Final Project Report

Team 05 B09901050 葉芳瑜

B09507018 羅文頡

1.Algortihm – QR decomposition algorithm introduction

We follow the algorithm outlined in the final project slide. First, we calculate the Euclidean distance of h_1^o. We compute the square sum of the data, then put it into a LUT to get the square root. Also put the square sum into another LUT to get the reciprocal of the Euclidean distance which is used for calculating the normalized factor e_1 (Q_{11}, Q_{21}, Q_{31}, Q_{41}).

$$\boldsymbol{e_1} = \boldsymbol{h_1}^{(0)}/\|\boldsymbol{h_1}^{(0)}\|$$

$Q_{11}, Q_{21}, Q_{31}, Q_{41}$

Then we calculate the inner product between e* and h_2^0, h_3^0, h_4^0 to get R_{12}, R_{13}, R_{14}. Next, use the formula below to obtain h_2^1,h_3^1,h_4^1.

$$\boldsymbol{h_2}^{(0)} \cdot \boldsymbol{e_1} = \boldsymbol{e_1^H h_2}^{(0)} \ //R_{12} \qquad \boldsymbol{h_2}^{(1)} = \boldsymbol{h_2}^{(0)} - (R_{12})\boldsymbol{e_1}$$
$$\boldsymbol{h_3}^{(0)} \cdot \boldsymbol{e_1} = \boldsymbol{e_1^H h_3}^{(0)} \ //R_{13} \qquad \boldsymbol{h_3}^{(1)} = \boldsymbol{h_3}^{(0)} - (R_{13})\boldsymbol{e_1}$$
$$\boldsymbol{h_4}^{(0)} \cdot \boldsymbol{e_1} = \boldsymbol{e_1^H h_4}^{(0)} \ //R_{14} \qquad \boldsymbol{h_4}^{(1)} = \boldsymbol{h_4}^{(0)} - (R_{14})\boldsymbol{e_1}$$

After finishing this iteration, we repeat the same process. Calculate the Euclidean distance of h_2^1 and then the normalized factor e_2 (Q_{12}, Q_{22}, Q_{32}, Q_{42}). At the end of the iteration, we calculate R_{23}, R_{24}, h_3^2 and h_4^2.

$$\boldsymbol{h_3}^{(1)} \cdot \boldsymbol{e_2} = \boldsymbol{e_2^H h_3}^{(1)} \ //R_{23} \qquad \boldsymbol{h_3}^{(2)} = \boldsymbol{h_3}^{(1)} - (R_{23})\boldsymbol{e_2}$$
$$\boldsymbol{h_4}^{(1)} \cdot \boldsymbol{e_2} = \boldsymbol{e_2^H h_4}^{(1)} \ //R_{24} \qquad \boldsymbol{h_4}^{(2)} = \boldsymbol{h_4}^{(1)} - (R_{24})\boldsymbol{e_2}$$

Then step to the third iteration, calculate the Euclidean distance of h_3^2, normalized factor e_3 (Q_{13}, Q_{23}, Q_{33}, Q_{43}), R_{34} and h_4^3.

$$\boldsymbol{h_4}^{(2)} \cdot \boldsymbol{e_3} = \boldsymbol{e_3^H h_4}^{(2)} \ //R_{34} \qquad \boldsymbol{h_4}^{(3)} = \boldsymbol{h_4}^{(2)} - (R_{34})\boldsymbol{e_3}$$

In the last iteration, we only calculate the Euclidean distance of h_4^3 and the normalized vector e_4 (Q_{14}, Q_{24}, Q_{34}, Q_{44}). Finally, we can obtain output r and y_hat value by multiplying Q^H and y.

