|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Team: SIS  Members: Thi-Chi Nguyen, Tien-Luan Vu, Van-Quy Quach, Trong-Uoc Bui  School Name: University of Electronic and Telecomunication  Address: 144 – Xuan Thuy Street, Cau Giay Distict, Ha Noi city, Viet Nam  Size of all team members’ T-shirt   |  |  | | --- | --- | | Thi-Chi Nguyen | M | | Tien-Luan Vu | M | | Van-Quy Quach | M | | Trong-Uoc Bui |  |   The tasks level of design: Level 1 and Level 2. |

**Level 1: Beginner ROM design**

Design ROM and inprove the Cordic v file to meet the input and output specification which is represented in table 6 -1.

1. **Architechture description:** Below is I/O schematic of X\_Cordic

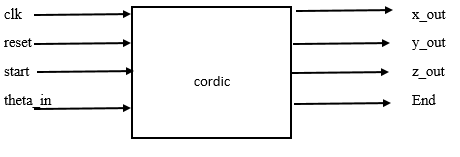


Figure 1. Top level schematic

In cordic block has two smaller block is *cordic\_data* and the *controller.* The main task is executed in cordic\_data. Block diagram is presented below describing the top level of the design of *cordic\_data*.

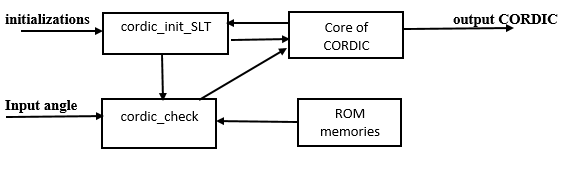


Figure 2. Block diagram of cordic\_data

1. **Explanation of the design circuit**

The design circuit step will follow the State Machine representation below:

|  |  |
| --- | --- |
|  | 1. Input*: Start, theta\_in* 2. Output: *cos\_out, sin\_out, End* 3. Wait Step: 4. If(*reset==*1) then 5. *Theta\_in;* 6. Goto Start; 7. Else 8. Goto Wait\_State; 9. End 10. Start Step: 11. If (*start* == 1) 12. Goto Initialization State; 13. Else 14. Goto wait state; 15. Initialization Step : 16. *Count* ++; 17. Goto Update state; 18. Update step: 19. Update initialized value of ; 20. 69; 21. Goto Checking State; 22. Checking Step: 23. Get value of *alpha* form ROM; 24. If(z\_in > theta\_in) 25. *alpha\_out = - alpha\_in* or*;* 26. *x\_delta\_out = (x\_in\_check >> address)* – 1; 27. *y\_delta\_out = (y\_in\_check >> address)* – 1; 28. Else 29. *alpha\_out = alpha\_in ;* 30. *x\_delta\_out = x\_in\_check >> address;* 31. *y\_delta\_out = y\_in\_check >> address;* 32. Goto Computing State; 33. Computing Step: 34. If (reset = 0) 35. *x\_out* = 0; 36. *y\_out* = 0; 37. else 38. if(*End* = 1) 39. *cos\_out = x\_out;* 40. *sin\_out = y\_out;* 41. *z\_out = z\_out;* 42. else 43. *x\_out = x\_in - y\_delta\_in;* 44. *y\_out = y\_in + x\_delta\_in;* 45. *z\_out = z\_in + alpha\_in* 46. Goto Update state |

Figure 3. State machine Representation of Cordic

X\_cordic architecture is based on the CORDIC method with some improvement to elimilate multiplers. This architecture will take the angle and the initializations of Cordic method such as , 69, it take 1 clock cycle to do this. The *Cordic\_init\_STL* play the transport role to keep the output of the loop number i as the input of the loop number for *cordic\_core* block. Checking block check whether the rotated angle is greater than the input angle or not to decide the direction and updated value to x and y on the next rotation and provide value of *zeta* for Cordic Core. Because the rotation angle are fixed so it need a ROM to save them. ROM has 2^4 \* 16 Cell. The main computation is implemented in Cordic core. It take 16 clock cycles to meet End signal enable finish.

1. **An appeal point and originality**

This architecture does not use any multipler and speed up from the frequence 112 Hz to the frequence 146 Hz.

1. **Critical path speed A circuit domain**
2. **HDL. Code** 
   1. **Cordic\_init\_SLT**

|  |
| --- |
| module codric\_init\_SLT (  select, // Select as output value from X\_init or X\_in  x\_init, // Initial value of x  y\_init, // Initial value of y  z\_init, // Initial value of z  x\_in, // Itarated value of x  y\_in, // Itarated value of y  z\_in, // Itarated value of z  x\_out, // Selected value of x  y\_out, // Selected value of y  z\_out // Selected value of z  );  //Inputs  input signed [15:0] x\_init;  input signed [15:0] y\_init;  input signed [15:0] z\_init;  input signed [15:0] x\_in;  input signed [15:0] y\_in;  input signed [15:0] z\_in;  input select;  //Inputs  output signed [15:0] x\_out;  output signed [15:0] y\_out;  output signed [15:0] z\_out;  // Wires  wire signed [15:0] x\_out;  wire signed [15:0] y\_out;  wire signed [15:0] z\_out;  // Select as output value from x\_init or x\_in  assign x\_out = selector\_v( x\_init, x\_in, select);  // Select as output value from y\_init or y\_in  assign y\_out = selector\_v( y\_init, y\_in, select);  // Select as output value from z\_init or z\_in  assign z\_out = selector\_v( z\_init, z\_in, select);  // Select as output value from \*\_init or \*\_in  function signed [15:0] selector\_v;  input signed [15:0] v\_init;  input signed [15:0] v\_in;  input select;    if (select) begin  selector\_v = v\_init;  end else begin  selector\_v = v\_in;  end  endfunction  endmodule |

