTPUT_WIDTH'd1021;
`SIN_TABLE_INPUT_WIDTH'd61: out <= `SIN_TABLE_OUTPUT_WIDTH'd1022;
`SIN_TABLE_INPUT_WIDTH'd62: out <= `SIN_TABLE_OUTPUT_WIDTH'd1023;
`SIN_TABLE_INPUT_WIDTH'd63: out <= `SIN_TABLE_OUTPUT_WIDTH'd1023;
`SIN_TABLE_INPUT_WIDTH'd64: out <= `SIN_TABLE_OUTPUT_WIDTH'd1023;
`SIN_TABLE_INPUT_WIDTH'd65: out <= `SIN_TABLE_OUTPUT_WIDTH'd1023;
`SIN_TABLE_INPUT_WIDTH'd66: out <= `SIN_TABLE_OUTPUT_WIDTH'd1022;
`SIN_TABLE_INPUT_WIDTH'd67: out <= `SIN_TABLE_OUTPUT_WIDTH'd1021;
`SIN_TABLE_INPUT_WIDTH'd68: out <= `SIN_TABLE_OUTPUT_WIDTH'd1020;
`SIN_TABLE_INPUT_WIDTH'd69: out <= `SIN_TABLE_OUTPUT_WIDTH'd1019;
`SIN_TABLE_INPUT_WIDTH'd70: out <= `SIN_TABLE_OUTPUT_WIDTH'd1017;
`SIN_TABLE_INPUT_WIDTH'd71: out <= `SIN_TABLE_OUTPUT_WIDTH'd1015;
`SIN_TABLE_INPUT_WIDTH'd72: out <= `SIN_TABLE_OUTPUT_WIDTH'd1012;
`SIN_TABLE_INPUT_WIDTH'd73: out <= `SIN_TABLE_OUTPUT_WIDTH'd1010;
`SIN_TABLE_INPUT_WIDTH'd74: out <= `SIN_TABLE_OUTPUT_WIDTH'd1007;
`SIN_TABLE_INPUT_WIDTH'd75: out <= `SIN_TABLE_OUTPUT_WIDTH'd1003;
`SIN_TABLE_INPUT_WIDTH'd76: out <= `SIN_TABLE_OUTPUT_WIDTH'd1000;
`SIN_TABLE_INPUT_WIDTH'd77: out <= `SIN_TABLE_OUTPUT_WIDTH'd996;
`SIN_TABLE_INPUT_WIDTH'd78: out <= `SIN_TABLE_OUTPUT_WIDTH'd992;
`SIN_TABLE_INPUT_WIDTH'd79: out <= `SIN_TABLE_OUTPUT_WIDTH'd987;
`SIN_TABLE_INPUT_WIDTH'd80: out <= `SIN_TABLE_OUTPUT_WIDTH'd983;
`SIN_TABLE_INPUT_WIDTH'd81: out <= `SIN_TABLE_OUTPUT_WIDTH'd977;
`SIN_TABLE_INPUT_WIDTH'd82: out <= `SIN_TABLE_OUTPUT_WIDTH'd972;
`SIN_TABLE_INPUT_WIDTH'd83: out <= `SIN_TABLE_OUTPUT_WIDTH'd967;
`SIN_TABLE_INPUT_WIDTH'd84: out <= `SIN_TABLE_OUTPUT_WIDTH'd961;
`SIN_TABLE_INPUT_WIDTH'd85: out <= `SIN_TABLE_OUTPUT_WIDTH'd954;
`SIN_TABLE_INPUT_WIDTH'd86: out <= `SIN_TABLE_OUTPUT_WIDTH'd948;
`SIN_TABLE_INPUT_WIDTH'd87: out <= `SIN_TABLE_OUTPUT_WIDTH'd941;
`SIN_TABLE_INPUT_WIDTH'd88: out <= `SIN_TABLE_OUTPUT_WIDTH'd934;
`SIN_TABLE_INPUT_WIDTH'd89: out <= `SIN_TABLE_OUTPUT_WIDTH'd927;
`SIN_TABLE_INPUT_WIDTH'd90: out <= `SIN_TABLE_OUTPUT_WIDTH'd920;
`SIN_TABLE_INPUT_WIDTH'd91: out <= `SIN_TABLE_OUTPUT_WIDTH'd912;
`SIN_TABLE_INPUT_WIDTH'd92: out <= `SIN_TABLE_OUTPUT_WIDTH'd904;
`SIN_TABLE_INPUT_WIDTH'd93: out <= `SIN_TABLE_OUTPUT_WIDTH'd896;
`SIN_TABLE_INPUT_WIDTH'd94: out <= `SIN_TABLE_OUTPUT_WIDTH'd887;
`SIN_TABLE_INPUT_WIDTH'd95: out <= `SIN_TABLE_OUTPUT_WIDTH'd879;
`SIN_TABLE_INPUT_WIDTH'd96: out <= `SIN_TABLE_OUTPUT_WIDTH'd870;
`SIN_TABLE_INPUT_WIDTH'd97: out <= `SIN_TABLE_OUTPUT_WIDTH'd861;
`SIN_TABLE_INPUT_WIDTH'd98: out <= `SIN_TABLE_OUTPUT_WIDTH'd851;
`SIN_TABLE_INPUT_WIDTH'd99: out <= `SIN_TABLE_OUTPUT_WIDTH'd842;
`SIN_TABLE_INPUT_WIDTH'd100: out <= `SIN_TABLE_OUTPUT_WIDTH'd832;
`SIN_TABLE_INPUT_WIDTH'd101: out <= `SIN_TABLE_OUTPUT_WIDTH'd822;
`SIN_TABLE_INPUT_WIDTH'd102: out <= `SIN_TABLE_OUTPUT_WIDTH'd812;
`SIN_TABLE_INPUT_WIDTH'd103: out <= `SIN_TABLE_OUTPUT_WIDTH'd802;
`SIN_TABLE_INPUT_WIDTH'd104: out <= `SIN_TABLE_OUTPUT_WIDTH'd791;
`SIN_TABLE_INPUT_WIDTH'd105: out <= `SIN_TABLE_OUTPUT_WIDTH'd781;
`SIN_TABLE_INPUT_WIDTH'd106: out <= `SIN_TABLE_OUTPUT_WIDTH'd770;
`SIN_TABLE_INPUT_WIDTH'd107: out <= `SIN_TABLE_OUTPUT_WIDTH'd759;
`SIN_TABLE_INPUT_WIDTH'd108: out <= `SIN_TABLE_OUTPUT_WIDTH'd748;
`SIN_TABLE_INPUT_WIDTH'd109: out <= `SIN_TABLE_OUTPUT_WIDTH'd737;
```

```
`SIN_TABLE_INPUT_WIDTH'd110: out <= `SIN_TABLE_OUTPUT_WIDTH'd725;
`SIN_TABLE_INPUT_WIDTH'd111: out <= `SIN_TABLE_OUTPUT_WIDTH'd714;
`SIN_TABLE_INPUT_WIDTH'd112: out <= `SIN_TABLE_OUTPUT_WIDTH'd702;
`SIN_TABLE_INPUT_WIDTH'd113: out <= `SIN_TABLE_OUTPUT_WIDTH'd690;
`SIN_TABLE_INPUT_WIDTH'd114: out <= `SIN_TABLE_OUTPUT_WIDTH'd679;
`SIN_TABLE_INPUT_WIDTH'd115: out <= `SIN_TABLE_OUTPUT_WIDTH'd667;
`SIN_TABLE_INPUT_WIDTH'd116: out <= `SIN_TABLE_OUTPUT_WIDTH'd655;
`SIN_TABLE_INPUT_WIDTH'd117: out <= `SIN_TABLE_OUTPUT_WIDTH'd642;
`SIN_TABLE_INPUT_WIDTH'd118: out <= `SIN_TABLE_OUTPUT_WIDTH'd630;