$$\underline{\hat{y}} = Q^H \underline{y} = \begin{bmatrix} Q_{11}^* & Q_{21}^* & Q_{31}^* & Q_{41}^* \\ Q_{12}^* & Q_{22}^* & Q_{32}^* & Q_{42}^* \\ Q_{13}^* & Q_{23}^* & Q_{33}^* & Q_{43}^* \\ Q_{14}^* & Q_{24}^* & Q_{34}^* & Q_{44}^* \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} Q_{11}^* y_1 + Q_{21}^* y_2 + Q_{31}^* y_3 + Q_{41}^* y_4 \\ Q_{12}^* y_1 + Q_{22}^* y_2 + Q_{32}^* y_3 + Q_{42}^* y_4 \\ Q_{13}^* y_1 + Q_{23}^* y_2 + Q_{33}^* y_3 + Q_{43}^* y_4 \\ Q_{14}^* y_1 + Q_{24}^* y_2 + Q_{34}^* y_3 + Q_{44}^* y_4 \end{bmatrix}$$

2.FXP setting

a. Regesiter
   (1) H: S1.14
   (2) Y: S1.14
   (3) R: S3.16
   (4) Q: S.15

b. Computation
   (1) Square: S1.10 * S1.10
   (2) Square root: 4.10 -> 3.16
   (3) Reciprocal of the square root 4.10 -> 3.8
   (4) Normalized vector: S1.10 * S3.8
   (5) Rij computation: S1.10 * S.11
   (6) H computation: S1.11 - S3.8 * S.11
   (7) y_hat computation: S1.10 * S.11

3.Hardware Scheduling

In this project, we try to utilize pipelined structure to minimize the overhead of every calculation. We will divide the entire calculation process into the below 6 stages. Since the data are all highly-dependent, we will run all the stages sequentially to maximize the calculation efficiency. Following is the schedule of one output process.

	H41	H42	H43	H44	Y4
H	H10 Receive	H20 Receive	H30 Receive	H40 Receive	
Y					
Rii	R11 Square	R11 Summation + Sqrt			
Rij					R12 Multiplication
Unit Vector			Q1 Multiplication	Q1 Summation	
R Output					
Y Output					

We will wait 15 cycles than start our computation at 16th cycle, whose loaded i_data is H41.

	H11	H12	H13	H14	Y1
H		H21 Multiplication	H21 Subtraction	H31 Multiplication	H31 Subtraction/ H41 Multiplication
Y					
Rii				R22 Square	R22 Summation + Sqrt
Rij	R12 Summation/ R13 Multiplication	R13 Summation	R14 Multiplication	R14 Summation	
Unit Vector					
R Output					
Y Output					Y11 Multiplication

	H21	H22	H23	H24	Y2
H	H41 Subtraction				H32 Multiplication
Y					
Rii					
Rij			R23 Multiplication	R23 Summation/ R24 Multiplication	R24 Summation
Unit Vector	Q2 Multiplication	Q2 Summation			
R Output					
Y Output	Y11 Addition/ Y21 Multiplication	Y21 Addition/ Y31 Multiplication	Y31 Addition/Y41 Multiplication	Y41 Addition/Y12 Multiplication	Y12 Multiplication

	H31	H32	H33	H34	Y3
H	H32 Subtraction / H42 Multiplication	H42 Subtraction			
Y					
Rii		R33 Square	R33 Summation + Sqrt		
Rij					
Unit Vector				Q3 Multiplication	Q3 Summation
R Output					
Y Output	Y22 Addition/ Y32 Multiplication	Y32 Addition/ Y42 Multiplication	Y42 Addition		

While loading the next set of data for next output, we continue calculating the above value.

	H41	H42	H43	H44	Y4
H			H43 Multiplication	H43 Subtraction	H Backup
Y					
Rii	R11 Square	R11 Summation + Sqrt			R44 Square
Rij					R12 Multiplication
Unit Vector			Q1 Multiplication	Q1 Summation	
R Output					
Y Output	Y13 Multiplication	y23 Multiplication	y23Addition/y33Multiplication	Y43 Multiplication	

When proceeding to the H4 cycle for the next set of data, we will calculate the value needed for both set of data. We ensure the module resources are fairly shared.