* 1. **Cordic\_data**

|  |
| --- |
| module cordic\_data(  clock, // Input system clock  reset, // Input system reset  End, // Input system End  theta\_in, // Input value of theta  x\_init, // Initial value of x  y\_init, // Initial value of y  z\_init, // Initial value of z  select, // Input value of select  address, // Input value of address  x\_out, // Output value of x  y\_out, // Output value of y  z\_out // Output value of z  );  input signed [15:0] x\_init;  input signed [15:0] y\_init;  input signed [15:0] z\_init;  input signed [15:0] theta\_in;  input [3:0] address;  input clock;  input reset;  input select;  input End;  output signed [15:0] x\_out;  output signed [15:0] y\_out;  output signed [15:0] z\_out;  wire signed [15:0] X1;  wire signed [15:0] Y1;  wire signed [15:0] Z1;  // wire signed [15:0] DELTA1;  wire signed [15:0] ALPHA1;  // wire signed [15:0] DELTA2;  wire signed [15:0] ALPHA2;  wire signed [15:0] X\_FB;  wire signed [15:0] Y\_FB;  wire signed [15:0] Z\_FB;    wire signed [15:0] y\_delta;  wire signed [15:0] x\_delta;  assign x\_out = X\_FB;  assign y\_out = Y\_FB;  assign z\_out = Z\_FB;  codric\_init\_SLT u\_codric\_init\_SLT(  .select (select ), // Select as output value from X\_init or X\_in  .x\_init ( x\_init ), // Initial value of x  .y\_init ( y\_init ), // Initial value of y  .z\_init ( z\_init ), // Initial value of z  .x\_in ( X\_FB ), // Itarated value of x  .y\_in ( Y\_FB ), // Itarated value of y  .z\_in ( Z\_FB ), // Itarated value of z  .x\_out ( X1 ), // Selected value of x  .y\_out ( Y1 ), // Selected value of y  .z\_out ( Z1 ) // Selected value of z  );  codric\_rom u\_codric\_rom(  .clock ( clock ), // clock input  .address ( address ), // ROM address (0 to 15)  .alpha\_out ( ALPHA1 ) // Alpha value : arctan(2^(-i))  // .delta\_out ( DELTA1 )  // Delta value : 2^(-i)  );  cordic\_check u\_cordic\_check(  .alpha\_in ( ALPHA1 ), // Input value of delta  .theta\_in ( theta\_in ), // Input value of theta  .x\_in\_check ( X1 ), // Input value of x\_check  .y\_in\_check ( Y1 ), // Input value of y\_check  .address ( address ), // Input value of alpha  .z\_in ( Z1 ), // Input value of z  .alpha\_out ( ALPHA2 ), // Output value of alpha  .x\_delta\_out (x\_delta), // Output value of x\_delta  .y\_delta\_out ( y\_delta ) // Output value of x\_delta    );  cordic\_core u\_cordic\_core(  .clock ( clock ), // clock  .xreset ( reset ), // reset (low active)  .End ( End ), // Input value of End  .alpha\_in ( ALPHA2 ), // Input value of alpha  .x\_delta\_in ( x\_delta ), // Input value of X\_delta  .y\_delta\_in ( y\_delta ), // Input value of y\_delta  .x\_in ( X1 ), // Input value of x  .y\_in ( Y1 ), // Input value of y  .z\_in ( Z1 ), // Input value of z  .x\_out ( X\_FB ), // Output value of x  .y\_out ( Y\_FB ), // Output value of y  .z\_out ( Z\_FB ) // Output value of z  );  endmodule |

* 1. **Cordic\_core**

|  |
| --- |
| module cordic\_core (  clock, // clock  xreset, // reset (low active)  End, // Input value of End  alpha\_in, // Input value of alpha  x\_delta\_in, // Input value of delta  y\_delta\_in,  x\_in, // Input value of x  y\_in, // Input value of y  z\_in, // Input value of z  x\_out, // Output value of x  y\_out, // Output value of y  z\_out // Output value of z  );  //Inputs  input clock;  input xreset;  input End;  input signed [15:0] x\_in;  input signed [15:0] y\_in;  input signed [15:0] z\_in;  input signed [15:0] alpha\_in;  input signed [15:0] x\_delta\_in;  input signed [15:0] y\_delta\_in;  //Outputs  output signed [15:0] x\_out;  output signed [15:0] y\_out;  output signed [15:0] z\_out;  //Regs  reg signed [15:0] x\_out;  reg signed [15:0] y\_out;  reg signed [15:0] z\_out;    always @ (posedge clock or negedge xreset) begin  if (xreset == 0) begin  x\_out <= 0;  end  else begin  if(End) begin  x\_out <= x\_out;  end  else begin  x\_out <= x\_in - y\_delta\_in;  end  end  end  always @ (posedge clock or negedge xreset) begin  if (xreset == 0) begin  y\_out <= 0;  end  else begin  if(End) begin  y\_out <= y\_out;  end  else begin  y\_out <= y\_in + x\_delta\_in;  end  end  end  always @ (posedge clock or negedge xreset) begin  if (xreset == 0) begin  z\_out <= 0;  end  else begin  if(End) begin  z\_out <= z\_out;  end  else begin  z\_out <= z\_in + alpha\_in;  end  end  end  endmodule |

* 1. **Cordic\_check**

|  |
| --- |
| module cordic\_check (  alpha\_in, // Input value of alpha  theta\_in, // Input value of theta  x\_in\_check,  y\_in\_check,  address,  z\_in, // Input value of z  alpha\_out, // Output value of alpha  x\_delta\_out, // Output value of delta  y\_delta\_out  );  //Inputs  // input signed [15:0] delta\_in;  input signed [15:0] alpha\_in;  input signed [15:0] theta\_in;  input signed [15:0] z\_in;  input signed [15:0] x\_in\_check;  input signed [15:0] y\_in\_check;  input [3:0] address;  //Outputs  output signed [15:0] x\_delta\_out;  output signed [15:0] y\_delta\_out;  output signed [15:0] alpha\_out;      //Check z\_in and decide to whether invert alpha or not  assign alpha\_out = alpha\_check( alpha\_in, theta\_in, z\_in);    //Check z\_in and decide to whether invert delta or not  assign x\_delta\_out = x\_delta\_check( x\_in\_check, theta\_in, z\_in);    //Check z\_in and decide to whether invert delta or not  assign y\_delta\_out = y\_delta\_check( y\_in\_check, theta\_in, z\_in);    //wether input z\_in is larger than theta or not  function signed [15:0] alpha\_check;  input signed [15:0] alpha\_in;  input signed [15:0] theta\_in;  input signed [15:0] z\_in;    if(z\_in > theta\_in) begin  alpha\_check = -alpha\_in;  end  else begin  alpha\_check = alpha\_in;  end  endfunction  //whether input z\_in is larger than theta or not  function signed [15:0] x\_delta\_check;  input signed [15:0] x\_in\_check;  input signed [15:0] theta\_in;  input signed [15:0] z\_in;    if(z\_in > theta\_in) begin    x\_delta\_check = -(x\_in\_check >> address)-1;  end  else begin  x\_delta\_check = x\_in\_check >> address;  end  endfunction    //whether input z\_in is larger than theta or not  function signed [15:0] y\_delta\_check;  input signed [15:0] y\_in\_check;  input signed [15:0] theta\_in;  input signed [15:0] z\_in;  if(z\_in > theta\_in) begin    y\_delta\_check = -(y\_in\_check >> address)-1;  end  else begin  y\_delta\_check = y\_in\_check >> address;  end  endfunction  endmodule |

* 1. **Cordic**

|  |
| --- |
| module cordic(  clock, // Input system clock  reset, // Input system reset  start, // Input system start  theta\_in, // Input value of theta  x\_init, // Initial value of x  y\_init, // Initial value of y  z\_init, // Initial value of z  x\_out, // Output value of x  y\_out, // Output value of y  z\_out, // Output value of z  End // Output value of End  );  input signed [15:0] x\_init;  input signed [15:0] y\_init;  input signed [15:0] z\_init;  input signed [15:0] theta\_in;  input clock;  input reset;  input start;  output signed [15:0] x\_out;  output signed [15:0] y\_out;  output signed [15:0] z\_out;  output End;  wire signed [15:0] theta\_out;  wire [3:0] ADDRESS;  wire SELECT;    // PI = 3.141592654 = 2 + 1.141592654(01.00100100001111)  assign theta\_out = (theta\_in <<1)+theta\_in+(theta\_in >>3)+(theta\_in >>6)+(theta\_in >>11)+(theta\_in >>12)+(theta\_in>>13)+(theta\_in >>14);  controler u\_controler(    .reset (reset), // Input system clock  .clock (clock), // Input system reset  .start ( start ), // Input value of start  .count ( ADDRESS ), // Output value of CNT  .select ( SELECT ), // Output value of select  .End ( End ) // Output value of END    );  cordic\_data u\_cordic\_data(  .clock ( clock ), // Input system clock  .reset ( reset ), // Input system reset  .End ( End ), // Input system End  .theta\_in ( theta\_out ), // Input value of theta  .x\_init ( x\_init ), // Initial value of x  .y\_init ( y\_init), // Initial value of y  .z\_init ( z\_init), // Initial value of z  .select ( SELECT ), // Input value of select  .address ( ADDRESS ), //Input value of address  .x\_out ( x\_out ), // Output value of x  .y\_out ( y\_out ), // Output value of y  .z\_out (z\_out ) // Output value of z  );  endmodule |

