**HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY**

**DEPARTMENT OF ELECTRONICS**

**--------------------------**

Logo, company name

Description automatically generated

**MICROPROCESSOR**

**REPORT FINAL PROJECT**

**AVALON MEMORY-MAPPED (AVALON-MM) INTERFACE SPECIFICATION**

**GROUP: TT01**

**SEMESTER: HK222**

**LECTURER: NGUYEN TUAN HUNG**

|  |  |  |
| --- | --- | --- |
| **STT** | **Student’s Name** | **Student’s ID** |
| 1 | Võ Viết Hưng | 2051076 |
| 2 | Nguyễn Đức Minh | 2051148 |

***HCM City, May 7th, 2023***

### **TABLE OF CONTENTS**

1. **ABSTRACT 1**
2. **CONCEPTS OF TRIGONOMETRIC IP** 
   1. **OVEVIEW 2**
   2. **CORDIC ALGORITHM 3**

**III. PREPARATION**

* 1. **ALGORITHM FOR HARDWARE IMPLEMENTATION 5**
  2. **CONTROL FLOW GRAPH 6**
  3. **DATA FLOW GRAPH 7**
  4. **DATAPATH DESIGN 8**
  5. **SINGLE-ASSIGNMENT 9**

**IV. RESULT**

* 1. **RTL VIEWER**
  2. **WAVEORM**

1. ***ABSTRACT***

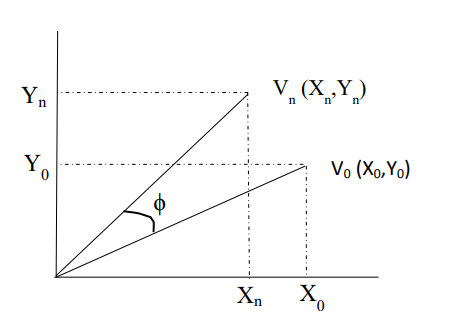
In this experiment, we attempt to create an IP for implementing Trigonometric function (sine, cosine and tangent) via using Avalon Memory Map protocol in fixed point. The inputs of the system include the values of Kn (Rotation algorithm gain constant) which is already imported inside the code structural, Angle (Input Signal), rst signal (reset), start signal and clock signal. All these values are used to control the CPU\_Nois2 of the DE10 standard kit and the output of the system are denoted as fn1, fn2 and Done signal (Flag).

1. ***CONCEPTS OF TRIGONOMETRIC IP***
   1. **OVERVIEW**

Calculation of sine and cosine of given angle is an essential requirement in many areas of real life. In signal processing, digital audio and high definition videos are based on sums of sine and cosine. Similarly, radio communication used in electronics is based on use of combinations of sine and cosine waves. Hardware provides highest speed due to its hardwired design. This report presents hardware design for calculating sine and cosine value of given angle using CORDIC algorithm with limited hardware usage.

As proposed by Behrooz Parhami, CORDIC is an iterative algorithm which is used to convert between polar and Cartesian coordinates, especially Trigonometric in order to compute trigonometric, linear and logarithmic functions. Due to limited hardware architecture, CORDIC requires least calculation and complicated modules as compared to the other such as LOOK-UP table, Taylor series expansion. Because for all calculation, CORDIC architecture just uses adder, subtractor and shifter circuits only.

* 1. **CORDIC ALGORITHM**

CORDIC works in two modes: Rotation mode and vectoring mode => In this project, CORDIC is used in rotation mode, which could compute the Cartesian coordinates of the target vector Vn by rotating the input vector V0 by an arbitrary angle ϕ.

