EE521800

Application Acceleration with
High-Level Synthesis

Lab#B
CORDIC

ID:110064513

1. Algorithm

- Derived from the general rotation transform

$$x' = x\cos(\varphi) - y\sin(\varphi)$$
$$y' = x\cos(\varphi) + y\sin(\varphi)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \cos(\varphi)\begin{bmatrix} 1 & -\tan(\varphi) \\ +\tan(\varphi) & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

- Iterative rotation transform

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = K_i \begin{bmatrix} 1 & -d_i 2^{-i} \\ +d_i 2^{-i} & 1 \end{bmatrix}\begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad K_i = \cos\left(\tan^{-1}(2^{-i})\right) = \frac{1}{\sqrt{1+2^{-2i}}}$$

- The product K_i approaches 0.6307 when the number of iteration goes to infinity. The algorithm has a gain depends on the number of iterations by

$$A_n = \prod_n \sqrt{1+2^{-2i}}$$

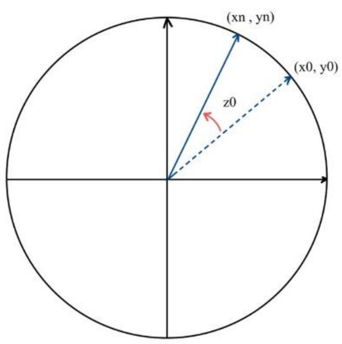
- Rotation mode (Given x_0, y_0 and desired rotation angle z_0)

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$

where $d_i = -1$ when $z_i < 0$，$d_i = +1$ when $z_i > 0$

$$x_n = A_n(x_0\cos(z_0) - y_0\sin(z_0))$$
$$y_n = A_n(x_0\sin(z_0) + y_0\cos(z_0))$$
$$z_n = 0$$
$$A_n = \prod_n \sqrt{1+2^{-2i}}$$

Final result of rotation mode is the vector position (x_n, y_n) with zero angle accumulator



➢ Vector mode (Given x_0 and y_0)

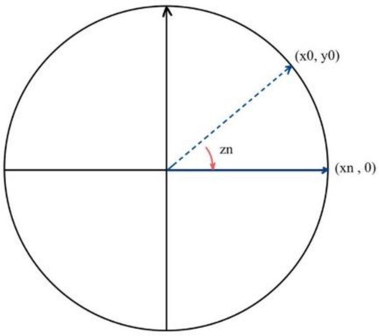
$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$

where $d_i = +1$ when $y_i < 0$，$d_i = -1$ when $y_i > 0$

$$\mathrm{x_n} = \mathrm{A_n}\sqrt{\mathrm{x_0}^2 + \mathrm{y_0}^2}$$
$$\mathrm{z_n} = \mathrm{z_0} + \tan^{-1}(\mathrm{y_0}/\mathrm{x_0})$$
$$\mathrm{y_n} = 0$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

Final result of vector mode is rotated angle with x-coordinate aligned vector



2. Explain the C++ code

➢ cordic.cpp

```
#include "cordic.h"
void cordic(THETA_TYPE theta, COS_SIN_TYPE &s, COS_SIN_TYPE &c)
{
```

a.

```
    #pragma HLS INTERFACE s_axilite port=return
    #pragma HLS INTERFACE s_axilite port=theta
    #pragma HLS INTERFACE s_axilite port=s
    #pragma HLS INTERFACE s_axilite port=c

    // Set the initial vector that we will rotate
    // current cos = I; current sin = Q
```

b.

```
    COS_SIN_TYPE current_cos = 0.60735;
    COS_SIN_TYPE current_sin = 0.0;
```

c.

```
    COS_SIN_TYPE factor = 1.0;
    // 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 < NUM_ITERATIONS; j++) {
        // Determine if we are rotating by a positive or negative angle
```

d.

```
        int sigma = (theta < 0) ? -1 : 1;

        // Multiply previous iteration by 2^(-j)
        COS_SIN_TYPE cos_shift = current_cos * sigma * factor;
        COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

        // Perform the rotation
        current_cos = current_cos - sin_shift;
        current_sin = current_sin + cos_shift;

        // Determine the new theta
        theta = theta - sigma * cordic_phase[j];
        factor = factor / 2;
    }

    // Set the final sine and cosine values
```

