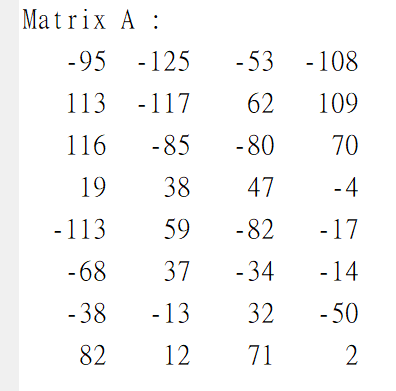
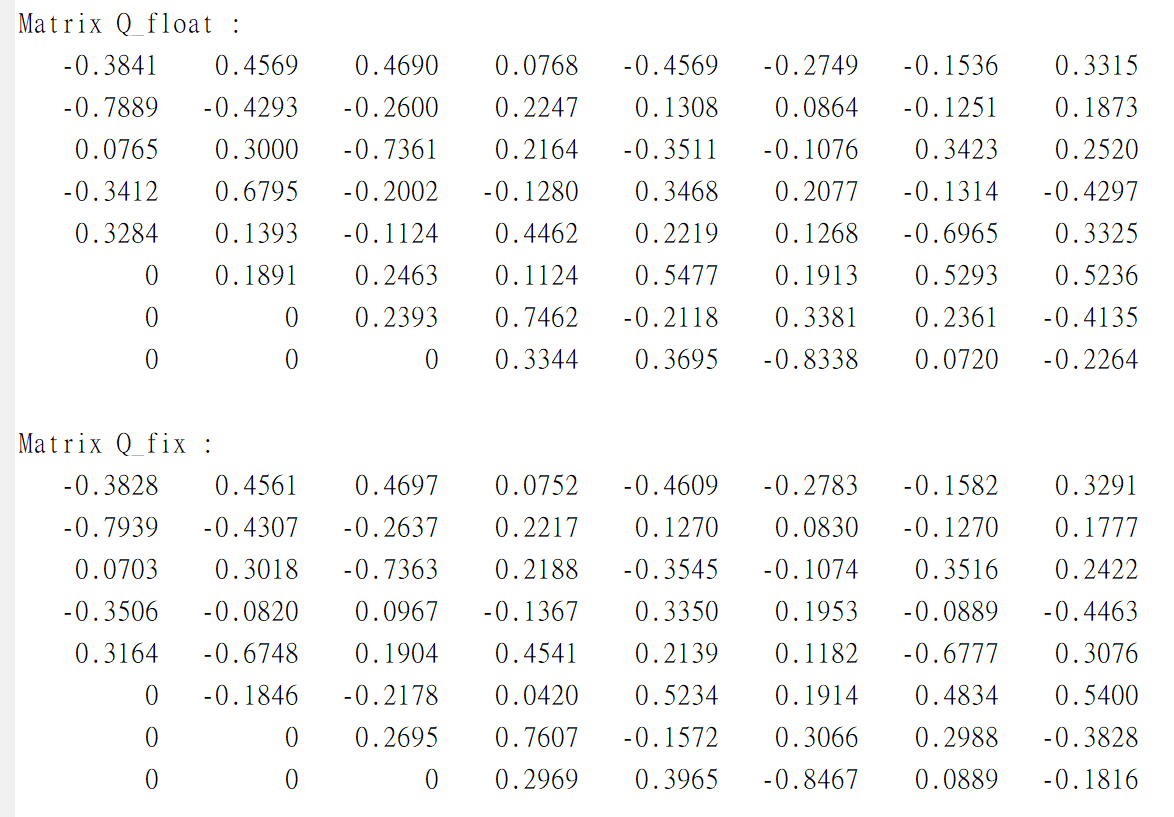
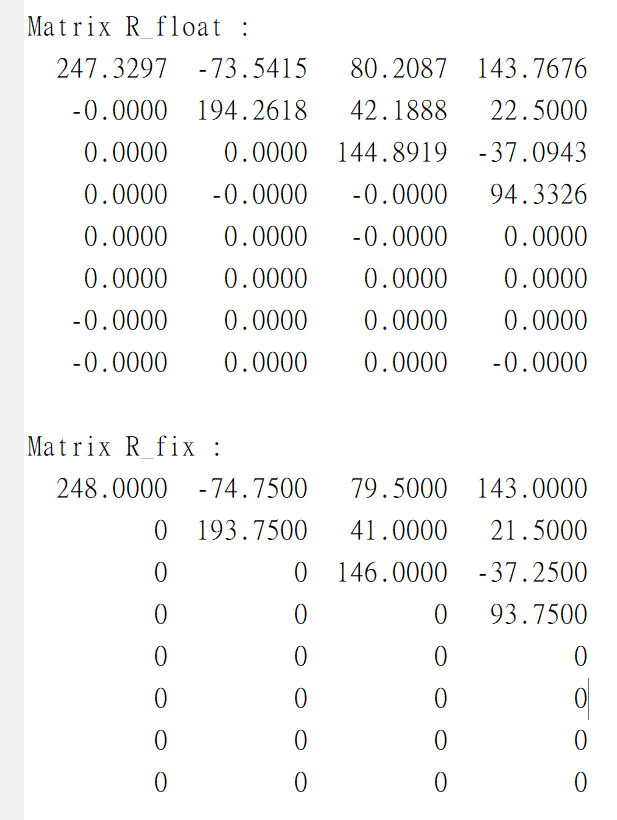
**VLSI DPS HW#4**

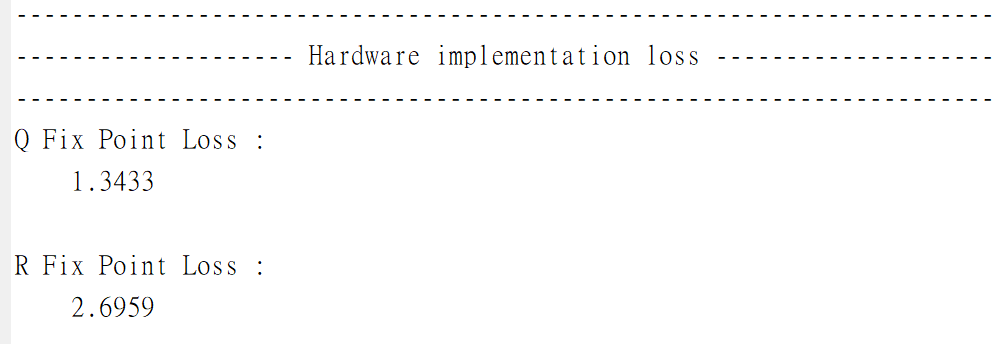
**電機四 4109064119 陳柏淳**

**1. Calculate the Frobenius Distance**

* Generate a random matrix A



* Q\_float and R\_float are generated from Q\*[A|I] =[R|Q]
* Q\_fix and R\_fix are generated from Cordic given’s rotation
* The Frobenius Distance (Euclidean Norm) is sqrt(trace((float-fix)\*(float-fix)'))



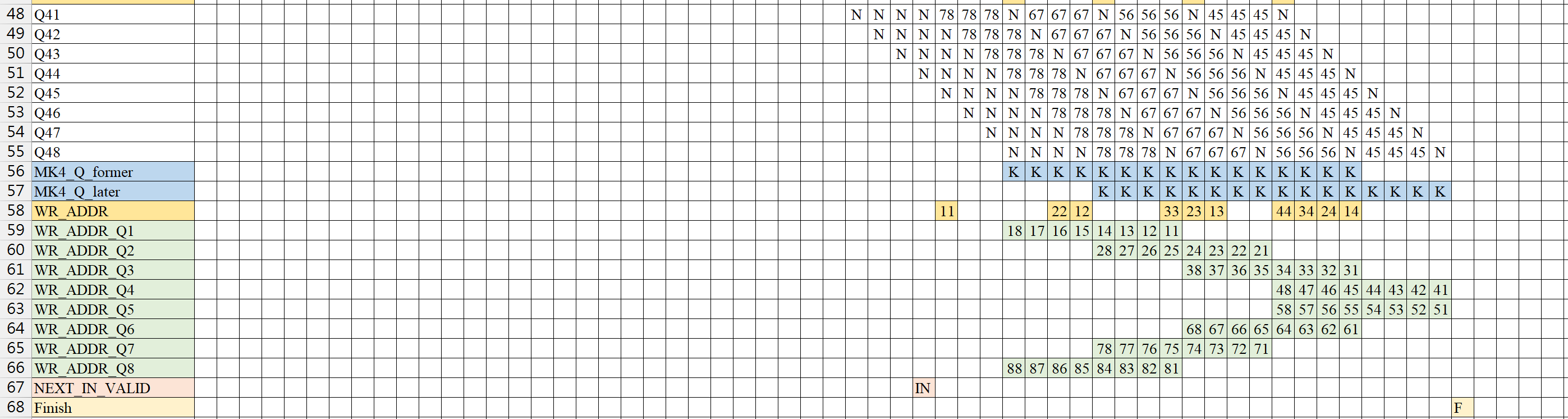
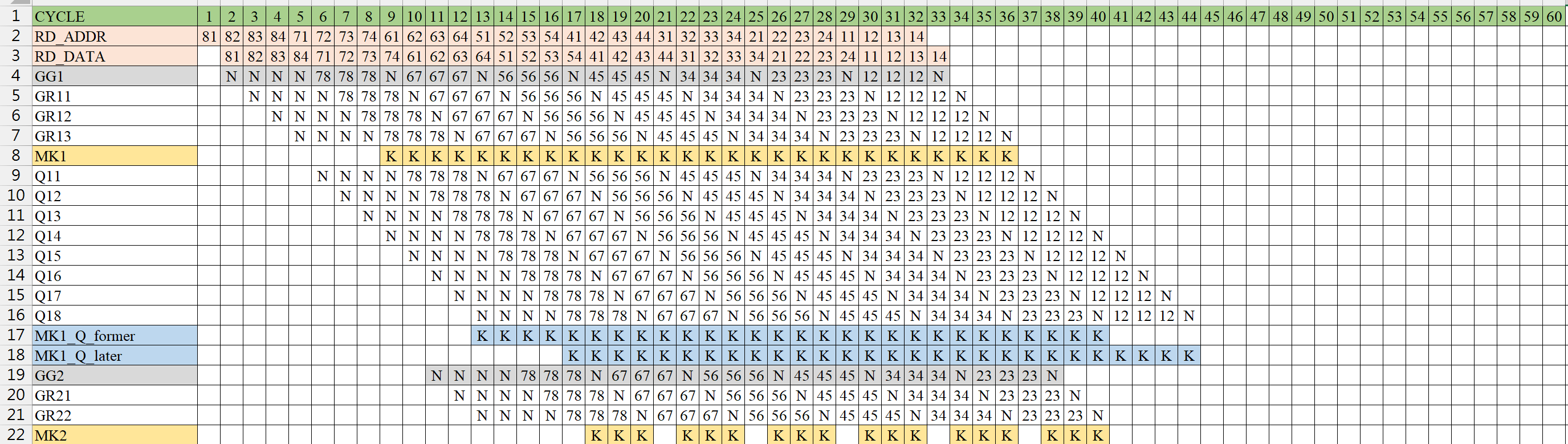
Q\_loss : 1.3433

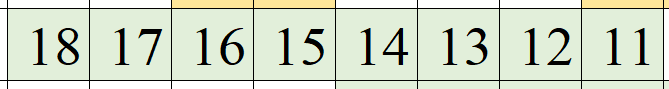
R\_loss : 2.6959

* Fix-point precision

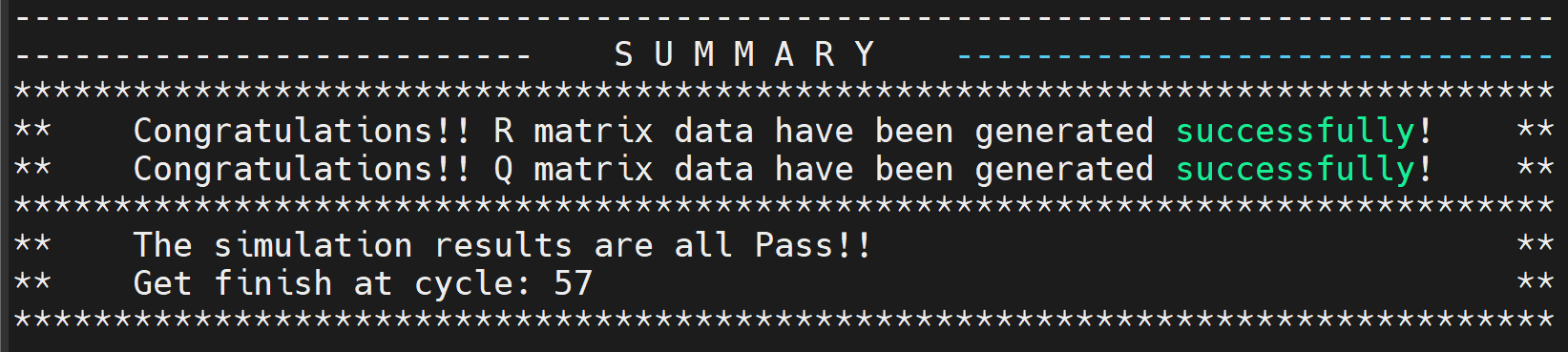
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data | Sign bit | Integer part | Fraction part | Length |
| A (input) | 1 | 7 | 0 | 8 |
| Q (output) | 1 | 1 | 10 | 12 |
| R (output) | 1 | 9 | 2 | 12 |
| K (parameter) | 1 | 0 | 9 | 10 |

**2. Timing diagram**



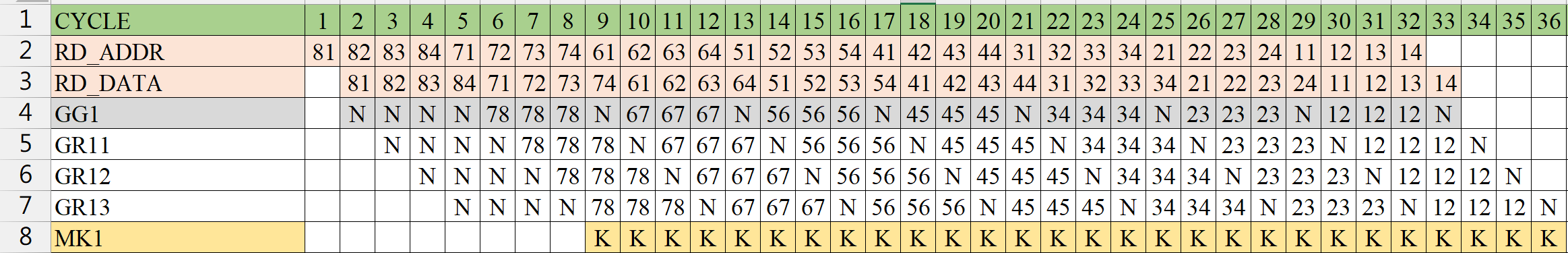
* N : No Operation (Idle)
* MK : Multiplied by K
*  : Perform rotation on 7th and 8th row
* : The matrix index of data write back to tb
* MK1\_Q\_former : process Q11~Q14
* MK1\_Q\_later : process Q15~Q18