**Fig 2.2.a. Rotation of vector V0 to Vn**

As shown in figure 2.2.a, V0 and Vn are initial vector and final vector repestively. Coordinates of final vector (Xn,Yn) can be calculated using following equations:

|  |
| --- |
| *Xn* = X0 cos ϕ – Y0 sin ϕ (1) |
| *Yn =* Y0 cos ϕ + X0 sin ϕ (2) |

These equations can be rewritten as

|  |
| --- |
| *Xn* = cos ϕ [X0 – Y0 tan ϕ] (3) |
| *Yn =* cos ϕ [Y0 + X0 tan ϕ] (4) |

Now if the angle of rotation is restricted to be such that tan ϕ = ±2*-i*, multiplication by tangent term can be reduced to simple shift for fixed point operations. If the angle of rotation (ϕ) is represented as Z0 then after rotating for i times:

|  |
| --- |
| *Zn* ≈ 0 (5) |

Considering eq. (3) and (4), if decision (di) at each iteration, i is which direction to rotate, then the term cos ϕ is constant (because cos ϕi = cos(-ϕi)). While the cos ϕi is independent of the direction of rotation, then iterative rotation can be represented as:

|  |
| --- |
| *Xi+1* = Ki [Xi – Yi di 2-i] (6) |
| *Yi+1* = Ki [Yi + Xi di 2-i] (7) |

Where *Ki* = cos(tan-1 2*-i*) = & *di* ϵ {-1,1}

Removing the scale constant from iterative equations yields a shift-add algorithm for vector rotation. The product of Ki’s can be treated as part of system processing gain. The rotation algorithm has a gain Kn of approximately 1.64676. The exact gain depends on number of rotations and can be calculated using relation

Therefore, if total gain for n iteration be Kn then eq. (1) and (2) can be rewritten as

|  |
| --- |
| *Xn* ≈ Kn [X0 cos Z0 – Y0 sin Z0] (8) |
| *Yn* ≈ Kn [Y0 cos Z0 + X0 sin Z0] (9) |

Let if we define a vector where Y0 = 0 then equations (8) and (9) become

|  |
| --- |
| *Xn* ≈ Kn X0 cos Z0 (8) |
| *Yn* ≈ Kn X0 sin Z0 (9) |

By setting X0 = we can calculate the value of sine and cosine of input angle Z0 as Y and X coordinate respectively of the final vector Vn.

1. ***PREPARATION***
   1. **ALGORITHM FOR HARDWARE IMPLEMENTATION**

We devide the algorithm into two similar code C for easy considering its graph:

int Cos\_func(int Z[], int Y[], int atan\_table[]){ (1)

int Z\_sign;

static int Cos\_res;

int X[16];

int Y\_shr;

for (int i = 0; i < 15; i++) {

(2) (3) (4)

Y\_shr = Y[i] >> i; (5)

Z\_sign = Z[i] >> 31; (6)

switch (Z\_sign) { (7)

case 1:

X[i + 1] = X[i] + Y\_shr; (8)

Z[i + 1] = Z[i] + atan\_table[i]; (9)

break;

case 0:

X[i + 1] = X[i] - Y\_shr; (10)

Z[i + 1] = Z[i] - atan\_table[i]; (11)

break;

}

}

return Cos\_res = X[15]; (12)

}

int Sin\_func(int Z[], int X[], int atan\_table[]){ (1)

int Z\_sign;

static int Sin\_res;

int Y[16];

int X\_shr;

for (int i = 0; i < 15; i++) {

(2) (3) (4)

X\_shr = X[i] >> i; (5)

Z\_sign = Z[i] >> 31; (6)

switch (Z\_sign) { (7)

case 1:

Y[i + 1] = Y[i] - X\_shr; (8)

Z[i + 1] = Z[i] + atan\_table[i]; (9)

break;

case 0:

Y[i + 1] = Y[i] + X\_shr; (10)

Z[i + 1] = Z[i] - atan\_table[i]; (11)

break;

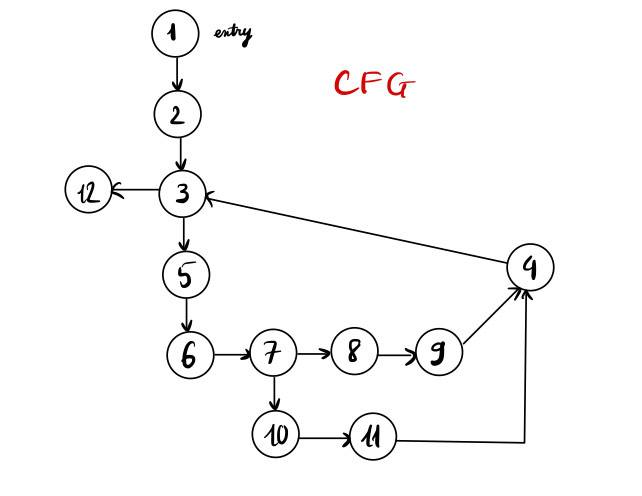
}

}

return Sin\_res = Y[15]; (12)

