Santa Clara University
Dept. of Electrical Engineering

ELEN 249: VLSI Architectures for Communications and Signal Processing Systems

Project Report on

Synthesizable Verilog RTL Design of AWGN

By: Prateek Sharma
W1112370

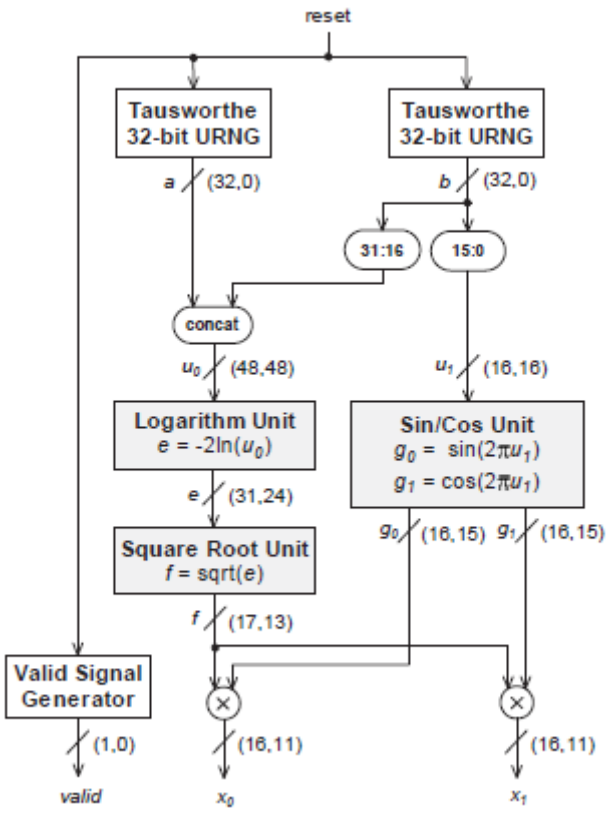
Problem Statement

Implement the MATLAB and RTL code of AWGN using Box-Muller Method. Following pseudo-code can be used to implement the design –

```
01: ---------------- Generate u0 and u1 ----------------
02: a  = taus(); b = taus();
03: u0 = concat(a,b[31:16]);
04: u1 = b[15:0];
05:
06: ------------- Evaluate a = -2ln(u0) ---------------
07:
08: # Range Reduction
09: exp_a = LeadingZeroDetector(u0)+1;
10: x_a   = u0 << exp_a;
11:
12: # Approximate -ln(x_a) where x_a = [1,2)
13: # Degree-2 piecewise polynomial
14: y_a = ((C2_a[x_a_B]*x_a)+C1_a[x_a_B])*x_a_B
15:       +C0_a[x_a_B];
16:
17: # Range Reconstruction
18: ln2 = ln(2);
19: a'  = exp_a*ln2;
20: a   = (a'-y_a)<<1;
21:
22: ------------- Evaluate f = sqrt(a) ---------------
23:
24: # Range Reduction
25: exp_f = 5-LeadingZeroDetector(a);
26: x_f'  = a >> exp_f;
27: x_f   = if(exp_f[0], x_f'>>1, x_f');
28:
29: # Approximate sqrt(x_f) where x_f = [1,4)
30: # Degree-1 piecewise polynomial
31: y_f = C1_f[x_f_B]*x_f_B+C0_f[x_f_B];
32:
33: # Range Reconstruction
34: exp_f' = if(exp_f[0], exp_f+1>>1, exp>>1);
35: f      = y_f << exp_f';
36:
37: ---------- Evaluate g0=sin(2*pi*u1) ------------
38: ----------          g1=cos(2*pi*u1) ------------
39:
40: # Range Reduction
41: quad  = u1[15:14];
42: x_g_a = u1[13:0];
43: x_g_b = (1-2^-14)-u1[13:0];
44:
45: # Approximate cos(x_g_a*pi/2) and cos(x_g_b*pi/2)
46: # where x_g_a, x_g_b = [0,1-2^-14]
47: # Degree-1 piecewise polynomial
48: y_g_a = C1_g[x_g_a_B]*x_g_a_B+C0_g[x_g_a_B];
49: y_g_b = C1_g[x_g_b_B]*x_g_b_B+C0_g[x_g_b_B];
50:
51: # Range Reconstruction
52: switch(seg)
53:   case 0:   g0 =  y_g_b; g1 =  y_g_a;
54:   case 1:   g0 =  y_g_a; g1 = -y_g_b;
55:   case 2:   g0 = -y_g_b; g1 = -y_b_a;
56:   case 3:   g0 = -y_g_a; g1 =  y_g_b;
57:
58: --------------- Compute x0 and x1 ----------------
59: x0 = f*g0; x1 = f*g1;
```

- Tausworthe Block is used to generate 32-bit Uniform Random Numbers.
- Natural Logs are calculated in Logarithm Unit.
- Square-root of the input is calculated in the Square Root Unit.
- Sin and Cosine values of input angle is found in Sin/Cosine Block.

The following flowchart shows how the design can be implemented with minimum bit-widths of all the signals.