**3. Clock cycles needed to complete one QR factorization**

****

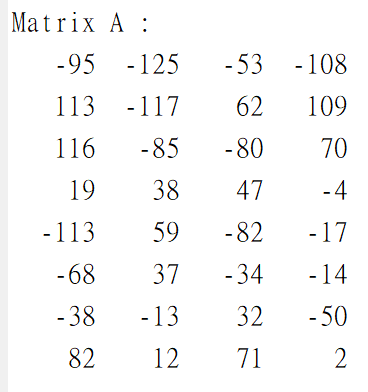
57 cycles (As the timing diagram)

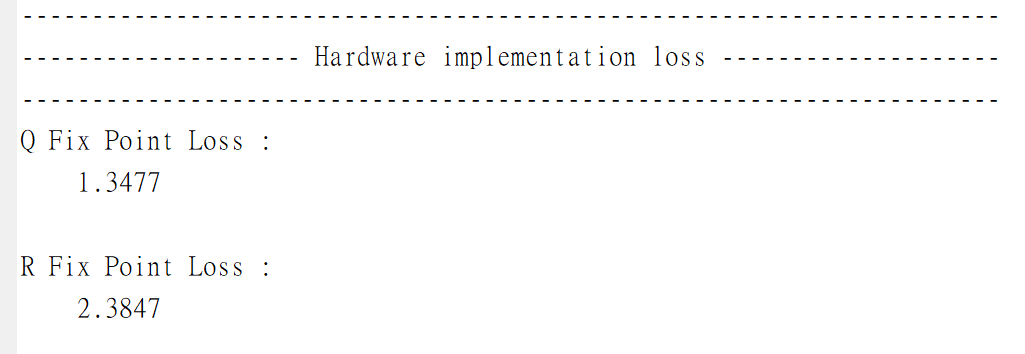
**4. The initiation interval of two successive QR factorizations**

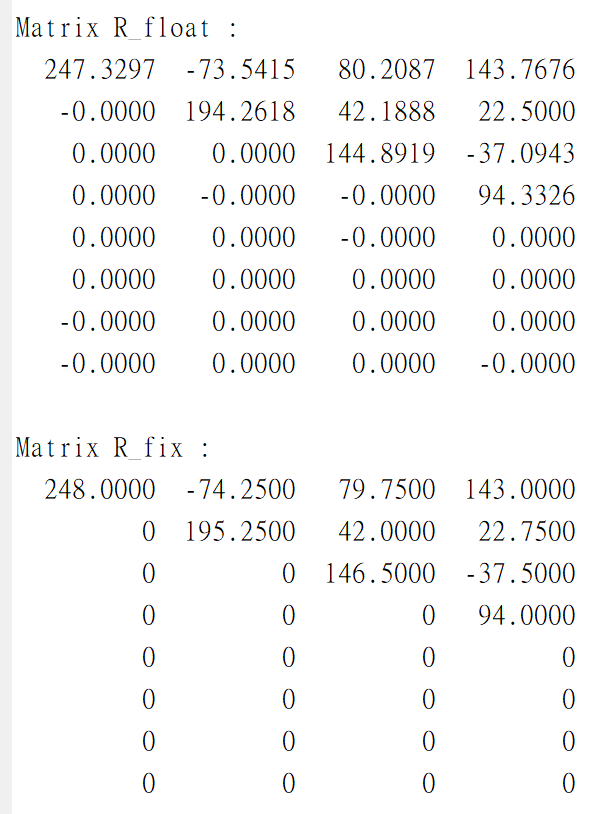
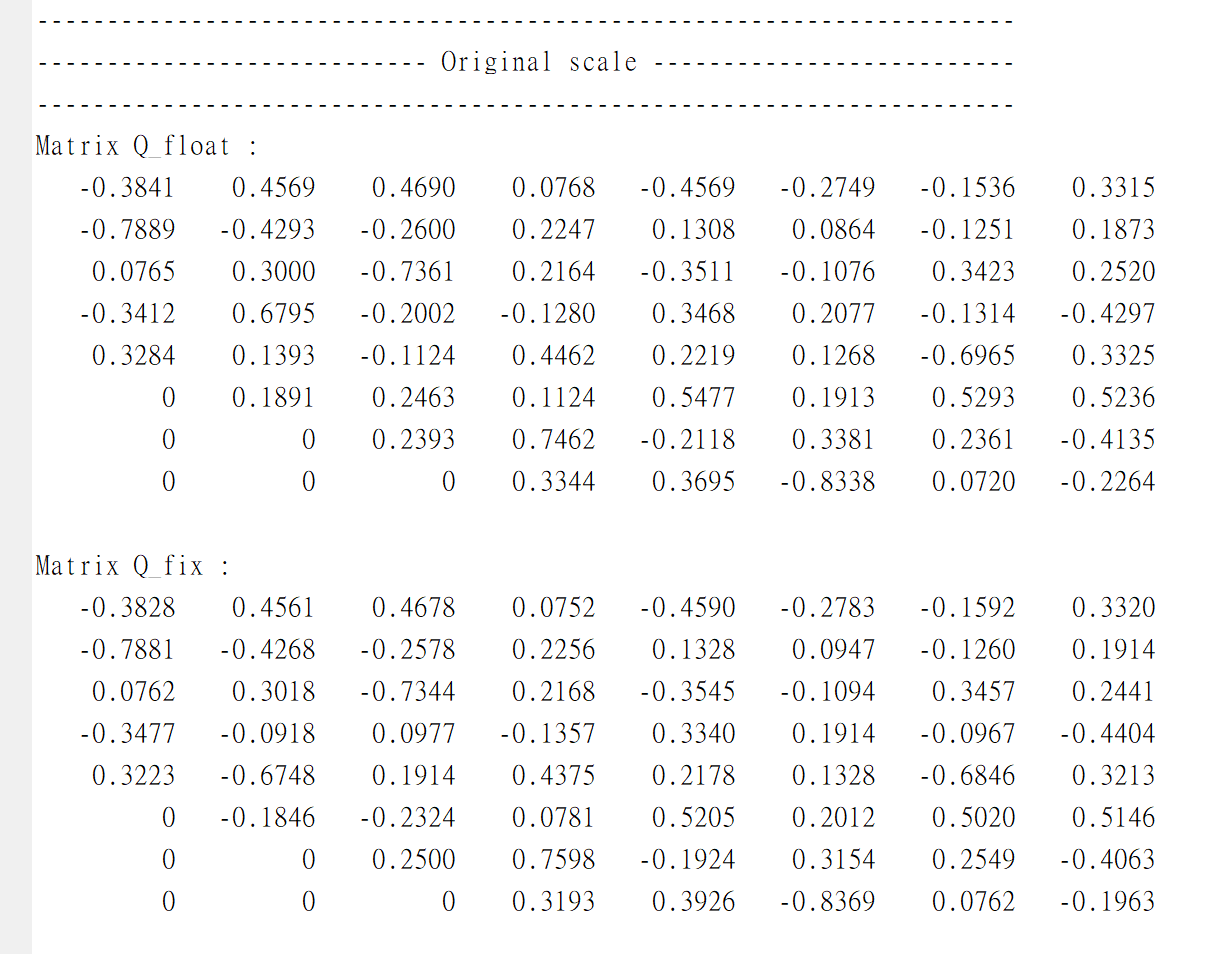
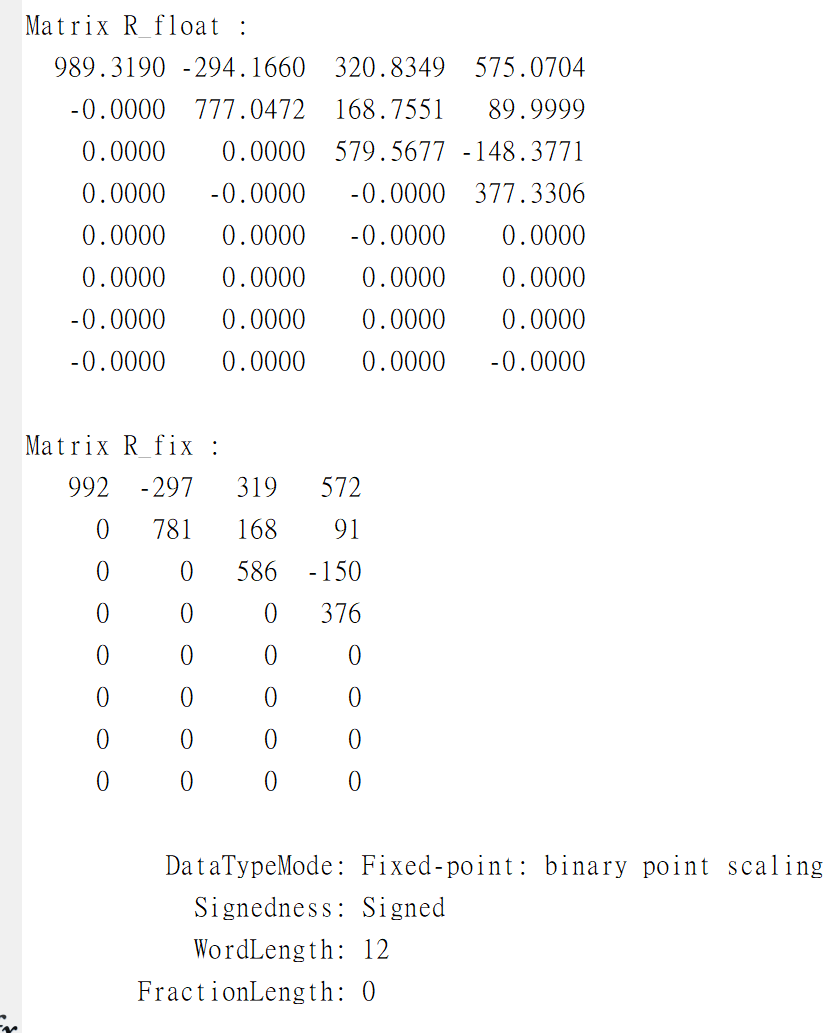
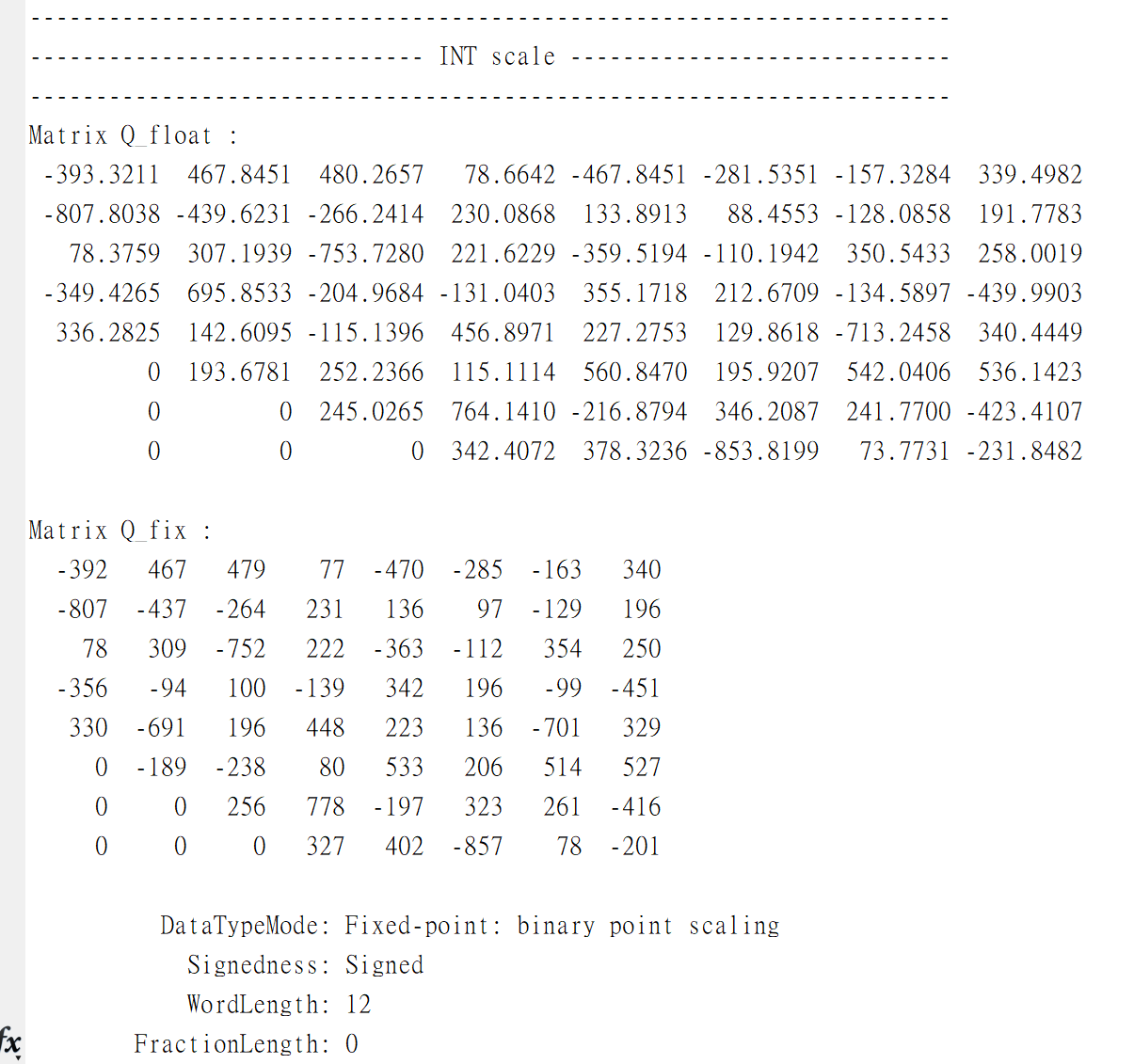
****

When GG1 performs all vectoring over, the next input is valid.

The initiation interval is 33 cycles.