}

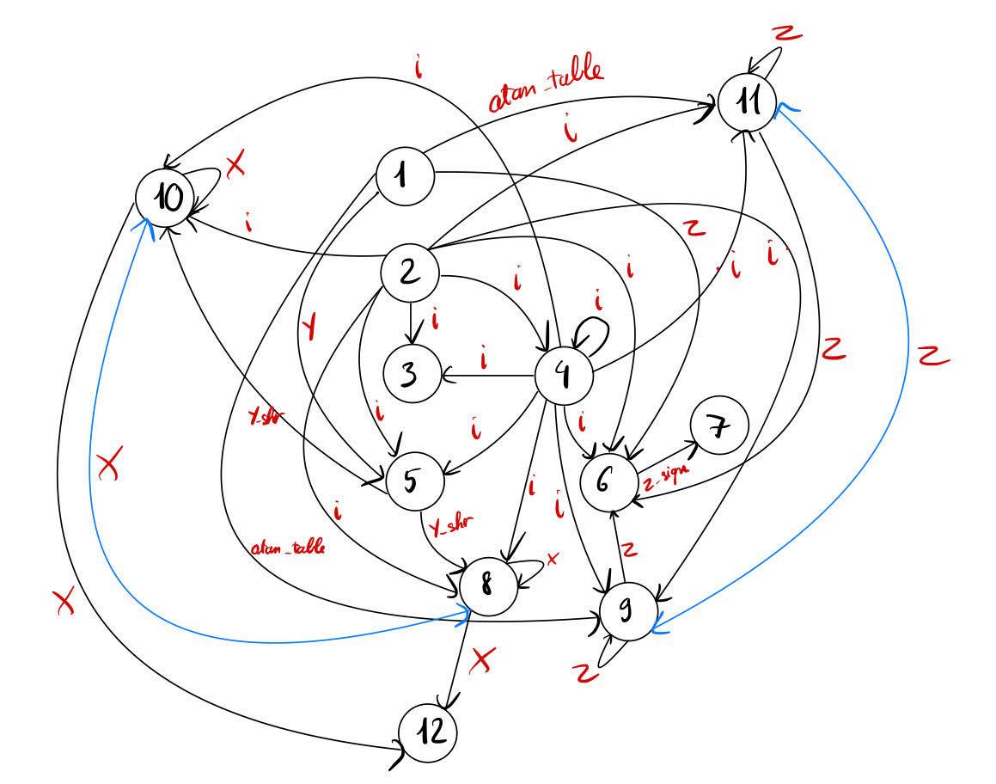
* 1. **CONTROL FLOW GRAPH:**

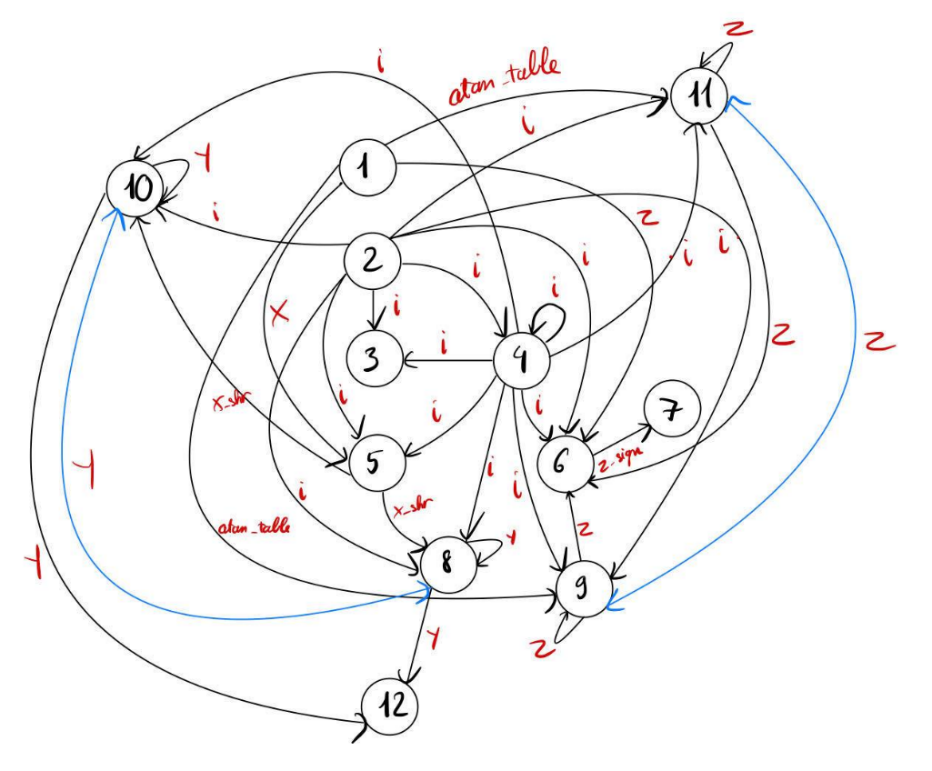
For both Sin\_func and Cos\_func:

In which:  
Operation 1: int Z[], int X[], int Y[], int atan\_table[];  
Operation 2: int i = 0;  
Operation 3: i < 15 ?  
Operation 4: i++;  
Operation 5: X\_shr = X[i] >> i; Y\_shr = Y[i] >> i;  
Operation 6: Z\_sign = Z[i] >> 31;   
Operation 7: Z\_sign?  
Operation 8: Y[i + 1] = Y[i] - X\_shr; or X[i + 1] = X[i] + Y\_shr;  
Operation 9: Z[i + 1] = Z[i] + atan\_table[i];

Operation 10: Y[i + 1] = Y[i] + X\_shr; or X[i + 1] = X[i] - Y\_shr;  
Operation 11: Z[i + 1] = Z[i] - atan\_table[i];  
Operation 12: return Cos\_res; or return Sin\_res;

