高階合成技術於應用加速 LabB Report

工科海洋所 R10525034 許家維

Topic: PP4FPGA-CORDIC

(github link: https://github.com/Jarwy/2022FALL_HLS_LabB)

Introduction

```
1.CORDIC
    A. Kernel Code
        (a) Cordic_baseline
        (b) Cordic_optimized1
        (c) Cordic_optimized2
    B. Questions
2.CORDIC_LUT
    A. Kernel Code
    B. Questions
```

圖一

這次實驗的主要目標是希望可以透過實驗去觀察並瞭解任意經度資料型態(arbitrary precision datatype)以及其對於運算效能、資源及準確度的影響。實驗透過使用 CORDIC 及 lookup table 來實現笛卡爾座標轉極座標的案例來進行學習。這份報告的架構(如圖一)大致會分成兩個部分，分別是 CORDIC 及 CORDIC_LUT。在第一個部分(CORDIC)會先介紹教學中要求我們自行完成的 CORDIC kernel code，其中包含 baseline 及三種優化版本。接著，會根據實驗結果來回答網站上要求回答的三個問題。在第二個部分(CORDIC_LUT)，也會先從 kernel code 介紹起，接著回答網站上所提出的問題。

1. CORDIC

A. Kernel Code

(a) cordic_baseline

```
#include "cordiccart2pol.h"
#include<iostream>
using namespace std;
data_t  FACTOR = 0.607252935008881256169 4;
data_t Kvalues[NO_ITER] = {          1,      0.5000000000,      0.25000000000,      0.12500000000,
                           0.06250000000,      0.0312500000,      0.01562500000,      0.00781250000,
                           0.00390625000,      0.0019531250,      0.00097656250,      0.00048828125,
                           0.000244140625,     1.220703125e-04,   6.103515625e-05,    3.0517578125e-05,
                           1.52587890e-05,     7.62939450e-06,    3.81469725e-06,     1.9073486300e-06,
                           9.53674315e-07,     4.76837157e-07,    2.38418578e-07,     1.1920928900e-07,
                           5.96046445e-08,     2.98023222e-08,    1.49011611e-08,     7.4505805500e-09,
                           3.72529028e-09,     1.86264514e-09,    9.31322570e-10,     4.6566128500e-10};

data_t angles[NO_ITER] = {  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,};


void cordiccart2pol(data_t x, data_t y, data_t &r,  data_t &theta)
{
#pragma HLS INTERFACE s_axilite port=x
#pragma HLS INTERFACE s_axilite port=y
#pragma HLS INTERFACE s_axilite port=r
#pragma HLS INTERFACE s_axilite port=theta
#pragma HLS INTERFACE ap_ctrl_none port=return

    printf("================Baseline================\n");

    data_t base_radian = 0.0;
    theta = 0.0;
    r = 0.0;

    //Convert all input coordinates to the range +pi/2 ~ -pi/2
    if(y >= 0) {
        //if input is in the 1st or 2nd quadrant, INVERT it 90 degrees
        data_t temp = x;
        x = y;
        y = -temp;
        base_radian = 1.57079632679;    //Remedy for angle conversion
    }else{
        //if input is in the 3rd or 4th quadrant, turn it forward 90 degrees
        data_t temp = y;
        y = x;
        x = -temp;
        base_radian = -1.57079632679;   //Remedy for angle conversion
    }

    for(int j = 0; j < 32 ; j++){

        int sign = (y < 0) ? 1 : -1;    //Determine rotate direction is positive or negative

        data_t temp_x = x;

        x = x - sign * Kvalues[j] * y;
        y = y + sign * Kvalues[j] * temp_x;

        theta = theta - sign *  angles[j];  //Update rotation angle
    }
    theta = theta + base_radian;
    r = x * FACTOR;
}
```

1

2

圖二

圖二是初版的 cordic kernel code，程式中自定義型態 data_t 為 float 型態。程式中程式有兩大部分，第一部份是用來將輸入座標轉換到 pi/2~-pi/2 範圍中，來確最終旋轉結果能逼近 0 度(座標接近 X 軸，Y 值逼近 0)。第二部分則是用來實現旋轉的功能。每次旋轉的角度都會累計到 theta 參數，theta 累計值加上最初的扣掉的校正徑度(base_radian)就會是輸出的 theta 值；

而最終的 X 座標值再除上旋轉所造成的 cordic gain 就會是 r 值。

(b) cordic_optimized1

cordic_optimized1 的 kernel code 與 baseline 完全相同，唯一的不同之處在於我將 data_t 的型態更改為 ap_fixed<16,4>，用來觀察 datatype 對運算效能、資源及準確度有甚麼影響，這部份比較會在問題的部分提到。