* 1. **Cordic\_sin\_cos**

|  |
| --- |
| module sin\_cos\_cordic(  clock, // Input system clock  reset, // Input system reset  start, // Input system start  theta\_in, // Input value of theta  cos\_out, // Output value of x  sin\_out, // Output value of y  End // Output value of End  );  input clock;  input reset;  input start;  input signed [15:0] theta\_in;  output signed [15:0] cos\_out;  output signed [15:0] sin\_out;  output End;  wire signed [15:0] cos\_out;  wire signed [15:0] sin\_out;  wire End;  cordic u\_cordic (  .clock (clock ), // Input system clock  .reset (reset ), // Input system reset  .start (start ), // Input system start  .theta\_in ( theta\_in ), // Input value of theta  .x\_init ( 16'b0010011011011101 ), // Initial value of x  .y\_init ( 16'b0000000000000000 ), // Initial value of y  .z\_init ( 16'b0000000000000000 ), // Initial value of z  .x\_out ( cos\_out ), // Output value of x  .y\_out ( sin\_out ), // Output value of y  .z\_out ( ), // Output value of z  .End ( End ) // Output value of End  );  endmodule |

* 1. **Controller**

|  |
| --- |
| module controler(  reset, // Input system clock  clock, // Input system reset  start, // Input value of start  count, // Output value of CNT  select, // Output value of select  End // Output value of END    );  //Input  input reset;  input clock;  input start;  //Outputs  output [3:0] count;  output select;  output End;  //Reg  reg [3:0] count;  reg count\_en;  reg End;    //Wire  wire select;  assign select = start;  always @(posedge clock or negedge reset)  begin  if(reset == 1'b0) begin  count\_en <= 0;  end  else begin  if(count == 13) begin  count\_en <= 0;  end  else if(start) begin  count\_en <= 1;  end  else begin  count\_en <= count\_en;  end  end  end  always @(posedge clock or negedge reset)  begin  if(reset == 1'b0) begin  count <= 0;  end  else begin  if(count == 13) begin  count <= 0;  end  else if (count\_en) begin  count <= count + 1;  end  else if(start) begin  count <= 1;  end  else begin  count <= 0;  end  end  end  always @(posedge clock or negedge reset)  begin  if(reset == 1'b0) begin  End <= 0;    end  else begin  if(count == 13) begin  End <= 1;  end  else begin  End <= 0;  end  end  end  endmodule |

* 1. **Cordic\_rom**

|  |
| --- |
| //`defineDvLen 16  `define CORDIC\_ROM\_DATA\_BW 16  module codric\_rom (  clock, // clock input  address, // ROM address (0 to 15  alpha\_out // Alpha value : arctan(2^(-i))  );  // Inputs  input clock;  input [3:0] address;  // Outputs  output [`CORDIC\_ROM\_DATA\_BW-1:0] alpha\_out;  // Regs  reg [`CORDIC\_ROM\_DATA\_BW-1:0] alpha\_out;  // ROM for alpha : arctan(2^(-i))  always @ ( negedge clock)  begin  case (address)  0:alpha\_out = 16'b0011001001000011 ;// 45 do  1:alpha\_out = 16'b0001110110101100 ;  2:alpha\_out = 16'b0000111110101101 ;  3:alpha\_out = 16'b0000011111110101 ;  4:alpha\_out = 16'b0000001111111110 ;  5:alpha\_out = 16'b0000000111111111 ;  6:alpha\_out = 16'b0000000011111111 ;  7:alpha\_out = 16'b0000000001111111 ;  8:alpha\_out = 16'b0000000000111111 ;  9:alpha\_out = 16'b0000000000011111 ;  10:alpha\_out = 16'b0000000000001111 ;  11:alpha\_out = 16'b0000000000000111 ;  12:alpha\_out = 16'b0000000000000011 ;  13:alpha\_out = 16'b0000000000000001 ;  14:alpha\_out = 16'b0000000000000000 ;  15:alpha\_out = 16'b0000000000000000 ;    endcase  end  endmodule |

* 1. **Sin\_cos\_cordic\_tb**

|  |
| --- |
| `timescale 1ns / 1ns  module cordic\_tb;  //Inputs  reg clock;  reg reset;  reg start;  reg signed [15:0] theta\_in;  //Outputs  wire signed [15:0] x\_out;  wire signed [15:0] y\_out;  wire End;  parameter STEP = 1000;  sin\_cos\_cordic u\_sin\_cos\_cordic(  clock, // Input system clock  reset, // Input system reset  start, // Input system start  theta\_in, // Input value of theta  x\_out, // Output value of x  y\_out, // Output value of y  End  );  always #(STEP/2) clock = ~clock;  initial begin  //initialize  #0 reset = 0;  clock = 0;  start = 0;  theta\_in = 0;  #(STEP) //theta\_in is PI/4 value  // theta\_in = 16'b0001000000000000;  //theta\_in is PI1/8 value  theta\_in = 16'b0000100000000000;  //theta\_in is PI2/8 value  //theta\_in = 16'b0001000000000000;  //theta\_in is PI3/8 value  //theta\_in = 16'b0001100000000000;  //theta\_in is PI4/8 value  //theta\_in = 16'b0010000000000000;  //theta\_in is -PI1/8 value  //theta\_in = 16'b1111100000000000;  //theta\_in is -PI2/8 value  //theta\_in = 16'b1111000000000000;  //theta\_in is -PI3/8 value  //theta\_in = 16'b1110100000000000;  //theta\_in is -PI4/8 value  //theta\_in = 16'b1110000000000000;  //theta\_in is 0 value  //theta\_in = 16'b0000000000000000;  //theta\_in is 1/128 value  //theta\_in = 16'b0000000010000000;  //theta\_in is 1/256 value  //theta\_in = 16'b0000000001000000;  //theta\_in is -1/128 value  //theta\_in = 16'b1111111110000000;  //theta\_in is -1/256 value  //theta\_in = 16'b1111111111000000;  //theta\_in is PI4/8 - 1/128 value  //theta\_in = 16'b0001111111010111;  //theta\_in is PI4/8 - 1/256 value  //theta\_in = 16'b0001111111101011;  //theta\_in is -PI4/8 + 1/128 value  //theta\_in = 16'b1110000000101001;  //theta\_in is -PI4/8 + 1/256 value  //theta\_in = 16'b1110000000010101;      reset = 1;    #(STEP) start = 1;  #(STEP) start = 0;  #(STEP\*14)  $finish;  end  initial  $monitor ( $stime,"theta\_in = %b x\_out = %b y\_out = %b End = %b",theta\_in, x\_out, y\_out, End);    endmodule |