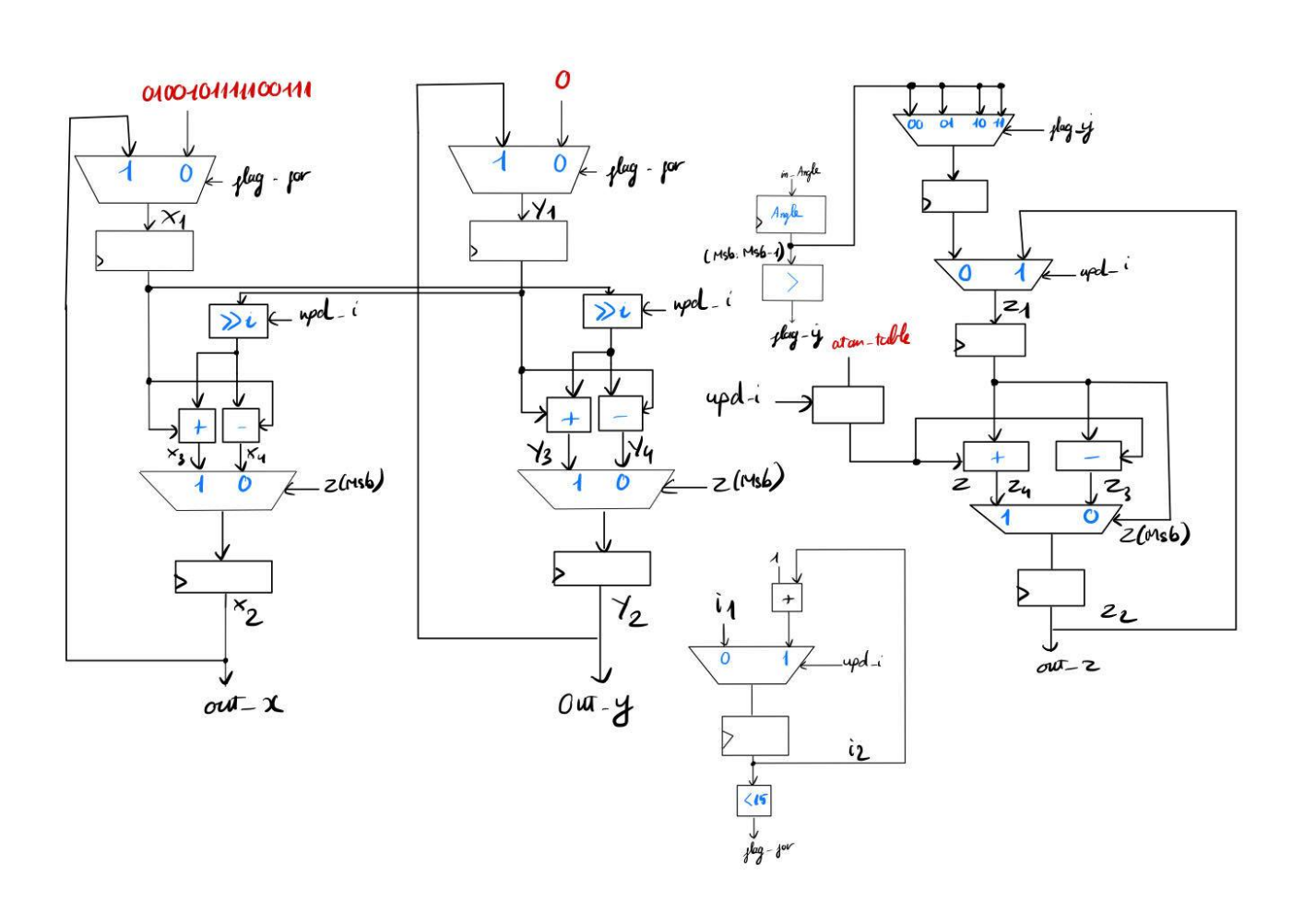
* 1. **DATA FLOW GRAPH:**

****For Cos\_func:

For Sin\_func:

* 1. **DATAPATH DESIGN:**

We co-oporate both func into one datapath in order to easy follow hear,



* 1. **SINGLE-ASSIGNMENT:**

For Cos\_func:

int Cos\_func(int Z1[], int Z2[], int Y[], int atan\_table[]){

int Z\_sign;

int Z3[16], Z4[16];

static int Cos\_res;

int X1[16], X2[16], X3[16], X4[16];

int Y\_shr;

for (int i1 = 0; merge(i1,i2) < 15;int i2 = merge(i1,i2)++) {

Y\_shr = Y[merge(i1,i2)] >> merge(i1,i2);

Z\_sign = merge(Z1[merge(i1,i2)],Z2[merge(i1,i2)],Z3[merge(i1,i2)],Z4[merge(i1,i2)]) >> 31;

switch (Z\_sign){

case 1:

X2[merge(i1,i2) + 1] = merge(X1[merge(i1,i2)], X2[merge(i1,i2)],X3[merge(i1,i2)],X4[merge(i1,i2)]) + Y\_shr;

Z3[merge(i1,i2) + 1] = merge(Z1[merge(i1,i2)], Z2[merge(i1,i2)],Z3[merge(i1,i2)],Z4[merge(i1,i2)]) + atan\_table[merge(i1,i2)];

break;

case 0:

X3[merge(i1,i2) + 1] = merge(X1[merge(i1,i2)], X2[merge(i1,i2)],X3[merge(i1,i2)],X4[merge(i1,i2)]) - Y\_shr;

Z4[merge(i1,i2) + 1] = merge(Z1[merge(i1,i2)], Z2[merge(i1,i2)],Z3[merge(i1,i2)],Z4[merge(i1,i2)]) - atan\_table[merge(i1,i2)];

break;

