ISSN 2278-3091
Volume 9, No.5, September - October 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse258952020.pdf
https://doi.org/10.30534/ijatcse/2020/258952020

Signal generator module based on CORDIC algorithm: Design, implementation, and verification using MATLAB and Verilog HDL

Phuong H. Lai[1], Binh A. Nguyen[2], Thien V. Truong[2], Tung V. Nguyen[2], Duong B. Nguyen[2], Duc D. Nhu[3]

[1]Dept. of Computing Fundamentals, FPT University, Hanoi, Vietnam;phuonglh17@fe.edu.vn
[2]ICT Department, FPT University, Hanoi, Vietnam; binhnase04865@fpt.edu.vn, thientvse04522@fpt.edu.vn, tungnvhe130151@fpt.edu.vn, duongnbhe130658@fpt.edu.vn
[3]Dept. of Multimedia Engineering, Dongguk University, Seoul, South Korea; nhudinhduc@mme.dongguk.edu

ABSTRACT

In this paper, we will design, implement, and verify the coordinate rotation digital computer (CORDIC) algorithm for the mixed signal generator module of the modern information system. First, a MATLAB tool will be developed to design parameters of CORDIC algorithm. Then, using these parameters, we will implement CORDIC in Verilog hardware description language (Verilog HDL). The verification will be done through both a testbench of our proposed hardware model and a calculation via MATLAB tool.

Key words: Application specific integrated design (ASIC),Field-programmable gate array (FPGA), MATLAB, Verilog hardware description language (Verilog HDL), CORDIC.

1. INTRODUCTION

Nowadays, the information system can be found anywhere around us which consists of many sub-modules with microcontroller or processor is the brain [1]. The signal generator is mandatory in every information system, and which can be implemented via several methods [2]. The hardware-efficient iterative method is the most effective famous and it has been used in many current applications around us such as calculation processor, robotics, 3D graphics, DSP, ARM based STM32G4, … due to its effective on speed and hardware cost [3].

Our research will focus on design, implementation, and verification of a system on chip design for signal generator module based on CORDIC algorithm [4,5,6].In our discussion, the HW module based on CORDIC algorithm is implemented to calculate the sine and cosine signal for an arbitrary angle. The proposed hardware model is flexible, easy-to-reuse to be the intellectual logic core. In our work, MATLAB R2020b will be used to design CORDIC algorithm and for mathematical computation. In addition, Model SimPE student version 10.4a tool with Verilog hardware description language is used to implement hardware model and a test bench for numerous input values.

The organization of our paper is as follows.In Section 2, we describe the preliminary of discrete-time signal and overview of CORDIC algorithm. The design, implementation, and verification will be proposed in Section 3. Finally, we conclude the conclusion and open the discussion in Section 4.

2. PRELIMINARY

A. Discrete-time signal and system

The overall concept of a modern information system is given in figure 1, which include three parts: sensor system, Analogue-front-end (AFE), and embedded processing part. Sensor system is needed to receive a driving signal which is sent from AFE part. The driving signal can be sine, cosine, or pulse signal for each design purpose, and the signal generator module is located between digital-to-analogue converter (DAC) and memory module (read-only-memory).

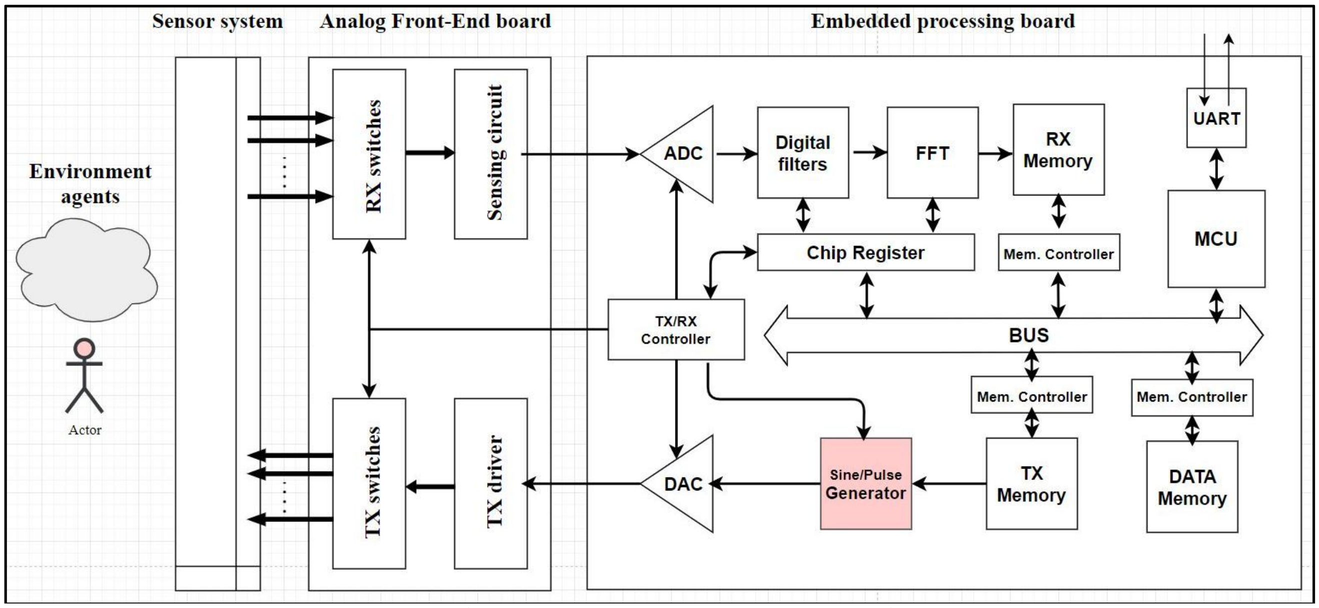


Figure 1: Location of signal generator module on modern information system.