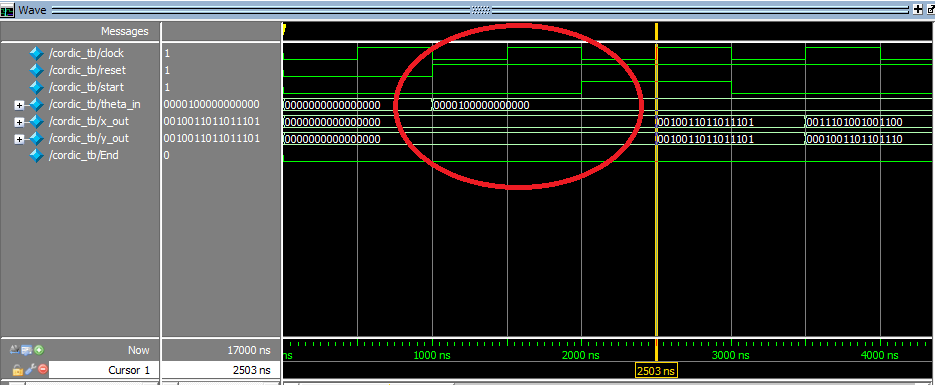
1. **Operation**

When *reset* is high *theta\_in* input is sampled by testbench and x\_Cordic processing is started as soon as meet a rising clock and *start* is high. Form starting, each loop take 1 clock cycle.

Regarding output CORDIC handling, there is *End* pin – when high means that the processing stop and give the final output.

Below is example of computing trigonometric function by x\_Cordic

Example: For angle = PI/8 we meet the output specification.



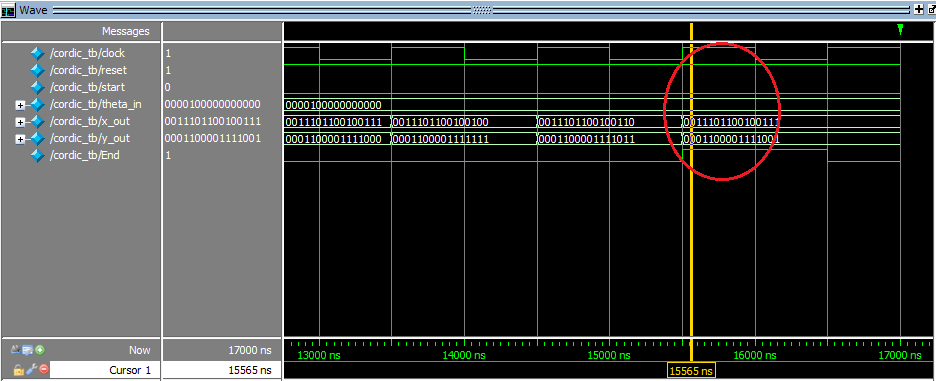
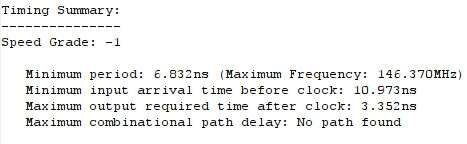
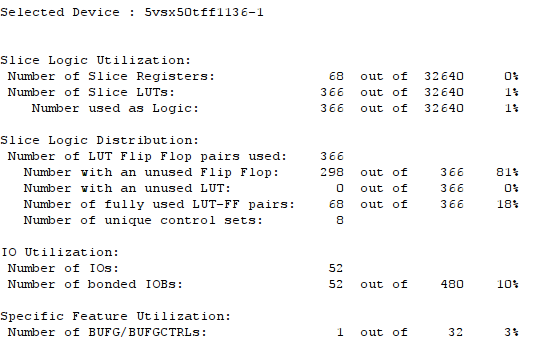


Figure 4. Cordic operation

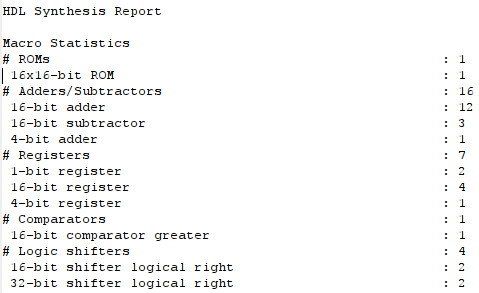
1. **Synthesize**
   1. **Timming**



* 1. **Source**



* 1. **Component**



|  |  |  |
| --- | --- | --- |
| **Design** | **Xilinx Virtex-5 (XC5VLX50-1ff324)** | |
| **NOS** | **MUF** |
| **Conventional CORDIC** | **178** | **109,005 MHz** |
| **Our design** | **361** | **146,370 MHz** |

**NOS stands for the number of slices. MUF stands for maximum operationg frequency in MHz**

**Level 2: For experts Trigonometric function output using a variety of algorithms.**

1. **Introduction**

Pramod Kumar Meher and Sang Yoon Park [1] said that it is observed that the hardware-complexity of barrel-shifters alone is nearly half of that of a CORDIC circuit. Therefore they suggested some techniques to minimize the complexity of barrel shifters. And, they have proved by experiment that a set of four selected micro-rotations is enough to achieve with maximum angular deviation . Terence K. Rodrigues and Earl E. Swartzlander. Jr [2] have proposed a simple implementation of the angle selection scheme that allows the Angle Recoding method to be used dynamically for any arbitrary angle of rotation. This methosd determind the boundaries of the contiguous ranges of to predict the ranges of (= by define as follows:

So if |Z| then |Z| is closer to and if |Z| <, then |Z| is closer to . This given the more than 50% reduction in the number of interations, but this is to define all the possible rangles for a given angle constant, we must find all the combinations of angle constants that include it.

1. **Architecture**

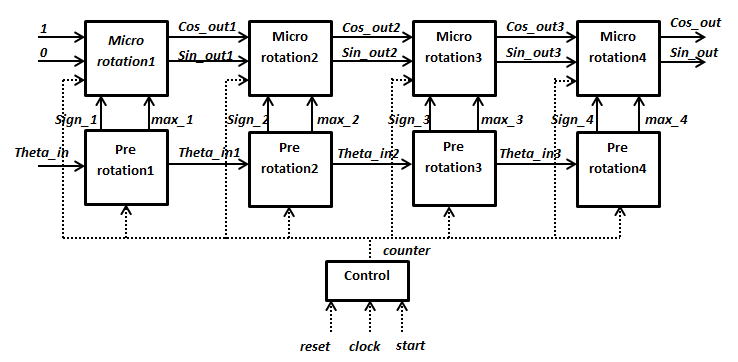


Figure 5. Architecture

As show in figure 4, our architecture has 9 main blocks: 4 pre\_rotation blocks, 4 micro\_rotation blocks and control block. In here, control block is aim to generated control signal. The pre\_rotation block is to find the direction and the magnitude of rotated angle in the micro rotation. Finally, the micro\_rotation block performs the rotation with the found rotated angle.

Next, the detail of above blocks will be clarified in the next section.