* **Matlab result**
* Random matrix A
* Hardware loss



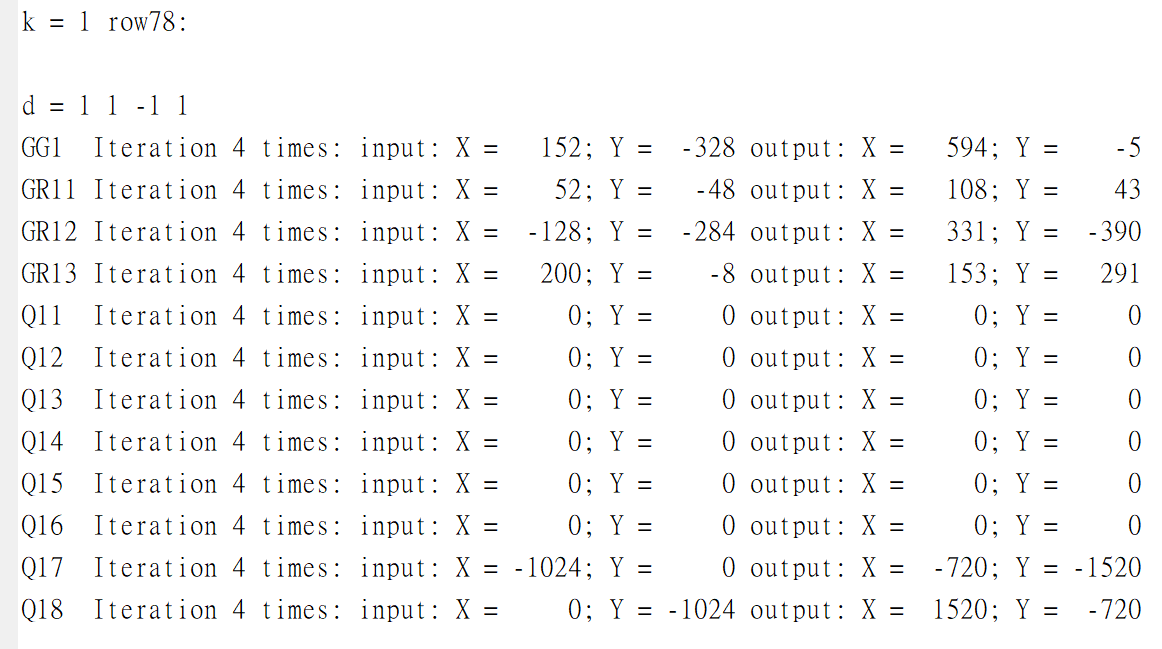
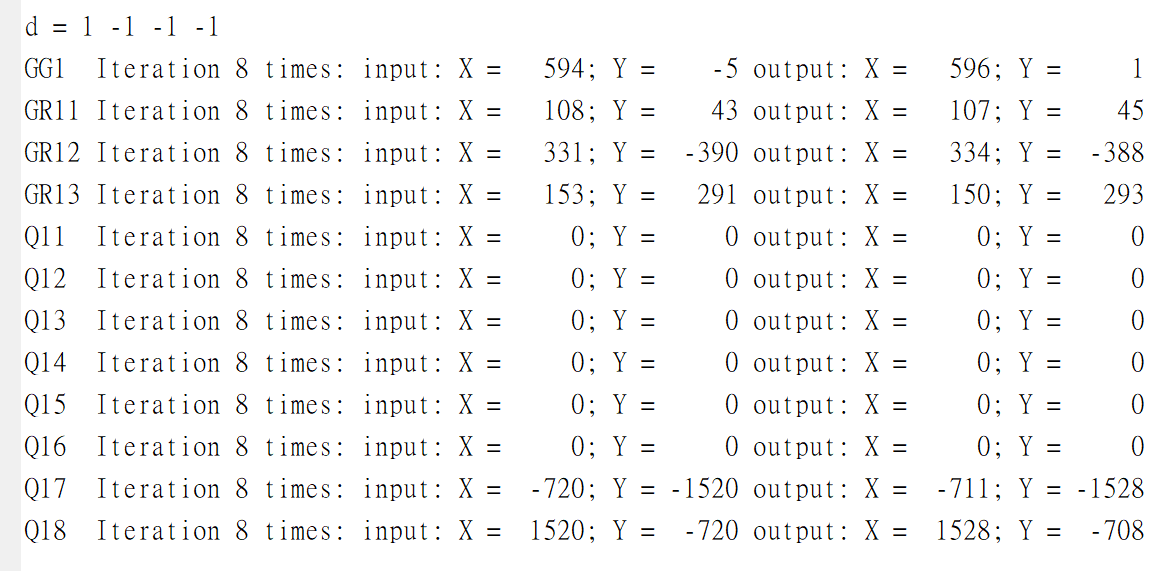
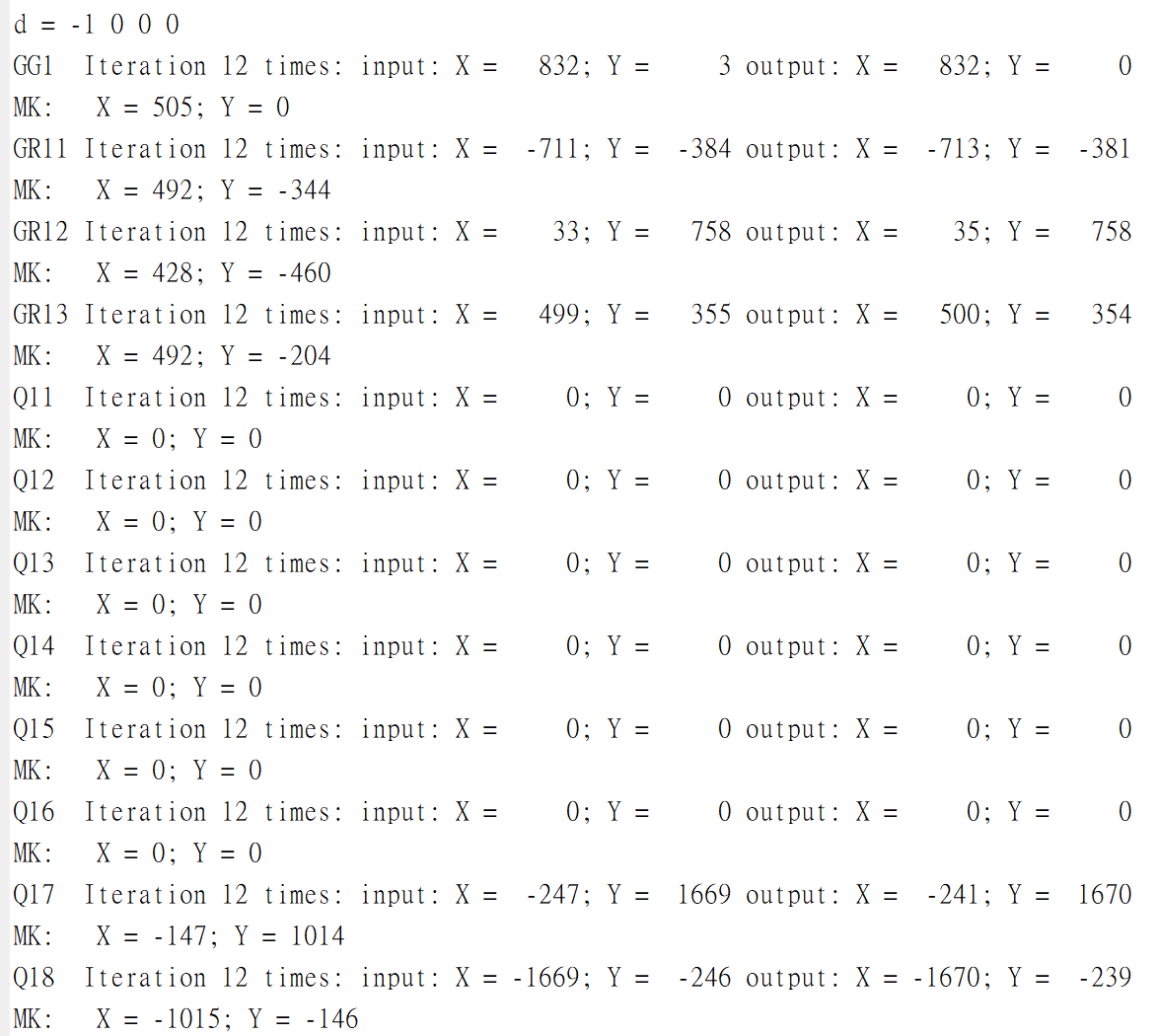


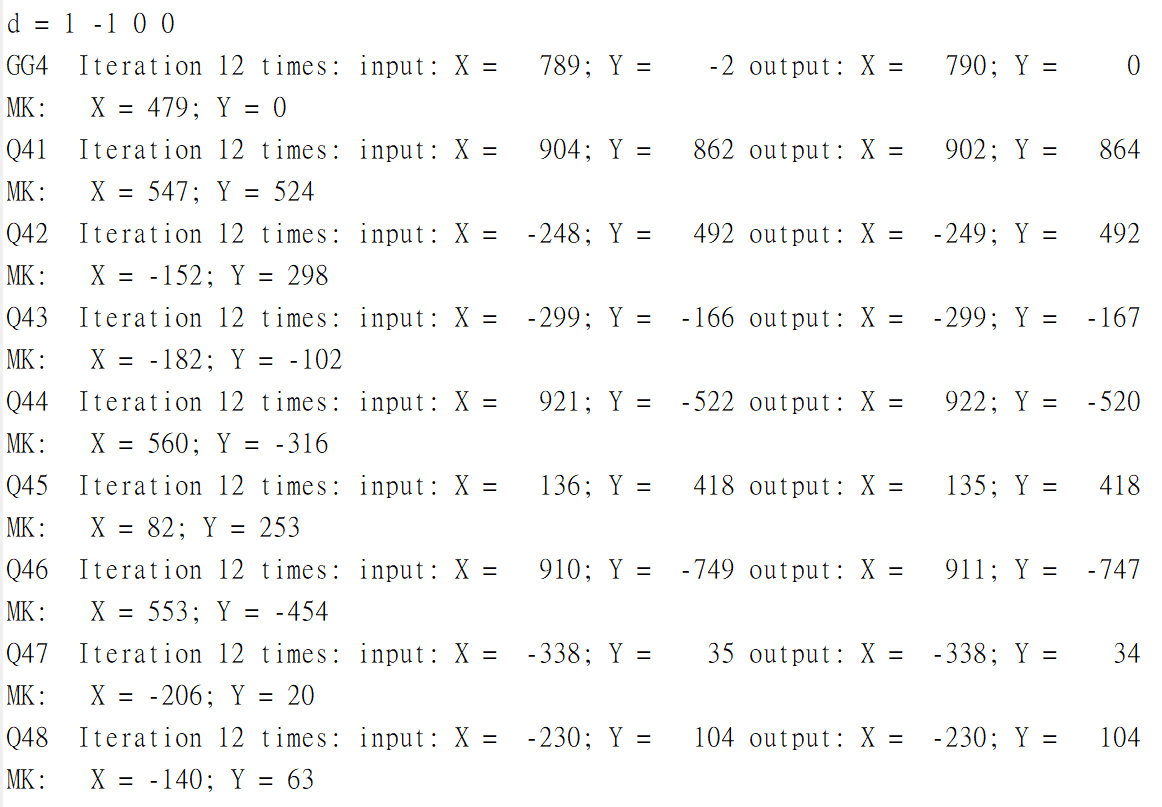
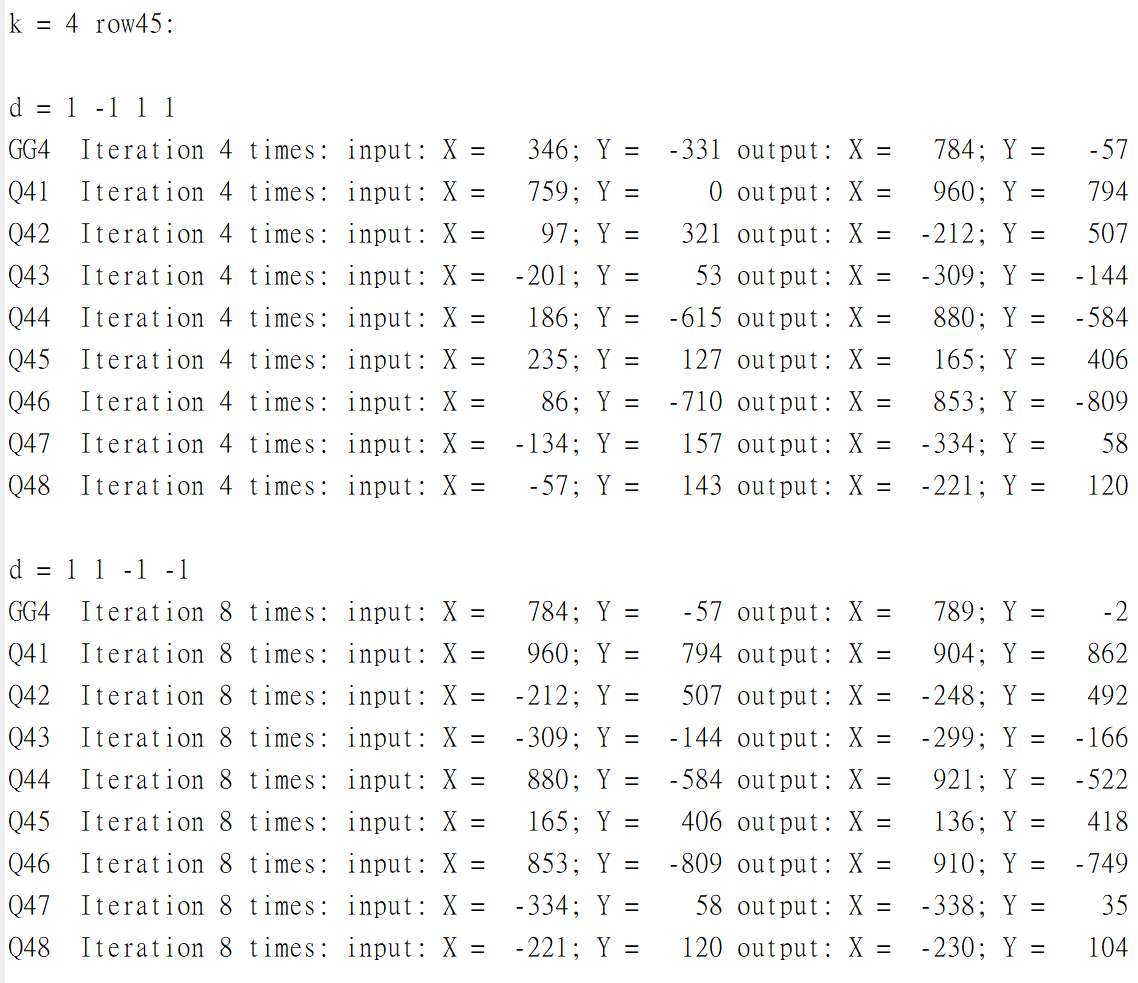
Shift to integer to simulate hardware performance

Actual values

* **Matlab result**

Record the values during iteration, which can be used for debugging in hardware implementation.