B. Overview of CORDIC algorithm

Coordinate rotation digital computer (CORDIC for short) is a hardware-efficient iterative method which uses rotations to calculate a wide range of mathematical elementary function. The CORDIC algorithm can be implemented by hardware description language by using shift-add operations, then it is widely used in pocket calculators, in FPGA or ASIC where we need to minimize the number of gates, or in many older systems with integer-only CPUs.

Figure 2 is a visible example of multiplication operation by cosine acts as a scaling factor. Generally, the mathematic equation for CORDIC algorithm is given as:

$$\begin{cases} z_{i+1} = z_i - d_i * \text{atan}(2^{-i}) \\ x_{i+1} = x_i - y_i * d_i * 2^{-i} \\ y_{i+1} = y_i + x_i * d_i * 2^{-i} \end{cases} \quad (1)$$

where $d_i = -1$ if $z_i < 0$ and $d_i = +1$ for others, $i = 0,1,\dots,n-1$; n is numbers of iterations. The equation (1) will give following result when $n \to \infty$.

$$\begin{cases} z_n = 0 \\ x_n = A_n(x_0 \cos(z_0) - y_0 \sin(z_0)), \\ y_n = A_n(y_0 \cos(z_0) + x_0 \sin(z_0)) \end{cases} \quad (2)$$

where $A_n = \prod_{i=0}^{N-1} \sqrt{1 + 2^{-2i}}$.

Consequently, if we choose z_0 as an arbitrary angle, $x_0 = \frac{1}{A_n}$, and $y_0 = 0$,we have following result:

$$\begin{cases} x_n = \cos(z_0) \\ y_n = \sin(z_0) \end{cases}. \quad (3)$$

In this research, we implement a hardware module based on equation (3) to calculate the sine and cosine value of an arbitrary angel.

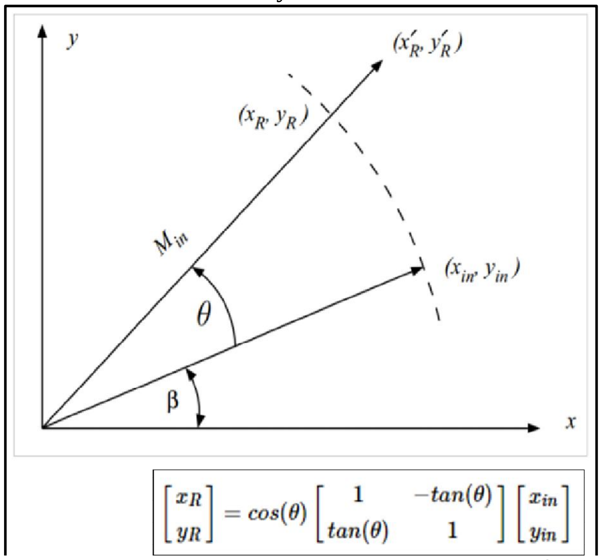

y
x
(x'_R, y'_R)
(x_R, y_R)
M_{in}
(x_{in}, y_{in})
θ
β
$\begin{bmatrix} xR \\ yR \end{bmatrix} = cos(\theta) \begin{bmatrix} 1 & -tan(\theta) \\ tan(\theta) & 1 \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}$

Figure 2: Multiplication by *cos* acts as a scaling factor.

3. SIGNAL GENRATOR MODULE BASED ON CORDIC ALGORITHM

A. Finite state machine chart of CORDIC algorithm

As the mathematical explanation in section 2, we have the corresponding finite state machine (FSM) of CORDIC algorithm with two states: IDLE and ITERATION as the upper part of figure 3. In addition, the algorithmic state machine (ASM) chartis analyzed in rest part of figure 3 which includes all the value of variables inputs, outputs, and their mathematical relation.From FSM and ASM, we can convert into FSM Moore machines as figure 4 which include next state logic, state register, and output logic,they are a good format for implementation via hardware description language.