	H11	H12	H13	H14	Y1	H21	H22
H		H21 Multiplication	H21 Subtraction	H31 Multiplication	H31 Subtraction/ H41 Multiplication	H41 Subtraction	
Y				Y Backup			
Rii	R44 Summation + Sqrt			R22 Square	R22 Summation + Sqrt		
Rij	R12 Summation/ R13 Multiplication	R13 Summation	R14 Multiplication	R14 Summation			
Unit Vector		Q4 Multiplication	Q4 Summation			Q2 Multiplication	Q2 Summation
R Output							
Y Output					Y11 Multiplication	Y11 Addition/Y21 Multiplication	Y21 Addition/ Y31 Multiplication
Y Output				Y14 Y24/Y34 /Y44 Multiplication	Y14 Y24/Y34 /Y44 Summation	Y11 Multiplication	
							o_rd_vld

Finally, we will get our 1st output at the data loading stage of the 3rd set. We raise the o_rd_vld at H22 (3rd).
The following provides a more detailed explanation:

a. Data Loading
  (1) Motivation
  As the SPEC specified, we will receive 200 data before we raise the o_last_data signal. To prevent missing out input data, we will have to make sure we design a structure to handle those data.

  (2) Schedule
  The input data sequence starts from H11->H12->H13->H14…., we can only start processing those data until we receive H41.
  Also, we have to prepare a H-backup register array and a Y-backup register array to store the incoming input value. To prevent data confliction, the data we received will be stored in the backup register array while loading data, and it will update the 'real' H data array after we no longer need the H value to perform calculation, which is after we calculate R44.

b. Euclidean Distance Calculation
  (1) Motivation
  We can divide this operation into 2 stages: square, summation + sqrt and the reciprocal value.

  (2) Square Value
  This will be the first stage of Euclidean distance calculation. We will only calculate the square value of real part and imaginary part in this stage to meet the clock period criteria.

  (3) Summation + Square Root
  This will be the second stage of Euclidean distance calculation. We will combine the summation and square root (and the reciprocal value) LUT search to obtain the needed value.

c. Unit vector
  (1) Motivation
  To calculate the unit-vector, we will perform four independent simple multiplication after we obtain the reciprocal of Euclidean Distance.

  (2) Multiplication
  For unit-vector calculation, first we utilize four independent complex multipliers to get the product value within 1 cycle.

  (3) Summation
  Then do the correct summation to obtain the real and imaginary part of the unit-vector.

d. Rij Calculation

(1) Motivation

We will have to perform dot product in this stage, which includes multiplication and summation of four product values. Therefore, we will split the calculation into 2 stages.

(2) Multiplication

We will also utilize 4 independent complex multipliers to calculate product value to fully utilize data paralleling to finish it within 1 cycle.

(3) Summation

We will sum up the 4 product values to obtain the real and imaginary part of Rij.

e. H projection entry update

(1) Motivation

We will have to perform orthogonal projection in this stage, which includes multiplication and subtraction of four product values. Therefore, we will split the calculation into 2 stages.

(2) Multiplication

This part will be same as the unit-vector multiplication stage.

(3) Subtraction

In this part, we will perform 4 parallel subtraction to calculate the new H value.

f. Y output

(1) Motivation

Y Output operation doesn't influence the entire schedule that much. We can simply use 1 multiplier and recursively perform multiplication of Y and H to calculate.

(2) Multiplication

We can perform this multiplication process after we finish calculate the unit vector. Therefore, Y multiplication will start right after we finish unit-vector multiplication.