(c) cordic_optimized2

```
void cordiccart2pol(data_t x, data_t y, data_t &r,  data_t &theta)
{
    printf("================Optimized2================\n");
    // Write your code here
    data_t base_radian = 0.0;
    theta = 0.0;
    r = 0.0;
    if(y >= 0) {
        data_t temp = x;
        x = y;
        y = -temp;
        base_radian = 1.57079632679;
    }else{
        data_t temp = y;
        y = x;
        x = -temp;
        base_radian = -1.57079632679;
    }

    for(int j = 0; j < 32 ; j++){

        data_t x_shift =  x >> j;
        data_t y_shift =  y >> j;

        if(y>=0){               //sign=-1
            x = x + y_shift;
            y = y - x_shift;
            theta = theta + angles[j];
        }else{                  //sign=1
            x = x - y_shift;
            y = y + x_shift;
            theta = theta - angles[j];
        }

    }
    theta = theta + base_radian;
    r = x * FACTOR;
}
```

圖三

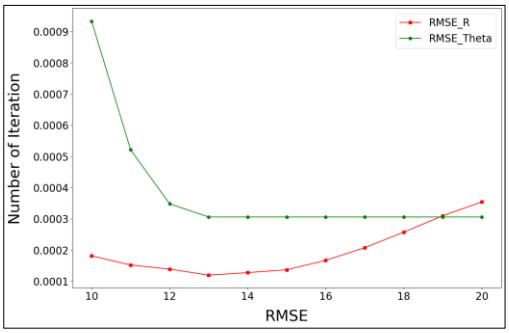
圖三是針對旋轉部份程式進行優化，將正轉反轉以 if 判斷是分開並將原先與 Kvalues 元素相乘的部份用位移代替，這樣一來，所有的乘除法運算都能以簡單的加減法代替。

B. Questions

- **Question 1:** One important design parameter is the number of rotations. Change that number to numbers between 10 and 20 and describe the trends. What happens to performance? Resource usage? Accuracy of the results? Why does the accuracy stop

improving after some number of iterations? Can you precisely state when that occurs?

Answer: 按照題目的意思，我將 iteration 的次數從 10 遞增到 20，並逐一記錄其效能、資源用量以及誤差(輸入測資我是用亂數產生 50 組範圍為 1~-1 間的 xy 值，並將測資存起來重複使用，所以每一次的測試皆公平)。根據測試結果，latency 與想像中相同，是會隨著 iteration 增加而增加的；而誤差的部份就沒有相同的線性關係了，從圖四可以看到，iteration 增加誤差並不一定會跟著減小，我猜想這個現象是因為當角度很接近零度時，增加或減少某個度數很有可能會讓角度值有正負值的轉換，導致離 0 度更遠的情況發生。參考下表(表一)以一個簡單的例子為例，假設今天輸入的座標是在 30 度的位置，可以注意到，第三個 iteration 的的終點角度為-2.511，而到了第四個 iteration 終點角度變為 4.614，反而離 0 度更遠了，因此 iteration 的次數與準確度並不是線性的關係，只能說 iteration 次數多，比較能夠保證角度能收斂到 0 度附近。



圖四

#iteration	起始角度	旋轉角度	終點角度
1	30	45(-)	-15
2	-15	26.565(+)	11.525
3	11.525	14.036(-)	-2.511
4	-2.511	7.125(+)	4.614
5	4.614	3.576(-)	1.038

表一

- **Question 2:** Another important design parameter is the data type of the variables. Is one data type sufficient for every variable or is it better for each variable to have a different type? Does the best data

type depend on the input data? What is the best technique for the designer to determine the data type(s)?

Answer: datatype 與運算效能、資源用量及準確度是絕對有關係的，使用比較高精度的 datatype 做運算雖然準確度會比較高，但相對的也會需要較多的運算資源及時間成本。比較圖五及圖六，兩個設計唯一的不同之處就只有資料型態的不同(從 float 改成 ap_fix)，在運算時間與資源用量上就有了非常大的區別，但使用 ap_fix 會造成誤差變大，兩者間的取捨並沒有甚麼標準答案，端看設計者的需求。接著

至於我是怎麼決定要用 ap_fix<16,4>的呢？我考慮到 theta 輸出的範圍會在 2pi(6.28)~-2pi(-6.28)，所以小數點前面的數字部份至少要有 3 個位元，再加上用來表示正負號的位元，因此總共要有 4 個位元來表示，而小數的位元數是經過測試誤差大小得到的。