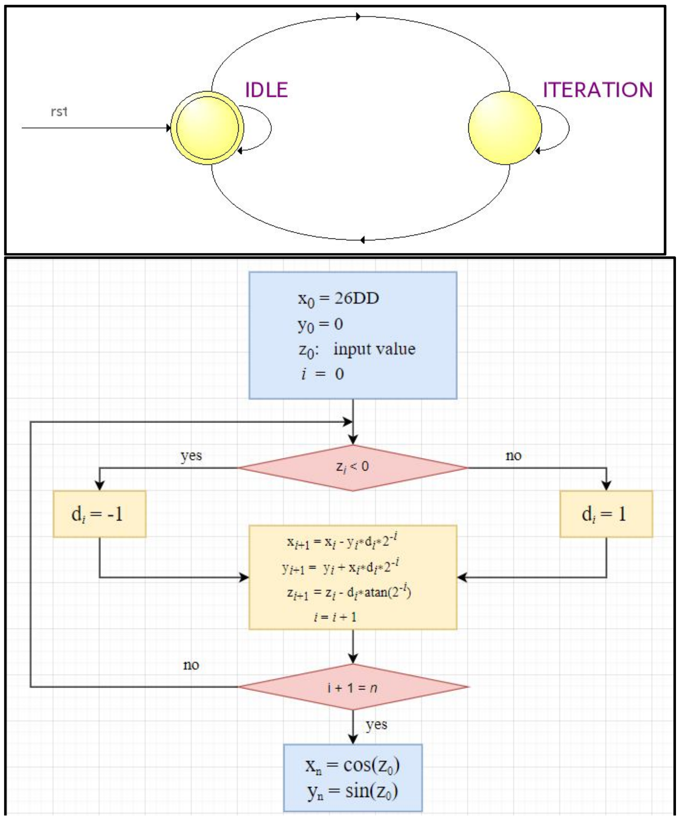


Figure 3: Finite state machine (FSM) and algorithmic state machine (ASM) charts of CORDIC algorithm.

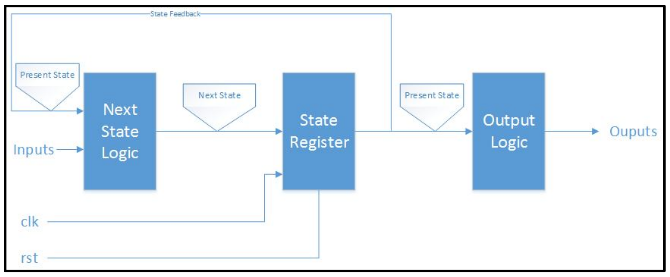
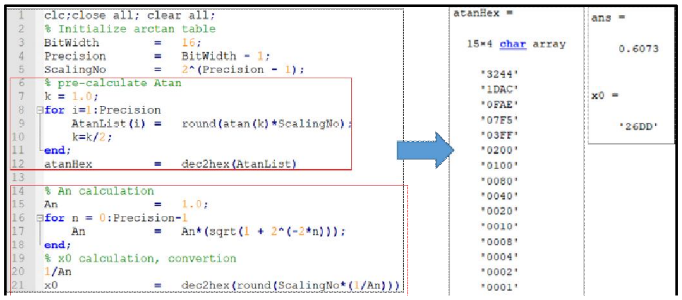


Figure 4:FSM Moore machines.



```
clc;close all; clear all;
% Initialize arctan table
BitWidth        =    16;
Precision       =    BitWidth - 1;
ScalingNo       =    2^(Precision - 1);
% pre-calculate Atan
k = 1.0;
for i=1:Precision
    AtanList(i) =    round(atan(k)*ScalingNo);
    k=k/2;
end;
atanHex         =    dec2hex(AtanList)

% An calculation
An              =    1.0;
for n = 0:Precision-1
    An          =    An*(sqrt(1 + 2^(-2*n)));
end;
% x0 calculation, convertion
1/An
x0              =    dec2hex(round(ScalingNo*(1/An)))
```

```
atanHex =

  15×4 char array

    '3244'
    '1DAC'
    '0FAE'
    '07F5'
    '03FF'
    '0200'
    '0100'
    '0080'
    '0040'
    '0020'
    '0010'
    '0008'
    '0004'
    '0002'
    '0001'
```

```
ans =

    0.6073

x0 =

    '26DD'
```

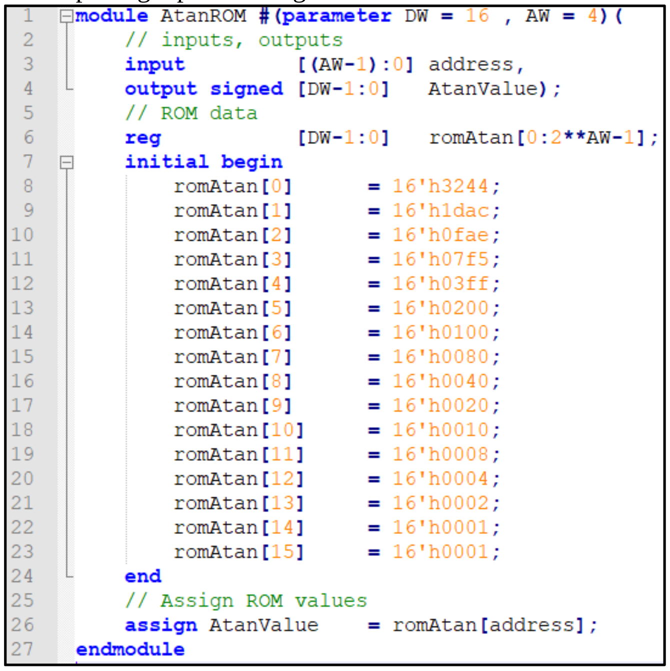
Figure 5:Design of CORDIC algorithm by MATLAB.