1. **32-bit Tausworthe URNG**

Uniform Random Numbers are generated using Tausworthe's algorithm. Initially seeds are passed into the tausworthe block to calculate the first number. Then the updated seeds are used to calculate the next random number. As we need 32-bit random numbers, all the variables in Tausworthe are of 32-bits. The following code is used to calculate the Tausworthe Random Number –

```
b0 <= ((( s0 << 13) ^ s0) >> 19);

s0 <= ((( s0 & 32'hFFFFFFFE) << 12 ) ^ b0);

b1 <= ((( s1 << 2) ^ s1) >> 25);

s1 <= ((( s1 & 32'hFFFF_FFF8) << 4 ) ^ b1);

b2 <= ((( s2 << 3) ^ s2) >> 11);

s2 <= ((( s2 & 32'hFFFF_FFF0) << 17 ) ^ b2);


Tout <= s0 ^ s1 ^ s2;
```

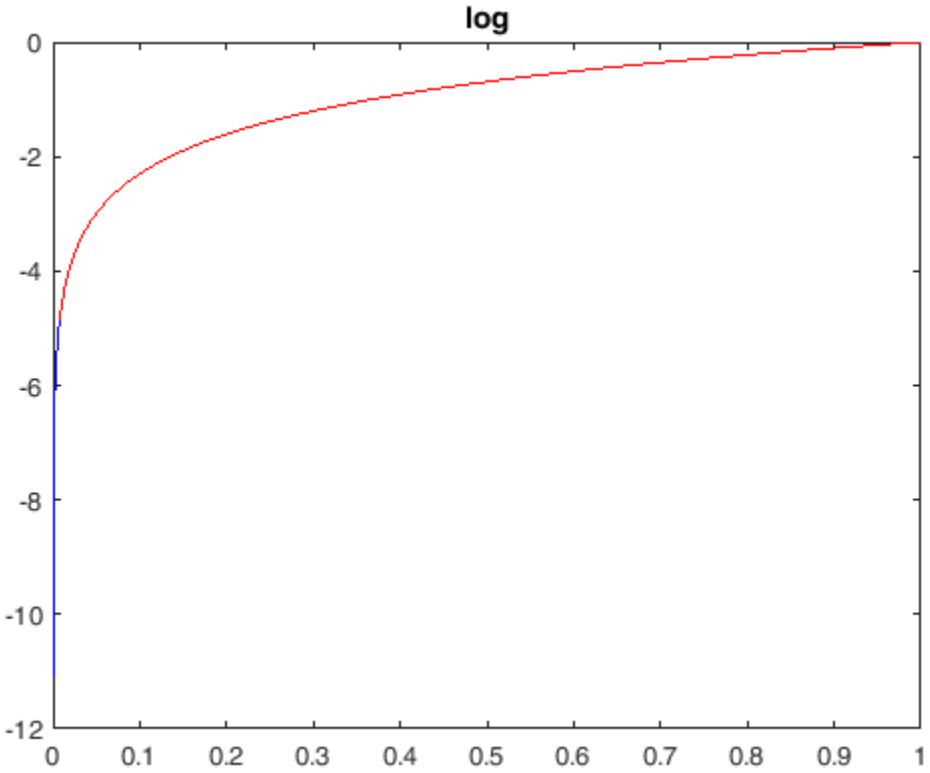
2. Logarithm Block

It's easy to calculate logarithmic values in software but to calculate the log values in a hardware design we have to bit-level calculations. For bit-level calculations in hardware it is easy if we can implement the function using only adders and multipliers.

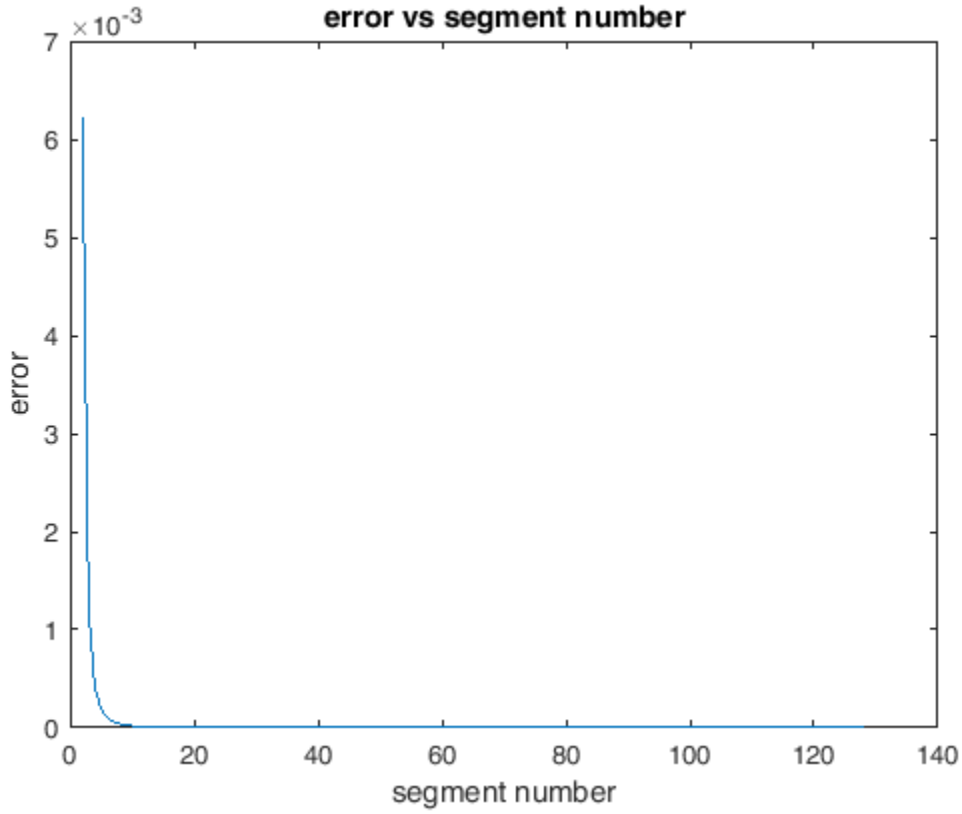
To implement a function using adders and multipliers we do the polynomial calculations. Functions can be defined using polynomials of various degrees. If we divide a function in segments and then calculate its polynomial, then we can get a better curve of the function. For this approach in the project I have used MATLAB functions like *polyfit* and *polyval*. The following code gives the values of polynomial coefficients for all the polynomials (Degree = 2) that are generated for the given number of segments of log function.

```
ns=2^7; %%number of segments
nr=2^9; %%number of points in each segment
np=2; %% polynomial order used

for i=1:ns          % For all segments
    %Start and End Points of segment
        xsegment_end   = i/ns;
        xsegment_step  = 1/(ns*nr);
        xsegment_start = (i-1)/ns;
    x(i,:)  =  xsegment_start:xsegment_step:xsegment_end;
    y(i,:)  =  log(x(i,:));     %Log values
    p(i,:)  =  polyfit(x(i,:),y(i,:),np);   %polynomial coef
     y1(i,:) = polyval(p(i,:),x(i,:));   %polynomial values

    %Log 0 is not defined therefore we need initialized
coefficients
    p(1,1) = -21474923943.6744;
    p(1,2) = 524289.593756354;
    p(1,3) = -14.4236942724212;

    Cc2(i,:) = ( -1 * p(i,1));
    Cc0(i,:) = ( -1 * p(i,3));

    %Convert coefficients into fixed point binary values
        C2fi(i,1) = fi(Cc2(i,1), 0, 30, 16);
        C2Binary (i,:) = bin(C2fi(i,1));

        C1fi(i,1) = fi(p(i,2), 0, 22, 13);
        C1Binary (i,:) = bin(C1fi(i,1));

        C0fi(i,1) = fi(Cc0(i,1), 0, 13, 10);
        C0Binary (i,:) = bin(C0fi(i,1));

    e(i)  = max(abs(y(i,:)- y1(i,:)));  %error
end
```