e.

```
    s = current_sin;  c = current_cos;
}
```

這個 code 所做的事情是使用 rotation mode 的性質來計算正弦與餘弦函數的數值，因 rotation mode 的輸出如下列所示：

$$x_n = A_n(x_0\cos(z_0) - y_0\sin(z_0))$$
$$y_n = A_n(x_0\sin(z_0) + y_0\cos(z_0))$$
$$z_n = 0$$
$$A_n = \prod_n \sqrt{1+2^{-2i}}$$

所以只要將 x_0 設為 1，y_0 設為 0 即可算出：

$$x_n = A_n\cos(z_0) \qquad y_n = A_n\sin(z_0)$$

a. 使用 axi-lite protocol 來傳送 data 與 control signals。

b. 若一開始就將 x_0 設為 $0.6735 \approx \frac{1}{A_n}$，就可使計算出的結果直接為所要求的正弦與餘弦值，不用再多除一個 A_n。

c. Initial 的 factor 指得是 $\tan 45^{\circ} = 2^{-i}$ i = 0

d. 即在計算下列式子：

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$

where $d_i = -1$ when $z_i < 0$，$d_i = +1$ when $z_i > 0$

e. 在所有 iteration 執行完後記下最終的正弦與餘弦值作為 output。

➢ cordic.h

```
#ifndef CORDIC_H
#define CORDIC_H
#include "ap_fixed.h"

typedef unsigned int UINTYPE_12;
typedef ap_fixed<12,2> THETA_TYPE;
typedef ap_fixed<12,2> COS_SIN_TYPE;
const int NUM_ITERATIONS=20;

const int NUM_DEGREE=90;
static THETA_TYPE cordic_phase[64]={0.78539816339744828000,0.46364760900080609000,0.24497866312686414000,0.12435499454676144000,0.06241880999595735000,
                                    0.03123983343026827700,0.01562372862047683100,0.00781234106010111110,0.00390623013196697180,0.00195312251647881880,
                                    0.00097656218955931946,0.00048828121119489829,0.00024414062014936177,0.00012207031189367021,0.00006103515617420877,
                                    0.00003051757811552610,0.00001525878906131576,0.00000762939453110197,0.00000381469726560650,0.00000190734863281019,
                                    0.00000095367431640596,0.00000047683715820309,0.00000023841857910156,0.00000011920928955078,0.00000005960464477539,
                                    0.00000002980232238770,0.00000001490116119385,0.00000000745058059692,0.00000000372529029846,0.00000000186264514923,
                                    0.00000000093132257462,0.00000000046566128731,0.00000000023283064365,0.00000000011641532183,0.00000000005820766091,
                                    0.00000000002910383046,0.00000000001455191523,0.00000000000727595761,0.00000000000363797881,0.00000000000181898940,
                                    0.00000000000090949470,0.00000000000045474735,0.00000000000022737368,0.00000000000011368684,0.00000000000005684342,
                                    0.00000000000002842171,0.00000000000001421085,0.00000000000000710543,0.00000000000000355271,0.00000000000000177636,
                                    0.00000000000000088818,0.00000000000000044409,0.00000000000000022204,0.00000000000000011102,0.00000000000000005551,
                                    0.00000000000000002776,0.00000000000000001388,0.00000000000000000694,0.00000000000000000347,0.00000000000000000173,
                                    0.00000000000000000087,0.00000000000000000043,0.00000000000000000022,0.00000000000000000011};

void cordic(THETA_TYPE theta, COS_SIN_TYPE &s, COS_SIN_TYPE &c);
#endif
```

