CORDIC

Task to be done:

Implementat CORDIC operation using HLS

Cordic: Coordinate Rotation DIgital Computer (CORDIC) is an efficient technique to calculate trigonometric, hyperbolic, and other mathematical functions.It is a digit-by-digit algorithm that produces one output digit per iteration.This allows us to tune the accuracy of the algorithm to the application requirements; additional iterations produce a more precise output result.

Accuracy is another common design evaluation metric alongside performance and resource usage.CORDIC performs simple computations using only addition, subtraction, bit shifting, and table lookups, which are efficient to implement in FPGAs and more generally in hardware.

Background

The main objective of CORDIC is to improve the performance of set of vector rotations in a two dimensional plane.By overlaying these rotations with some simple control decisions, we can perform a variety of fundamental operations, e.g., trigonometric, hyperbolic, and logarithmic functions, real and complex multiplication, and matrix decompositions and factorizations. CORDIC has been used in a wide range of applications including signal processing, robotics, communications, and many scientific computations. CORDIC is commonly used in FPGA design since it has a small resource usage.

Calculation:

DATE: / /
PAGE

We perform an iterative series of rotation, each of the subsequent rotation uses an increasingly smaller angle, which means that every iteration adds a bit more-precision to the output value.

In two dimension the rotation matrix is

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

The iterative algorithm rotates a vector v to some target of angle which depends on the function that CORDIC is performing. One rotion is a matrix vector multiplication in the form $v_i = R_i \cdot V_{i-1}$. Thus in each iteration of the CORDIC we perform the following operation to perform one rotation which is the matrix vector multiply

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$\therefore \; x_i = x_{i-1}\cos\theta - y_{i-1}\sin\theta.$$

$$y_i = x_{i-1}\sin\theta + y_{i-1}\cos\theta$$

PAGE

The main thing to note that the hardware just swap x and y by taking 2's complement of y which is actually an adder to ease the computation

So to make the computation easy we can rewrite our rotation matrix for general θ as :-

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{bmatrix} \left(\frac{1}{\sqrt{1+\tan^2\theta}} \right.$$

Here we take an assumption that $\tan(\theta_i) = 2^{-i}$

$$\therefore R(\theta_i) = \begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix} \frac{1}{\sqrt{1+2^{-2i}}}$$

If the rotation is -ve

$$R(\theta_i) = \begin{bmatrix} 1 & 2^{-i} \\ -2^{-i} & 1 \end{bmatrix} \frac{1}{\sqrt{1+2^{-2i}}}$$

DATE: / /
PAGE

Let $K_i = \frac{1}{\sqrt{1+2^{-2i}}}$ which is the scalibity.

For the series of rotation the scalibity constant (K_i) multiply and the resultant constant is :-

$$\prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}} = K(n)$$

$$\lim_{n \to \infty} K(n) \cong 0.607$$

K can be precompiled and stored or if we do sufficient number of iteration / rotation, then it tends to its limiting value.

To get θ_i we use $\theta_i = \tan^{-1}(2^{-i})$

According to the value of ϕ, we can perform +ve rotation if required angle θ is $> \phi$ or a -ve rotation if $\theta < \phi$.

Example Table for Rotating angle,Scaling factor and CORDIC gain

i	2^{-i}	Rotating Angle	Scaling Factor	CORDIC Gain
0	1.0	45.000°	1.41421	1.41421
1	0.5	26.565°	1.11803	1.58114
2	0.25	14.036°	1.03078	1.62980
3	0.125	7.125°	1.00778	1.64248
4	0.0625	3.576°	1.00195	1.64569
5	0.03125	1.790°	1.00049	1.64649
6	0.015625	0.895°	1.00012	1.64669

Now we find cos(45) and sin(45) with CORDIC operation using HLS and optimize it accordingly.