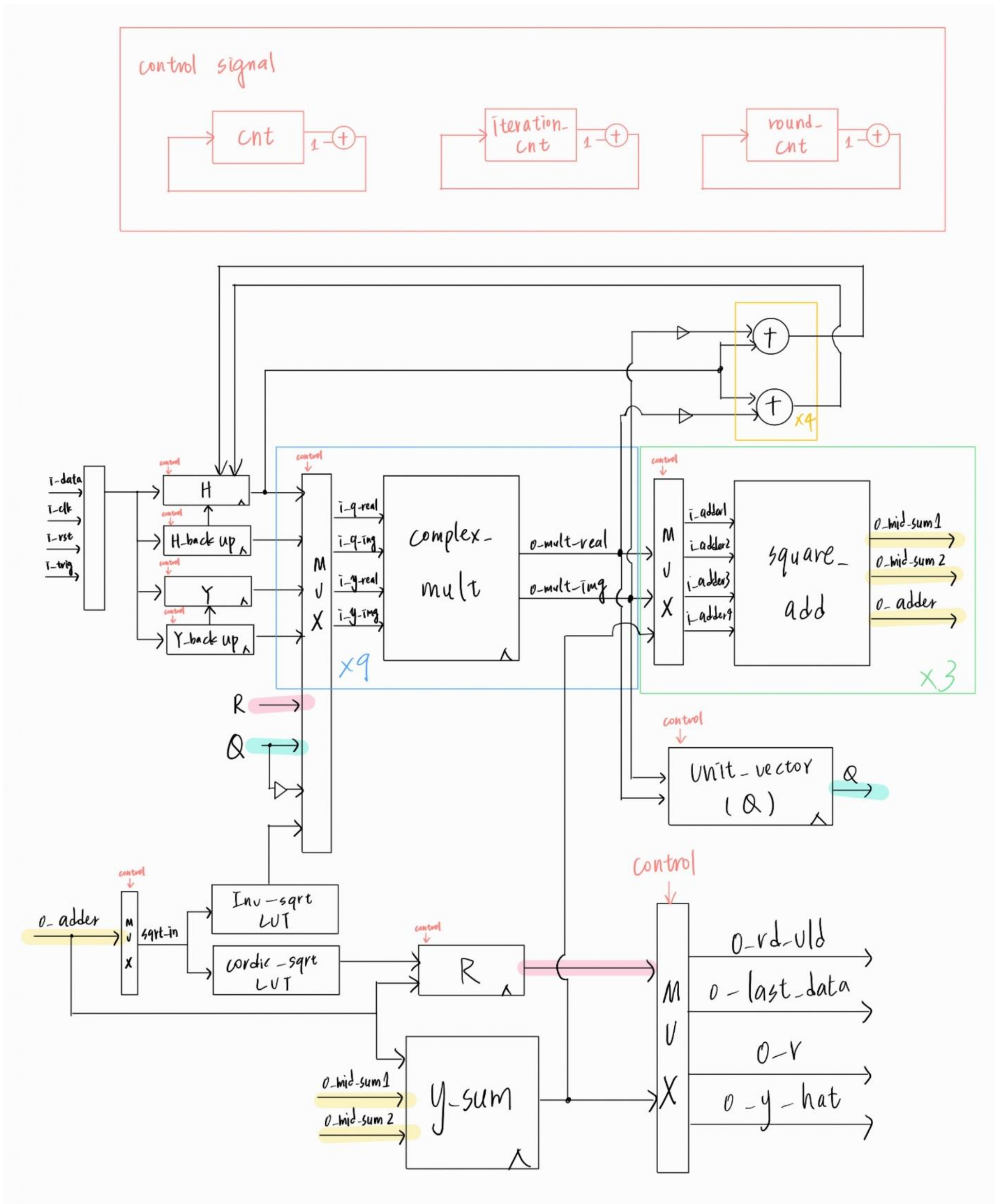
(3) Summation

We will recursively add up the multiplication after we finish calculating multiplication.

g. Overhead Solution

The entire calculation will last 25 cycles, which means that we have to calculate the 4^{th} iteration part and the 1^{st} iteration part simultaneously. To ensure data correctness, we will let the 1^{st} iteration load H value from the backup register array to perform calculation. Until we finish 4^{th} iteration, we will then update the H value.

4.Hardware block diagram



5.Area/Power/Latency report

- Area: 827604.71 um^2

```
************************* Analyze Floorplan *************************
    Die Area(um^2)             : 1145175.43
    Core Area(um^2)            : 827604.71
    Chip Density (Counting Std Cells and MACROs and IOs): 72.269%
    Core Density (Counting Std Cells and MACROs): 100.000%
    Average utilization        : 100.000%
    Number of instance(s)      : 100605
    Number of Macro(s)         : 0
    Number of IO Pin(s)        : 533
    Number of Power Domain(s) : 0
************************* Estimation Results *************************
**********************************************************************
```