B. Implementation proposed module by Verilog HDL

In this subsection, we will implement a hardware module based on CORDIC algorithm to calculate the sine and cosine value of an arbitrary angle. The iteration for CORDIC algorithm is chosen as sixteen.

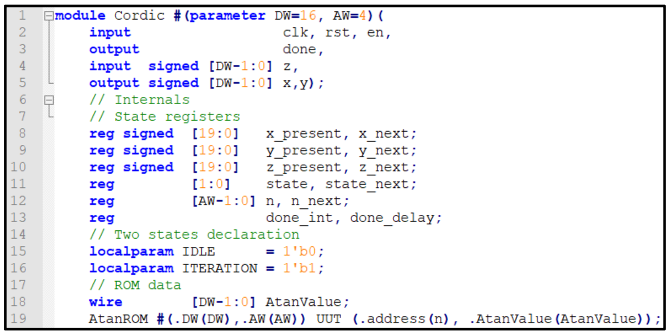
As the analysis of CORDIC algorithm in section 2, we develop a MATLAB calculation tool to design CORDIC algorithm parameters. The outputs of MATLAB design are ROM memory for Atan numbers, CORDIC parameters such as *x0* in integer machine format, they are explained in detail as figure 5 with clear explanations. Using the outputs from MATLAB design, Atan ROM module is implemented via Verilog HDL in figure 6.

The hardware module based on CORDIC algorithm is implemented in figure 7, 8, 9, and 10. In figure 7, the inputs, outputs, and the ROM are declared where the input, angle value, is denoted as *z*, the sine value is denoted as *x*, and the cosine value is denoted as *y*. The state registers, current state and next states are declared which all are clear as highlight in figure 7. Corresponding to HDL format of FSM Moore machines, the Verilog HDL implementation for state registers part is in figure 8, for next stages logic part is in figure 9, and the output logic part is in figure 10.



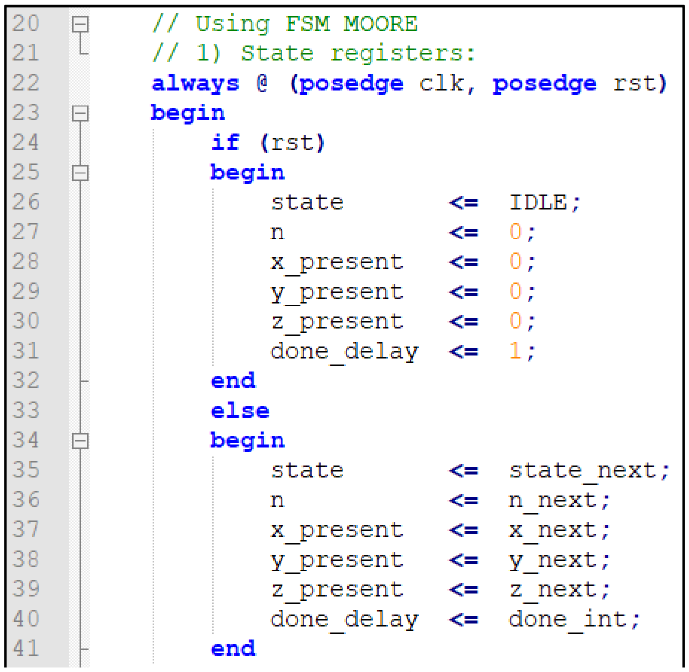
```
module AtanROM #(parameter DW = 16 , AW = 4)(
    // inputs, outputs
    input           [(AW-1):0] address,
    output signed [DW-1:0]   AtanValue);
    // ROM data
    reg             [DW-1:0]   romAtan[0:2**AW-1];
    initial begin
        romAtan[0]      = 16'h3244;
        romAtan[1]      = 16'h1dac;
        romAtan[2]      = 16'h0fae;
        romAtan[3]      = 16'h07f5;
        romAtan[4]      = 16'h03ff;
        romAtan[5]      = 16'h0200;
        romAtan[6]      = 16'h0100;
        romAtan[7]      = 16'h0080;
        romAtan[8]      = 16'h0040;
        romAtan[9]      = 16'h0020;
        romAtan[10]     = 16'h0010;
        romAtan[11]     = 16'h0008;
        romAtan[12]     = 16'h0004;
        romAtan[13]     = 16'h0002;
        romAtan[14]     = 16'h0001;
        romAtan[15]     = 16'h0001;
    end
    // Assign ROM values
    assign AtanValue    = romAtan[address];
endmodule
```

Figure 6: Implementation of Atan ROM by Verilog HDL.