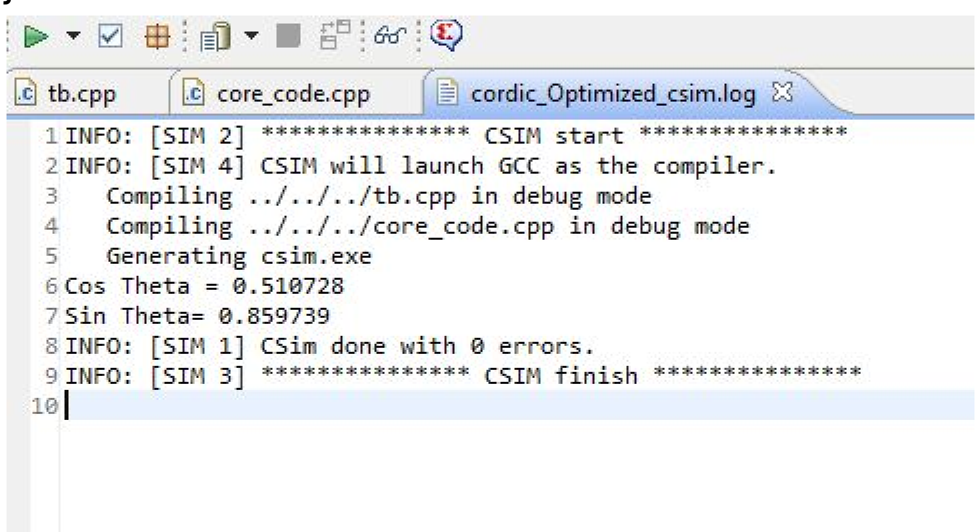
Cordic code:

```
#include <math.h>

// The cordic phase array holds the angle for the current rotation

void cordic(float theta, float cordic_phase[6] ,float angle[2])
{
// Set the initial vector that we will rotate

        float current_cos = angle[0];
        float current_sin = angle[1];

// This loop iteratively rotates the initial vector to find the
```

```
// sine and cosine values corresponding to the input theta angle

for (int j = 0; j < 6; j++) {
// Multiply previous iteration by 2ˆ(−j). This is equivalent to
// a right shift by j on a fixed−point number.

        float cos_shift = current_cos*pow(0.5,j);
        float sin_shift = current_sin*pow(0.5,j);

// Determine if we are rotating by a positive or negative angle
if(theta >= 0) {
// Perform the rotation
current_cos = current_cos - sin_shift;
current_sin = current_sin + cos_shift;
// Determine the new theta
theta =  cordic_phase[j];

} else {

current_cos = current_cos + sin_shift;
current_sin = current_sin - cos_shift;
theta = cordic_phase[j];
}
}

// Set the final sine and cosine values
angle[1] = current_sin;
angle[0] = current_cos;
}
```

Testbench

```
#include<stdio.h>

void cordic(float theta,float codic_phase[6],float angle[2]);

int main(){
	float angle[2] ={1,0};
	float theta = 45.0;
	float cordic_phase[6] =
{26.565,-14.036,7.125,-3.576,-1.79,0.895};
	cordic(theta,cordic_phase,angle);

	printf("Cos Theta = %f\n",(float)angle[0]*0.60735);
	printf("Sin Theta= %f\n",(float)angle[1]*0.60735);


	return 0;

}
```




```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../tb.cpp in debug mode
   Compiling ../../../core_code.cpp in debug mode
   Generating csim.exe
Cos Theta = 0.510728
Sin Theta= 0.859739
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Synthesis Report of the Code:

Synthesis Report for 'cordic'

General Information

Date: Mon Nov 23 17:21:18 2020
Version: 2019.1 (Build 2552052 on Fri May 24 15:28:33 MDT 2019)
Project: cordic_project
Solution: solution1
Product family: zynq
Target device: xc7z020-clg484-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.514	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
272	272	272	272	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	118	-
FIFO	-	-	-	-	-
Instance	17	67	5688	9052	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	305	-
Register	-	-	633	-	-
Total	17	67	6321	9475	0
Available	280	220	106400	53200	0
Utilization (%)	6	30	5	17	0

Interface

Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	cordic	return value
ap_rst	in	1	ap_ctrl_hs	cordic	return value
ap_start	in	1	ap_ctrl_hs	cordic	return value
ap_done	out	1	ap_ctrl_hs	cordic	return value
ap_idle	out	1	ap_ctrl_hs	cordic	return value
ap_ready	out	1	ap_ctrl_hs	cordic	return value
theta	in	32	ap_none	theta	scalar
cordic_phase_address0	out	3	ap_memory	cordic_phase	array
cordic_phase_ce0	out	1	ap_memory	cordic_phase	array
cordic_phase_q0	in	32	ap_memory	cordic_phase	array
angle_address0	out	1	ap_memory	angle	array
angle_ce0	out	1	ap_memory	angle	array
angle_we0	out	1	ap_memory	angle	array
angle_d0	out	32	ap_memory	angle	array
angle_q0	in	32	ap_memory	angle	array
angle_address1	out	1	ap_memory	angle	array
angle_ce1	out	1	ap_memory	angle	array
angle_we1	out	1	ap_memory	angle	array
angle_d1	out	32	ap_memory	angle	array
angle_q1	in	32	ap_memory	angle	array

Here the latency is very high so as the utilization of flip flops and LUTs.

Optimization of Cordic:
To optimize the above code we include "*ap_fixed*" instead of "*float*".Also for the operation 2^(-j) we can use bit shift instead of math:power operation because we are using fixed numbers of bits.

To remove the ap_start ,ap_done,ap_idle , ap_ready we can use axi interface.

Optimized Code:

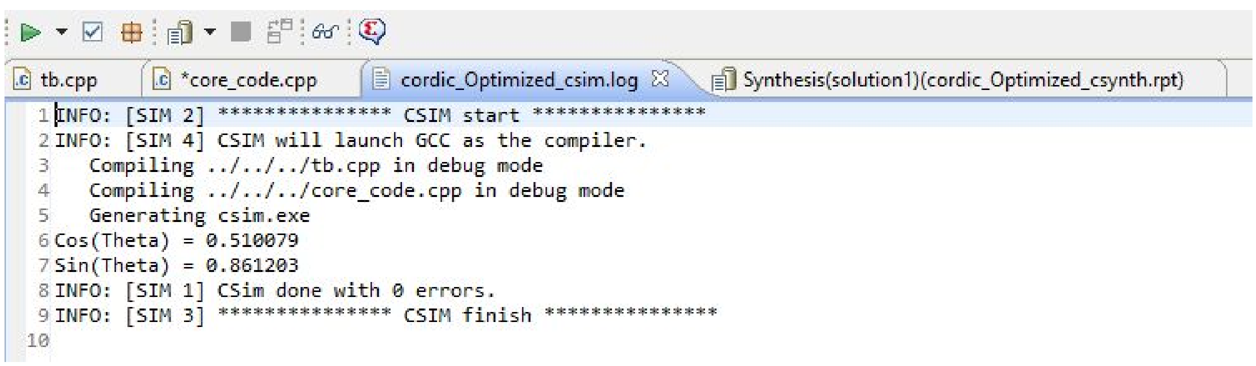
```
#include <ap_fixed.h>
typedef ap_fixed<16,7> fix;