a. 設定變數的形式為 12bits 且整數位為 2bits 的 fixed point。

b. 設定 cordic 所要執行的 iteration 數為 20，iteration 數越高所計算出來的數值越準確，但相對的 cost 也會上升。

c. 將 $\tan^{-1} 2^{-i}$ 寫成 LUT 在每個 iteration 取出對應的值去做加減。

➢ Testbench

```
#include <math.h>
#include"cordic.h"
#include <stdio.h>
#include <stdlib.h>
using namespace std;

double abs_double(double var){
    if ( var < 0)
    var = -var;
    return var;
}
int main(int argc, char **argv)
{
    FILE *fp;

    COS_SIN_TYPE s;         //sine
    COS_SIN_TYPE c;         //cos
    THETA_TYPE radian;      //radian versuin of degree

    //zs=sin, zc=cos using math.h in VivadoHLS
    double zs, zc;          // sine and cos values calculated from math.

    //Error checking
    double Total_Error_Sin=0.0;
    double Total_error_Cos=0.0;
    double error_sin=0.0, error_cos=0.0;

    fp=fopen("C:/Users/user/Desktop/out.dat","w");
    for(int i=1;i<NUM_DEGREE;i++) {
        radian = i*M_PI/180;
        cordic(radian, s, c);
        zs = sin((double)radian);
        zc = cos((double)radian);
        error_sin=(abs_double((double)s-zs)/zs)*100.0;
        error_cos=(abs_double((double)c-zc)/zc)*100.0;
        Total_Error_Sin=Total_Error_Sin+error_sin;
        Total_error_Cos=Total_error_Cos+error_cos;
        fprintf(fp, "degree=%d, radian=%f, cos=%f, sin=%f\n", i, (double)radian, (double)c, (double)s);
    }

    fclose(fp);
    printf ("Total_Error_Sin=%f, Total_error_Cos=%f, \n", Total_Error_Sin, Total_error_Cos);
    return 0;
}
```

a. 將想要計算的角度轉換為徑度並放入 kernel 做計算。

b. 計算 cordic 所計算出的結果與真實數值的誤差。

3. Timing and utilization

➢ Timing

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.188 ns	2.70 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
62	62	0.620 us	0.620 us	63	63	no

Detail

Instance

Loop

可看到計算所需要的 latency 為 62 個 clock cycle。

➢ Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	-	0	175	-
FIFO	-	-	-	-	-
Instance	0	-	97	114	-
Memory	0	-	10	10	-
Multiplexer	-	-	-	131	-
Register	-	-	98	-	-
Total	0	2	205	430	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	~0	~0	0

4. Optimize

- Unroll the loop

```
for (int j = 0; j < NUM_ITERATIONS; j++) {
    #pragma HLS unroll
    // Determine if we are rotating by a positive or negative angle
    int sigma = (theta < 0) ? -1 : 1;

    // Multiply previous iteration by 2^(-j)
    COS_SIN_TYPE cos_shift = current_cos * sigma * factor;
    COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

    // Perform the rotation
    current_cos = current_cos - sin_shift;
    current_sin = current_sin + cos_shift;

    // Determine the new theta
    theta = theta - sigma * cordic_phase[j];

    factor = factor / 2;
}
```

- Timing

⊟ Timing

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.188 ns	2.70 ns

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
8	8	80.000 ns	80.000 ns	9	9	no

⊟ Detail

⊞ Instance

⊞ Loop

可以發現將 loop unroll 之後計算所需要的 latency 從原本的 62 個 clock cycle 下降到 8 個 clock cycle。

➢ Utilization

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	919	-
FIFO	-	-	-	-	-
Instance	0	-	97	114	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	53	-
Register	-	-	201	-	-
Total	0	0	298	1086	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	2	0

可以發現在 latency 下降的同時 utilization 也會上升，所以我們必須在 timing 與 area 間做出取捨。

5. Problem、observed and learned

➢ 原本 github 上的 source 中並沒有下 protocol 的 pragma 所以合成出來的 input 與 output 訊號皆為 register，一開始我使用 GPIO port 讀寫資料，但我發現 GPIO 在控制 ctrl 訊號時很容易出錯，導致我會在錯誤的時間讀到結果，使的計算結果錯，所以後來我選擇使用 axi-lite 來讀寫 data 與控制訊號，並成功地解決的這些問題。

➢ 更熟悉每個 protocol 的運作方式，以及對應的 host code 寫法