`SIN_TABLE_INPUT_WIDTH'd119: out <= `SIN_TABLE_OUTPUT_WIDTH'd618;
`SIN_TABLE_INPUT_WIDTH'd120: out <= `SIN_TABLE_OUTPUT_WIDTH'd605;
`SIN_TABLE_INPUT_WIDTH'd121: out <= `SIN_TABLE_OUTPUT_WIDTH'd593;
`SIN_TABLE_INPUT_WIDTH'd122: out <= `SIN_TABLE_OUTPUT_WIDTH'd581;
`SIN_TABLE_INPUT_WIDTH'd123: out <= `SIN_TABLE_OUTPUT_WIDTH'd568;
`SIN_TABLE_INPUT_WIDTH'd124: out <= `SIN_TABLE_OUTPUT_WIDTH'd556;
`SIN_TABLE_INPUT_WIDTH'd125: out <= `SIN_TABLE_OUTPUT_WIDTH'd543;
`SIN_TABLE_INPUT_WIDTH'd126: out <= `SIN_TABLE_OUTPUT_WIDTH'd530;
`SIN_TABLE_INPUT_WIDTH'd127: out <= `SIN_TABLE_OUTPUT_WIDTH'd518;
`SIN_TABLE_INPUT_WIDTH'd128: out <= `SIN_TABLE_OUTPUT_WIDTH'd505;
`SIN_TABLE_INPUT_WIDTH'd129: out <= `SIN_TABLE_OUTPUT_WIDTH'd493;
`SIN_TABLE_INPUT_WIDTH'd130: out <= `SIN_TABLE_OUTPUT_WIDTH'd480;
`SIN_TABLE_INPUT_WIDTH'd131: out <= `SIN_TABLE_OUTPUT_WIDTH'd467;
`SIN_TABLE_INPUT_WIDTH'd132: out <= `SIN_TABLE_OUTPUT_WIDTH'd455;
`SIN_TABLE_INPUT_WIDTH'd133: out <= `SIN_TABLE_OUTPUT_WIDTH'd442;
`SIN_TABLE_INPUT_WIDTH'd134: out <= `SIN_TABLE_OUTPUT_WIDTH'd430;
`SIN_TABLE_INPUT_WIDTH'd135: out <= `SIN_TABLE_OUTPUT_WIDTH'd418;
`SIN_TABLE_INPUT_WIDTH'd136: out <= `SIN_TABLE_OUTPUT_WIDTH'd405;
`SIN_TABLE_INPUT_WIDTH'd137: out <= `SIN_TABLE_OUTPUT_WIDTH'd393;
`SIN_TABLE_INPUT_WIDTH'd138: out <= `SIN_TABLE_OUTPUT_WIDTH'd381;
`SIN_TABLE_INPUT_WIDTH'd139: out <= `SIN_TABLE_OUTPUT_WIDTH'd368;
`SIN_TABLE_INPUT_WIDTH'd140: out <= `SIN_TABLE_OUTPUT_WIDTH'd356;
`SIN_TABLE_INPUT_WIDTH'd141: out <= `SIN_TABLE_OUTPUT_WIDTH'd344;
`SIN_TABLE_INPUT_WIDTH'd142: out <= `SIN_TABLE_OUTPUT_WIDTH'd333;
`SIN_TABLE_INPUT_WIDTH'd143: out <= `SIN_TABLE_OUTPUT_WIDTH'd321;
`SIN_TABLE_INPUT_WIDTH'd144: out <= `SIN_TABLE_OUTPUT_WIDTH'd309;
`SIN_TABLE_INPUT_WIDTH'd145: out <= `SIN_TABLE_OUTPUT_WIDTH'd298;
`SIN_TABLE_INPUT_WIDTH'd146: out <= `SIN_TABLE_OUTPUT_WIDTH'd286;
`SIN_TABLE_INPUT_WIDTH'd147: out <= `SIN_TABLE_OUTPUT_WIDTH'd275;
`SIN_TABLE_INPUT_WIDTH'd148: out <= `SIN_TABLE_OUTPUT_WIDTH'd264;
`SIN_TABLE_INPUT_WIDTH'd149: out <= `SIN_TABLE_OUTPUT_WIDTH'd253;
`SIN_TABLE_INPUT_WIDTH'd150: out <= `SIN_TABLE_OUTPUT_WIDTH'd242;
`SIN_TABLE_INPUT_WIDTH'd151: out <= `SIN_TABLE_OUTPUT_WIDTH'd232;
`SIN_TABLE_INPUT_WIDTH'd152: out <= `SIN_TABLE_OUTPUT_WIDTH'd221;
`SIN_TABLE_INPUT_WIDTH'd153: out <= `SIN_TABLE_OUTPUT_WIDTH'd211;
`SIN_TABLE_INPUT_WIDTH'd154: out <= `SIN_TABLE_OUTPUT_WIDTH'd201;
`SIN_TABLE_INPUT_WIDTH'd155: out <= `SIN_TABLE_OUTPUT_WIDTH'd191;
`SIN_TABLE_INPUT_WIDTH'd156: out <= `SIN_TABLE_OUTPUT_WIDTH'd181;
`SIN_TABLE_INPUT_WIDTH'd157: out <= `SIN_TABLE_OUTPUT_WIDTH'd172;
`SIN_TABLE_INPUT_WIDTH'd158: out <= `SIN_TABLE_OUTPUT_WIDTH'd162;
`SIN_TABLE_INPUT_WIDTH'd159: out <= `SIN_TABLE_OUTPUT_WIDTH'd153;
`SIN_TABLE_INPUT_WIDTH'd160: out <= `SIN_TABLE_OUTPUT_WIDTH'd144;
`SIN_TABLE_INPUT_WIDTH'd161: out <= `SIN_TABLE_OUTPUT_WIDTH'd136;
`SIN_TABLE_INPUT_WIDTH'd162: out <= `SIN_TABLE_OUTPUT_WIDTH'd127;
`SIN_TABLE_INPUT_WIDTH'd163: out <= `SIN_TABLE_OUTPUT_WIDTH'd119;
`SIN_TABLE_INPUT_WIDTH'd164: out <= `SIN_TABLE_OUTPUT_WIDTH'd111;
`SIN_TABLE_INPUT_WIDTH'd165: out <= `SIN_TABLE_OUTPUT_WIDTH'd103;
`SIN_TABLE_INPUT_WIDTH'd166: out <= `SIN_TABLE_OUTPUT_WIDTH'd96;
```

```
`SIN_TABLE_INPUT_WIDTH'd167: out <= `SIN_TABLE_OUTPUT_WIDTH'd89;
`SIN_TABLE_INPUT_WIDTH'd168: out <= `SIN_TABLE_OUTPUT_WIDTH'd82;
`SIN_TABLE_INPUT_WIDTH'd169: out <= `SIN_TABLE_OUTPUT_WIDTH'd75;
`SIN_TABLE_INPUT_WIDTH'd170: out <= `SIN_TABLE_OUTPUT_WIDTH'd69;
`SIN_TABLE_INPUT_WIDTH'd171: out <= `SIN_TABLE_OUTPUT_WIDTH'd62;
`SIN_TABLE_INPUT_WIDTH'd172: out <= `SIN_TABLE_OUTPUT_WIDTH'd56;
`SIN_TABLE_INPUT_WIDTH'd173: out <= `SIN_TABLE_OUTPUT_WIDTH'd51;
`SIN_TABLE_INPUT_WIDTH'd174: out <= `SIN_TABLE_OUTPUT_WIDTH'd46;
`SIN_TABLE_INPUT_WIDTH'd175: out <= `SIN_TABLE_OUTPUT_WIDTH'd40;