* 1. **Pre\_rotation block.**

The following figure will illustrated the process in pre\_rotation block more clearly:

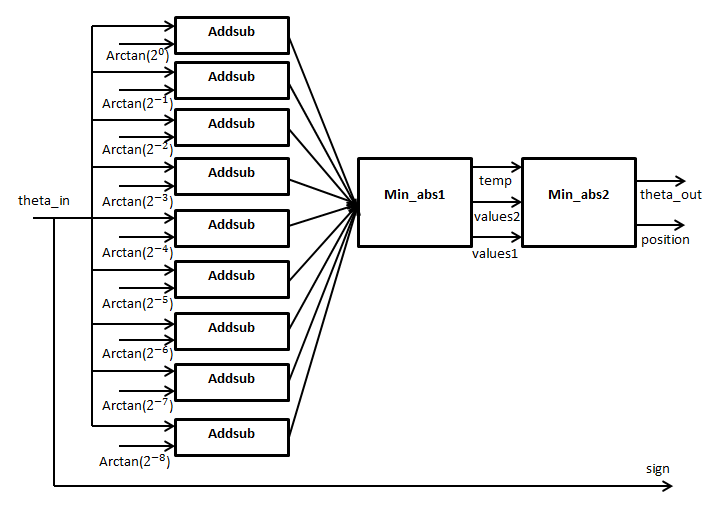


Figure 6. Pre\_rotation

In our architecture, depending on the sign of the theta\_in in each micro rotation, we will perform 9 addition or subtraction between this angle with 9 angles which have values of arctan() with i in range from 0 to 8, and we denote 9 results by angle\_temp(i). After that, they were used in min\_abs1 block to find a set containing two values and have the minimum absolute values. Because the values of arctan() is decrease when i increase, 9 values of angle\_temp(i) is monotonic. Therefore, if all values of angle\_temp(i) is in the same direction, we will choose the initial and final value to return to min\_abs2 block. If not, we will return two consecutive results which have opposite sign. Because suppose that we find angle\_temp(k) and angle\_temp(k+1) which has different sign, there are two cases:

If then angle\_temp(k) has minimum absolute in the set of negative values and angle\_temp(k+1) has minimum absolute in the set of positive values.

If then angle\_temp(k) has minimum absolute in the set of negative values and angle\_temp(k+1) has minimum absolute in the set of positive values.

In both case, output of min\_abs1 block always has the values which has minimum absolute.

The last block in pre\_rotation is min\_abs2. It perform compare the absolute of two values in output of min\_abs1 block and return the position of angle we choose to rotate to support for micro\_rotation block.

In addion, the sign of theta\_in is stored to determine the direction of rotation in micro\_rotation block.

* 1. **Micro\_rotation block.**

This block is illustrated by the following figure:

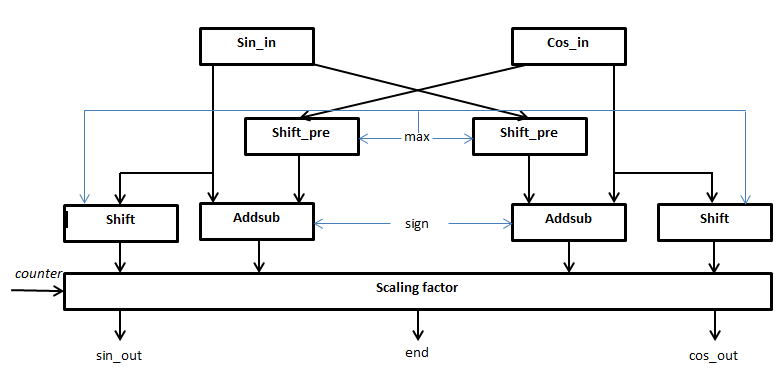


Figure 7. Micro\_rotation block

This block contain three main processes: pre\_shift, addsub and multiplied with scaling factor. Firstly, sin\_in and cos\_in values is pre\_shift max bit by load (16-max) bit less significant bit from the value of sin\_in and cos\_in, other bits is equal the sign bit of sin\_in and cos\_in. After that, we will perform addition or subtraction depending on the value of sign. Finally, The results after addsub is multiplied with scaling factor.

The scaling factor is

To gain the accuracy of 8 bit, we will use with. In the cases we will use the true values which were stored in ROM.

* 1. **Control block**

The control block is simplied a generation of counter signal to count the number of clock cycles and controls blocks to perform in time. In addition, counter signal also detect the point of time the system is finished and end signal is equal 1.

1. **An appeal point and originality**

In our architecture, by find the rotated angle that nearest input theta\_in in each micro rotation, we have already reduced the number of iteration from 9 to 4 iterations, so we could reduced delay of the system. For the reason that each iteration consumes only one clock cycles, our system consumes only 4 clock cycles to gain the results. When compare with 14 clock cycles consuming in the architecture in level 1, It is can be easy to seen that our architecture can gain lower delay than that one in level 1.

orther appeal point in our architecture is the way to find the element that has smallest absolute from a set of 9 elements by searching two consecutive results which have opposite sign. If we can not find, we can return the set containing the initial and final element. This step helps us to reduce the complexity of comparators from 16 bit to only 1 bit. Therefore, it reduces delay to find the element which has the smallest absolute.

1. **Critical path speed a circuit domain**

Timing constraint: Default period analysis for Clock 'clock'

Clock period: 7.427ns (frequency: 134.644MHz)

Total number of paths / destination ports: 8724094 / 256

-------------------------------------------------------------------------

Delay: 7.427ns (Levels of Logic = 38)

Source: pre\_rotation2/min\_abs\_step1/values\_out1\_0 (FF)

Destination: pre\_rotation3/min\_abs\_step1/values\_out2\_14 (FF)

Source Clock: clock rising

Destination Clock: clock rising

Data Path: pre\_rotation2/min\_abs\_step1/values\_out1\_0 to pre\_rotation3/min\_abs\_step1/values\_out2\_14

1. **HDL code (VHDL)**
   1. **Addsub**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity addsub is  port(  sign: in std\_logic;  theta\_in: in std\_logic\_vector(15 downto 0);  angle\_in: in std\_logic\_vector(15 downto 0);  angle\_temp: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture addsub of addsub is  begin  angle\_temp <= theta\_in-angle\_in when (sign='0')  else  theta\_in+angle\_in;  end architecture;  -------------------------------------------------------- |

* 1. **Shift**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_arith.all;  entity shift is  port(  inbus: in std\_logic\_vector(15 downto 0);  round: in std\_logic\_vector(3 downto 0);  outbus: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture shift of shift is  begin  outbus <= inbus when ( round="0000")  else inbus(15)&inbus(15)&inbus(14 downto 1) when (round="0001")  else inbus(15)&inbus(15)&inbus(15)&inbus(14 downto 2) when (round="0010")  else inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(14 downto 3) when (round="0011")  else inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(14 downto 4) when (round="0100")  else inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(14 downto 5) when (round="0101")  else inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(14 downto 6) when (round="0110")  else inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(14 downto 7) when (round="0111")  else inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(15)&inbus(14 downto 8) when (round="1000")  else (others=>'0');  end architecture; |