}

}

return Cos\_res = merge(X1[15], X2[15], X3[15], X4[15]);

}

For Sin\_func:

int Sin\_func(int Z1[], int Z2[], int X[], int atan\_table[]){

int Z\_sign;

int Z3[16], int Z4[16];

static int Sin\_res;

int Y1[16], Y2[16], Y3[16], Y4[16];

int X\_shr;

for (int i1 = 0; merge(i1,i2) < 15;int i2 = merge(i1,i2)++) {

X\_shr = X[merge(i1,i2)] >> merge(i1,i2);

Z\_sign = merge(Z1[merge(i1,i2)],Z2[merge(i1,i2)],Z3[merge(i1,i2)],Z4[merge(i1,i2)]) >> 31;

switch (Z\_sign){

case 1:

Y2[merge(i1,i2) + 1] = merge(Y1[merge(i1,i2)], Y2[merge(i1,i2)],Y3[merge(i1,i2)],Y4[merge(i1,i2)]) + X\_shr;

Z3[merge(i1,i2) + 1] = merge(Z1[merge(i1,i2)], Z2[merge(i1,i2)],Z3[merge(i1,i2)],Z4[merge(i1,i2)]) + atan\_table[merge(i1,i2)];

break;

case 0:

Y3[merge(i1,i2) + 1] = merge(Y1[merge(i1,i2)], Y2[merge(i1,i2)],Y3[merge(i1,i2)],Y4[merge(i1,i2)]) - Y\_shr;

Z4[merge(i1,i2) + 1] = merge(Z1[merge(i1,i2)], Z2[merge(i1,i2)],Z3[merge(i1,i2)],Z4[merge(i1,i2)]) - atan\_table[merge(i1,i2)];

break;

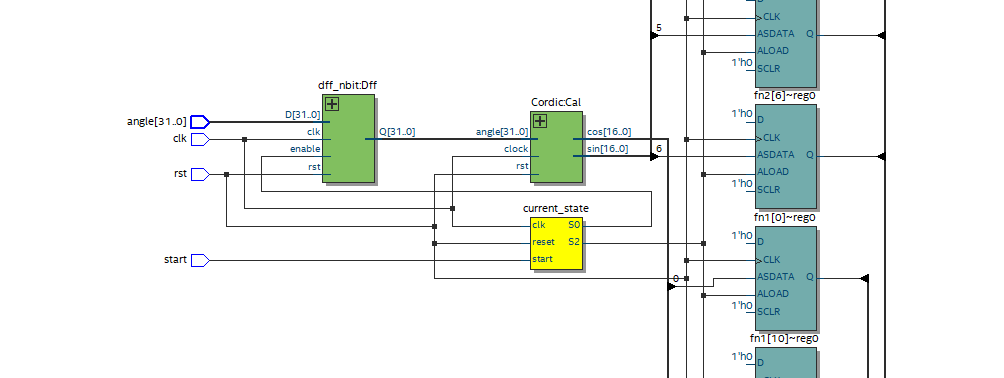
}

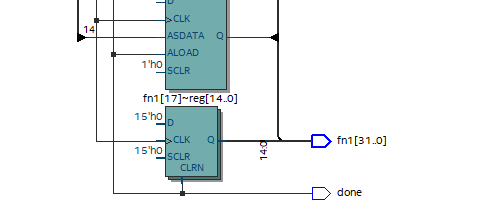
}

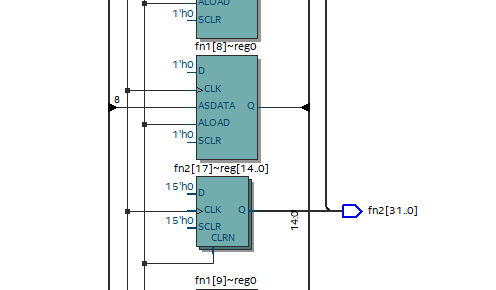
return Sin\_res = merge(Y1[15], Y2[15], Y3[15], Y4[15]);