`SIN_TABLE_INPUT_WIDTH'd176: out <= `SIN_TABLE_OUTPUT_WIDTH'd36;
`SIN_TABLE_INPUT_WIDTH'd177: out <= `SIN_TABLE_OUTPUT_WIDTH'd31;
`SIN_TABLE_INPUT_WIDTH'd178: out <= `SIN_TABLE_OUTPUT_WIDTH'd27;
`SIN_TABLE_INPUT_WIDTH'd179: out <= `SIN_TABLE_OUTPUT_WIDTH'd23;
`SIN_TABLE_INPUT_WIDTH'd180: out <= `SIN_TABLE_OUTPUT_WIDTH'd20;
`SIN_TABLE_INPUT_WIDTH'd181: out <= `SIN_TABLE_OUTPUT_WIDTH'd16;
`SIN_TABLE_INPUT_WIDTH'd182: out <= `SIN_TABLE_OUTPUT_WIDTH'd13;
`SIN_TABLE_INPUT_WIDTH'd183: out <= `SIN_TABLE_OUTPUT_WIDTH'd11;
`SIN_TABLE_INPUT_WIDTH'd184: out <= `SIN_TABLE_OUTPUT_WIDTH'd8;
`SIN_TABLE_INPUT_WIDTH'd185: out <= `SIN_TABLE_OUTPUT_WIDTH'd6;
`SIN_TABLE_INPUT_WIDTH'd186: out <= `SIN_TABLE_OUTPUT_WIDTH'd4;
`SIN_TABLE_INPUT_WIDTH'd187: out <= `SIN_TABLE_OUTPUT_WIDTH'd3;
`SIN_TABLE_INPUT_WIDTH'd188: out <= `SIN_TABLE_OUTPUT_WIDTH'd2;
`SIN_TABLE_INPUT_WIDTH'd189: out <= `SIN_TABLE_OUTPUT_WIDTH'd1;
`SIN_TABLE_INPUT_WIDTH'd190: out <= `SIN_TABLE_OUTPUT_WIDTH'd0;
`SIN_TABLE_INPUT_WIDTH'd191: out <= `SIN_TABLE_OUTPUT_WIDTH'd0;
`SIN_TABLE_INPUT_WIDTH'd192: out <= `SIN_TABLE_OUTPUT_WIDTH'd0;
`SIN_TABLE_INPUT_WIDTH'd193: out <= `SIN_TABLE_OUTPUT_WIDTH'd0;
`SIN_TABLE_INPUT_WIDTH'd194: out <= `SIN_TABLE_OUTPUT_WIDTH'd1;
`SIN_TABLE_INPUT_WIDTH'd195: out <= `SIN_TABLE_OUTPUT_WIDTH'd2;
`SIN_TABLE_INPUT_WIDTH'd196: out <= `SIN_TABLE_OUTPUT_WIDTH'd3;
`SIN_TABLE_INPUT_WIDTH'd197: out <= `SIN_TABLE_OUTPUT_WIDTH'd5;
`SIN_TABLE_INPUT_WIDTH'd198: out <= `SIN_TABLE_OUTPUT_WIDTH'd7;
`SIN_TABLE_INPUT_WIDTH'd199: out <= `SIN_TABLE_OUTPUT_WIDTH'd9;
`SIN_TABLE_INPUT_WIDTH'd200: out <= `SIN_TABLE_OUTPUT_WIDTH'd12;
`SIN_TABLE_INPUT_WIDTH'd201: out <= `SIN_TABLE_OUTPUT_WIDTH'd15;
`SIN_TABLE_INPUT_WIDTH'd202: out <= `SIN_TABLE_OUTPUT_WIDTH'd18;
`SIN_TABLE_INPUT_WIDTH'd203: out <= `SIN_TABLE_OUTPUT_WIDTH'd21;
`SIN_TABLE_INPUT_WIDTH'd204: out <= `SIN_TABLE_OUTPUT_WIDTH'd25;
`SIN_TABLE_INPUT_WIDTH'd205: out <= `SIN_TABLE_OUTPUT_WIDTH'd29;
`SIN_TABLE_INPUT_WIDTH'd206: out <= `SIN_TABLE_OUTPUT_WIDTH'd33;
`SIN_TABLE_INPUT_WIDTH'd207: out <= `SIN_TABLE_OUTPUT_WIDTH'd38;
`SIN_TABLE_INPUT_WIDTH'd208: out <= `SIN_TABLE_OUTPUT_WIDTH'd43;
`SIN_TABLE_INPUT_WIDTH'd209: out <= `SIN_TABLE_OUTPUT_WIDTH'd48;
`SIN_TABLE_INPUT_WIDTH'd210: out <= `SIN_TABLE_OUTPUT_WIDTH'd54;
`SIN_TABLE_INPUT_WIDTH'd211: out <= `SIN_TABLE_OUTPUT_WIDTH'd59;
`SIN_TABLE_INPUT_WIDTH'd212: out <= `SIN_TABLE_OUTPUT_WIDTH'd65;
`SIN_TABLE_INPUT_WIDTH'd213: out <= `SIN_TABLE_OUTPUT_WIDTH'd72;
`SIN_TABLE_INPUT_WIDTH'd214: out <= `SIN_TABLE_OUTPUT_WIDTH'd78;
`SIN_TABLE_INPUT_WIDTH'd215: out <= `SIN_TABLE_OUTPUT_WIDTH'd85;
`SIN_TABLE_INPUT_WIDTH'd216: out <= `SIN_TABLE_OUTPUT_WIDTH'd92;
`SIN_TABLE_INPUT_WIDTH'd217: out <= `SIN_TABLE_OUTPUT_WIDTH'd100;
`SIN_TABLE_INPUT_WIDTH'd218: out <= `SIN_TABLE_OUTPUT_WIDTH'd107;
`SIN_TABLE_INPUT_WIDTH'd219: out <= `SIN_TABLE_OUTPUT_WIDTH'd115;
`SIN_TABLE_INPUT_WIDTH'd220: out <= `SIN_TABLE_OUTPUT_WIDTH'd123;
`SIN_TABLE_INPUT_WIDTH'd221: out <= `SIN_TABLE_OUTPUT_WIDTH'd131;
`SIN_TABLE_INPUT_WIDTH'd222: out <= `SIN_TABLE_OUTPUT_WIDTH'd140;
`SIN_TABLE_INPUT_WIDTH'd223: out <= `SIN_TABLE_OUTPUT_WIDTH'd149;
```

```
      `SIN_TABLE_INPUT_WIDTH'd224: out <= `SIN_TABLE_OUTPUT_WIDTH'd158;
      `SIN_TABLE_INPUT_WIDTH'd225: out <= `SIN_TABLE_OUTPUT_WIDTH'd167;
      `SIN_TABLE_INPUT_WIDTH'd226: out <= `SIN_TABLE_OUTPUT_WIDTH'd176;
      `SIN_TABLE_INPUT_WIDTH'd227: out <= `SIN_TABLE_OUTPUT_WIDTH'd186;
      `SIN_TABLE_INPUT_WIDTH'd228: out <= `SIN_TABLE_OUTPUT_WIDTH'd196;
      `SIN_TABLE_INPUT_WIDTH'd229: out <= `SIN_TABLE_OUTPUT_WIDTH'd206;
      `SIN_TABLE_INPUT_WIDTH'd230: out <= `SIN_TABLE_OUTPUT_WIDTH'd216;
      `SIN_TABLE_INPUT_WIDTH'd231: out <= `SIN_TABLE_OUTPUT_WIDTH'd226;
      `SIN_TABLE_INPUT_WIDTH'd232: out <= `SIN_TABLE_OUTPUT_WIDTH'd237;
      `SIN_TABLE_INPUT_WIDTH'd233: out <= `SIN_TABLE_OUTPUT_WIDTH'd248;