Matlab code (only core part):

* Floating-point ( Q\*[A|I] = [R|Q] )

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | % Floating point QR factorization using Q\*[A|I] = [R|Q]  **function** R\_Q **=** float\_QR**(**row\_R**,** col\_R**,** row\_Q**,** Q**,** A**)**  % Q\*[A|I] = [R|Q]  R\_Q **=** **[**A Q**];**  % Perform Givens rotations  **for** p\_float **=** 1 **:** col\_R  **for** q\_float **=** **(**row\_R**-**1**)** **:** **(-**1**)** **:** p\_float  Q **=** eye**(**row\_Q**);**  theta **=** atan2**(**R\_Q**(**q\_float**+**1**,** p\_float**),** R\_Q**(**q\_float**,** p\_float**));**  % Givens Q  Q**(**q\_float **,** q\_float **)** **=** cos**(**theta**);**  Q**(**q\_float **,** q\_float**+**1**)** **=** sin**(**theta**);**  Q**(**q\_float**+**1**,** q\_float **)** **=** **-**sin**(**theta**);**  Q**(**q\_float**+**1**,** q\_float**+**1**)** **=** cos**(**theta**);**  % Q\*[A|I] = [R|Q]  R\_Q **=** Q **\*** R\_Q**;**  **end**  **end**  **end** |

* Fixed-point (Cordic Given’s rotation)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91 | % Fixed point QR factorization with the CORDIC  **function** **[**Q\_cordic**,** R\_cordic**]** **=** Cordic\_QR**(**K\_cordic**,** Q\_scaled**,** R\_scaled**,** row\_R**,** col\_R**,** col\_Q**,** iter\_num**,** R\_sign**,** R\_len**,** R\_frac**,** Q\_sign**,** Q\_len**,** Q\_frac**)**  F **=** fimath**(**'RoundingMethod'**,**'Floor'**);**  Q\_cordic **=** Q\_scaled**;**  R\_cordic **=** R\_scaled**;**    % Eliminate A(q+1,p) by A(q,p)  **for** p\_fix **=** 1 **:** col\_R  **for** q\_fix **=** **(**row\_R**-**1**)** **:** **(-**1**)** **:** p\_fix  % Column q and column q+1 are rotated 180 degrees  **if** R\_cordic**(**q\_fix**,**p\_fix**)** **<** 0  **for** reverse\_R **=** p\_fix **:** col\_R  R\_cordic**(**q\_fix **,**reverse\_R**)** **=** **-**R\_cordic**(**q\_fix **,**reverse\_R**);**  R\_cordic**(**q\_fix**+**1**,**reverse\_R**)** **=** **-**R\_cordic**(**q\_fix**+**1**,**reverse\_R**);**  **end**  **for** reverse\_Q **=** p\_fix **:** col\_Q  Q\_cordic**(**q\_fix **,**reverse\_Q**)** **=** **-**Q\_cordic**(**q\_fix **,**reverse\_Q**);**  Q\_cordic**(**q\_fix**+**1**,**reverse\_Q**)** **=** **-**Q\_cordic**(**q\_fix**+**1**,**reverse\_Q**);**  **end**  **end**  x\_pre **=** zeros**(**col\_Q**,** 1**);**  y\_pre **=** zeros**(**col\_Q**,** 1**);**  **for** iter **=** 0 **:** iter\_num**-**1  % vectoring mode  x\_vect **=** R\_cordic**(**q\_fix **,** p\_fix**);**  y\_vect **=** R\_cordic**(**q\_fix**+**1**,** p\_fix**);**  **[**X\_vect**,** Y\_vect**,** d**]** **=** GG**(**x\_vect**,** y\_vect**,** R\_len**,** R\_frac**,** iter**);**  **if** iter **==** iter\_num**-**1  R\_cordic**(**q\_fix**,** p\_fix**)** **=** fi**((**X\_vect**\***K\_cordic**),**R\_sign**,**R\_len**,**R\_frac**,** F**);**  R\_cordic**(**q\_fix**+**1**,**p\_fix**)=** fi**((**Y\_vect**\***K\_cordic**),**R\_sign**,**R\_len**,**R\_frac**,** F**);**  **else**  R\_cordic**(**q\_fix **,** p\_fix**)** **=** X\_vect**;**  R\_cordic**(**q\_fix**+**1**,** p\_fix**)** **=** Y\_vect**;**  **end**  % rotation mode  **for** rot\_R **=** 1 **:** **(**col\_R**-**p\_fix**)**  x\_rot\_R **=** R\_cordic**(**q\_fix **,** p\_fix**+**rot\_R**);**  y\_rot\_R **=** R\_cordic**(**q\_fix**+**1**,** p\_fix**+**rot\_R**);**  **[**X\_rot\_R**,** Y\_rot\_R**]** **=** GR**(**x\_rot\_R**,** y\_rot\_R**,** d**,** R\_len**,** R\_frac**,** iter**);**    **if** iter **==** iter\_num**-**1  R\_cordic**(**q\_fix**,**p\_fix**+**rot\_R**)=**fi**((**X\_rot\_R**\***K\_cordic**),**R\_sign**,**R\_len**,** R\_frac**,**F**);**  R\_cordic**(**q\_fix**+**1**,**p\_fix**+**rot\_R**)=**fi**((**Y\_rot\_R**\***K\_cordic**),**R\_sign**,**R\_len**,**R\_frac**,**F**);**  **else**  R\_cordic**(**q\_fix **,** p\_fix**+**rot\_R**)** **=** X\_rot\_R**;**  R\_cordic**(**q\_fix**+**1**,** p\_fix**+**rot\_R**)** **=** Y\_rot\_R**;**  **end**  **end**  % compute Q (As the processing of R)  **for** rot\_Q **=** 1 **:** col\_Q  x\_rot\_Q **=** Q\_cordic**(**q\_fix **,** rot\_Q**);**  y\_rot\_Q **=** Q\_cordic**(**q\_fix**+**1**,** rot\_Q**);**  **[**X\_rot\_Q**,** Y\_rot\_Q**]** **=** GR**(**x\_rot\_Q**,** y\_rot\_Q**,** d**,** Q\_len**,** Q\_frac**,** iter**);**    **if** iter **==** iter\_num**-**1  Q\_cordic**(**q\_fix**,**rot\_Q**)=**fi**((**X\_rot\_Q**\***K\_cordic**),**Q\_sign**,**Q\_len**,**Q\_frac**,**F**);**  Q\_cordic**(**q\_fix**+**1**,**rot\_Q**)=**fi**((**Y\_rot\_Q**\***K\_cordic**),**Q\_sign**,**Q\_len**,**Q\_frac**,**F**);**  **else**  Q\_cordic**(**q\_fix **,** rot\_Q**)** **=** X\_rot\_Q**;**  Q\_cordic**(**q\_fix**+**1**,** rot\_Q**)** **=** Y\_rot\_Q**;**  **end**  **end**  **end**  **end**  **end**  **end**  % GG : vectoring mode  **function** **[**X**,** Y**,** d**]** **=** GG**(**x**,** y**,** len**,** frac**,** iter**)**  d **=** **-**sign**(**x **\*** y**);**  X **=** x **-** d **\*** bitsra**(**y**,** iter**);**  Y **=** y **+** d **\*** bitsra**(**x**,** iter**);**    F **=** fimath**(**'RoundingMethod'**,**'Floor'**);**  X **=** fi**(**X**,** 1**,** len**,** frac**,** F**);**  Y **=** fi**(**Y**,** 1**,** len**,** frac**,** F**);**  **end**  % GR : rotation mode  **function** **[**X**,** Y**]** **=** GR**(**x**,** y**,** d**,** len**,** frac**,** iter**)**  X **=** x **-** d **\*** bitsra**(**y**,** iter**);**  Y **=** y **+** d **\*** bitsra**(**x**,** iter**);**    F **=** fimath**(**'RoundingMethod'**,**'Floor'**);**  X **=** fi**(**X**,** 1**,** len**,** frac**,** F**);**  Y **=** fi**(**Y**,** 1**,** len**,** frac**,** F**);**  **end** |

* **Hardware implementation**

1. Hardware architecture ( s=[1 1 0]t, d=[1 0 0]t )

A matrix input

Identity matrix input (I8)

R matrix output

Q matrix output

GG

GR

GR

GR

Q

Q

GG(vectoring mode) : One GG contains 4 micro-rotation modules, when iteration over, it outputs to above and multiplied by K(MK module), the rest of data will be transmitted to the right GR and Q modules.

D

MK

d

Xi Yi iter

xo yo

d

D

MK

d

Xi Yi iter

xo yo

GR(rotation mode) : GR is almost the same module as GG, the only difference is rotation direction d, d will input from the left module(GG or GR).

* There are 4 GG (vectoring mode) and (6+32) GR (rotation mode)
* GG output the direction of vector rotation and transmits it to GR. In each iteration, the rotation angle is halved to approach the target angle.

2. System diagram

wr\_Q\_data\_1~8

12

Q\_RAM

wr\_Q\_1~8

wr\_Q\_addr\_1~8

3

clk

rst

en

R\_RAM

wr\_R

wr\_R\_row\_addr

wr\_R\_col\_addr

wr\_R\_data

3

2

8

A\_ROM

rd\_A

rd\_A\_row\_addr

rd\_A\_col\_addr

rd\_A\_data

3

2

8

qr\_cordic

GG1

GG2

GG3

GG4

GR11

GR12

GR13

GR21

GR22

GR31

Q11

Q21

Q31

Q41

Q12

Q22

Q32

Q42

Q13

Q23

Q33

Q43

Q14

Q24

Q34

Q44

Q17

Q27

Q37

Q47

Q18

Q28

Q38

Q48

Q15

Q25

Q35

Q45

Q16

Q26

Q36

Q46

MK\_Q1

MK\_Q2

MK\_Q3

MK\_Q4

MK\_Q5

MK\_Q6

MK\_Q7

MK\_Q8

MK1

MK2

MK3

MK4

qr\_cordic\_tb

3. Verilog code (only core part):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | **module** GG\_one\_iter **#(**  **parameter** R\_LEN **=** 12**,**  **parameter** R\_FRAC **=** 2  **)(**  **input** **signed** **[**R\_LEN**-**1**:**0**]** xi**,**  **input** **signed** **[**R\_LEN**-**1**:**0**]** yi**,**  **input** **[**3**:**0**]** iter**,**  **output** **[**1**:**0**]** d**,**  **output** **reg** **signed** **[**R\_LEN**-**1**:**0**]** xo**,**  **output** **reg** **signed** **[**R\_LEN**-**1**:**0**]** yo  **);**  **assign** d **=** **(**yi **==** ’d0**)** **?** ’d2 **:** xi**[**R\_LEN**-**1**]** **^** yi**[**R\_LEN**-**1**];**  **always** **@(\*)** **begin**  **if(**d **==** ‘d2**)** **begin**  xo **=** xi**;**  yo **=** yi**;**  **end**  **else** **if(**d **==** ‘d1**)** **begin**  xo **=** xi **-** **(**yi **>>>** iter**);**  yo **=** yi **+** **(**xi **>>>** iter**);**  **end**  **else** **begin**  xo **=** xi **+** **(**yi **>>>** iter**);**  yo **=** yi **-** **(**xi **>>>** iter**);**  **end**  **end**  **endmodule** |

a. GG module

b. GR module

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | **module** GR\_one\_iter **#(**  **parameter** R\_LEN **=** 12**,**  **parameter** R\_FRAC **=** 2  **)**  **(** **input** **signed** **[**R\_LEN**-**1**:**0**]** xi**,**  **input** **signed** **[**R\_LEN**-**1**:**0**]** yi**,**  **input** **[**3**:**0**]** iter**,**  **input** **[**1**:**0**]** d**,**  **output** **reg** **signed** **[**R\_LEN**-**1**:**0**]** xo**,**  **output** **reg** **signed** **[**R\_LEN**-**1**:**0**]** yo  **);**  **always** **@(\*)** **begin**  **if(**d **==** ‘d2**)** **begin**  xo **=** xi**;**  yo **=** yi**;**  **end**  **else** **if(**d **==** ‘d1**)** **begin**  xo **=** xi **-** **(**yi **>>>** iter**);**  yo **=** yi **+** **(**xi **>>>** iter**);**  **end**  **else** **begin**  xo **=** xi **+** **(**yi **>>>** iter**);**  yo **=** yi **-** **(**xi **>>>** iter**);**  **end**  **end**  **endmodule** |

c. Q module is the same module as GR

d. MK (multiplied by K)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **module** MK **#(**  **parameter** R\_LEN **=** 12**,**  **parameter** R\_FRAC **=** 2**,**  **parameter** K\_LEN **=** 10**,**  **parameter** K\_FRAC **=** 9  **)(**  **input** **signed** **[**R\_LEN**-**1**:**0**]** xi**,**  **input** **signed** **[**R\_LEN**-**1**:**0**]** yi**,**  **output** **signed** **[**R\_LEN**-**1**:**0**]** xo**,**  **output** **signed** **[**R\_LEN**-**1**:**0**]** yo  **);**  **localparam** **signed** K **=** 10'b0\_100110111**;** // K = 0.607421875  **wire** **signed** **[**R\_LEN**+**K\_LEN**-**1**:**0**]** xo\_0**;**  **wire** **signed** **[**R\_LEN**+**K\_LEN**-**1**:**0**]** yo\_0**;**  **assign** xo\_0 **=** xi **\*** K**;**  **assign** yo\_0 **=** yi **\*** K**;**  // truncate to R\_LEN bits  **assign** xo **=** xo\_0**[**R\_LEN**+**K\_FRAC**-**1**:**K\_FRAC**];**  **assign** yo **=** yo\_0**[**R\_LEN**+**K\_FRAC**-**1**:**K\_FRAC**];**  **endmodule** |

e. GG iteration control

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | // iteration times  **always** **@(posedge** clk **or** **posedge** rst**)** **begin**  **if** **(**rst**)** **begin**  iter\_gg1 **<=** 'd0**;**  **end**  **else** **if(**ROT\_wire**)** **begin**  **if(**nop\_gg1**)** **begin**  iter\_gg1 **<=** 'd0**;**  **end**  **else** **if(**iter\_last\_gg1**)** **begin**  iter\_gg1 **<=** iter\_gg1 **+** 'd1**;**  **end**  **else** **begin**  iter\_gg1 **<=** iter\_gg1 **+** ITER\_ONE\_CYCLE**;**  **end**  **end**  **else** **begin**  iter\_gg1 **<=** 'd0**;**  **end**  **end**  // GG1 input data xi, yi  **always** **@(posedge** clk **or** **posedge** rst**)** **begin**  **if** **(**rst**)** **begin**  xi\_gg1 **<=** 'd0**;**  yi\_gg1 **<=** 'd0**;**  **end**  **else** **if(**OP\_wire**)** **begin**  **case(**iter\_gg1**)**  0**:** **begin**  **if(**start\_gg1**)** **begin**  xi\_gg1 **<=** 'd0**;**  yi\_gg1 **<=** rd\_A\_data\_ext**;**  **end**  **else** **if(**nop\_gg1 **&&** **!**finish\_gg1**)** **begin**  xi\_gg1 **<=** rd\_A\_data\_ext**;**  yi\_gg1 **<=** yo\_gg1**;**  **end**  **else** **begin**  xi\_gg1 **<=** xo\_gg1**;**  yi\_gg1 **<=** yo\_gg1**;**  **end**  **end**  ITER\_K**:** **begin**  **if(**finish\_gg1**)** **begin**  xi\_gg1 **<=** xo\_gg1**;**  yi\_gg1 **<=** yo\_gg1**;**  **end**  **else** **begin**  xi\_gg1 **<=** rd\_A\_data\_ext**;**  yi\_gg1 **<=** xo\_mk1**;**  **end**  **end**  **default:** **begin**  xi\_gg1 **<=** xo\_gg1**;**  yi\_gg1 **<=** yo\_gg1**;**  **end**  **endcase**  **end**  **else** **begin**  xi\_gg1 **<=** 'd0**;**  yi\_gg1 **<=** 'd0**;**  **end**  **end**  // GG1 mk\_count  **always** **@(posedge** clk **or** **posedge** rst**)** **begin**  **if** **(**rst**)** **begin**  mk\_count\_gg1 **<=** 'd0**;**  **end**  **else** **if(**multk\_gg1**)** **begin**  mk\_count\_gg1 **<=** mk\_count\_gg1 **+** 'd1**;**  **end**  **end** |