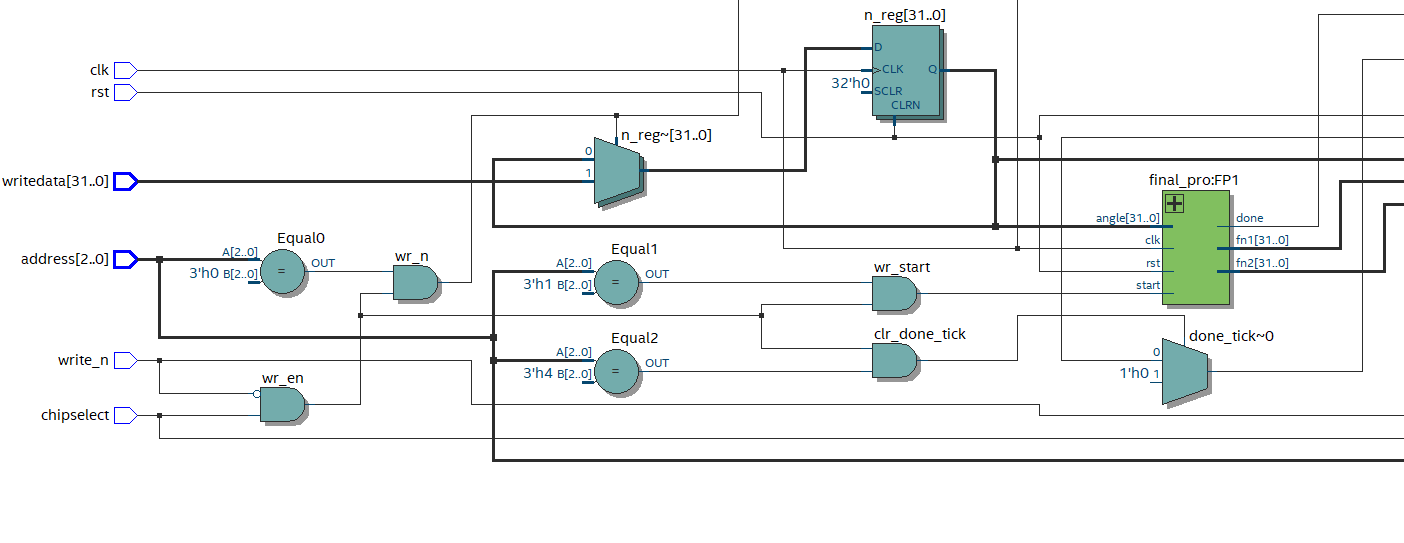
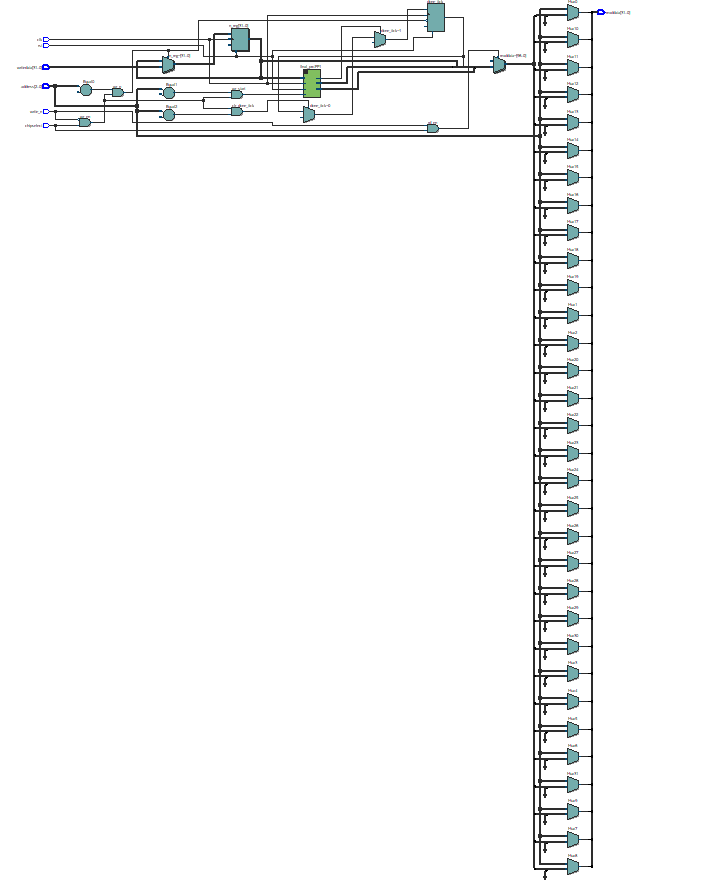
}