      `SIN_TABLE_INPUT_WIDTH'd234: out <= `SIN_TABLE_OUTPUT_WIDTH'd258;
      `SIN_TABLE_INPUT_WIDTH'd235: out <= `SIN_TABLE_OUTPUT_WIDTH'd270;
      `SIN_TABLE_INPUT_WIDTH'd236: out <= `SIN_TABLE_OUTPUT_WIDTH'd281;
      `SIN_TABLE_INPUT_WIDTH'd237: out <= `SIN_TABLE_OUTPUT_WIDTH'd292;
      `SIN_TABLE_INPUT_WIDTH'd238: out <= `SIN_TABLE_OUTPUT_WIDTH'd303;
      `SIN_TABLE_INPUT_WIDTH'd239: out <= `SIN_TABLE_OUTPUT_WIDTH'd315;
      `SIN_TABLE_INPUT_WIDTH'd240: out <= `SIN_TABLE_OUTPUT_WIDTH'd327;
      `SIN_TABLE_INPUT_WIDTH'd241: out <= `SIN_TABLE_OUTPUT_WIDTH'd339;
      `SIN_TABLE_INPUT_WIDTH'd242: out <= `SIN_TABLE_OUTPUT_WIDTH'd350;
      `SIN_TABLE_INPUT_WIDTH'd243: out <= `SIN_TABLE_OUTPUT_WIDTH'd362;
      `SIN_TABLE_INPUT_WIDTH'd244: out <= `SIN_TABLE_OUTPUT_WIDTH'd375;
      `SIN_TABLE_INPUT_WIDTH'd245: out <= `SIN_TABLE_OUTPUT_WIDTH'd387;
      `SIN_TABLE_INPUT_WIDTH'd246: out <= `SIN_TABLE_OUTPUT_WIDTH'd399;
      `SIN_TABLE_INPUT_WIDTH'd247: out <= `SIN_TABLE_OUTPUT_WIDTH'd411;
      `SIN_TABLE_INPUT_WIDTH'd248: out <= `SIN_TABLE_OUTPUT_WIDTH'd424;
      `SIN_TABLE_INPUT_WIDTH'd249: out <= `SIN_TABLE_OUTPUT_WIDTH'd436;
      `SIN_TABLE_INPUT_WIDTH'd250: out <= `SIN_TABLE_OUTPUT_WIDTH'd449;
      `SIN_TABLE_INPUT_WIDTH'd251: out <= `SIN_TABLE_OUTPUT_WIDTH'd461;
      `SIN_TABLE_INPUT_WIDTH'd252: out <= `SIN_TABLE_OUTPUT_WIDTH'd474;
      `SIN_TABLE_INPUT_WIDTH'd253: out <= `SIN_TABLE_OUTPUT_WIDTH'd486;
      `SIN_TABLE_INPUT_WIDTH'd254: out <= `SIN_TABLE_OUTPUT_WIDTH'd499;
      `SIN_TABLE_INPUT_WIDTH'd255: out <= `SIN_TABLE_OUTPUT_WIDTH'd511;
      default: out <= `SIN_TABLE_OUTPUT_WIDTH'hx;
    endcase
  end
endmodule



//This file was automatically generated by
//Matlab on 01-Sep-2009 16:16:19.
`include "tri_table.vh"

module tri_table(
    input [`TRI_TABLE_INPUT_RANGE]       in,
    output reg [`TRI_TABLE_OUTPUT_RANGE] out);

  always @(in) begin
    casez(in)
      `TRI_TABLE_INPUT_WIDTH'd0: out <= `TRI_TABLE_OUTPUT_WIDTH'd512;
      `TRI_TABLE_INPUT_WIDTH'd1: out <= `TRI_TABLE_OUTPUT_WIDTH'd519;
      `TRI_TABLE_INPUT_WIDTH'd2: out <= `TRI_TABLE_OUTPUT_WIDTH'd527;
      `TRI_TABLE_INPUT_WIDTH'd3: out <= `TRI_TABLE_OUTPUT_WIDTH'd535;
      `TRI_TABLE_INPUT_WIDTH'd4: out <= `TRI_TABLE_OUTPUT_WIDTH'd543;
      `TRI_TABLE_INPUT_WIDTH'd5: out <= `TRI_TABLE_OUTPUT_WIDTH'd551;
      `TRI_TABLE_INPUT_WIDTH'd6: out <= `TRI_TABLE_OUTPUT_WIDTH'd559;
      `TRI_TABLE_INPUT_WIDTH'd7: out <= `TRI_TABLE_OUTPUT_WIDTH'd567;
```

```
`TRI_TABLE_INPUT_WIDTH'd8: out <= `TRI_TABLE_OUTPUT_WIDTH'd575;
`TRI_TABLE_INPUT_WIDTH'd9: out <= `TRI_TABLE_OUTPUT_WIDTH'd583;
`TRI_TABLE_INPUT_WIDTH'd10: out <= `TRI_TABLE_OUTPUT_WIDTH'd591;
`TRI_TABLE_INPUT_WIDTH'd11: out <= `TRI_TABLE_OUTPUT_WIDTH'd599;
`TRI_TABLE_INPUT_WIDTH'd12: out <= `TRI_TABLE_OUTPUT_WIDTH'd607;
`TRI_TABLE_INPUT_WIDTH'd13: out <= `TRI_TABLE_OUTPUT_WIDTH'd615;
`TRI_TABLE_INPUT_WIDTH'd14: out <= `TRI_TABLE_OUTPUT_WIDTH'd623;
`TRI_TABLE_INPUT_WIDTH'd15: out <= `TRI_TABLE_OUTPUT_WIDTH'd631;
`TRI_TABLE_INPUT_WIDTH'd16: out <= `TRI_TABLE_OUTPUT_WIDTH'd639;
`TRI_TABLE_INPUT_WIDTH'd17: out <= `TRI_TABLE_OUTPUT_WIDTH'd647;
`TRI_TABLE_INPUT_WIDTH'd18: out <= `TRI_TABLE_OUTPUT_WIDTH'd655;
`TRI_TABLE_INPUT_WIDTH'd19: out <= `TRI_TABLE_OUTPUT_WIDTH'd663;
`TRI_TABLE_INPUT_WIDTH'd20: out <= `TRI_TABLE_OUTPUT_WIDTH'd671;
`TRI_TABLE_INPUT_WIDTH'd21: out <= `TRI_TABLE_OUTPUT_WIDTH'd679;
`TRI_TABLE_INPUT_WIDTH'd22: out <= `TRI_TABLE_OUTPUT_WIDTH'd687;
`TRI_TABLE_INPUT_WIDTH'd23: out <= `TRI_TABLE_OUTPUT_WIDTH'd695;
`TRI_TABLE_INPUT_WIDTH'd24: out <= `TRI_TABLE_OUTPUT_WIDTH'd703;
`TRI_TABLE_INPUT_WIDTH'd25: out <= `TRI_TABLE_OUTPUT_WIDTH'd711;
`TRI_TABLE_INPUT_WIDTH'd26: out <= `TRI_TABLE_OUTPUT_WIDTH'd719;
`TRI_TABLE_INPUT_WIDTH'd27: out <= `TRI_TABLE_OUTPUT_WIDTH'd727;