* 1. **Min\_abs1**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity min\_abs1 is  port(  counter: in std\_logic\_vector(3 downto 0);  clock: in std\_logic;  angle0: in std\_logic\_vector(15 downto 0);  angle1: in std\_logic\_vector(15 downto 0);  angle2: in std\_logic\_vector(15 downto 0);  angle3: in std\_logic\_vector(15 downto 0);  angle4: in std\_logic\_vector(15 downto 0);  angle5: in std\_logic\_vector(15 downto 0);  angle6: in std\_logic\_vector(15 downto 0);  angle7: in std\_logic\_vector(15 downto 0);  angle8: in std\_logic\_vector(15 downto 0);  new\_max: out std\_logic\_vector(3 downto 0);  values\_out1: out std\_logic\_vector(15 downto 0);  values\_out2: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture min\_abs1 of min\_abs1 is  begin  process(angle0,clock)  begin  if (rising\_edge(clock)) then  if (counter/="0000") then  if angle0(15)/=angle4(15) then  if angle0(15)/=angle2(15) then  if angle0(15)/=angle1(15) then  new\_max <="0000";  values\_out1 <= angle0;  values\_out2 <= angle1;  else  new\_max <="0001";  values\_out1 <= angle1;  values\_out2 <= angle2;  end if;  else  if(angle2(15)/=angle3(15)) then  new\_max <= "0010";  values\_out1 <= angle2;  values\_out2 <= angle3;  else  new\_max <= "0011";  values\_out1 <= angle3;  values\_out2 <= angle4;  end if;  end if;  elsif( angle4(15)/=angle8(15)) then  if angle4(15)/=angle6(15) then  if angle4(15)/=angle5(15) then  new\_max <= "0100";  values\_out1 <= angle4;  values\_out2 <= angle5;  else  new\_max <= "0101";  values\_out1 <= angle5;  values\_out2 <= angle6;  end if;  else  if (angle6(15)/=angle7(15)) then  new\_max <="0110";  values\_out1 <= angle6;  values\_out2 <= angle7;  else  new\_max <="0111";  values\_out1 <= angle7;  values\_out2 <= angle8;  end if;  end if;  else  new\_max <= "1000";  values\_out1 <=angle0;  values\_out2 <= angle8;  end if;  end if;  end if;  end process;  end architecture; |

* 1. **Min\_abs2**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity min\_abs2 is  port(  values\_out1: in std\_logic\_vector(15 downto 0);  values\_out2: in std\_logic\_vector(15 downto 0);  new\_max: in std\_logic\_vector(3 downto 0);  max: out std\_logic\_vector(3 downto 0);  values\_out: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture min\_abs2 of min\_abs2 is  signal sum, sub: std\_logic\_vector(15 downto 0);  component sum\_sub is  port(  values\_out1: in std\_logic\_vector(15 downto 0);  values\_out2: in std\_logic\_vector(15 downto 0);  sum: out std\_logic\_vector(15 downto 0);  sub: out std\_logic\_vector(15 downto 0)  );  end component;begin  sum\_sub1: sum\_sub port map(  values\_out1 => values\_out1,  values\_out2 => values\_out2,  sum => sum,  sub => sub);  values\_out <= values\_out1 when ((new\_max="1000" and sub(15)/=values\_out1(15)) or (new\_max/="1000" and sum(15)/=values\_out1(15)))  else values\_out2;  max <= "1000" when ((new\_max="1000") and (sub(15)=values\_out1(15)))  else "0000" when ((new\_max="1000") and (sub(15)/=values\_out1(15)))  else new\_max+"0001" when ((new\_max/="1000" and sum(15)=values\_out1(15)))  else new\_max;  end architecture; |

* 1. **Cordic**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity cordic is  port(  sin\_in: std\_logic\_vector(15 downto 0);  cos\_in: std\_logic\_vector(15 downto 0);  max: in std\_logic\_vector(3 downto 0);  sign: in std\_logic;  cos\_out: out std\_logic\_vector(15 downto 0);  sin\_out: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture cordic of cordic is  component shift is  port(  inbus: in std\_logic\_vector(15 downto 0);  round: in std\_logic\_vector(3 downto 0);  outbus: out std\_logic\_vector(15 downto 0)  );  end component;  component addsub is  port(  sign: in std\_logic;  theta\_in: in std\_logic\_vector(15 downto 0);  angle\_in: in std\_logic\_vector(15 downto 0);  angle\_temp: out std\_logic\_vector(15 downto 0)  );  end component;  component scaling\_factor is  port(  max: in std\_logic\_vector(3 downto 0);  sin\_in: in std\_logic\_vector(15 downto 0);  cos\_in: in std\_logic\_vector(15 downto 0);  sin\_out: out std\_logic\_vector(15 downto 0);  cos\_out: out std\_logic\_vector(15 downto 0)  );  end component;  begin  not\_sign <= not sign;  shift1: shift port map (  inbus => sin\_in,  round => max,  outbus => sin\_shift  );  shift2: shift port map (  inbus => cos\_in,  round => max,  outbus => cos\_shift  );  addsub1: addsub port map (  sign => not\_sign,  theta\_in => cos\_in,  angle\_in => sin\_shift,  angle\_temp => t\_cos  );  addsub2: addsub port map (  sign => sign,  theta\_in => sin\_in,  angle\_in => cos\_shift,  angle\_temp => t\_sin  );  scaling\_factor1: scaling\_factor port map(  max => max,  sin\_in => t\_sin,  cos\_in => t\_cos,  cos\_out => cos\_out,  sin\_out => sin\_out  );  end architecture; |

* 1. **K1**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity k1 is  port(  sin\_in: in std\_logic\_vector(14 downto 0);  cos\_in: in std\_logic\_vector(14 downto 0);  sin\_temp1: out std\_logic\_vector(15 downto 0);  cos\_temp1: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture k1 of k1 is  signal sin\_temp: std\_logic\_vector(15 downto 0);  signal cos\_temp: std\_logic\_vector(15 downto 0);  begin    sin\_temp <= (sin\_in(14)&sin\_in)  +(sin\_in(14)&sin\_in(14)&sin\_in(14)&sin\_in(14)&sin\_in(14)&sin\_in(14)&sin\_in(14 downto 5));  cos\_temp <= (cos\_in(14)&cos\_in)  + (cos\_in(14)&cos\_in(14)&cos\_in(14)&cos\_in(14)&cos\_in(14)&cos\_in(14)&cos\_in(14 downto 5));  sin\_temp1 <= sin\_temp+(sin\_temp(15)&sin\_temp(15)&sin\_temp(15 downto 2));  cos\_temp1 <= cos\_temp+(cos\_temp(15)&cos\_temp(15)&cos\_temp(15 downto 2));  end architecture; |

* 1. **Sum\_sub**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity sum\_sub is  port(  values\_out1: in std\_logic\_vector(15 downto 0);  values\_out2: in std\_logic\_vector(15 downto 0);  sum: out std\_logic\_vector(15 downto 0);  sub: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture sum\_sub of sum\_sub is  begin  sum <= values\_out1+values\_out2;  sub <= values\_out1-values\_out2;  end architecture; |