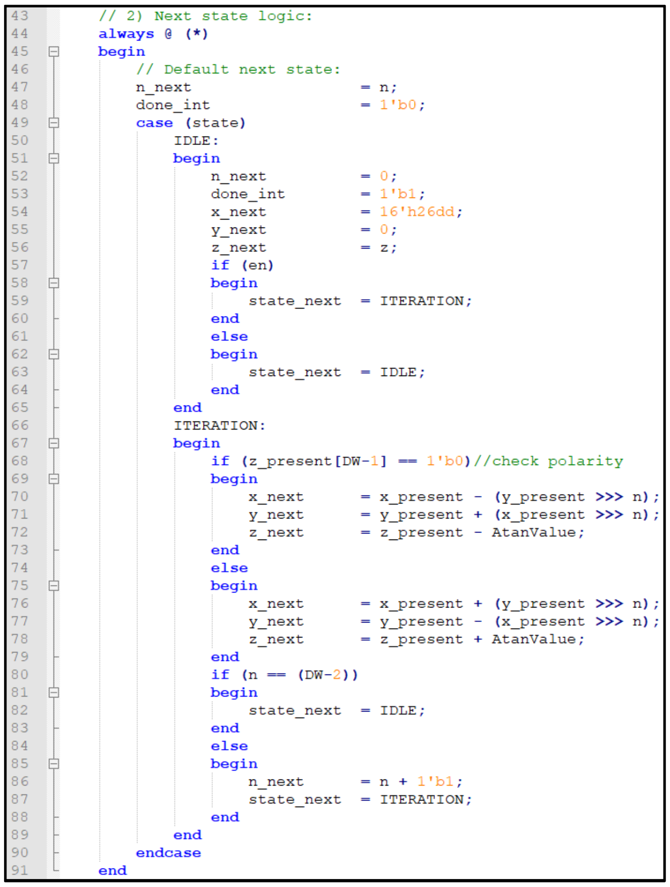
```
module Cordic #(parameter DW=16, AW=4)(
    input                   clk, rst, en,
    output                  done,
    input  signed [DW-1:0] z,
    output signed [DW-1:0] x,y);
    // Internals
    // State registers
    reg signed  [19:0]   x_present, x_next;
    reg signed  [19:0]   y_present, y_next;
    reg signed  [19:0]   z_present, z_next;
    reg         [1:0]    state, state_next;
    reg         [AW-1:0] n, n_next;
    reg                  done_int, done_delay;
    // Two states declaration
    localparam IDLE      = 1'b0;
    localparam ITERATION = 1'b1;
    // ROM data
    wire        [DW-1:0] AtanValue;
    AtanROM #(.DW(DW),.AW(AW)) UUT (.address(n), .AtanValue(AtanValue));
```

Figure 7: Declaration of inputs, outputs, state registers.



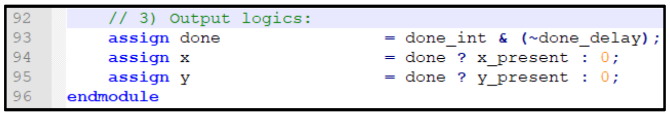
```
    // Using FSM MOORE
    // 1) State registers:
    always @ (posedge clk, posedge rst)
    begin
        if (rst)
        begin
            state       <=  IDLE;
            n           <=  0;
            x_present   <=  0;
            y_present   <=  0;
            z_present   <=  0;
            done_delay  <=  1;
        end
        else
        begin
            state       <=  state_next;
            n           <=  n_next;
            x_present   <=  x_next;
            y_present   <=  y_next;
            z_present   <=  z_next;
            done_delay  <=  done_int;
        end
```

Figure 8: State registers part of CORDIC module.