`TRI_TABLE_INPUT_WIDTH'd28: out <= `TRI_TABLE_OUTPUT_WIDTH'd735;
`TRI_TABLE_INPUT_WIDTH'd29: out <= `TRI_TABLE_OUTPUT_WIDTH'd743;
`TRI_TABLE_INPUT_WIDTH'd30: out <= `TRI_TABLE_OUTPUT_WIDTH'd751;
`TRI_TABLE_INPUT_WIDTH'd31: out <= `TRI_TABLE_OUTPUT_WIDTH'd759;
`TRI_TABLE_INPUT_WIDTH'd32: out <= `TRI_TABLE_OUTPUT_WIDTH'd767;
`TRI_TABLE_INPUT_WIDTH'd33: out <= `TRI_TABLE_OUTPUT_WIDTH'd775;
`TRI_TABLE_INPUT_WIDTH'd34: out <= `TRI_TABLE_OUTPUT_WIDTH'd783;
`TRI_TABLE_INPUT_WIDTH'd35: out <= `TRI_TABLE_OUTPUT_WIDTH'd791;
`TRI_TABLE_INPUT_WIDTH'd36: out <= `TRI_TABLE_OUTPUT_WIDTH'd799;
`TRI_TABLE_INPUT_WIDTH'd37: out <= `TRI_TABLE_OUTPUT_WIDTH'd807;
`TRI_TABLE_INPUT_WIDTH'd38: out <= `TRI_TABLE_OUTPUT_WIDTH'd815;
`TRI_TABLE_INPUT_WIDTH'd39: out <= `TRI_TABLE_OUTPUT_WIDTH'd823;
`TRI_TABLE_INPUT_WIDTH'd40: out <= `TRI_TABLE_OUTPUT_WIDTH'd831;
`TRI_TABLE_INPUT_WIDTH'd41: out <= `TRI_TABLE_OUTPUT_WIDTH'd839;
`TRI_TABLE_INPUT_WIDTH'd42: out <= `TRI_TABLE_OUTPUT_WIDTH'd847;
`TRI_TABLE_INPUT_WIDTH'd43: out <= `TRI_TABLE_OUTPUT_WIDTH'd855;
`TRI_TABLE_INPUT_WIDTH'd44: out <= `TRI_TABLE_OUTPUT_WIDTH'd863;
`TRI_TABLE_INPUT_WIDTH'd45: out <= `TRI_TABLE_OUTPUT_WIDTH'd871;
`TRI_TABLE_INPUT_WIDTH'd46: out <= `TRI_TABLE_OUTPUT_WIDTH'd879;
`TRI_TABLE_INPUT_WIDTH'd47: out <= `TRI_TABLE_OUTPUT_WIDTH'd887;
`TRI_TABLE_INPUT_WIDTH'd48: out <= `TRI_TABLE_OUTPUT_WIDTH'd895;
`TRI_TABLE_INPUT_WIDTH'd49: out <= `TRI_TABLE_OUTPUT_WIDTH'd903;
`TRI_TABLE_INPUT_WIDTH'd50: out <= `TRI_TABLE_OUTPUT_WIDTH'd911;
`TRI_TABLE_INPUT_WIDTH'd51: out <= `TRI_TABLE_OUTPUT_WIDTH'd919;
`TRI_TABLE_INPUT_WIDTH'd52: out <= `TRI_TABLE_OUTPUT_WIDTH'd927;
`TRI_TABLE_INPUT_WIDTH'd53: out <= `TRI_TABLE_OUTPUT_WIDTH'd935;
`TRI_TABLE_INPUT_WIDTH'd54: out <= `TRI_TABLE_OUTPUT_WIDTH'd943;
`TRI_TABLE_INPUT_WIDTH'd55: out <= `TRI_TABLE_OUTPUT_WIDTH'd951;
`TRI_TABLE_INPUT_WIDTH'd56: out <= `TRI_TABLE_OUTPUT_WIDTH'd959;
`TRI_TABLE_INPUT_WIDTH'd57: out <= `TRI_TABLE_OUTPUT_WIDTH'd967;
`TRI_TABLE_INPUT_WIDTH'd58: out <= `TRI_TABLE_OUTPUT_WIDTH'd975;
`TRI_TABLE_INPUT_WIDTH'd59: out <= `TRI_TABLE_OUTPUT_WIDTH'd983;
`TRI_TABLE_INPUT_WIDTH'd60: out <= `TRI_TABLE_OUTPUT_WIDTH'd991;
`TRI_TABLE_INPUT_WIDTH'd61: out <= `TRI_TABLE_OUTPUT_WIDTH'd999;
`TRI_TABLE_INPUT_WIDTH'd62: out <= `TRI_TABLE_OUTPUT_WIDTH'd1007;
`TRI_TABLE_INPUT_WIDTH'd63: out <= `TRI_TABLE_OUTPUT_WIDTH'd1015;
`TRI_TABLE_INPUT_WIDTH'd64: out <= `TRI_TABLE_OUTPUT_WIDTH'd1023;
```

```
`TRI_TABLE_INPUT_WIDTH'd65: out <= `TRI_TABLE_OUTPUT_WIDTH'd1015;
`TRI_TABLE_INPUT_WIDTH'd66: out <= `TRI_TABLE_OUTPUT_WIDTH'd1007;
`TRI_TABLE_INPUT_WIDTH'd67: out <= `TRI_TABLE_OUTPUT_WIDTH'd999;
`TRI_TABLE_INPUT_WIDTH'd68: out <= `TRI_TABLE_OUTPUT_WIDTH'd991;
`TRI_TABLE_INPUT_WIDTH'd69: out <= `TRI_TABLE_OUTPUT_WIDTH'd983;
`TRI_TABLE_INPUT_WIDTH'd70: out <= `TRI_TABLE_OUTPUT_WIDTH'd975;
`TRI_TABLE_INPUT_WIDTH'd71: out <= `TRI_TABLE_OUTPUT_WIDTH'd967;
`TRI_TABLE_INPUT_WIDTH'd72: out <= `TRI_TABLE_OUTPUT_WIDTH'd959;
`TRI_TABLE_INPUT_WIDTH'd73: out <= `TRI_TABLE_OUTPUT_WIDTH'd951;
`TRI_TABLE_INPUT_WIDTH'd74: out <= `TRI_TABLE_OUTPUT_WIDTH'd942;
`TRI_TABLE_INPUT_WIDTH'd75: out <= `TRI_TABLE_OUTPUT_WIDTH'd934;
`TRI_TABLE_INPUT_WIDTH'd76: out <= `TRI_TABLE_OUTPUT_WIDTH'd926;
`TRI_TABLE_INPUT_WIDTH'd77: out <= `TRI_TABLE_OUTPUT_WIDTH'd918;
`TRI_TABLE_INPUT_WIDTH'd78: out <= `TRI_TABLE_OUTPUT_WIDTH'd910;
`TRI_TABLE_INPUT_WIDTH'd79: out <= `TRI_TABLE_OUTPUT_WIDTH'd902;
`TRI_TABLE_INPUT_WIDTH'd80: out <= `TRI_TABLE_OUTPUT_WIDTH'd894;
`TRI_TABLE_INPUT_WIDTH'd81: out <= `TRI_TABLE_OUTPUT_WIDTH'd886;
`TRI_TABLE_INPUT_WIDTH'd82: out <= `TRI_TABLE_OUTPUT_WIDTH'd878;
`TRI_TABLE_INPUT_WIDTH'd83: out <= `TRI_TABLE_OUTPUT_WIDTH'd870;
`TRI_TABLE_INPUT_WIDTH'd84: out <= `TRI_TABLE_OUTPUT_WIDTH'd862;
`TRI_TABLE_INPUT_WIDTH'd85: out <= `TRI_TABLE_OUTPUT_WIDTH'd854;
`TRI_TABLE_INPUT_WIDTH'd86: out <= `TRI_TABLE_OUTPUT_WIDTH'd846;