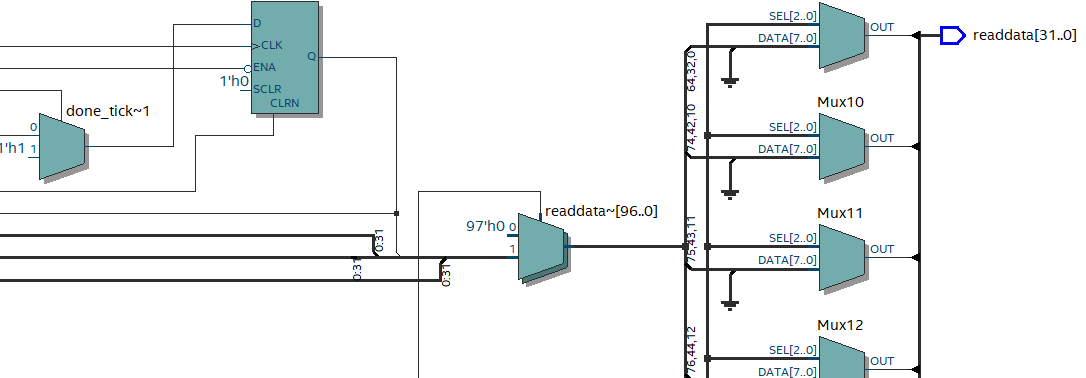
1. ***RESULT:***
   1. **RTL VIEWER:**

****

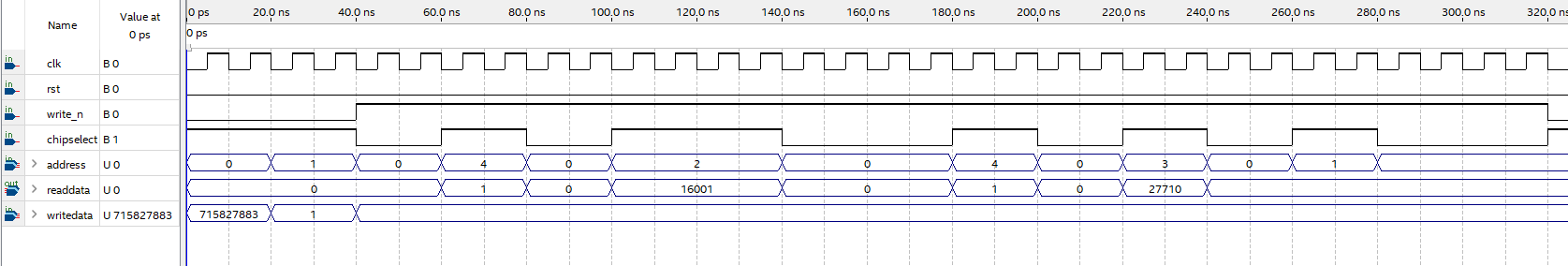
****

****

****

****

* 1. **WAVEFFORM:**

******

While write\_n signal is active low => enable the processor to write the data into reg (regn ) => value of angle 60° is insert to address 1 (the register store value of input)

Then we enable the register in address 2 which allowed the processor to load the data of angle into CORDIC module to calculate.

After 2 clock cycle, we check the register which represent for DONE flag to ensure wether the processor are done or not.

When it done, we active high chip select, write\_n is also active high and write into address value 2 in the purpose to read the data in the register 2 (cos value) and it give back the value 16001 which is nearly the same as the function cos(60°) .

***\_\*\_THE END\_\*\_***