If we plot the graph of the polynomial functions, then we get a similar graph like the logarithm function-



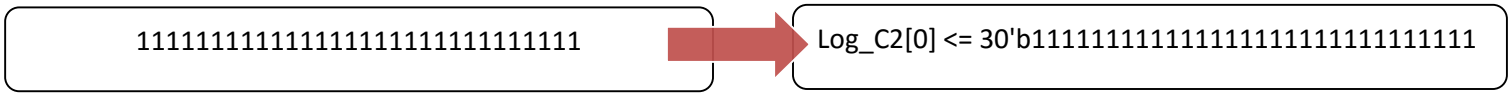
The graph for error in the actual log values and the ones calculated from the polynomial is –



Thus, we get the polynomial equations of logarithm function. The coefficients of these polynomial equations are needed for the Verilog design so I have written scripts in ***python*** language that would give you Verilog statements which can be directly pasted into the Verilog code. An example of such python script is –

```
import math

#File to be written
fw = open('C0_Log_FIFO.txt', 'w')

#File to be read
fd = open('LOG_C0.txt', 'r')

n = ';\n'
b = "13'b"
x = 0
ad = '\t\tLog_C0['
for line in fd:

        data = str(line)
        data = data.strip()
        RHS = str(b + str(data) + n)
        loc = str( x )
        x = x + 1
        LHS = str(ad + loc + '] <= ')
        fw.write( LHS + RHS )

fw.close()
fd.close()
```

The above python code will do the following –



Similar calculations were done for finding polynomial coefficients of Square-Root and Sin/Cos Units.

From the reference paper, the number of segments needed for log is 256. The minimal bit-widths of each coefficient is calculated in the reference paper and I have used the same values. Thus, there is a need to design memory in Verilog which will store all the coefficient values.

For memory design I have used SystemVerilog so that I can create memories using 2-dimensional arrays. I have used Synopsys's VCS and Design Compiler for hardware design and test. Following are the area and timing reports of Logarithm block generated from the tool –

```
****************************************
Report : area
Design : LOG_POLY
Version: I-2013.12-SP1
Date   : Mon Mar 21 06:39:15 2016
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

    lsi_10k (File: /opt/synopsys-2013/app/syn/libraries/syn/lsi_10k.db)

Number of ports:                          81
Number of nets:                         5704
Number of cells:                        5123
Number of combinational cells:          4770
Number of sequential cells:              348
Number of macros/black boxes:              0
Number of buf/inv:                      1210
Number of references:                     37

Combinational area:              72514.000000
Buf/Inv area:                     2538.000000
Noncombinational area:            3432.000000
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 75946.000000
Total area:                 undefined
```