- Power: 46.8 mW

```
                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           8.795e-04 1.348e-03 1.689e-06 2.229e-03 ( 4.76%)
register                   0.0182 2.792e-04 1.225e-04    0.0186 (39.78%)  i
combinational              0.0156    0.0100 3.262e-04    0.0260 (55.46%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  =    0.0117   (24.93%)
  Cell Internal Power  =    0.0347   (74.11%)
  Cell Leakage Power   = 4.505e-04   ( 0.96%)
                         ---------
Total Power            =    0.0468  (100.00%)

X Transition Power     =    0.0000
Glitching Power        = 1.532e-04

Peak Power             =    0.6422
Peak Time              =   650.648

1
report_power -verbose > try_active.power
# exitInformation: Defining new variable 'CYCLE'. (CMD-041)
pt_shell>
```

- Latency: 100161.5 ns

```
$finish called from file "./testfixture.v", line 263.
$finish at simulation time             100161500
           V C S   S i m u l a t i o n   R e p o r t
Time: 100161500 ps
CPU Time:    277.420 seconds;       Data structure size:  10.0Mb
Tue Dec 19 17:44:12 2023
CPU time: 8.562 seconds to compile + 3.001 seconds to elab + 1.479
[b9507018@cad30 05_POST]$
```

6.Improvements

a. Conjugate dealing
   When dealing with conjugate complex numbers, we employ a method that involves reversing all bits and omitting the ‘+1’ step. We've observed that this approach doesn't have a

substantial impact on the error rate, yet it enables us to decrease the number of adders. It represents a trade-off.

b. Multiplier bit length
First, we employed 24-bit * 24-bit multipliers, achieving a commendable level of accuracy. Nevertheless, the area and latency of these multipliers proved to be excessively large. Consequently, we opted for a transition to 12-bit * 12-bit multipliers, accepting a higher error rate in exchange for a substantial reduction in both area and latency.

c. Register bit length
Since we only need to do 12-bit * 12-bit multiplication, we downsized the bit length of the H, Q and Y register from 24-bit (S1.22) to 16-bit(S1.14).

d. Square root algorithm
For the square root step in the algorithm, we initially attempted the CORDIC methods of implementation. The CORDIC (Coordinate Rotation Digital Computer) Method utilize the simple adder and right shift to simulate trigonometric functions calculations. The CORIDC method could run in rotation mode and Vectoring Mode. When using Hyperbolic coordinates in rotation mode, we can utilize obtain the sqrt (X^2 – Y^2) value. By substituting X and Y into A+1 and A-1, we can obtain the sqrt value of A.
However, then we find the latency for square root module is too long. Subsequently, we transitioned to using a LUT, opting for a larger area to reduce latency. The LUT is specifically designed for a 14-bit input (4.10) to produce a 19-bit output (3.16).

e. Replace division with multiplication and LUT
When computing the unit vector (Q), division becomes a critical operation. However, employing a traditional divisor consumes too many cycles. Achieving a one-cycle completion would result in excessive latency. To address this, we opted to use a Look-Up Table (LUT) as an alternative. Instead of performing division directly, we transformed it into a multiplication by the reciprocal of the Euclidean distance. For this purpose, we constructed a LUT with a 14-bit (4.10) input, representing the square sum of four data points, and an 11-bit (3.8) output, indicating the reciprocal of the square root of the sum.

f. Separation for multiplication and addition
In our initial schedule, we observed a sequential pattern of performing multiplication followed by up to four addition operations. This sequential path resulted in excessive length. Consequently, we made adjustment to our tight schedule by moving the addition operations to the next cycle. While this modification increased the time required for a single output, it contributed to an overall reduction in cycle time, bringing us closer to the optimal boundary of 5ns.

g. Multiplication module sharing
In our planned schedule, we acknowledged the need to perform up to 9 multiplications (inner product) within a single cycle. To address this requirement, we developed a specialized

module for complex number inner product computation. Moreover, we introduced resource sharing across different cycles to optimize area utilization. The module is designed to handle multiplication and addition in separate cycles. Although this extension results in a two-cycle operation, it effectively reduces latency, meeting our targeted boundary.

h. Addition module sharing
We've designed a module for computing the sum of four data points in our addition operations. This module is crafted to share resources across different cycles, resulting in improved area performance.