那還有沒有更省的方法呢？以這次實驗為例，如果能夠知道使用者會輸入的座標值範圍，那我們就能使用剛剛好的資料型別去承接輸入值，來達到更省資源的目的。

Modules & Loops	Latency(cycles)	Latency(ns)	Interval	BRAM	DSP	FF	LUT	URAM
cordiccart2pol	491	4.910E3	492	2	10	1471	2428	0
cordiccart2pol_Pipeline_VITIS_LOOP_53_1	482	4.820E3	482	2	5	760	1029	0

圖五 cordic_baseline 效能資源評估表 (RMSE_R: 2.19e-7, RMSE_Θ: 5.98e-4)

Modules & Loops	Latency(cycles)	Latency(ns)	Interval	BRAM	DSP	FF	LUT	URAM
cordiccart2pol	104	1.040E3	105	0	3	170	388	0
cordiccart2pol_Pipeline_VITIS_LOOP_43_1	99	990.000	99	0	2	131	236	0

圖六 cordic_opt1 效能資源評估表 (RMSE_R: 4.39e-4, RMSE_Θ: 3.99e-4)

- **Question 3:** What is the effect of using simple operations (add and shift) in the CORDIC as opposed to multiply and divide? How does resource usage change? Performance? Accuracy?

Answer: 比較圖六與圖七，兩個設計的區別是 opt2 將 opt1 中所使用到的乘除法改用較為簡單的位移與加減法來取代，又能進一步縮短運算時長並節省資源，但以位移取代除法的方法也會讓誤差變大，一樣是需要根據使用者的需求去做取捨。

Modules & Loops	Latency(cycles)	Latency(ns)	Interval	BRAM	DSP	FF	LUT	URAM
cordiccart2pol	39	390.000	40	0	1	142	521	0
cordiccart2pol_Pipeline_VITIS_LOOP_38_1	34	340.000	34	0	0	103	369	0

圖七 cordic_opt2 效能資源評估表(RMSE_R: 2.06e-3, RMSE_Θ: 6.36e-4)

2. CORDIC_LUT

A. Kernel Code

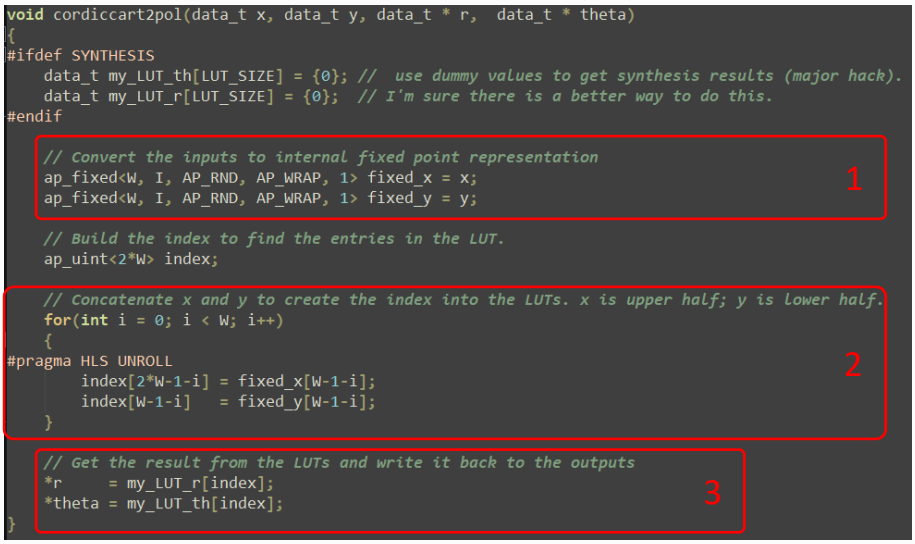
```
void init_cart2pol_LUTs(data_t my_LUT_th[LUT_SIZE], data_t my_LUT_r[LUT_SIZE])
{
    // Fill the LUT with appropriate values
    float step = 0.5/FRACTIONAL_BITS; //
    int num_steps = (int) 2/step; // x and y are normalized between -1 and 1

    float min_theta, max_theta = 0;
    float min_r, max_r = 0;

    // Fill the LUT values with their appropriate R and theta values
    for(int i=0; i<LUT_SIZE; i++){
        ap_uint<2*W> index = i;
        ap_fixed<W, I, AP_RND, AP_WRAP, 1> fixed_x;
        ap_fixed<W, I, AP_RND, AP_WRAP, 1> fixed_y;
        cout<<"index[15]"<<index[15]<<endl;
        for(int j = 0; j < W; j++)
            {
                fixed_x[W-1-j] = index[2*W-1-j];
                //cout<<"index[2*W-1-j]"<<index[2*W-1-j]<<endl;
                fixed_y[W-1-j] = index[W-1-j];
            }

        float _x = fixed_x;
        //cout<<"_x"<<_x<<endl;
        float _y = fixed_y;
        //cout<<"_y"<<_y<<endl;

        if((_x == 0) & (_y == 0)){
            my_LUT_th[index] = 0;
            my_LUT_r[index]  = 0;
        }
        else{
            my_LUT_th[index] = atan2f(_y, _x);
            my_LUT_r[index]  = sqrtf((_y*_y)+(_x*_x));
        }
    }
}
```