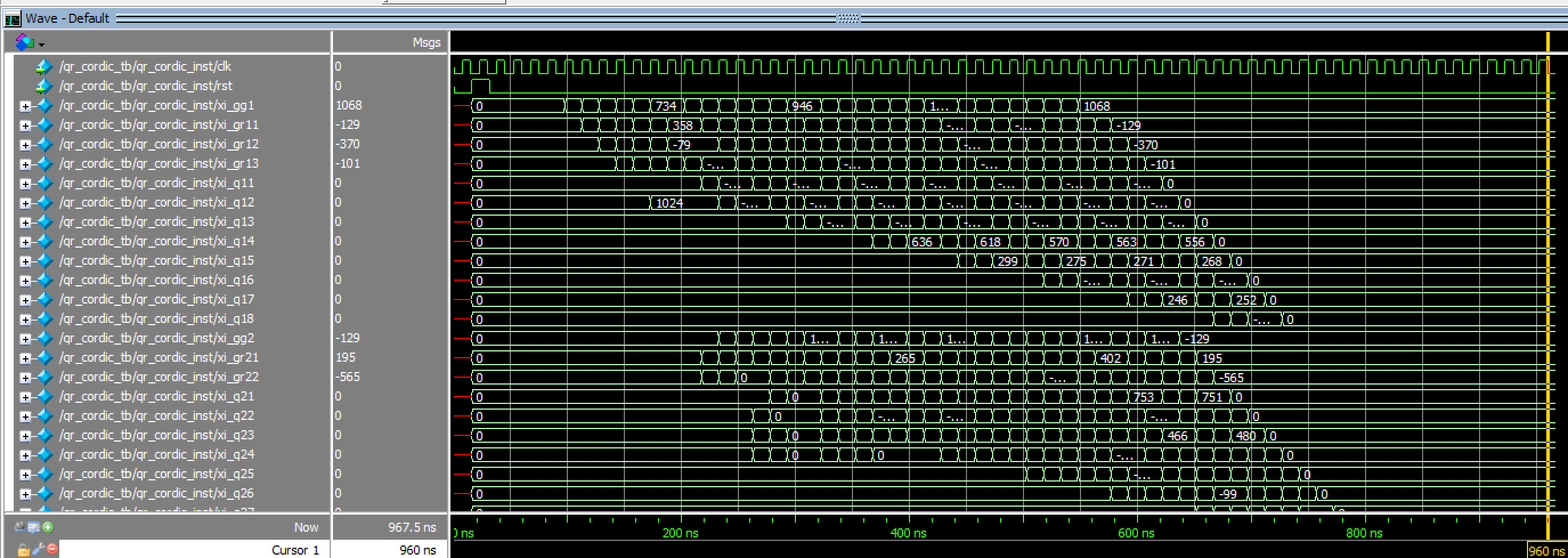
f. GR iteration control

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | // data propagated from left to right  **always** **@(posedge** clk **or** **posedge** rst**)** **begin**  **if** **(**rst**)** **begin**  iter\_gr11 **<=** 'd0**;**  nop\_gr11 **<=** 'd0**;**  d1\_gr11 **<=** 'd0**;**  d2\_gr11 **<=** 'd0**;**  d3\_gr11 **<=** 'd0**;**  d4\_gr11 **<=** 'd0**;**  neg\_gr11 **<=** 'd0**;**  mk\_count\_gr11 **<=** 'd0**;**  **end**  **else** **begin**  iter\_gr11 **<=** iter\_gg1**;**  nop\_gr11 **<=** nop\_gg1**;**  d1\_gr11 **<=** d1\_gg1**;**  d2\_gr11 **<=** d2\_gg1**;**  d3\_gr11 **<=** d3\_gg1**;**  d4\_gr11 **<=** d4\_gg1**;**  neg\_gr11 **<=** neg\_gg1**;**  mk\_count\_gr11 **<=** mk\_count\_gg1**;**  **end**  **end**  // GR11 input data xi, yi  **always** **@(posedge** clk **or** **posedge** rst**)** **begin**  **if** **(**rst**)** **begin**  xi\_gr11 **<=** 'd0**;**  yi\_gr11 **<=** 'd0**;**  **end**  **else** **if(**OP\_wire**)** **begin**  **case(**iter\_gr11**)**  0**:** **begin**  **if(**start\_gr11\_reg**)** **begin**  xi\_gr11 **<=** 'd0**;**  yi\_gr11 **<=** rd\_A\_data\_ext**;**  **end**  **else** **if(**nop\_gr11 **&&** **!**finish\_gr11**)** **begin**  xi\_gr11 **<=** rd\_A\_data\_ext**;**  yi\_gr11 **<=** yo\_gr11**;**  **end**  **else** **begin**  xi\_gr11 **<=** xo\_gr11**;**  yi\_gr11 **<=** yo\_gr11**;**  **end**  **end**  ITER\_K**:** **begin**  **if(**finish\_gr11**)** **begin**  xi\_gr11 **<=** xo\_mk1**;**  yi\_gr11 **<=** yo\_mk1**;**  **end**  **else** **begin**  xi\_gr11 **<=** rd\_A\_data\_ext**;**  yi\_gr11 **<=** xo\_mk1**;**  **end**  **end**  **default:** **begin**  xi\_gr11 **<=** xo\_gr11**;**  yi\_gr11 **<=** yo\_gr11**;**  **end**  **endcase**  **end**  **else** **begin**  xi\_gr11 **<=** 'd0**;**  xi\_gr11 **<=** 'd0**;**  **end**  **end** |

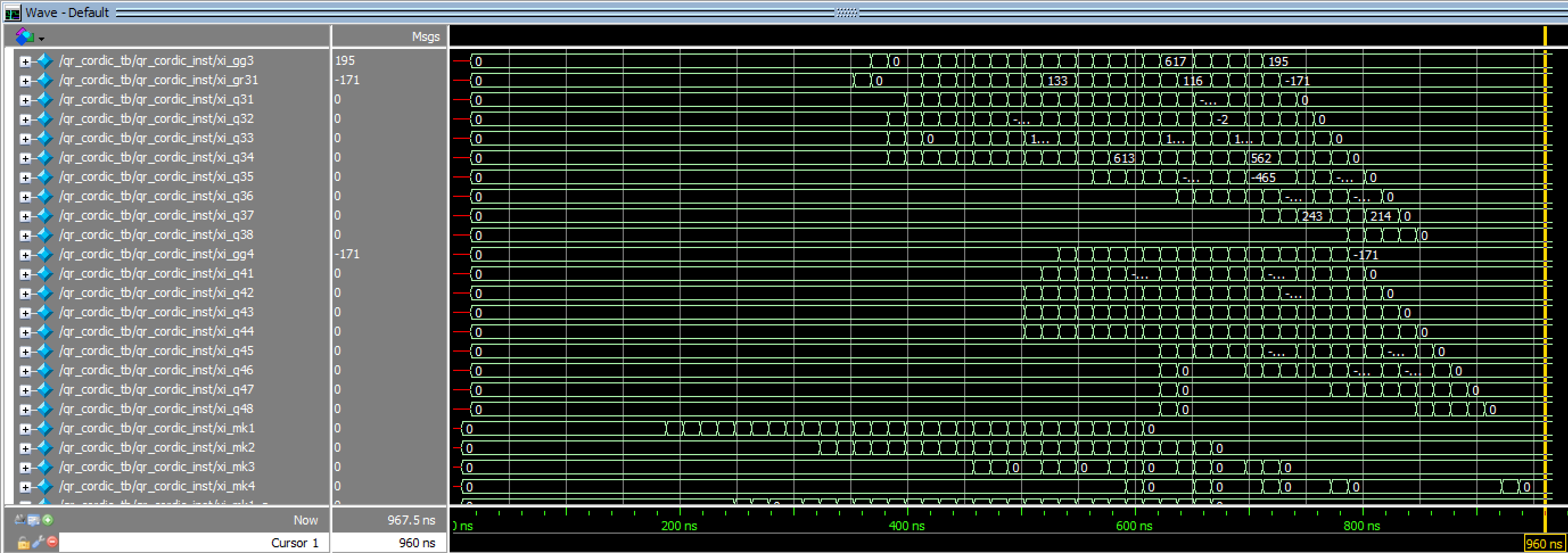
4. RTL Simulation

a. Simulation waveform (Data propagate form left to right PE (GG, GR, Q))

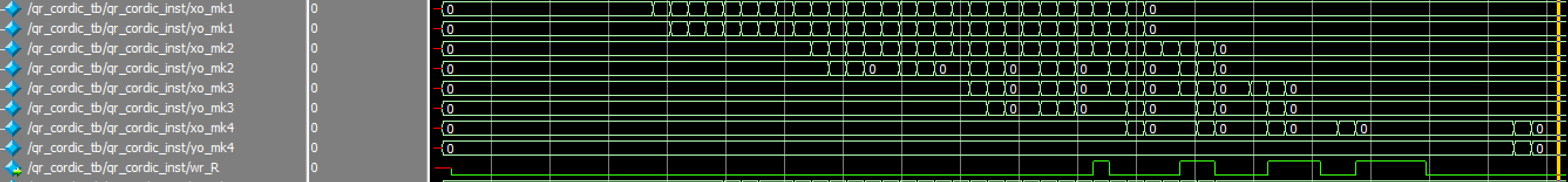
* Data propagate : GG1 →Q18, GG2 →Q28



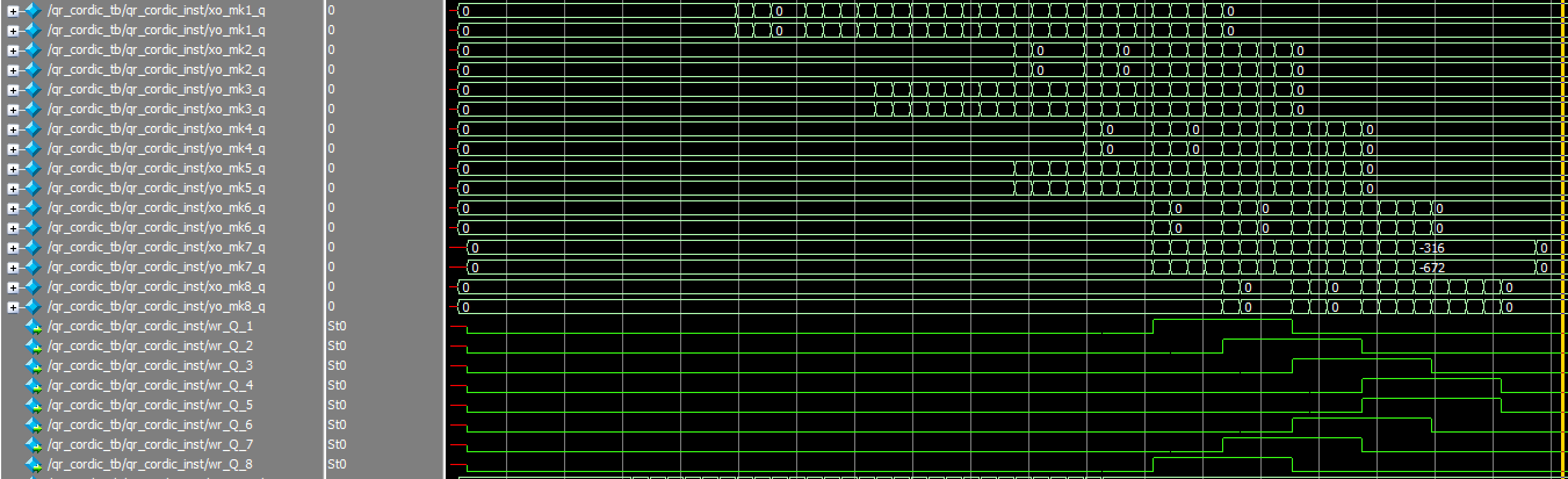
* Data propagate : GG3 →Q38, GG4 →Q48



* Output (MK output R)



* Output (MK output Q)



* Data flow & data scheduling

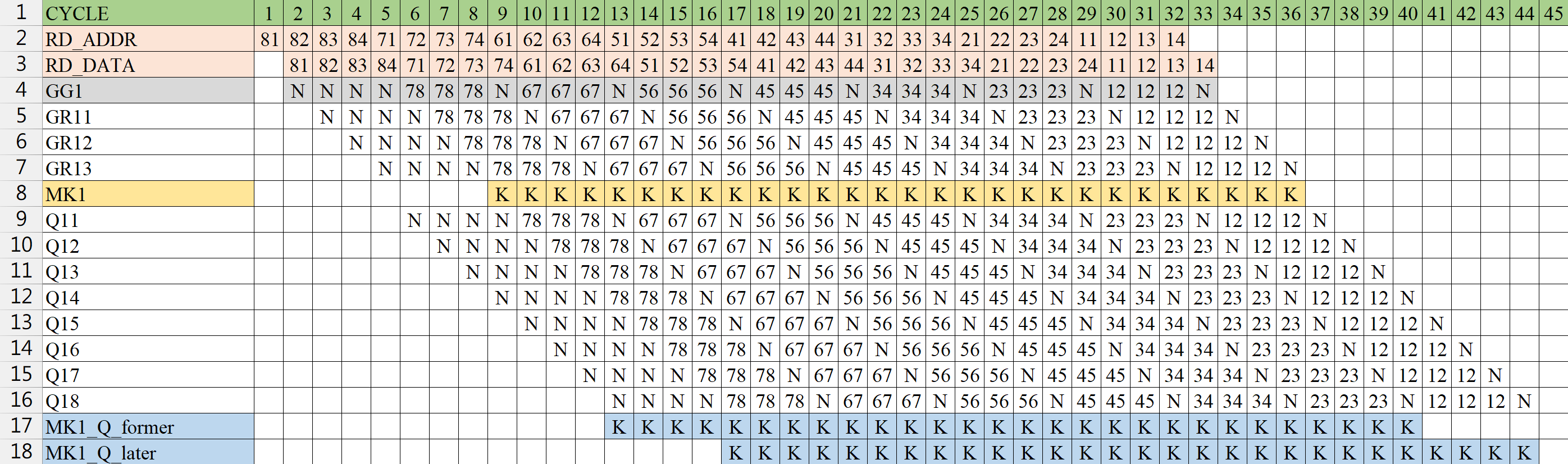
GG1→GR11→GR12→GR13→Q11→Q12→Q13→…..→Q18

-------- GG2→GR21→GR22→Q21→Q22→Q23→…..→Q28

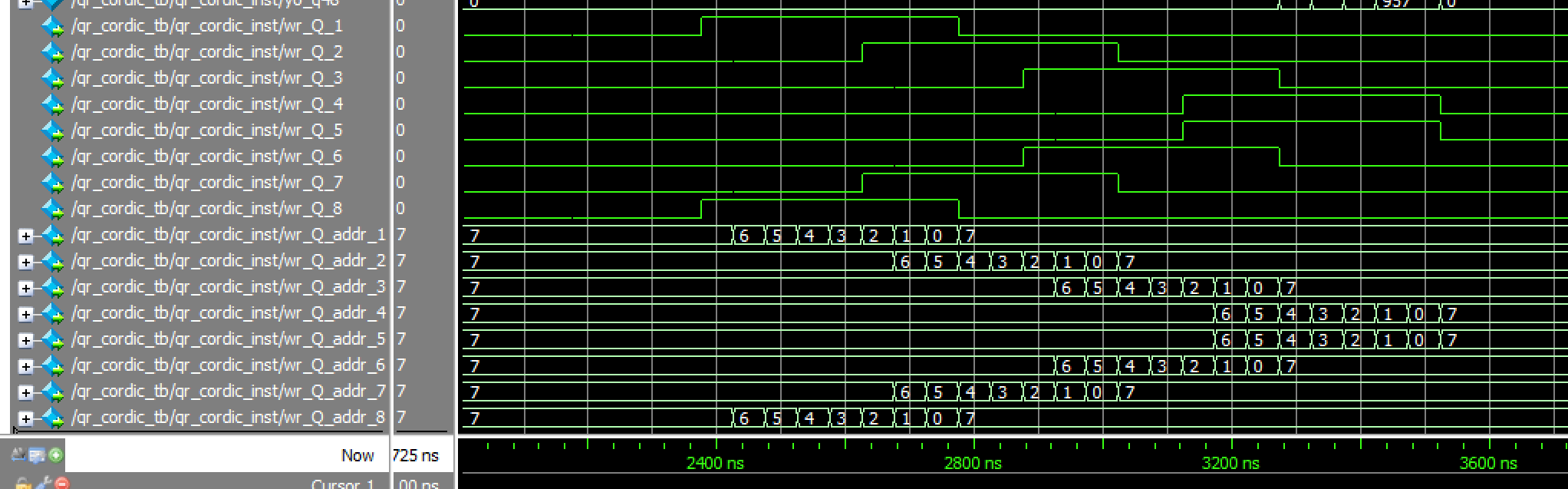
---------------- GG3→GR31→Q31→Q32→Q33→…..→Q38