Logarithm Unit Area Report

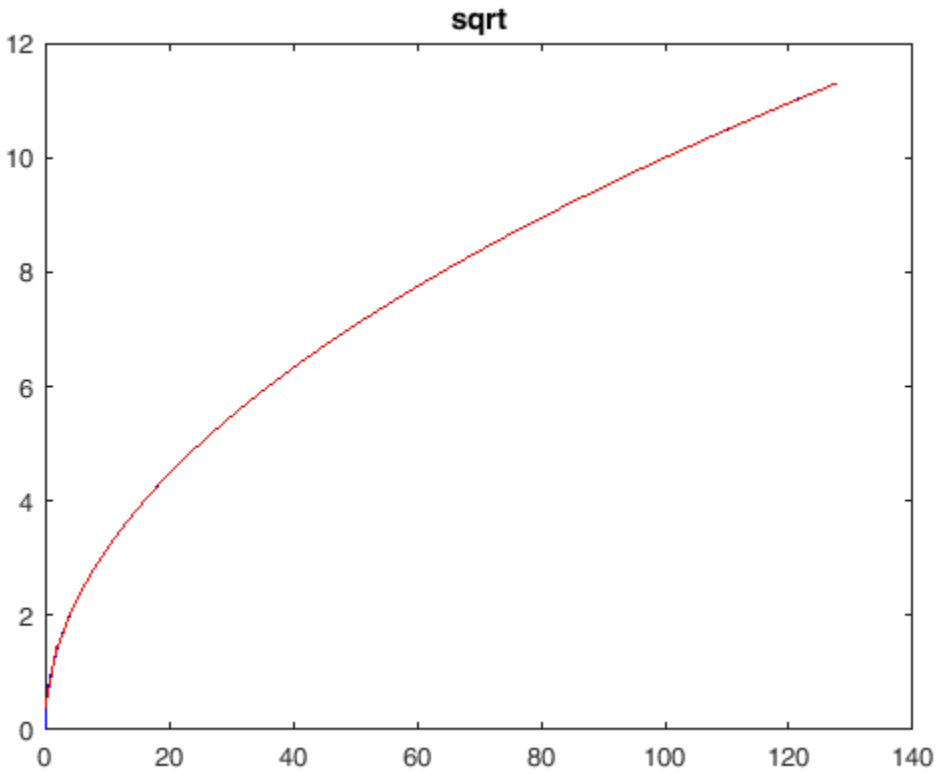
```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : LOG_POLY
Version: I-2013.12-SP1
Date   : Mon Mar 21 06:39:15 2016
****************************************

Operating Conditions: nom_pvt   Library: lsi_10k
Wire Load Model Mode: top

  Startpoint: LogIn[36] (input port clocked by clk)
  Endpoint: LogInSquare_reg[91]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

data arrival time                                         99.20

clock clk (rise edge)                          100.00     100.00
clock network delay (ideal)                      0.00     100.00
LogInSquare_reg[91]/CP (FD1)                     0.00     100.00 r
library setup time                              -0.80      99.20
data required time                                         99.20
---------------------------------------------------------------------
data required time                                         99.20
data arrival time                                         -99.20
---------------------------------------------------------------------
slack (MET)                                                0.00
```

Logarithm Unit Timing Report

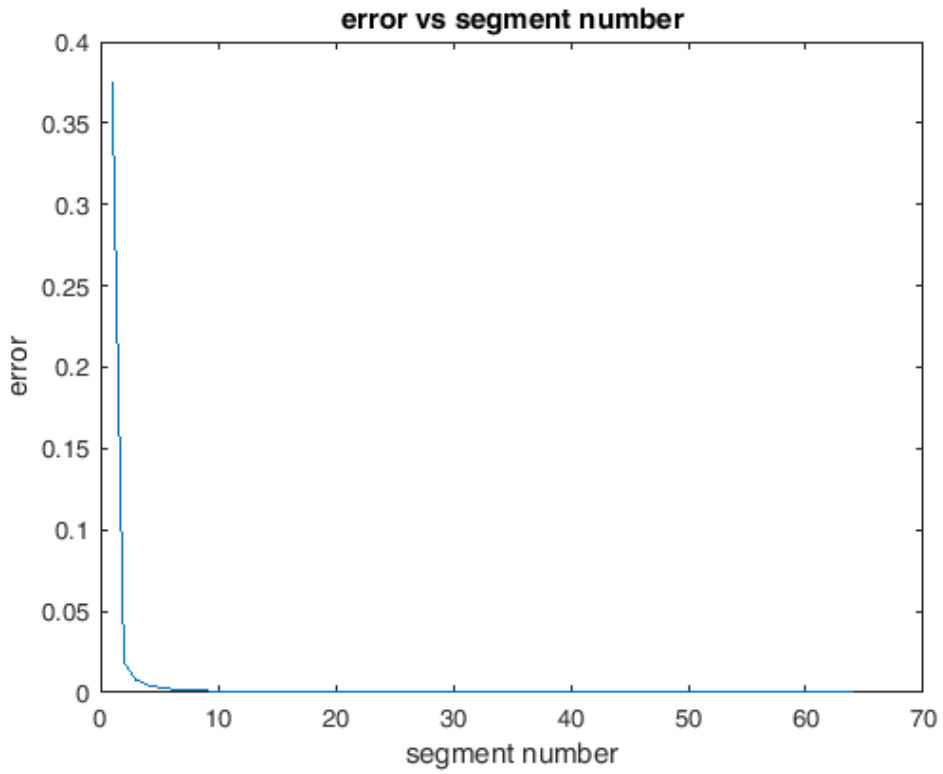
3. Square-Root Block

```
ns=2^6; %%number of segments
nr=2^8; %%number of points in each segment
np=1; %% polynomial order used

for i=1:(ns)
xsegment_end   = 2*i;
xsegment_step  = 1/(nr);
xsegment_start = 2*(i-1);

x(i,:)  =  xsegment_start:xsegment_step:xsegment_end;
y(i,:)  =  sqrt(x(i,:));
p(i,:)  =  polyfit(x(i,:),y(i,:),np);

y1(i,:) = polyval(p(i,:),x(i,:));

%%polynomial
C1fi(i,1) = fi(p(i,1), 0, 12, 12);
C1Binary (i,:) = bin(C1fi(i,1));

C0fi(i,1) = fi(p(i,2), 0, 20, 17);
C0Binary (i,:) = bin(C0fi(i,1));

e(i)  = max(abs(y(i,:)- y1(i,:)));
end
```

If we plot the graph of the polynomial functions, then we get a similar graph like the square-root function-



The graph for error in the actual sqrt values and the ones calculated from the polynomial is –