`TRI_TABLE_INPUT_WIDTH'd87: out <= `TRI_TABLE_OUTPUT_WIDTH'd838;
`TRI_TABLE_INPUT_WIDTH'd88: out <= `TRI_TABLE_OUTPUT_WIDTH'd830;
`TRI_TABLE_INPUT_WIDTH'd89: out <= `TRI_TABLE_OUTPUT_WIDTH'd822;
`TRI_TABLE_INPUT_WIDTH'd90: out <= `TRI_TABLE_OUTPUT_WIDTH'd814;
`TRI_TABLE_INPUT_WIDTH'd91: out <= `TRI_TABLE_OUTPUT_WIDTH'd806;
`TRI_TABLE_INPUT_WIDTH'd92: out <= `TRI_TABLE_OUTPUT_WIDTH'd797;
`TRI_TABLE_INPUT_WIDTH'd93: out <= `TRI_TABLE_OUTPUT_WIDTH'd789;
`TRI_TABLE_INPUT_WIDTH'd94: out <= `TRI_TABLE_OUTPUT_WIDTH'd781;
`TRI_TABLE_INPUT_WIDTH'd95: out <= `TRI_TABLE_OUTPUT_WIDTH'd773;
`TRI_TABLE_INPUT_WIDTH'd96: out <= `TRI_TABLE_OUTPUT_WIDTH'd765;
`TRI_TABLE_INPUT_WIDTH'd97: out <= `TRI_TABLE_OUTPUT_WIDTH'd757;
`TRI_TABLE_INPUT_WIDTH'd98: out <= `TRI_TABLE_OUTPUT_WIDTH'd749;
`TRI_TABLE_INPUT_WIDTH'd99: out <= `TRI_TABLE_OUTPUT_WIDTH'd741;
`TRI_TABLE_INPUT_WIDTH'd100: out <= `TRI_TABLE_OUTPUT_WIDTH'd733;
`TRI_TABLE_INPUT_WIDTH'd101: out <= `TRI_TABLE_OUTPUT_WIDTH'd725;
`TRI_TABLE_INPUT_WIDTH'd102: out <= `TRI_TABLE_OUTPUT_WIDTH'd717;
`TRI_TABLE_INPUT_WIDTH'd103: out <= `TRI_TABLE_OUTPUT_WIDTH'd709;
`TRI_TABLE_INPUT_WIDTH'd104: out <= `TRI_TABLE_OUTPUT_WIDTH'd701;
`TRI_TABLE_INPUT_WIDTH'd105: out <= `TRI_TABLE_OUTPUT_WIDTH'd693;
`TRI_TABLE_INPUT_WIDTH'd106: out <= `TRI_TABLE_OUTPUT_WIDTH'd685;
`TRI_TABLE_INPUT_WIDTH'd107: out <= `TRI_TABLE_OUTPUT_WIDTH'd677;
`TRI_TABLE_INPUT_WIDTH'd108: out <= `TRI_TABLE_OUTPUT_WIDTH'd669;
`TRI_TABLE_INPUT_WIDTH'd109: out <= `TRI_TABLE_OUTPUT_WIDTH'd661;
`TRI_TABLE_INPUT_WIDTH'd110: out <= `TRI_TABLE_OUTPUT_WIDTH'd652;
`TRI_TABLE_INPUT_WIDTH'd111: out <= `TRI_TABLE_OUTPUT_WIDTH'd644;
`TRI_TABLE_INPUT_WIDTH'd112: out <= `TRI_TABLE_OUTPUT_WIDTH'd636;
`TRI_TABLE_INPUT_WIDTH'd113: out <= `TRI_TABLE_OUTPUT_WIDTH'd628;
`TRI_TABLE_INPUT_WIDTH'd114: out <= `TRI_TABLE_OUTPUT_WIDTH'd620;
`TRI_TABLE_INPUT_WIDTH'd115: out <= `TRI_TABLE_OUTPUT_WIDTH'd612;
`TRI_TABLE_INPUT_WIDTH'd116: out <= `TRI_TABLE_OUTPUT_WIDTH'd604;
`TRI_TABLE_INPUT_WIDTH'd117: out <= `TRI_TABLE_OUTPUT_WIDTH'd596;
`TRI_TABLE_INPUT_WIDTH'd118: out <= `TRI_TABLE_OUTPUT_WIDTH'd588;
`TRI_TABLE_INPUT_WIDTH'd119: out <= `TRI_TABLE_OUTPUT_WIDTH'd580;
`TRI_TABLE_INPUT_WIDTH'd120: out <= `TRI_TABLE_OUTPUT_WIDTH'd572;
`TRI_TABLE_INPUT_WIDTH'd121: out <= `TRI_TABLE_OUTPUT_WIDTH'd564;
```

```
`TRI_TABLE_INPUT_WIDTH'd122: out <= `TRI_TABLE_OUTPUT_WIDTH'd556;
`TRI_TABLE_INPUT_WIDTH'd123: out <= `TRI_TABLE_OUTPUT_WIDTH'd548;
`TRI_TABLE_INPUT_WIDTH'd124: out <= `TRI_TABLE_OUTPUT_WIDTH'd540;
`TRI_TABLE_INPUT_WIDTH'd125: out <= `TRI_TABLE_OUTPUT_WIDTH'd532;
`TRI_TABLE_INPUT_WIDTH'd126: out <= `TRI_TABLE_OUTPUT_WIDTH'd524;
`TRI_TABLE_INPUT_WIDTH'd127: out <= `TRI_TABLE_OUTPUT_WIDTH'd516;
`TRI_TABLE_INPUT_WIDTH'd128: out <= `TRI_TABLE_OUTPUT_WIDTH'd507;
`TRI_TABLE_INPUT_WIDTH'd129: out <= `TRI_TABLE_OUTPUT_WIDTH'd499;
`TRI_TABLE_INPUT_WIDTH'd130: out <= `TRI_TABLE_OUTPUT_WIDTH'd491;
`TRI_TABLE_INPUT_WIDTH'd131: out <= `TRI_TABLE_OUTPUT_WIDTH'd483;
`TRI_TABLE_INPUT_WIDTH'd132: out <= `TRI_TABLE_OUTPUT_WIDTH'd475;
`TRI_TABLE_INPUT_WIDTH'd133: out <= `TRI_TABLE_OUTPUT_WIDTH'd467;
`TRI_TABLE_INPUT_WIDTH'd134: out <= `TRI_TABLE_OUTPUT_WIDTH'd459;
`TRI_TABLE_INPUT_WIDTH'd135: out <= `TRI_TABLE_OUTPUT_WIDTH'd451;
`TRI_TABLE_INPUT_WIDTH'd136: out <= `TRI_TABLE_OUTPUT_WIDTH'd443;
`TRI_TABLE_INPUT_WIDTH'd137: out <= `TRI_TABLE_OUTPUT_WIDTH'd435;
`TRI_TABLE_INPUT_WIDTH'd138: out <= `TRI_TABLE_OUTPUT_WIDTH'd427;
`TRI_TABLE_INPUT_WIDTH'd139: out <= `TRI_TABLE_OUTPUT_WIDTH'd419;
`TRI_TABLE_INPUT_WIDTH'd140: out <= `TRI_TABLE_OUTPUT_WIDTH'd411;
`TRI_TABLE_INPUT_WIDTH'd141: out <= `TRI_TABLE_OUTPUT_WIDTH'd403;
`TRI_TABLE_INPUT_WIDTH'd142: out <= `TRI_TABLE_OUTPUT_WIDTH'd395;
`TRI_TABLE_INPUT_WIDTH'd143: out <= `TRI_TABLE_OUTPUT_WIDTH'd387;