-------------------------- GG4→Q41→Q42→Q43→…..→Q48

* Corresponding Timing diagram

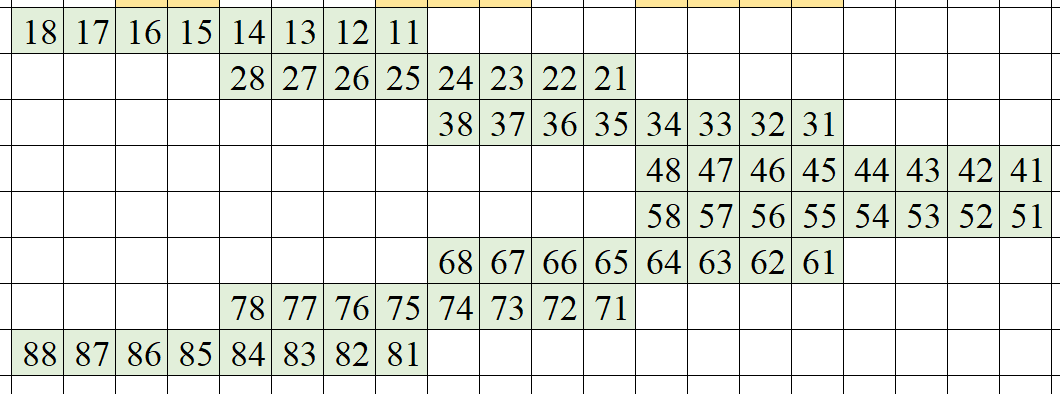


* Write Q matrix data back

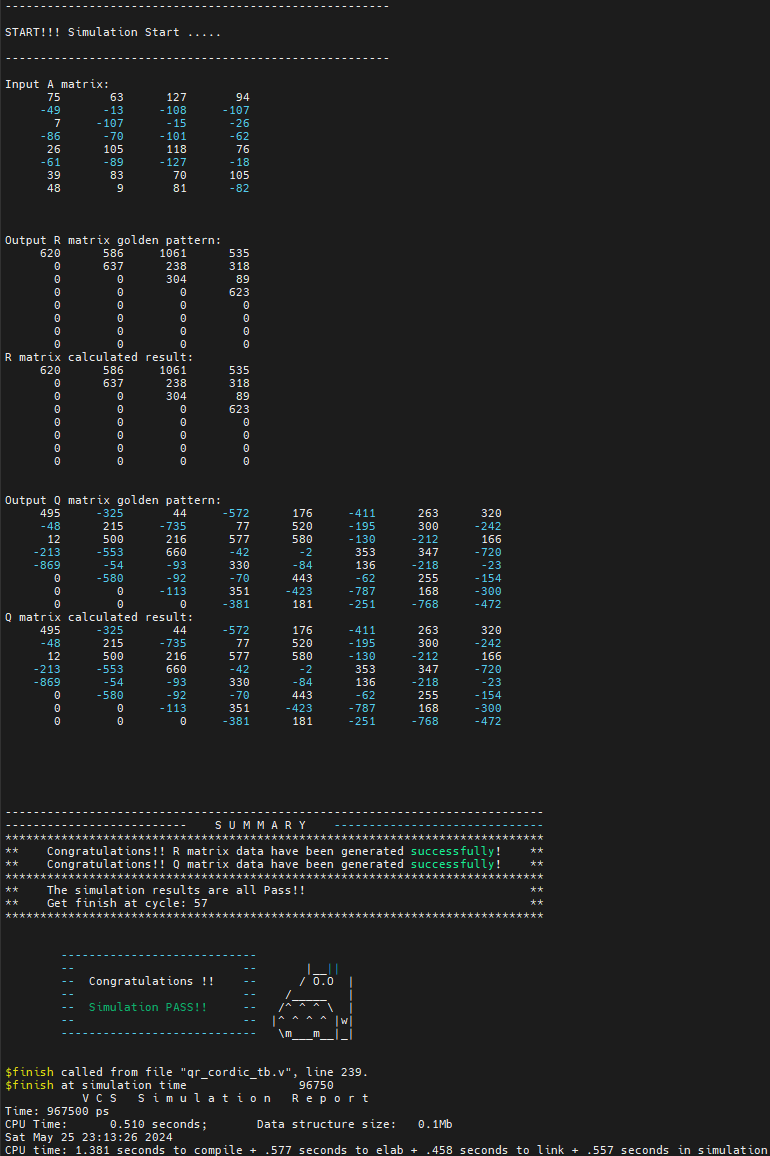
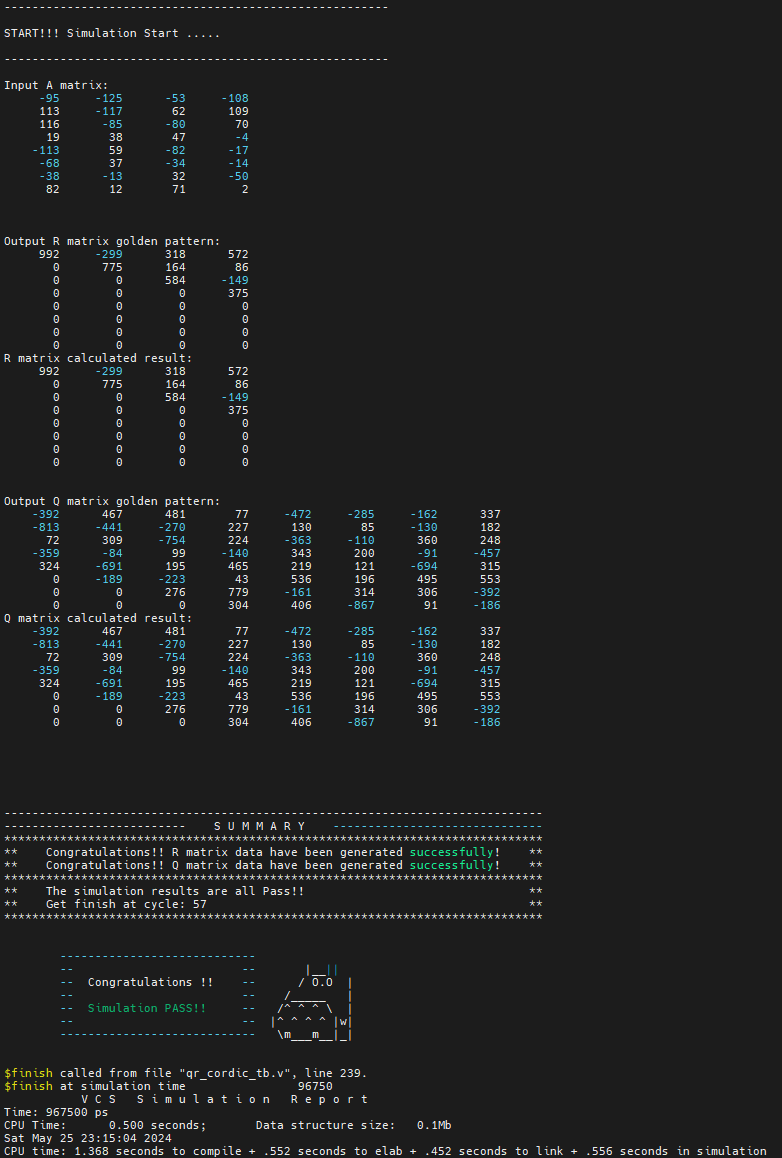


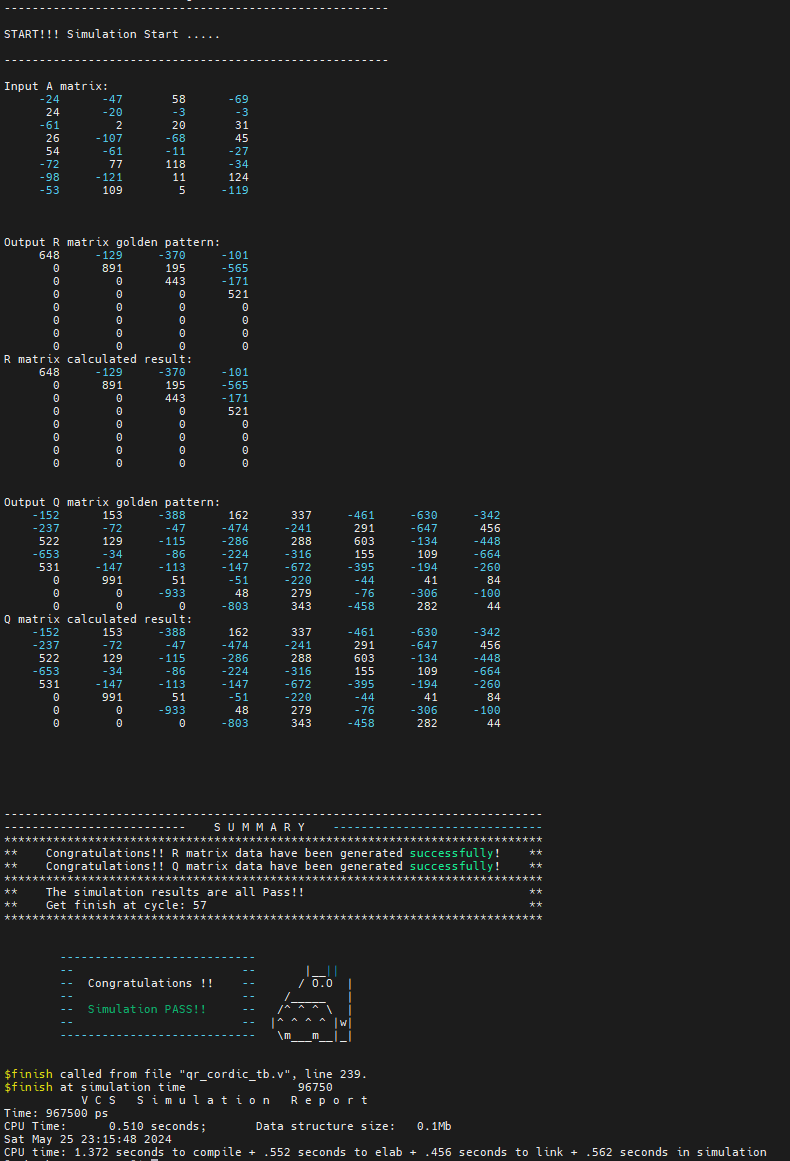
Each row has an independent write enable signal, since multiple data outputs from Q occur simultaneously in the systolic array.

* Corresponding Timing diagram



b. Simulation transcript (4 patterns)

Pattern 1: Pattern 2:

Pattern 3: Pattern4:

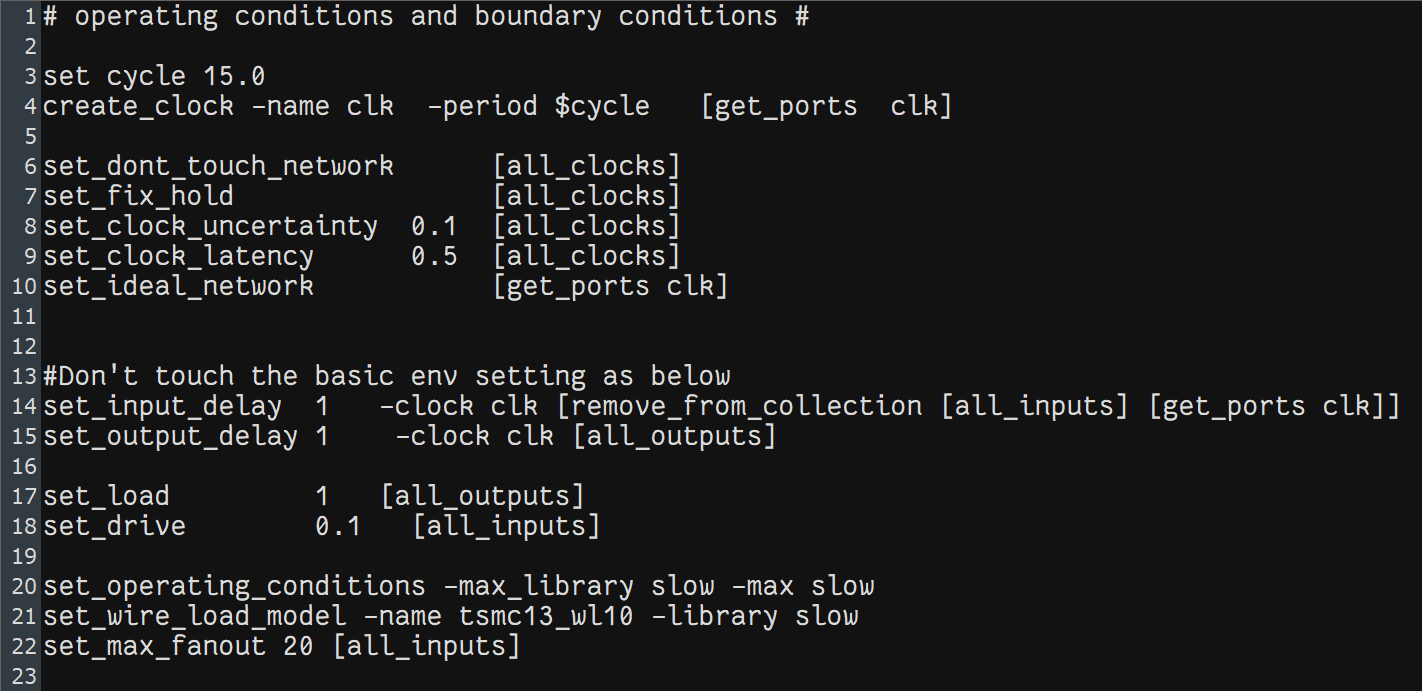
* Both R and Q are all pass in RTL simulation
* Clock rate : 15ns
* Cycles : 57
* Run time : 967.5ns

RTL simulation Commands (VCS):

vcs -full64 -R -sverilog qr\_cordic\_tb.v qr\_cordic.v +access+r +vcs+fsdbon +fsdb+mda +fsdbfile+qr\_cordic.fsdb