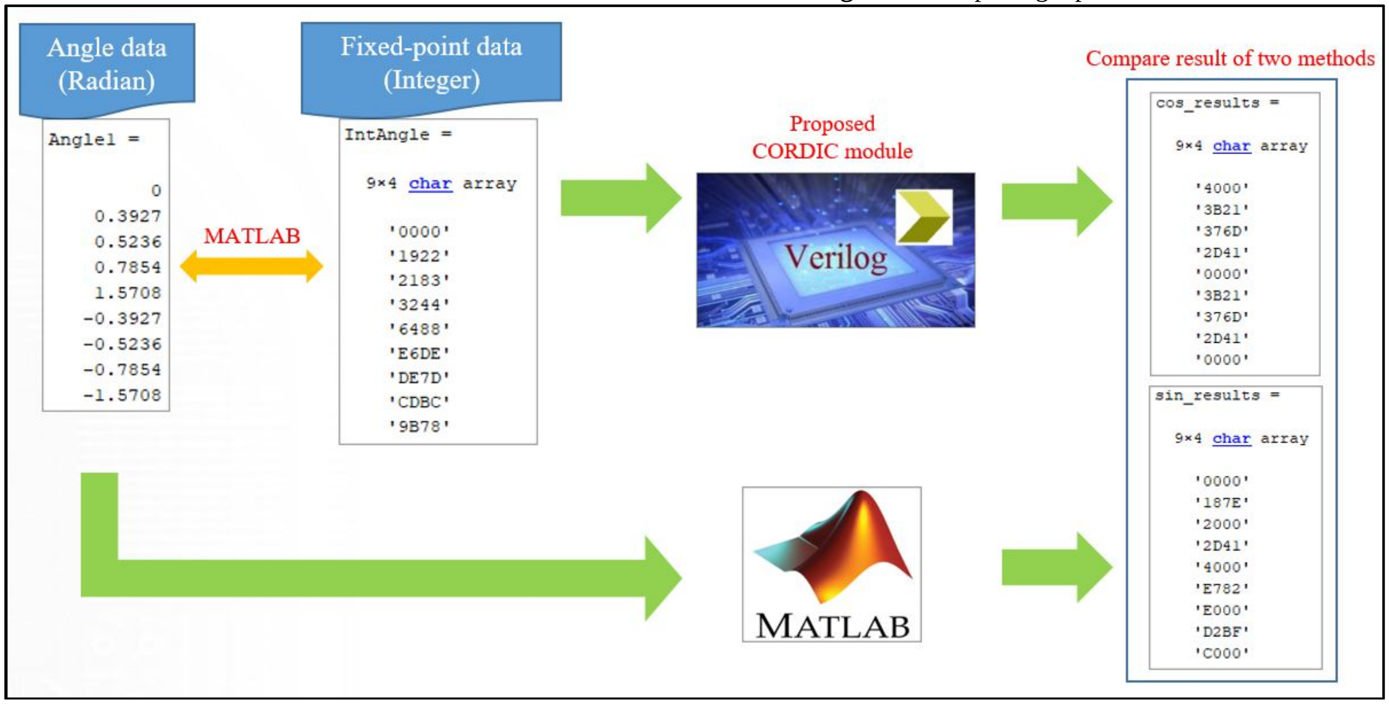
```
    // 2) Next state logic:
    always @ (*)
    begin
        // Default next state:
        n_next                  = n;
        done_int                = 1'b0;
        case (state)
            IDLE:
            begin
                n_next          = 0;
                done_int        = 1'b1;
                x_next          = 16'h26dd;
                y_next          = 0;
                z_next          = z;
                if (en)
                begin
                    state_next  = ITERATION;
                end
                else
                begin
                    state_next  = IDLE;
                end
            end
            ITERATION:
            begin
                if (z_present[DW-1] == 1'b0)//check polarity
                begin
                    x_next      = x_present - (y_present >>> n);
                    y_next      = y_present + (x_present >>> n);
                    z_next      = z_present - AtanValue;
                end
                else
                begin
                    x_next      = x_present + (y_present >>> n);
                    y_next      = y_present - (x_present >>> n);
                    z_next      = z_present + AtanValue;
                end
                if (n == (DW-2))
                begin
                    state_next  = IDLE;
                end
                else
                begin
                    n_next      = n + 1'b1;
                    state_next  = ITERATION;
                end
            end
        endcase
    end
```

Figure 9:Next stage logic partof CORDIC module.



```
    // 3) Output logics:
    assign done                 = done_int & (~done_delay);
    assign x                    = done ? x_present : 0;
    assign y                    = done ? y_present : 0;
endmodule
```

Figure 10: Output logic part of CORDIC module.



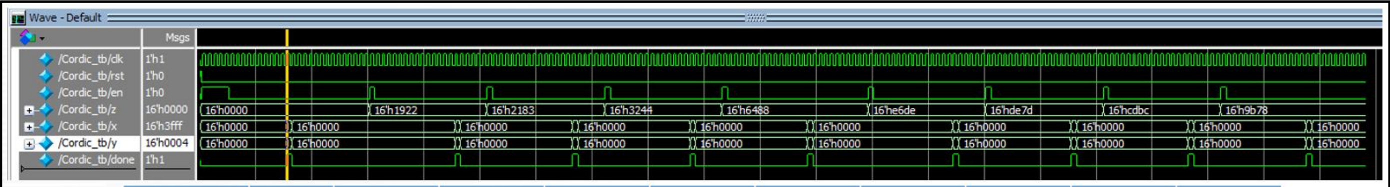
```
Angle1 =

        0
   0.3927
   0.5236
   0.7854
   1.5708
  -0.3927
  -0.5236
  -0.7854
  -1.5708
```

```
IntAngle =

  9×4 char array

    '0000'
    '1922'
    '2183'
    '3244'
    '6488'
    'E6DE'
    'DE7D'
    'CDBC'
    '9B78'
```

```
cos_results =

  9×4 char array

    '4000'
    '3B21'
    '376D'
    '2D41'
    '0000'
    '3B21'
    '376D'
    '2D41'
    '0000'
```

```
sin_results =

  9×4 char array

    '0000'
    '187E'
    '2000'
    '2D41'
    '4000'
    'E782'
    'E000'
    'D2BF'
    'C000'
```

Figure 11: Verification system to test proposed CORDIC module.



	Angle	0000	1922	2183	3244	6488	E6DE	DE7D	CDBC	9B78
MATLAB	Cos	4000	3B21	376D	2D41	0000	3B21	376D	2D41	0000
	Sine	0000	187E	2000	2D41	4000	E782	E000	D2BF	C000
Verilog	Cos	3FFF	3B1F	376F	2D42	0002	3B23	3F6F	2D43	FFFF
	Sine	0004	1880	1FFD	2D41	3FFF	E785	E002	D2BE	BFFF

Figure 12: Wave form result for a testbench of proposed CORDIC module in comparison with MATLAB result.