`TRI_TABLE_INPUT_WIDTH'd144: out <= `TRI_TABLE_OUTPUT_WIDTH'd379;
`TRI_TABLE_INPUT_WIDTH'd145: out <= `TRI_TABLE_OUTPUT_WIDTH'd371;
`TRI_TABLE_INPUT_WIDTH'd146: out <= `TRI_TABLE_OUTPUT_WIDTH'd362;
`TRI_TABLE_INPUT_WIDTH'd147: out <= `TRI_TABLE_OUTPUT_WIDTH'd354;
`TRI_TABLE_INPUT_WIDTH'd148: out <= `TRI_TABLE_OUTPUT_WIDTH'd346;
`TRI_TABLE_INPUT_WIDTH'd149: out <= `TRI_TABLE_OUTPUT_WIDTH'd338;
`TRI_TABLE_INPUT_WIDTH'd150: out <= `TRI_TABLE_OUTPUT_WIDTH'd330;
`TRI_TABLE_INPUT_WIDTH'd151: out <= `TRI_TABLE_OUTPUT_WIDTH'd322;
`TRI_TABLE_INPUT_WIDTH'd152: out <= `TRI_TABLE_OUTPUT_WIDTH'd314;
`TRI_TABLE_INPUT_WIDTH'd153: out <= `TRI_TABLE_OUTPUT_WIDTH'd306;
`TRI_TABLE_INPUT_WIDTH'd154: out <= `TRI_TABLE_OUTPUT_WIDTH'd298;
`TRI_TABLE_INPUT_WIDTH'd155: out <= `TRI_TABLE_OUTPUT_WIDTH'd290;
`TRI_TABLE_INPUT_WIDTH'd156: out <= `TRI_TABLE_OUTPUT_WIDTH'd282;
`TRI_TABLE_INPUT_WIDTH'd157: out <= `TRI_TABLE_OUTPUT_WIDTH'd274;
`TRI_TABLE_INPUT_WIDTH'd158: out <= `TRI_TABLE_OUTPUT_WIDTH'd266;
`TRI_TABLE_INPUT_WIDTH'd159: out <= `TRI_TABLE_OUTPUT_WIDTH'd258;
`TRI_TABLE_INPUT_WIDTH'd160: out <= `TRI_TABLE_OUTPUT_WIDTH'd250;
`TRI_TABLE_INPUT_WIDTH'd161: out <= `TRI_TABLE_OUTPUT_WIDTH'd242;
`TRI_TABLE_INPUT_WIDTH'd162: out <= `TRI_TABLE_OUTPUT_WIDTH'd234;
`TRI_TABLE_INPUT_WIDTH'd163: out <= `TRI_TABLE_OUTPUT_WIDTH'd226;
`TRI_TABLE_INPUT_WIDTH'd164: out <= `TRI_TABLE_OUTPUT_WIDTH'd217;
`TRI_TABLE_INPUT_WIDTH'd165: out <= `TRI_TABLE_OUTPUT_WIDTH'd209;
`TRI_TABLE_INPUT_WIDTH'd166: out <= `TRI_TABLE_OUTPUT_WIDTH'd201;
`TRI_TABLE_INPUT_WIDTH'd167: out <= `TRI_TABLE_OUTPUT_WIDTH'd193;
`TRI_TABLE_INPUT_WIDTH'd168: out <= `TRI_TABLE_OUTPUT_WIDTH'd185;
`TRI_TABLE_INPUT_WIDTH'd169: out <= `TRI_TABLE_OUTPUT_WIDTH'd177;
`TRI_TABLE_INPUT_WIDTH'd170: out <= `TRI_TABLE_OUTPUT_WIDTH'd169;
`TRI_TABLE_INPUT_WIDTH'd171: out <= `TRI_TABLE_OUTPUT_WIDTH'd161;
`TRI_TABLE_INPUT_WIDTH'd172: out <= `TRI_TABLE_OUTPUT_WIDTH'd153;
`TRI_TABLE_INPUT_WIDTH'd173: out <= `TRI_TABLE_OUTPUT_WIDTH'd145;
`TRI_TABLE_INPUT_WIDTH'd174: out <= `TRI_TABLE_OUTPUT_WIDTH'd137;
`TRI_TABLE_INPUT_WIDTH'd175: out <= `TRI_TABLE_OUTPUT_WIDTH'd129;
`TRI_TABLE_INPUT_WIDTH'd176: out <= `TRI_TABLE_OUTPUT_WIDTH'd121;
`TRI_TABLE_INPUT_WIDTH'd177: out <= `TRI_TABLE_OUTPUT_WIDTH'd113;
`TRI_TABLE_INPUT_WIDTH'd178: out <= `TRI_TABLE_OUTPUT_WIDTH'd105;
```

```
`TRI_TABLE_INPUT_WIDTH'd179: out <= `TRI_TABLE_OUTPUT_WIDTH'd97;
`TRI_TABLE_INPUT_WIDTH'd180: out <= `TRI_TABLE_OUTPUT_WIDTH'd89;
`TRI_TABLE_INPUT_WIDTH'd181: out <= `TRI_TABLE_OUTPUT_WIDTH'd81;
`TRI_TABLE_INPUT_WIDTH'd182: out <= `TRI_TABLE_OUTPUT_WIDTH'd72;
`TRI_TABLE_INPUT_WIDTH'd183: out <= `TRI_TABLE_OUTPUT_WIDTH'd64;
`TRI_TABLE_INPUT_WIDTH'd184: out <= `TRI_TABLE_OUTPUT_WIDTH'd56;
`TRI_TABLE_INPUT_WIDTH'd185: out <= `TRI_TABLE_OUTPUT_WIDTH'd48;
`TRI_TABLE_INPUT_WIDTH'd186: out <= `TRI_TABLE_OUTPUT_WIDTH'd40;
`TRI_TABLE_INPUT_WIDTH'd187: out <= `TRI_TABLE_OUTPUT_WIDTH'd32;
`TRI_TABLE_INPUT_WIDTH'd188: out <= `TRI_TABLE_OUTPUT_WIDTH'd24;
`TRI_TABLE_INPUT_WIDTH'd189: out <= `TRI_TABLE_OUTPUT_WIDTH'd16;
`TRI_TABLE_INPUT_WIDTH'd190: out <= `TRI_TABLE_OUTPUT_WIDTH'd8;
`TRI_TABLE_INPUT_WIDTH'd191: out <= `TRI_TABLE_OUTPUT_WIDTH'd0;
`TRI_TABLE_INPUT_WIDTH'd192: out <= `TRI_TABLE_OUTPUT_WIDTH'd8;
`TRI_TABLE_INPUT_WIDTH'd193: out <= `TRI_TABLE_OUTPUT_WIDTH'd16;
`TRI_TABLE_INPUT_WIDTH'd194: out <= `TRI_TABLE_OUTPUT_WIDTH'd24;
`TRI_TABLE_INPUT_WIDTH'd195: out <= `TRI_TABLE_OUTPUT_WIDTH'd32;
`TRI_TABLE_INPUT_WIDTH'd196: out <= `TRI_TABLE_OUTPUT_WIDTH'd40;
`TRI_TABLE_INPUT_WIDTH'd197: out <= `TRI_TABLE_OUTPUT_WIDTH'd48;
`TRI_TABLE_INPUT_WIDTH'd198: out <= `TRI_TABLE_OUTPUT_WIDTH'd56;
`TRI_TABLE_INPUT_WIDTH'd199: out <= `TRI_TABLE_OUTPUT_WIDTH'd64;
`TRI_TABLE_INPUT_WIDTH'd200: out <= `TRI_TABLE_OUTPUT_WIDTH'd72;
`TRI_TABLE_INPUT_WIDTH'd201: out <= `TRI_TABLE_OUTPUT_WIDTH'd80;