Following are the area and timing reports of Square-Root block generated from the tool –

```
****************************************
Report : area
Design : SQRT_POLY
Version: I-2013.12-SP1
Date   : Mon Mar 21 07:04:39 2016
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

    lsi_10k (File: /opt/synopsys-2013/app/syn/libraries/syn/lsi_10k.db)

Number of ports:                           50
Number of nets:                           591
Number of cells:                          437
Number of combinational cells:            347
Number of sequential cells:                88
Number of macros/black boxes:               0
Number of buf/inv:                         82
Number of references:                      51

Combinational area:               5385.000000
Buf/Inv area:                      264.000000
Noncombinational area:             868.000000
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  6253.000000
Total area:                 undefined
```

Square-Root Unit Area Report

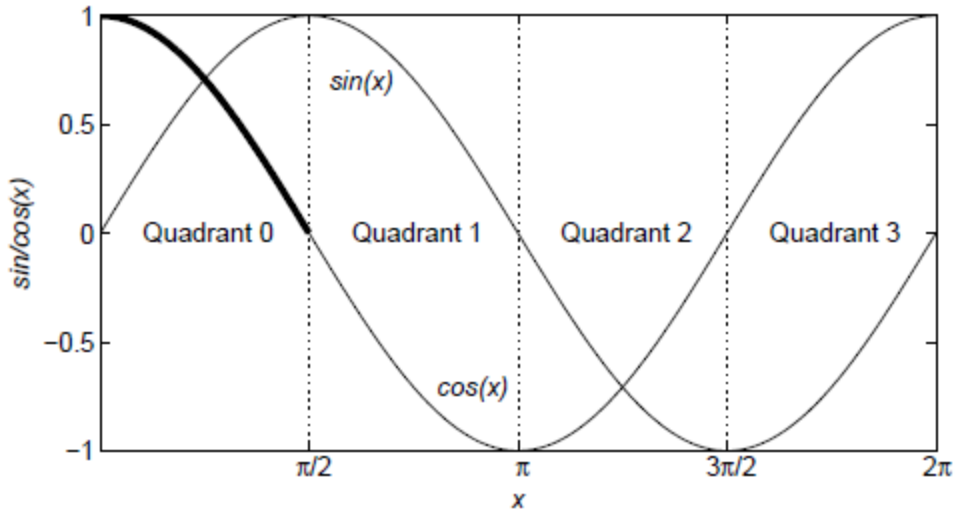
```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : SQRT_POLY
Version: I-2013.12-SP1
Date   : Mon Mar 21 07:04:39 2016
****************************************

Operating Conditions: nom_pvt   Library: lsi_10k
Wire Load Model Mode: top

  Startpoint: RootIn[27] (input port clocked by clk)
  Endpoint: Term1_reg[38]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max
```

data arrival time		34.20
clock clk (rise edge)	35.00	35.00
clock network delay (ideal)	0.00	35.00
Term1_reg[38]/CP (FD1)	0.00	35.00 r
library setup time	-0.80	34.20
data required time		34.20
data required time		34.20
data arrival time		-34.20
slack (MET)		0.00

Square-Root Unit Timing Report

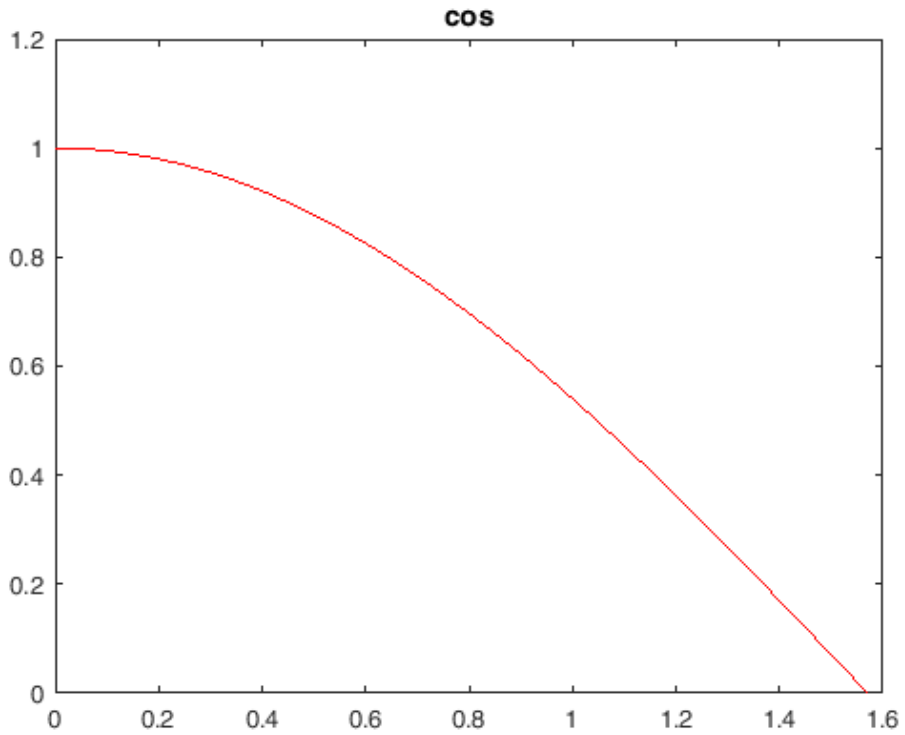
4. Sin/Cosine Block

As shown in figure below, Sine and Cosine waveforms are identical with a phase shift of 90. Also we can observe that the cosine wave is symmetrical in quadrants. Thus, if we can get polynomial functions for cosine from 0 to 90 degrees range then we can calculate sine and cosine values from the polynomials.