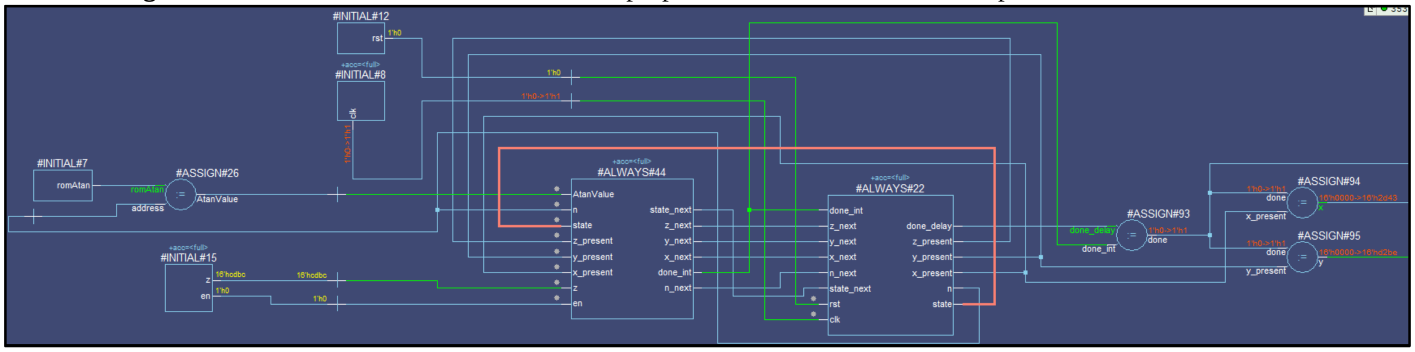
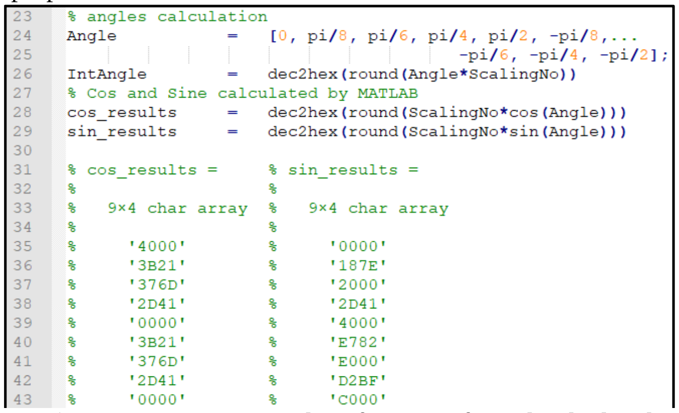


Figure 13: Schematic view of our proposed CORDIC module.

C. Verification of proposed CORDIC module

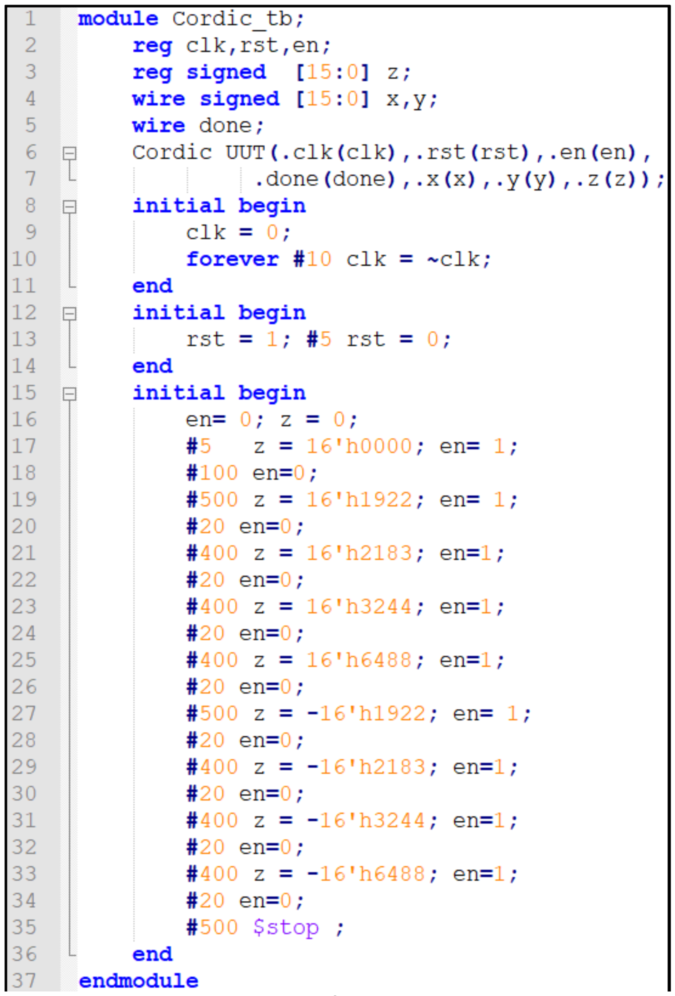
To verify our proposed system, we will use two methods. First method is to use MATLAB to calculate the sine and cosine values for several arbitrary angles. Second method is to develop a testbench from our proposed module. As the result of two method, we can estimate the correction of our proposed module. The detail of verification method can be seen in figure 11. The MATLAB tool is used for data conversion between real numbers and these integer format to be used in HDL.