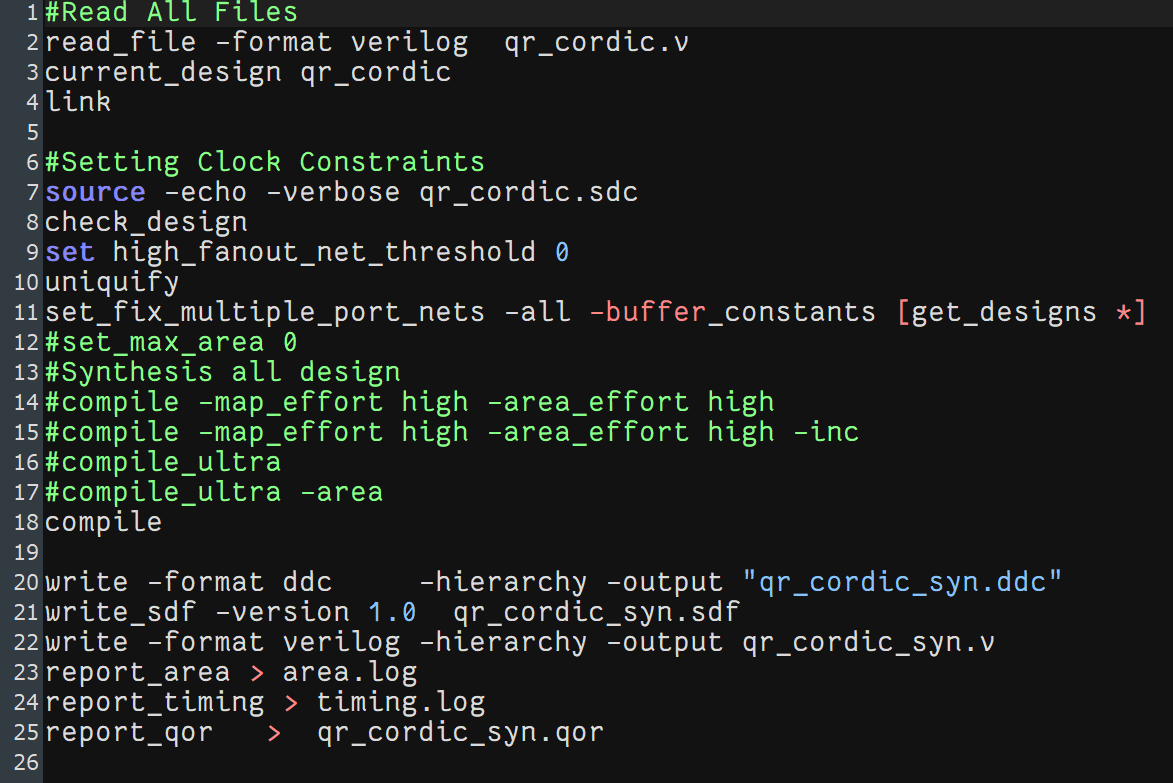
5. Synthesis

a. Synopsys Design Constraints (SDC)



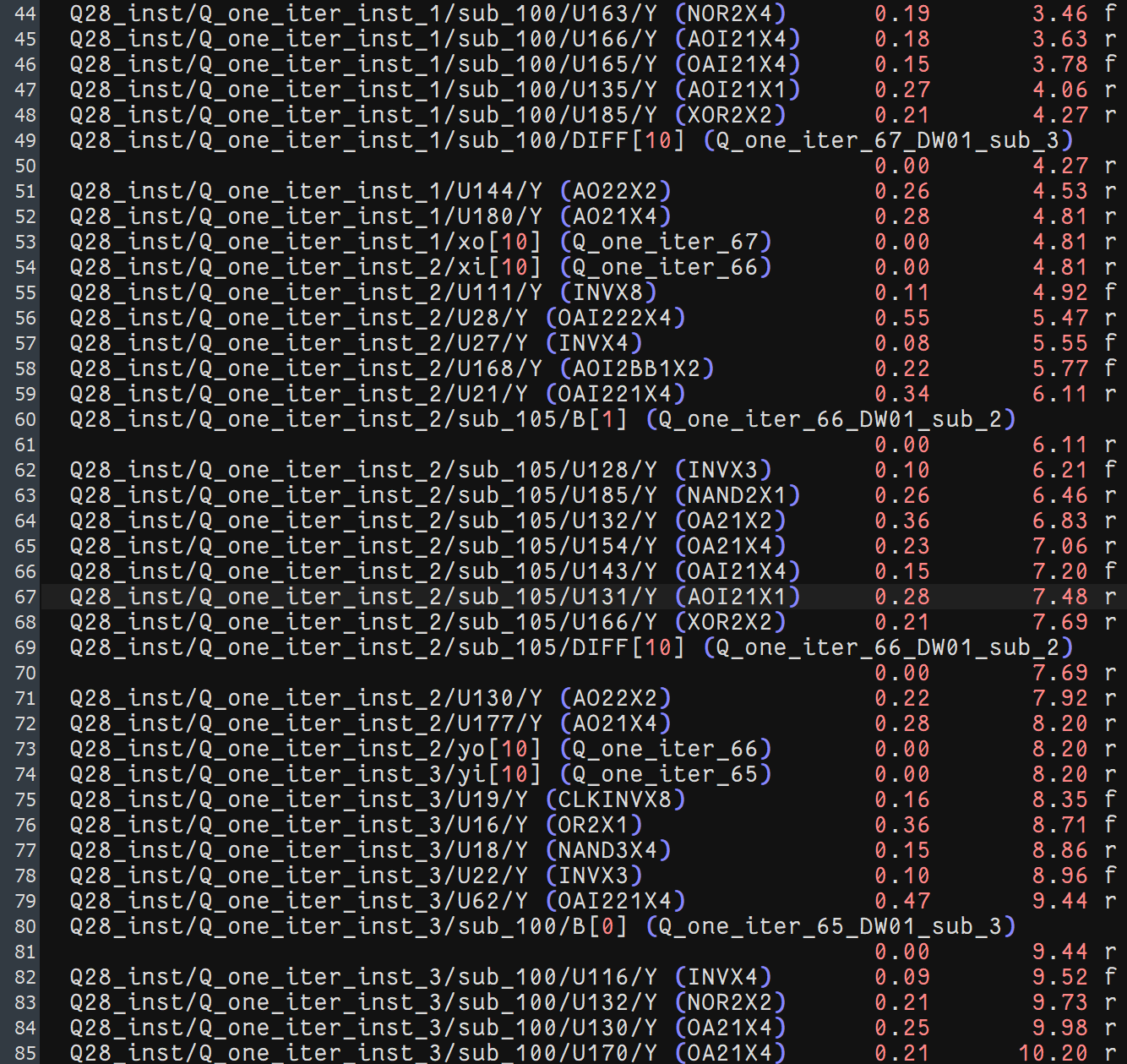
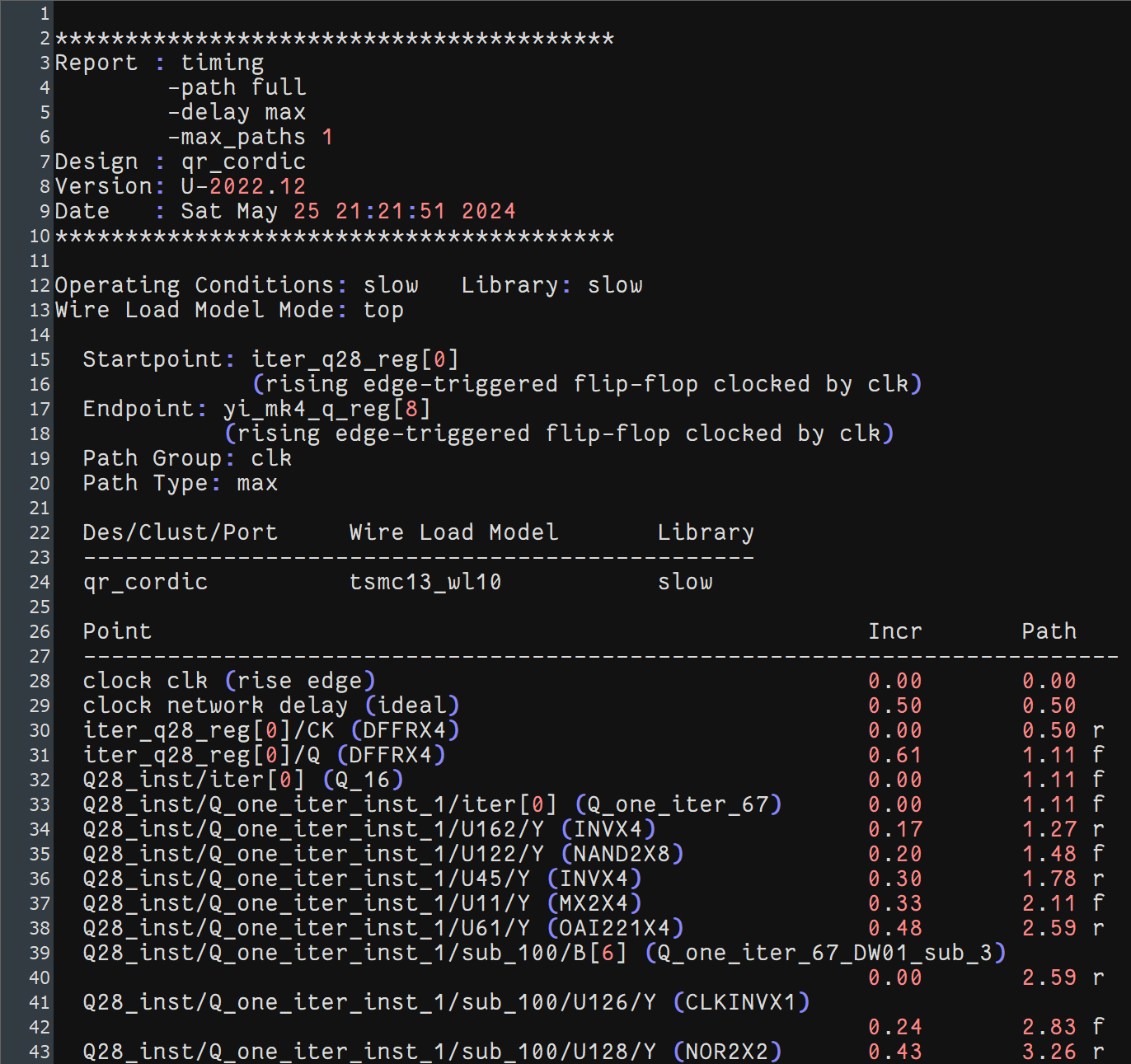
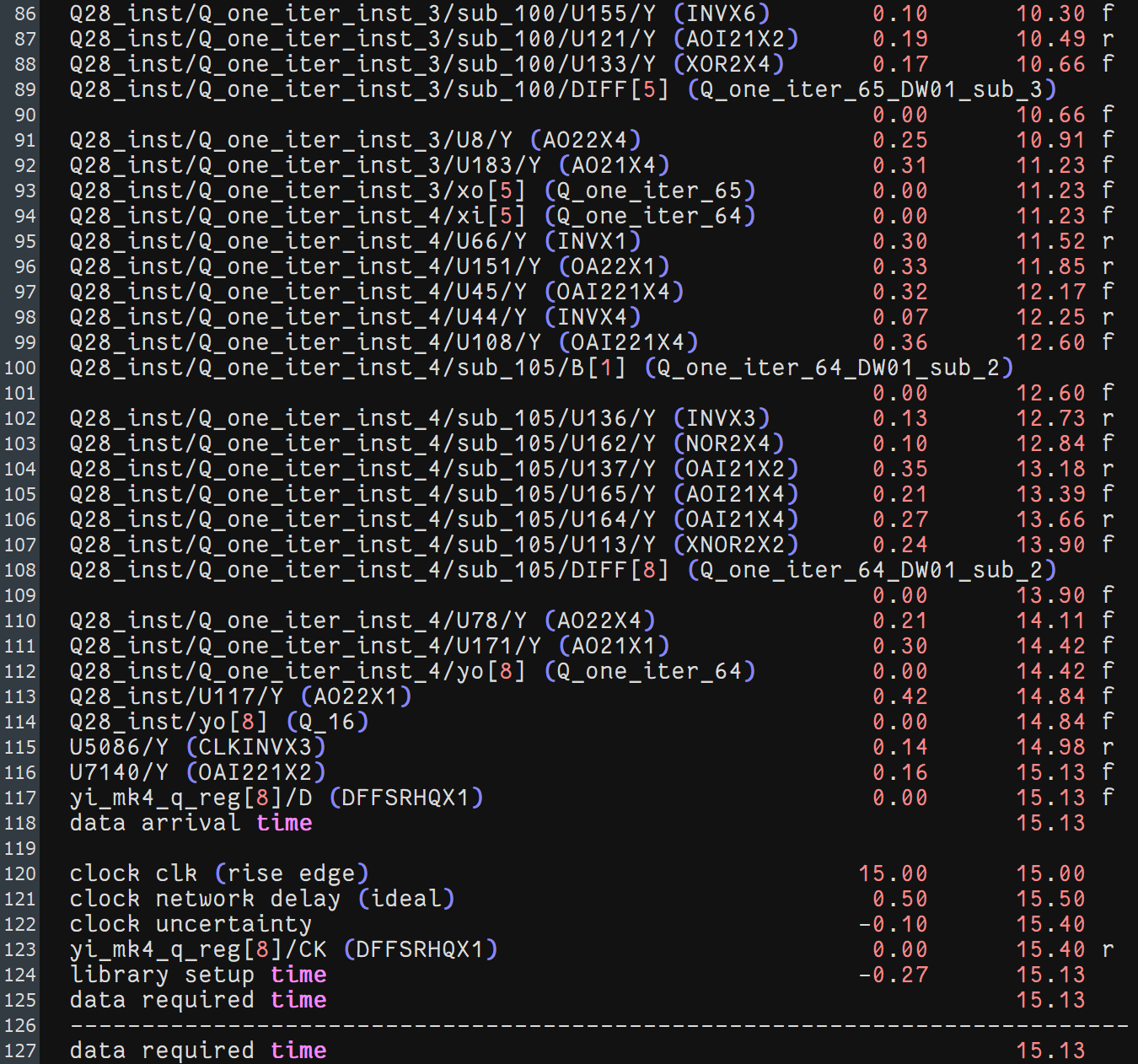
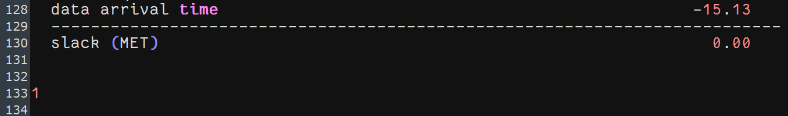
* Clock rate = 15ns
* I/O delay = 1ns