`TRI_TABLE_INPUT_WIDTH'd202: out <= `TRI_TABLE_OUTPUT_WIDTH'd88;
`TRI_TABLE_INPUT_WIDTH'd203: out <= `TRI_TABLE_OUTPUT_WIDTH'd96;
`TRI_TABLE_INPUT_WIDTH'd204: out <= `TRI_TABLE_OUTPUT_WIDTH'd104;
`TRI_TABLE_INPUT_WIDTH'd205: out <= `TRI_TABLE_OUTPUT_WIDTH'd112;
`TRI_TABLE_INPUT_WIDTH'd206: out <= `TRI_TABLE_OUTPUT_WIDTH'd120;
`TRI_TABLE_INPUT_WIDTH'd207: out <= `TRI_TABLE_OUTPUT_WIDTH'd128;
`TRI_TABLE_INPUT_WIDTH'd208: out <= `TRI_TABLE_OUTPUT_WIDTH'd136;
`TRI_TABLE_INPUT_WIDTH'd209: out <= `TRI_TABLE_OUTPUT_WIDTH'd144;
`TRI_TABLE_INPUT_WIDTH'd210: out <= `TRI_TABLE_OUTPUT_WIDTH'd152;
`TRI_TABLE_INPUT_WIDTH'd211: out <= `TRI_TABLE_OUTPUT_WIDTH'd160;
`TRI_TABLE_INPUT_WIDTH'd212: out <= `TRI_TABLE_OUTPUT_WIDTH'd168;
`TRI_TABLE_INPUT_WIDTH'd213: out <= `TRI_TABLE_OUTPUT_WIDTH'd176;
`TRI_TABLE_INPUT_WIDTH'd214: out <= `TRI_TABLE_OUTPUT_WIDTH'd184;
`TRI_TABLE_INPUT_WIDTH'd215: out <= `TRI_TABLE_OUTPUT_WIDTH'd192;
`TRI_TABLE_INPUT_WIDTH'd216: out <= `TRI_TABLE_OUTPUT_WIDTH'd200;
`TRI_TABLE_INPUT_WIDTH'd217: out <= `TRI_TABLE_OUTPUT_WIDTH'd208;
`TRI_TABLE_INPUT_WIDTH'd218: out <= `TRI_TABLE_OUTPUT_WIDTH'd216;
`TRI_TABLE_INPUT_WIDTH'd219: out <= `TRI_TABLE_OUTPUT_WIDTH'd224;
`TRI_TABLE_INPUT_WIDTH'd220: out <= `TRI_TABLE_OUTPUT_WIDTH'd232;
`TRI_TABLE_INPUT_WIDTH'd221: out <= `TRI_TABLE_OUTPUT_WIDTH'd240;
`TRI_TABLE_INPUT_WIDTH'd222: out <= `TRI_TABLE_OUTPUT_WIDTH'd248;
`TRI_TABLE_INPUT_WIDTH'd223: out <= `TRI_TABLE_OUTPUT_WIDTH'd256;
`TRI_TABLE_INPUT_WIDTH'd224: out <= `TRI_TABLE_OUTPUT_WIDTH'd264;
`TRI_TABLE_INPUT_WIDTH'd225: out <= `TRI_TABLE_OUTPUT_WIDTH'd272;
`TRI_TABLE_INPUT_WIDTH'd226: out <= `TRI_TABLE_OUTPUT_WIDTH'd280;
`TRI_TABLE_INPUT_WIDTH'd227: out <= `TRI_TABLE_OUTPUT_WIDTH'd288;
`TRI_TABLE_INPUT_WIDTH'd228: out <= `TRI_TABLE_OUTPUT_WIDTH'd296;
`TRI_TABLE_INPUT_WIDTH'd229: out <= `TRI_TABLE_OUTPUT_WIDTH'd304;
`TRI_TABLE_INPUT_WIDTH'd230: out <= `TRI_TABLE_OUTPUT_WIDTH'd312;
`TRI_TABLE_INPUT_WIDTH'd231: out <= `TRI_TABLE_OUTPUT_WIDTH'd320;
`TRI_TABLE_INPUT_WIDTH'd232: out <= `TRI_TABLE_OUTPUT_WIDTH'd328;
`TRI_TABLE_INPUT_WIDTH'd233: out <= `TRI_TABLE_OUTPUT_WIDTH'd336;
`TRI_TABLE_INPUT_WIDTH'd234: out <= `TRI_TABLE_OUTPUT_WIDTH'd344;
`TRI_TABLE_INPUT_WIDTH'd235: out <= `TRI_TABLE_OUTPUT_WIDTH'd352;
```

```
        `TRI_TABLE_INPUT_WIDTH'd236: out <= `TRI_TABLE_OUTPUT_WIDTH'd360;
        `TRI_TABLE_INPUT_WIDTH'd237: out <= `TRI_TABLE_OUTPUT_WIDTH'd368;
        `TRI_TABLE_INPUT_WIDTH'd238: out <= `TRI_TABLE_OUTPUT_WIDTH'd376;
        `TRI_TABLE_INPUT_WIDTH'd239: out <= `TRI_TABLE_OUTPUT_WIDTH'd384;
        `TRI_TABLE_INPUT_WIDTH'd240: out <= `TRI_TABLE_OUTPUT_WIDTH'd392;
        `TRI_TABLE_INPUT_WIDTH'd241: out <= `TRI_TABLE_OUTPUT_WIDTH'd400;
        `TRI_TABLE_INPUT_WIDTH'd242: out <= `TRI_TABLE_OUTPUT_WIDTH'd408;
        `TRI_TABLE_INPUT_WIDTH'd243: out <= `TRI_TABLE_OUTPUT_WIDTH'd416;
        `TRI_TABLE_INPUT_WIDTH'd244: out <= `TRI_TABLE_OUTPUT_WIDTH'd424;
        `TRI_TABLE_INPUT_WIDTH'd245: out <= `TRI_TABLE_OUTPUT_WIDTH'd432;
        `TRI_TABLE_INPUT_WIDTH'd246: out <= `TRI_TABLE_OUTPUT_WIDTH'd440;
        `TRI_TABLE_INPUT_WIDTH'd247: out <= `TRI_TABLE_OUTPUT_WIDTH'd448;
        `TRI_TABLE_INPUT_WIDTH'd248: out <= `TRI_TABLE_OUTPUT_WIDTH'd456;
        `TRI_TABLE_INPUT_WIDTH'd249: out <= `TRI_TABLE_OUTPUT_WIDTH'd464;
        `TRI_TABLE_INPUT_WIDTH'd250: out <= `TRI_TABLE_OUTPUT_WIDTH'd472;
        `TRI_TABLE_INPUT_WIDTH'd251: out <= `TRI_TABLE_OUTPUT_WIDTH'd480;
        `TRI_TABLE_INPUT_WIDTH'd252: out <= `TRI_TABLE_OUTPUT_WIDTH'd488;
        `TRI_TABLE_INPUT_WIDTH'd253: out <= `TRI_TABLE_OUTPUT_WIDTH'd496;
        `TRI_TABLE_INPUT_WIDTH'd254: out <= `TRI_TABLE_OUTPUT_WIDTH'd504;
        `TRI_TABLE_INPUT_WIDTH'd255: out <= `TRI_TABLE_OUTPUT_WIDTH'd512;
        default: out <= `TRI_TABLE_OUTPUT_WIDTH'hx;
      endcase
    end
endmodule
```