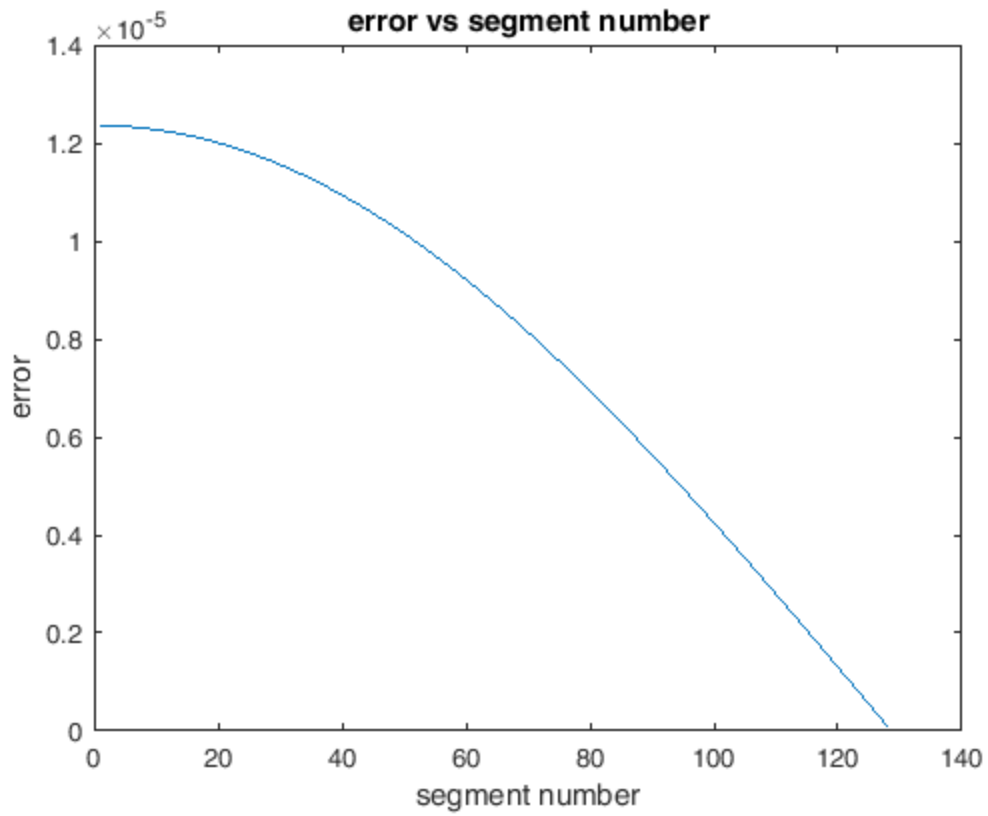


```
ns=2^7; %%number of segments
nr=2^6; %%number of points in each segment
np=1; %% polynomial order used
arg_max = pi/2;

for i=1:(ns)

    xsegment_start = (i-1)/ns*arg_max;
    xsegment_end   = i/ns*arg_max;
    xsegment_step  = 1/(ns*nr)*arg_max;

    x(i,:)  =  xsegment_start:xsegment_step:xsegment_end;
    y(i,:)  =  cos(x(i,:));
    p(i,:)  =  polyfit(x(i,:),y(i,:),np);

    y1(i,:) = polyval(p(i,:),x(i,:));

        %%polynomial
        Cc1(i,:) = ( -1 * p(i,1));
        C1fi(i,1) = fi(Cc1(i,1), 0, 12, 11);
        C1Binary (i,:) = bin(C1fi(i,1));

        C0fi(i,1) = fi(p(i,2), 0, 19, 18);
        C0Binary (i,:) = bin(C0fi(i,1));

    e(i)  = max(abs(y(i,:)- y1(i,:)));
end
```

If we plot the graph of the polynomial functions, then we get a similar graph like the cosine function-



The graph for error in the actual cosine values and the ones calculated from the polynomial is –



Following are the area and timing reports of Sin/Cosine block generated from the tool –

```
****************************************
Report : area
Design : SinBlock
Version: I-2013.12-SP1
Date   : Mon Mar 21 07:11:36 2016
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

    lsi_10k (File: /opt/synopsys-2013/app/syn/libraries/syn/lsi_10k.db)

Number of ports:                           35
Number of nets:                           646
Number of cells:                          579
Number of combinational cells:            545
Number of sequential cells:                33
Number of macros/black boxes:               0
Number of buf/inv:                        116
Number of references:                      23

Combinational area:               2532.000000
Buf/Inv area:                      154.000000
Noncombinational area:             329.000000
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  2861.000000
Total area:                 undefined
```

Sin-Cos Unit Area Report

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : SinBlock
Version: I-2013.12-SP1
Date   : Mon Mar 21 07:11:36 2016
****************************************

Operating Conditions: nom_pvt   Library: lsi_10k
Wire Load Model Mode: top

  Startpoint: x[8] (input port clocked by clk)
  Endpoint: Term1_reg[25]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max
```