* 1. **Scaling\_factor**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity scaling\_factor is  port(  max: in std\_logic\_vector(3 downto 0);  sin\_in: in std\_logic\_vector(15 downto 0);  cos\_in: in std\_logic\_vector(15 downto 0);  sin\_out: out std\_logic\_vector(15 downto 0);  cos\_out: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture scaling\_factor of scaling\_factor is  component k1 is  port(  sin\_in: in std\_logic\_vector(14 downto 0);  cos\_in: in std\_logic\_vector(14 downto 0);  sin\_temp1: out std\_logic\_vector(15 downto 0);  cos\_temp1: out std\_logic\_vector(15 downto 0)  );  end component;  component shift is  port(  inbus: in std\_logic\_vector(15 downto 0);  round: in std\_logic\_vector(3 downto 0);  outbus: out std\_logic\_vector(15 downto 0)  );  end component;  signal sin\_temp1,cos\_temp1: std\_logic\_vector(15 downto 0);  signal t\_sin\_in,t\_cos\_in,t\_sin\_in1,t\_cos\_in1,t\_sin\_in2,t\_cos\_in2: std\_logic\_vector(15 downto 0);  begin  scaling\_shift1: shift port map(  inbus => cos\_in,  round => max,  outbus => t\_cos\_in1  );  scaling\_shift2: shift port map(  inbus => sin\_in,  round => max,  outbus => t\_sin\_in1  );  scaling\_shift3: shift port map(  inbus => t\_cos\_in1,  round => max,  outbus => t\_cos\_in2  );  scaling\_shift4: shift port map(  inbus => t\_sin\_in1,  round => max,  outbus => t\_sin\_in2  );  scaling\_shift5: shift port map(  inbus => t\_cos\_in2,  round => "0001",  outbus => t\_cos\_in  );  scaling\_shift6: shift port map(  inbus => t\_sin\_in2,  round => "0001",  outbus => t\_sin\_in  );  k11: k1 port map(  sin\_in => sin\_in(15 downto 1),  cos\_in => cos\_in(15 downto 1),  sin\_temp1 => sin\_temp1,  cos\_temp1 => cos\_temp1  );  sin\_out <= sin\_temp1+(sin\_in(15)&sin\_in(15)&sin\_in(15)&sin\_in(15)&sin\_in(15 downto 4)) when (max="0000")  else sin\_temp1+(sin\_in(15)&sin\_in(15)&sin\_in(15 downto 2)) when (max="0001")  else sin\_in when (max="1000")  else sin\_in-t\_sin\_in;    cos\_out <= cos\_temp1+(cos\_in(15)&cos\_in(15)&cos\_in(15)&cos\_in(15)&cos\_in(15 downto 4)) when (max="0000")  else cos\_temp1+(cos\_in(15)&cos\_in(15)&cos\_in(15 downto 2)) when (max="0001")  else cos\_in when (max="1000")  else cos\_in-t\_cos\_in;  end architecture; |

* 1. **Pre\_rotation**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_arith.all;  entity pre\_rotation is  port(  counter: in std\_logic\_vector(3 downto 0);  clock: in std\_logic;  theta\_in: in std\_logic\_vector(15 downto 0);  sign: out std\_logic;  max: out std\_logic\_vector(3 downto 0);  values\_out: out std\_logic\_vector(15 downto 0)  );  end entity;  architecture pre\_rotation of pre\_rotation is  signal angle\_temp0,angle\_temp1,angle\_temp2,angle\_temp3,angle\_temp4,angle\_temp5,angle\_temp6,angle\_temp7,angle\_temp8: std\_logic\_vector(15 downto 0);--9 gia tri angle sau khi add/sub  signal new\_max: std\_logic\_vector(3 downto 0);--gia tri nhan duoc sau buoc 1 khoi min\_abs  signal t\_sign: std\_logic;  signal values\_out1,values\_out2: std\_logic\_vector(15 downto 0);  component addsub is  port(  sign: in std\_logic;  theta\_in: in std\_logic\_vector(15 downto 0);  angle\_in: in std\_logic\_vector(15 downto 0);  angle\_temp: out std\_logic\_vector(15 downto 0)  );  end component;  component min\_abs1 is  port(  counter: in std\_logic\_vector(3 downto 0);  clock: in std\_logic;  angle0: in std\_logic\_vector(15 downto 0);  angle1: in std\_logic\_vector(15 downto 0);  angle2: in std\_logic\_vector(15 downto 0);  angle3: in std\_logic\_vector(15 downto 0);  angle4: in std\_logic\_vector(15 downto 0);  angle5: in std\_logic\_vector(15 downto 0);  angle6: in std\_logic\_vector(15 downto 0);  angle7: in std\_logic\_vector(15 downto 0);  angle8: in std\_logic\_vector(15 downto 0);  new\_max: out std\_logic\_vector(3 downto 0);  values\_out1: out std\_logic\_vector(15 downto 0):=(others=>'0');  values\_out2: out std\_logic\_vector(15 downto 0):=(others=>'0')  );  end component;  component min\_abs2 is  port(  values\_out1: in std\_logic\_vector(15 downto 0);  values\_out2: in std\_logic\_vector(15 downto 0);  new\_max: in std\_logic\_vector(3 downto 0);  max: out std\_logic\_vector(3 downto 0);  values\_out: out std\_logic\_vector(15 downto 0)  );  end component;  begin  t\_sign <= theta\_in(15);  addsub0: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0000000000010100",  angle\_temp => angle\_temp8  );  addsub1: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0000000000101000",  angle\_temp => angle\_temp7  );  addsub2: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0000000001010001",  angle\_temp => angle\_temp6  );  addsub3: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0000000010100100",  angle\_temp => angle\_temp5  );  addsub4: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0000000101000101",  angle\_temp => angle\_temp4  );  addsub5: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0000001010001000",  angle\_temp => angle\_temp3  );  addsub6: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0000010011111101",  angle\_temp => angle\_temp2  );  addsub7: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0000100101110010",  angle\_temp => angle\_temp1  );  addsub8: addsub port map(  sign => t\_sign,  theta\_in => theta\_in,  angle\_in => "0001000000000000",  angle\_temp => angle\_temp0  );  min\_abs\_step1: min\_abs1 port map(  counter => counter,  clock => clock,  angle0 => angle\_temp0,  angle1 => angle\_temp1,  angle2 => angle\_temp2,  angle3 => angle\_temp3,  angle4 => angle\_temp4,  angle5 => angle\_temp5,  angle6 => angle\_temp6,  angle7 => angle\_temp7,  angle8 => angle\_temp8,  new\_max => new\_max,  values\_out1 => values\_out1,  values\_out2 => values\_out2  );  min\_abs\_step2: min\_abs2 port map(  values\_out1 => values\_out1,  values\_out2 => values\_out2,  new\_max => new\_max,  max => max,  values\_out => values\_out  );  sign <= not theta\_in(15);  end architecture; |