1

2

圖八

如圖八，在 cart2pol_LUTs 函式中大概有兩個部份，簡單來說第一個部份是用來產生 x y 座標值的，而第二個部份使用 atan2f 及 sqrtf 來分別算出 thrta 及 r 值並填入 lookup table 中。



圖八

如圖九，cordiccart2pol 函式第一部份先將輸入的 xy 座標都轉為 ap_fixed 的資料型態，接著第二部份將 fixed_x/y 轉換成能夠去 lookup table 查表的 index。得到 index 後，第三部份就拿著 index 去 lookup table 查表。

B. Questions

- How does the input data type affect the size of the LUT? How does the output data type affect the size of the LUT? Precisely describe the relationship between input/output data types and the number of bits required for the LUT.

  Answer: 假設輸入用 8 bits 表示，我們共有 xy 兩個輸入，那麼 LUT 的 size 會需要 2^8*2^8=2^16 個空間，而每一個空間需要 8 bits，因此 LUT 會需要 2^16*8=2^19 bits 的空間。

- The testbench assumes that the inputs x, y are normalized between [-1,1]. What is the minimum number of integer bits required for x and y? What is the minimal number of integer bits for the output data type R and Theta?

  Answer: x 與 y 的整數部份應該只需要 2bits，一個表示符號，另一個表示數字；並且由於 x 及 y 被限制在[-1,1]，所以 r 可能出現的範圍為[-1.414, 1.414]，因此 r 的整數部份應該也只需要

2bits；而 theta 的範圍為[-pi, pi]，因此 theta 的整數部份會需要 3bits。

- Modify the number of fractional bits for the input and output data types. How does the precision of the input and output data types affect the accuracy (RMSE) results?

Answer: 比較圖九與圖十可以發現減少用來代表小數部份的位元數會導致運算誤差變大。其中圖九是用 5 bits 來表示小數，而圖十只用 4 bits 來表示小數。

```
Testbench min_theta=-3.1316, max_theta=3.1416
Testbench min_r=0.0000, max_r=1.4142
   RMSE(R)            RMSE(Theta)
0.023094084113836 0.051045734435320
```

圖九

```
Testbench min_theta=-3.1316, max_theta=3.1416
Testbench min_r=0.0000, max_r=1.4142
   RMSE(R)            RMSE(Theta)
0.045045133680105 0.094583898782730
```

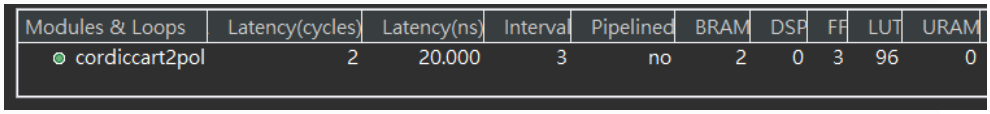
圖十

- What is the performance (throughput, latency) of the LUT implementation. How does this change as the input and output data types change?

Answer: 由於 LUT 法不需要根據輸入去及實作運算，只需要查表就可以吐出結果，所以 latency 而 throughput 高。比較圖十一及圖十二可以看到，輸入輸出的資料型態的改變對於 latency 及 throughput 上沒有太大的幫助，但 data bits 越少，LUT 所需的 size 就越小，會明顯影響 BRAM 的用量。

Modules & Loops	Latency(cycles)	Latency(ns)	Interval	BRAM	DSP	FF	LUT	URAM
cordiccart2pol	2	20.000	3	8	0	3	104	0

圖十一



Modules & Loops	Latency(cycles)	Latency(ns)	Interval	Pipelined	BRAM	DSP	FF	LUT	URAM
cordiccart2pol	2	20.000	3	no	2	0	3	96	0

圖十二

- What advantages/disadvantages of the CORDIC implementation compared to the LUT-based implementation?

Answer: CORDIC 方法的優點是精度比較高，缺點是要花比較多時間運算，而且在 DSP、FF、LUT 的使用量也較高。而使用 LUT 查表法的優點是 latency 短且 DSP、FF、LUT 的用量低，但缺點是的精度低且 BRAM 用量較高。