```
data arrival time                                                  47.27

clock clk (rise edge)                                 50.00        50.00
clock network delay (ideal)                            0.00        50.00
Term1_reg[25]/CP (FDS2L)                               0.00        50.00 r
library setup time                                    -1.60        48.40
data required time                                                 48.40
---------------------------------------------------------------------------
data required time                                                 48.40
data arrival time                                                 -47.27
---------------------------------------------------------------------------
slack (MET)                                                        1.13
```

Sin-Cos Unit Timing Report

System Performance Parameters

	Logarithm	Square-Root	Sin/Cos
Area	**75946 gates**	**6253 gates**	**2861 gates**
Critical Path	**99.20ns**	**34.20ns**	**47.27ns**

Logarithm unit has 48-bit input and Logarithm polynomials are of degree 2 hence it needs more number of gates and its critical path is maximum of all others. I have pipelined the Log Unit so that it can have minimum critical path possible. The throughput of the Log Unit is 6 clock cycles while the throughput of Square-Root Unit is 4 clock cycles and that of Sin/Cos Unit is 2 clock cycles. As the critical path of Log Unit is maximum, we have to take clock period of more than 99.20ns for the design.

Now that we have all the blocks of the design available, we can start integrating all the blocks to get AWGN samples. After pipelining and re-timing all the blocks to get best performance with least area, we get design throughput to be 16 clock cycles. The design is highly pipelined so we will get outputs every cycle after the frst output comes.

Following are the area and timing reports of AWGN generated from the tool –

```
****************************************
Report : area
Design : AWGN
Version: I-2013.12-SP1
Date   : Mon Mar 21 07:34:46 2016
****************************************

Information: Updating design information... (UID-85)
Warning: Design 'AWGN' contains 1 high-fanout nets. A fanout number of 1000 will be used for de
lay calculations involving these nets. (TIM-134)
Library(s) Used:

    lsi_10k (File: /opt/synopsys-2013/app/syn/libraries/syn/lsi_10k.db)

Number of ports:                         35
Number of nets:                        1232
Number of cells:                        589
Number of combinational cells:           55
Number of sequential cells:             526
Number of macros/black boxes:             0
Number of buf/inv:                       49
Number of references:                    16

Combinational area:              88341.000000
Buf/Inv area:                     3322.000000
Noncombinational area:           12614.000000
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                100955.000000
Total area:                 undefined
```

AWGN Area Report

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : AWGN
Version: I-2013.12-SP1
Date   : Mon Mar 21 07:34:47 2016
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: nom_pvt   Library: lsi_10k
Wire Load Model Mode: top

  Startpoint: LogIn_reg[1]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: L1/LogInSquare_reg[90]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max
```

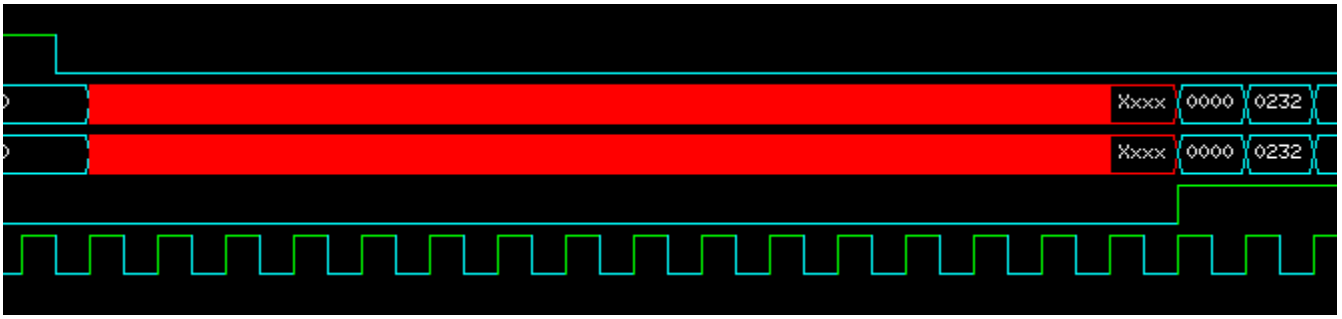
```
  data arrival time                                                  99.19

  clock clk (rise edge)                                  100.00     100.00
  clock network delay (ideal)                              0.00     100.00
  L1/LogInSquare_reg[90]/CP (FD1)                          0.00     100.00 r
  library setup time                                      -0.80      99.20
  data required time                                                 99.20
  ---------------------------------------------------------------------------
  data required time                                                 99.20
  data arrival time                                                 -99.19
  ---------------------------------------------------------------------------
  slack (MET)                                                        0.01
```

AWGN Timing Report

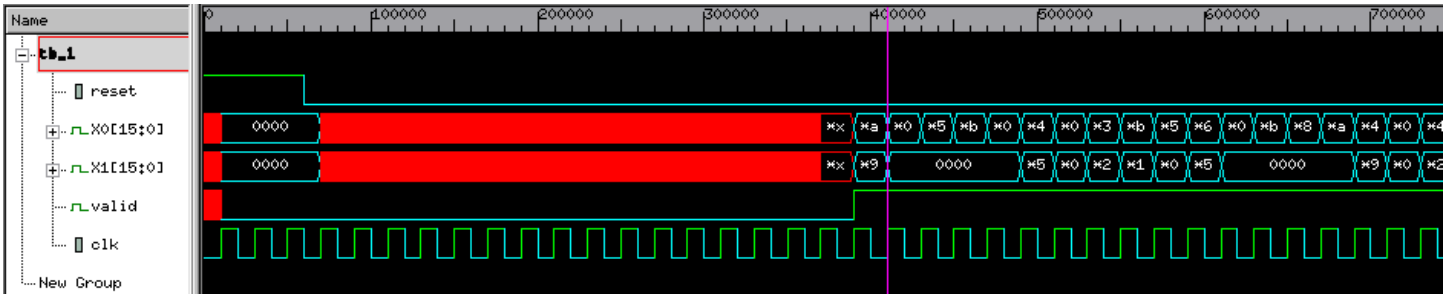
```
****************************************
Report : resources
Design : AWGN
Version: I-2013.12-SP1
Date   : Mon Mar 21 07:37:06 2016
****************************************

Resource Sharing Report for design AWGN in file
        /DCNFS/users/student/psharma/AWGN/Polyfit/AWGN.sv

===============================================================================
|          |              |            | Contained    |                      |
| Resource | Module       | Parameters | Resources    | Contained Operations |
===============================================================================
| r184     | DW02_mult    | A_width=15 |              | mult_68              |
|          |              | B_width=17 |              |                      |
| r186     | DW02_mult    | A_width=15 |              | mult_69              |
|          |              | B_width=17 |              |                      |
===============================================================================


Implementation Report
===============================================================================
|                  |                | Current          | Set              |
| Cell             | Module         | Implementation   | Implementation   |
===============================================================================
| mult_69          | DW02_mult      | csa              |                  |
| mult_68          | DW02_mult      | csa              |                  |
===============================================================================
```