MATLAB based verification are calculated in figure 14 where the array of arbitrary inputs is defined and converted into integer data. The testbench is implemented in figure 15 via Verilog HDL. The waveform of testbenchis shown in figure 12 where one important thing to consider is that the throughput delay clocks equals to the number of iterations we defined (sixteen). In figure 12, the results of two proposedmethods are shown in a table. As can be seen from table, the result of two methods are mostly same and we can conclude that our proposed method is believable.Finally, the schematic view of our proposed module and testbench can be found in figure 13.



```
% angles calculation
Angle             =   [0, pi/8, pi/6, pi/4, pi/2, -pi/8,...
                                              -pi/6, -pi/4, -pi/2];
IntAngle          =   dec2hex(round(Angle*ScalingNo))
% Cos and Sine calculated by MATLAB
cos_results       =   dec2hex(round(ScalingNo*cos(Angle)))
sin_results       =   dec2hex(round(ScalingNo*sin(Angle)))

% cos_results =       % sin_results =
%                     %
%   9×4 char array    %   9×4 char array
%                     %
%     '4000'          %     '0000'
%     '3B21'          %     '187E'
%     '376D'          %     '2000'
%     '2D41'          %     '2D41'
%     '0000'          %     '4000'
%     '3B21'          %     'E782'
%     '376D'          %     'E000'
%     '2D41'          %     'D2BF'
%     '0000'          %     'C000'
```

Figure 14: Sine/cosinevalues for a set of angelscalculated by MATLAB.



```
module Cordic_tb;
    reg clk,rst,en;
    reg signed  [15:0] z;
    wire signed [15:0] x,y;
    wire done;
    Cordic UUT(.clk(clk),.rst(rst),.en(en),
             .done(done),.x(x),.y(y),.z(z));
    initial begin
        clk = 0;
        forever #10 clk = ~clk;
    end
    initial begin
        rst = 1; #5 rst = 0;
    end
    initial begin
        en= 0; z = 0;
        #5   z = 16'h0000; en= 1;
        #100 en=0;
        #500 z = 16'h1922; en= 1;
        #20 en=0;
        #400 z = 16'h2183; en=1;
        #20 en=0;
        #400 z = 16'h3244; en=1;
        #20 en=0;
        #400 z = 16'h6488; en=1;
        #20 en=0;
        #500 z = -16'h1922; en= 1;
        #20 en=0;
        #400 z = -16'h2183; en=1;
        #20 en=0;
        #400 z = -16'h3244; en=1;
        #20 en=0;
        #400 z = -16'h6488; en=1;
        #20 en=0;
        #500 $stop ;
    end
endmodule
```

Figure 15: A test bench of proposed CORDIC module.

4. CONCLUSION

In the short research, we have explained a signal generator module for the modern information system based on CORDIC algorithm, the proposed module is very simple, just to calculate the function values for arbitrary angle. In future research, we will consider CORDIC module for more complex task such as design for a full signal where we need to use some advantage techniques such as pipelined and multi-cores.

All the steps of design, implementation, verification has given in detail via MATLAB and Modelsim tool simulation. As the result of testbench in Verilog and calculation in MATLAB, the result of hardware design is guaranteed to be IP core for ASIC and FPGA. We hope that the short research will help students, engineers have a good reference on system on chip design. All source codes are available to share via reasonable request.

ACKNOWLEDGEMENT

This work is supported byFPT University, Hanoi, Vietnam.

Conflict of Interest

On behalf of all authors, corresponding author declares that there is no conflict of interest to publish this research.

REFERENCES

1. Kiran Kumar V G et al., "**FPGA Implementation of Simple Encryption Scheme for Resource-Constrained Devices**" Int. J. of Advanced Trends in Computer Science and Engineering, 9(4), 2020, 5631 – 5639.
2. Nguyen, D.M., Kim, S. “**A novel construction for quantum stabilizer codes based on binary formalism**”. Int. J. of Modern Phys. B. vol. 33(8), 2020.
3. Kantabutra, V. "**On hardware for computing Exponential and Trigonometric Functions,**" IEEE Transactions on Computers, vol. 45, no. 3, march, 1996, pp. 328-339.
4. Hieu V.D.et. al. “**Design and verification of novel classical error control codes using VERILOG Hardware Description Language**”. Int. J. of Advanced Trends in Computer Science and Engineering, vol. 9(4), 2020.
5. Zhang, C., Han, J. “**Design and Implementation of Hybrid CORDIC algorithm based on phased rotation estimation for NCO**”. The Scientific World Journal, 2014.
6. Chetan D., Hari K. M.,"**Design and Analysis of Double Precision Floating Point Division Operator Based on CORDIC Algorithm**". Int. J. of Science and Research (IJSR), vol. 3(7), 2014, pp. 1378 - 1381.