void cordic_Optimized (fix theta,fix cordic_phase[6],fix angle[2]){

		#pragma HLS INTERFACE ap_ctrl_none port = return
		#pragma HLS INTERFACE axis register both port =
theta
		#pragma HLS INTERFACE axis register both port =
cordic_phase

	fix current_cos =angle[0];
	fix current_sin = angle[1];

	for(int j=0;j<6;j++){

		#pragma HLS_UNROLL		// for loop is used as a
sequential

		fix cos_shift = current_cos >>j;
```

```
        fix sin_shift = current_sin >> j;


    if(theta >= 0) {
    // Perform the rotation
    current_cos = current_cos - sin_shift;
    current_sin = current_sin + cos_shift;
    // Determine the new theta
    theta =  cordic_phase[j];

    } else {
    // Perform the rotation
    current_cos = current_cos + sin_shift;
    current_sin = current_sin - cos_shift;
    // Determine the new theta
    theta = cordic_phase[j];

    }

}
    angle[1]= current_sin;
    angle[0]= current_cos;
}
```

Testbench

```
#include<stdio.h>
#include<ap_fixed.h>
typedef ap_fixed<16,7> fix;
```

```
void cordic_Optimized(fix theta,fix codic_phase[6],fix angle[2]);
int main(){
	fix angle[2] ={1,0};
	fix theta = 45.0;
	fix cordic_phase[6] =
{(fix)26.565,(fix)-14.036,(fix)7.125,(fix)-3.576,(fix)-1.79,(fix)0.895};
	cordic_Optimized(theta,cordic_phase,angle);

	printf("Cos(Theta) = %f\n",(float)angle[0]*0.60735);
	printf("Sin(Theta) = %f\n",(float)angle[1]*0.60735);



	return 0;

}
```




```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../tb.cpp in debug mode
   Compiling ../../../core_code.cpp in debug mode
   Generating csim.exe
Cos(Theta) = 0.510079
Sin(Theta) = 0.861203
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Synthesis Report of code

tb.cpp | *core_code.cpp | cordic_Optimized_csim.log | Synthesis(solution1)(cordic_Optimized_csynth.rpt)

Synthesis Report for 'cordic_Optimized'

General Information

Date:	Sun Nov 29 18:04:59 2020
Version:	2019.1 (Build 2552052 on Fri May 24 15:28:33 MDT 2019)
Project:	cordic_project
Solution:	solution1
Product family:	zynq
Target device:	xc7z020-clg484-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.880	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
8	8	8	8	none

Detail

- Instance
- Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	213	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	165	-
Register	-	-	142	-	-
Total	0	0	142	378	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

Interface

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_none	cordic_Optimized	return value
ap_rst_n	in	1	ap_ctrl_none	cordic_Optimized	return value
theta_V_TDATA	in	16	axis	theta_V	scalar
theta_V_TVALID	in	1	axis	theta_V	scalar
theta_V_TREADY	out	1	axis	theta_V	scalar
cordic_phase_V_TDATA	in	16	axis	cordic_phase_V	pointer
cordic_phase_V_TVALID	in	1	axis	cordic_phase_V	pointer
cordic_phase_V_TREADY	out	1	axis	cordic_phase_V	pointer
angle_V_address0	out	1	ap_memory	angle_V	array
angle_V_ce0	out	1	ap_memory	angle_V	array
angle_V_we0	out	1	ap_memory	angle_V	array
angle_V_d0	out	16	ap_memory	angle_V	array
angle_V_q0	in	16	ap_memory	angle_V	array
angle_V_address1	out	1	ap_memory	angle_V	array
angle_V_ce1	out	1	ap_memory	angle_V	array
angle_V_we1	out	1	ap_memory	angle_V	array
angle_V_d1	out	16	ap_memory	angle_V	array
angle_V_q1	in	16	ap_memory	angle_V	array

Here the latency decreased in a good significant amount after optimization,Timing reduced and the use of flip flops and LUTs also decreased with large numbers,also this time it is not using DSP48E and BRAM_18K.