AWGN Resource Sharing Report

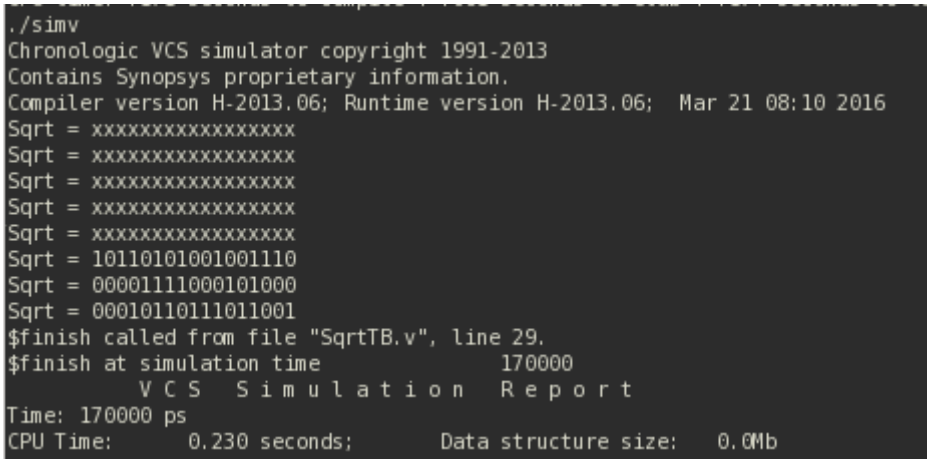
Verilog RTL Simulation Results



Throughput = 16 Clock Cycles



Simulation result of first few samples of AWGN



Simulation Result of Square-Root Unit

For the above figure, simulation of square root unit, first input was 31'hFFFF_FFFE which is equal to real value of 128. Square root of 128 is equal to 11.3137 which corresponds to 10110101000001010 in binary with Q13 format. The first output that we get has value of 10110101001001110. Thus, there is an error of last 6-bits for square root.

Thus, the required precision is not achieved with the polynomial coefficients that we have taken. To achieve more precise results, we can increase the degree of the polynomial or we can increase the number of segments of the function. Increasing number of segments of a function will cost us more memory than increasing the degree of polynomial.

The final result obtained from AWGN RTL design is compared with the AWGN MATLAB design. The difference between the two values was of 8 bits, which is too much error for the design specifications we have chosen.

MATLAB Design

```
% Tausworthe Seeds
s0 = fi(2, 0, 32, 0);
s1 = fi(7, 0, 32, 0);
s2 = fi(5, 0, 32, 0);
b0 = fi(0, 0, 32, 0);
b1 = fi(0, 0, 32, 0);
b2 = fi(0, 0, 32, 0);

%Taus Output
Taus = fi(0, 0, 32, 0);

%Taus Constants
a= fi(4294967294, 0, 32, 0);
b= fi(4294967288, 0, 32, 0);
c= fi(4294967280, 0, 32, 0);

%Log Input
    U0 = fi(0, 0, 48, 48);
%Sin/Cos Input
    U1 = fi(0, 0, 16, 16);

    e = fi(0, 0, 31, 24); % Log output
    f = fi(0, 0, 17, 13); % Sqrt output
    g0 = fi(0, 1, 16,15); % Sin output
    g1 = fi(0, 1, 16,15); % Cos output
    x0 = fi(0, 1, 16, 11); % AWGN output
    x1 = fi(0, 1, 16, 11); % AWGN output


for i=1:1000       %Get 1000 Sampl

    b0 = bitror(bitxor(bitrol(s0, 13), s0),19);
    s0 = bitxor(bitrol(bitand(s0,a),12),b0);

    b1 = bitror(bitxor(bitrol(s1, 2), s1),25);
    s1 = bitxor(bitrol(bitand(s1,b),4),b1);

    b2 = bitror(bitxor(bitrol(s2, 3), s2),11);
    s2 = bitxor(bitrol(bitand(s2,c),17),b2);

    Taus = bitxor(bitxor(s0,s1),s2);

    %URNG = Taus.bin;
    U0.bin = [Taus.bin, Taus.bin(17:32)];
    U1.bin = Taus.bin(1:16);

    %LOGARITHM BLOCK
    e.data = (-2) * log(U0.data);

    %SquareRoot Block
    f.data = sqrt(e.data);

    %Sin and Cos
    g0.data = sin(2*pi*U1.data);
    g1.data = cos(2*pi*U1.data);
```

```
    %Multiplication
    x0.data = f.data .* g0.data;
    x1.data = f.data .* g1.data;

    X(i,1) = x0.data;
    X0Binary (i,:) = x0.bin;
    X1Binary (i,:) = x1.bin;
    %X(i,2) = x1.data;
end

histogram(X)
```