* 1. **system**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_arith.all;  entity system is  port(  reset: in std\_logic;  start: in std\_logic;  clock: in std\_logic;  theta\_in: in std\_logic\_vector(15 downto 0);  cos\_out: out std\_logic\_vector(15 downto 0);  sin\_out: out std\_logic\_vector(15 downto 0);  t\_end: out std\_logic:='0'  );  end entity;  architecture system of system is  signal t\_end1: std\_logic:='0';  signal t\_values\_out1,t\_values\_out2,t\_values\_out3,t\_values\_out4: std\_logic\_vector(15 downto 0);  signal sign1,sign2,sign3,sign4: std\_logic;  signal max1,max2,max3,max4: std\_logic\_vector(3 downto 0);  signal sin\_out1,sin\_out2,sin\_out3,sin\_out4: std\_logic\_vector(15 downto 0  signal cos\_out1,cos\_out2,cos\_out3,cos\_out4: std\_logic\_vector(15 downto 0);  signal counter: std\_logic\_vector(3 downto 0);  component control is  port(  reset: in std\_logic;  clock: in std\_logic;  start: in std\_logic;  counter: out std\_logic\_vector(3 downto 0)  );  end component;  component pre\_rotation is  port(  counter: in std\_logic\_vector(3 downto 0);  clock: in std\_logic;  theta\_in: in std\_logic\_vector(15 downto 0);  sign: out std\_logic;  max: out std\_logic\_vector(3 downto 0);  values\_out: out std\_logic\_vector(15 downto 0)  );  end component;  component cordic is  port(  sin\_in: std\_logic\_vector(15 downto 0);  cos\_in: std\_logic\_vector(15 downto 0);  max: in std\_logic\_vector(3 downto 0);  sign: in std\_logic;  cos\_out: out std\_logic\_vector(15 downto 0);  sin\_out: out std\_logic\_vector(15 downto 0)  );  end component;  begin  control1: control port map(  reset => reset,  clock => clock,  start => start,  counter => counter  );  pre\_rotation1: pre\_rotation port map(  counter => counter,  clock => clock,  theta\_in => theta\_in,  sign => sign1,  max => max1,  values\_out => t\_values\_out1  );  rotation1: cordic port map(  sin\_in => x"0000",  cos\_in => x"4000",  max => max1,  sign => sign1,  cos\_out => cos\_out1,  sin\_out => sin\_out1  );  pre\_rotation2: pre\_rotation port map(  counter => counter,  clock => clock,  theta\_in => t\_values\_out1,  sign => sign2,  max => max2,  values\_out => t\_values\_out2  );  rotation2: cordic port map(  sin\_in => sin\_out1,  cos\_in => cos\_out1,  max => max2,  sign => sign2,  cos\_out => cos\_out2,  sin\_out => sin\_out2  );  pre\_rotation3: pre\_rotation port map(  counter => counter,  clock => clock,  theta\_in => t\_values\_out2,  sign => sign3,  max => max3,  values\_out => t\_values\_out3  );  rotation3: cordic port map(  sin\_in => sin\_out2,  cos\_in => cos\_out2,  max => max3,  sign => sign3,  cos\_out => cos\_out3,  sin\_out => sin\_out3  );  pre\_rotation4: pre\_rotation port map(  counter => counter,  clock => clock,  theta\_in => t\_values\_out3,  sign => sign4,  max => max4,  values\_out => t\_values\_out4  );  rotation4: cordic port map(  sin\_in => sin\_out3,  cos\_in => cos\_out3,  max => max4,  sign => sign4,  cos\_out => cos\_out4,  sin\_out => sin\_out4  );  t\_end1 <= '1' when (counter>"0100")  else '0';  t\_end <= t\_end1;  cos\_out <= cos\_out4 when (t\_end1='1')  else (others =>'0');  sin\_out <= sin\_out4 when (t\_end1 ='1')  else (others=>'0');    end architecture; |

1. **The display of a simulation waveform which the design is operating**

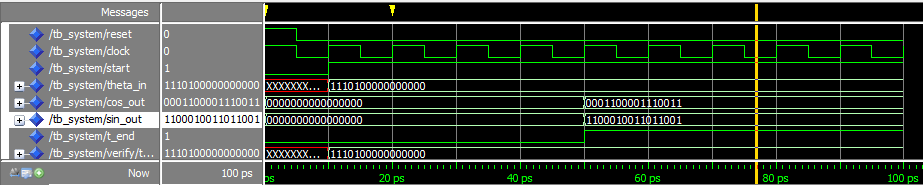


Figure 8. Simulation waveform

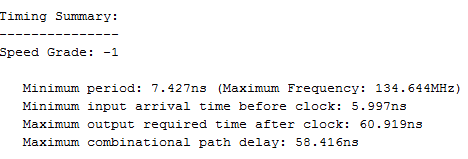
With theta\_in=”1110010000000000” then sin\_out = “1100010011011001” and cos\_out=”0001100001110011”.

We have the result table:

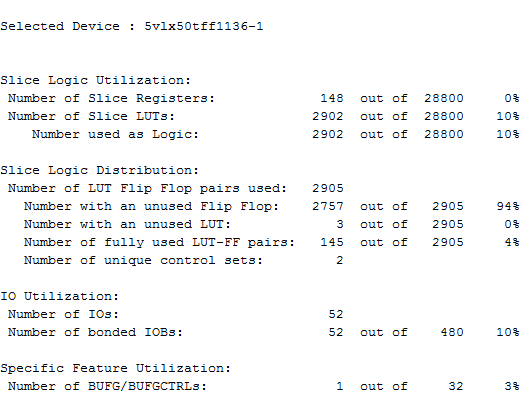
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Theta\_in | Output | | True value | |
| Cos\_out | Sin\_out | Cos\_out | Sin\_out |
|  | 0000000000000000 | 0011111111111101 | 0000000000000000 | 0100000000000000 |
|  | 0000000001111110 | 0011111111111100 | 0000000011001001 | 0011111111111111 |
|  | 0000000101111110 | 0011111111111100 | 0000000110010010 | 0011111111111011 |
|  | 0001100001110001 | 0011101100100111 | 0001100001111101 | 0011101100100000 |
|  | 0010110100010011 | 0010110101101101 | 0010110101000001 | 0010110101000001 |
|  | 0011101100110111 | 0001100001010000 | 0011101100100000 | 0001100001111101 |
|  | 0011111111111100 | 0000000110000001 | 0011111111111011 | 0000000011001001 |
|  | 0100000000000000 | 0000000011000001 | 0011111111111110 | 1111111100110111 |
| 0 | 0100000000000000 | 0000000000000000 | 0100000000000000 | 0000000000000000 |
|  | 0011111111111100 | 1111111010000001 | 0011111111111110 | 1111111100110111 |
|  | 0011111111111111 | 1111111101000000 | 0011111111111011 | 1111111001101110 |
|  | 0011101100111000 | 1110011110110010 | 0011101100100000 | 1110011110000011 |
|  | 0010110101101110 | 1101001011101101 | 0010110101000001 | 1101001010111111 |
|  | 0001100001110011 | 1100010011011001 | 0001100001111101 | 1100010011100000 |
|  | 0000000011000000 | 1100000000000100 | 0000000110010010 | 1100000000000010 |
|  | 0000000010000000 | 1100000000000011 | 0000000011001001 | 1100000000000010 |
|  | 0000000000000000 | 1100000000000011 | 0000000000000000 | 1100000000000000 |

|  |
| --- |
| *Table 2: Result* |

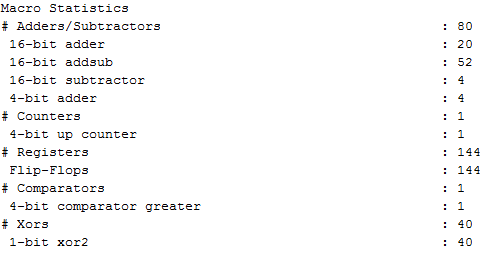
1. **Synthesize**
   1. **Timming**



* 1. **Source**



* 1. **Component**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Thank you\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*