b. Design compiler synthesis .tcl



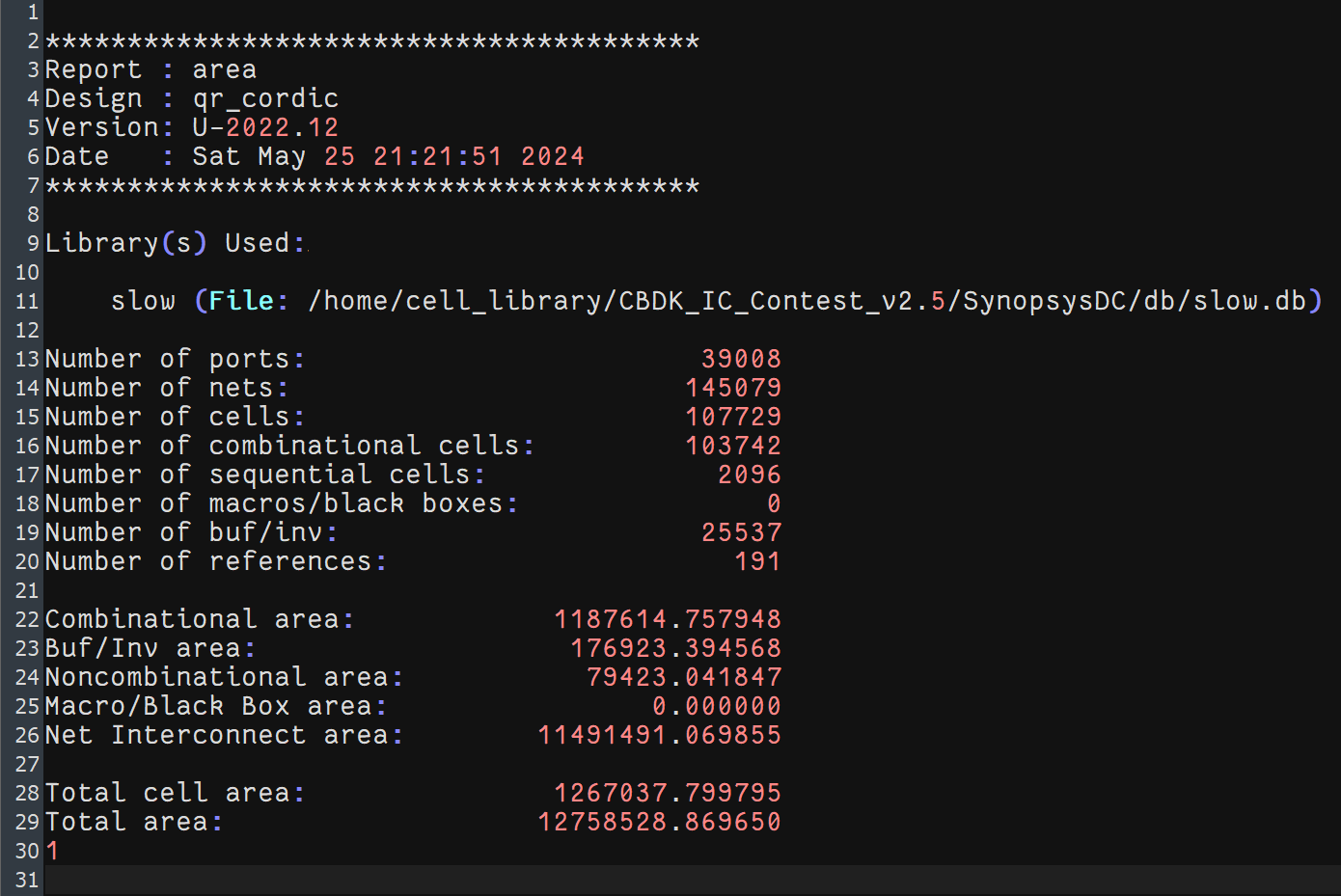
* Use Standard synthesis run without specifying additional optimization efforts for mapping or area.
* Synthesis on the CBDK\_IC\_Contest\_v2.5 and TSMC 130nm process

c. Timing report



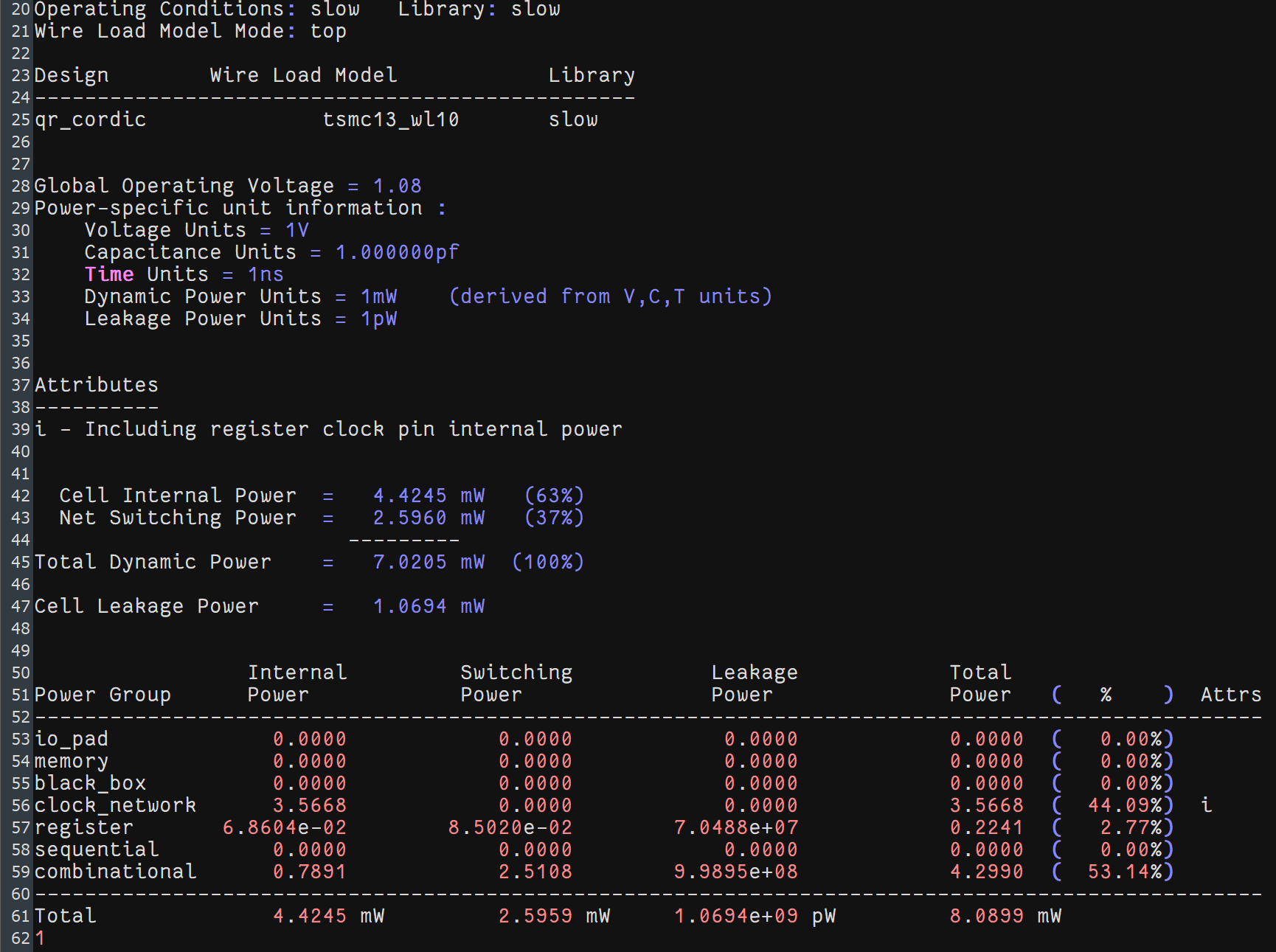
* Clock rate = 15ns
* Slack **≧** 0ns

d. Area report



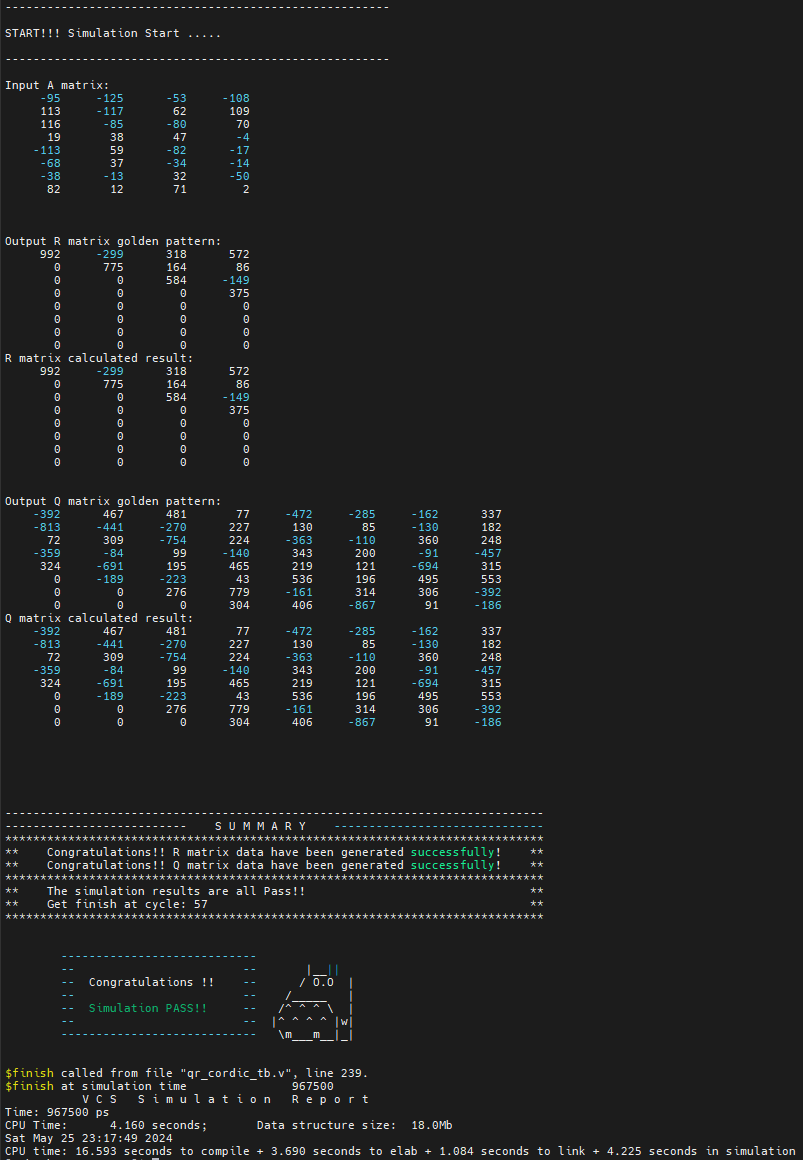
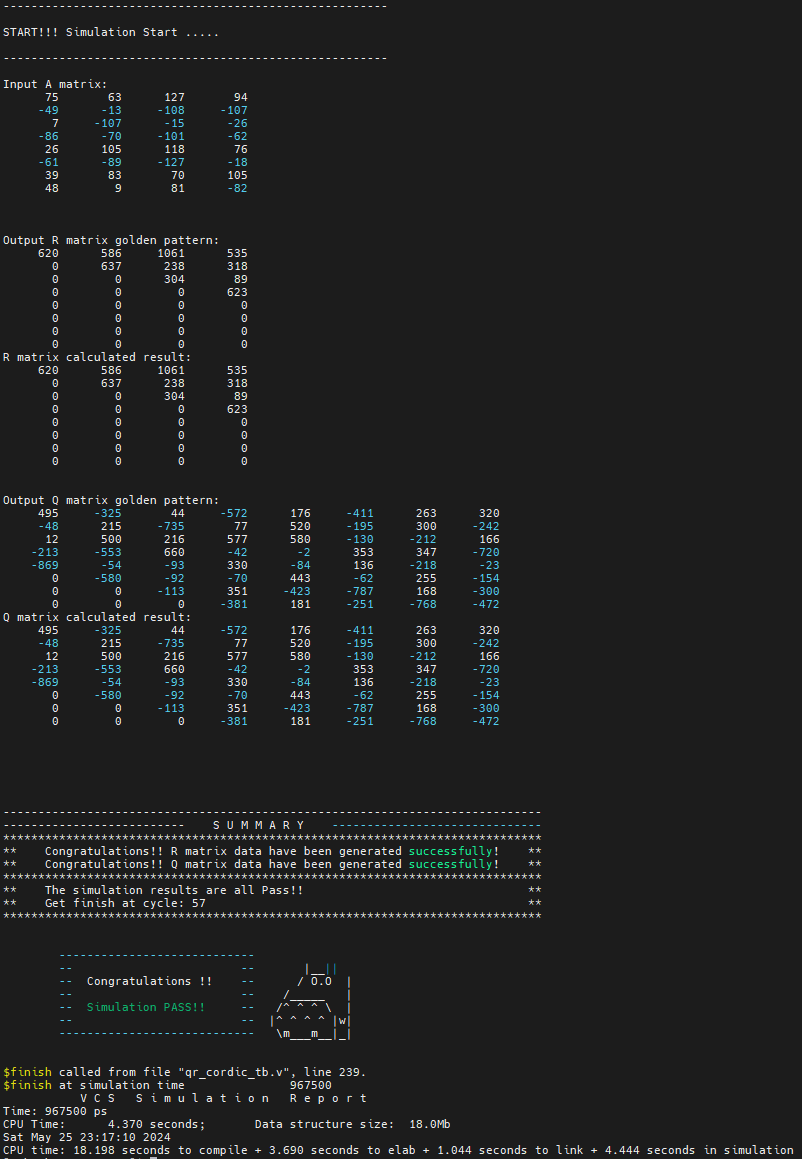
* Area: 12567037.8 µm² (clock rate:15ns, standard compile)

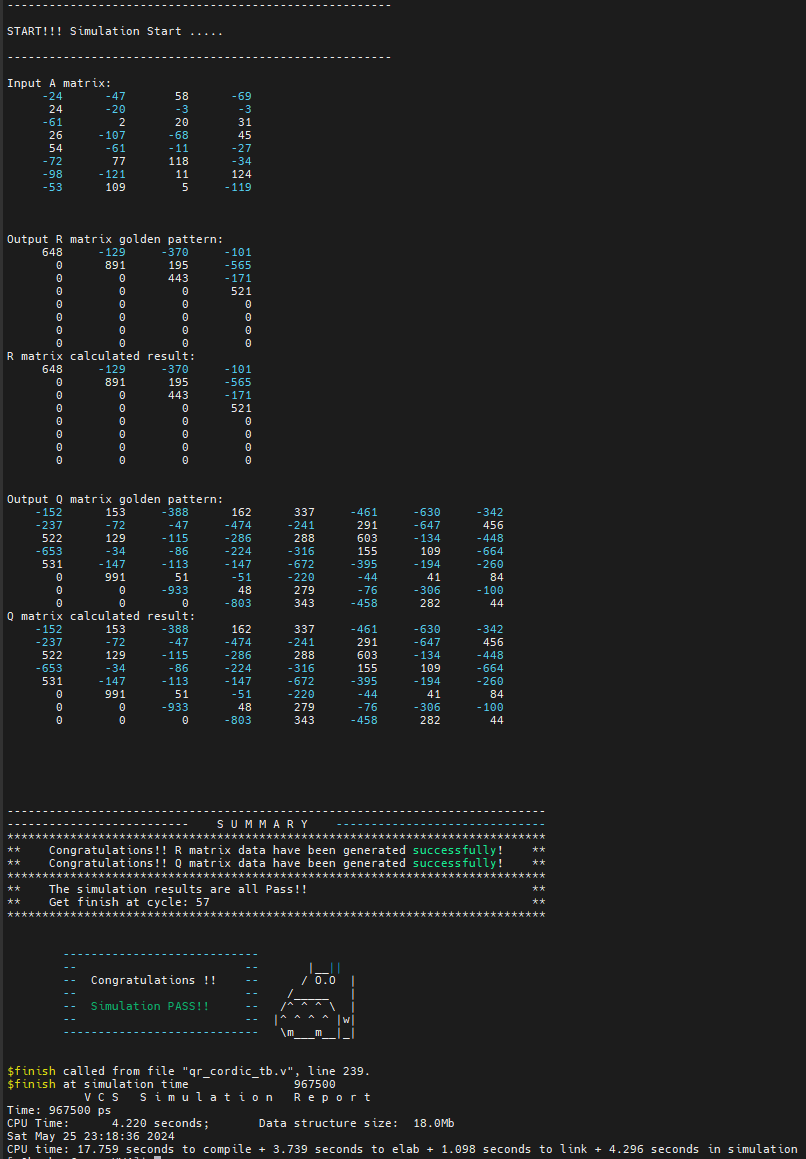
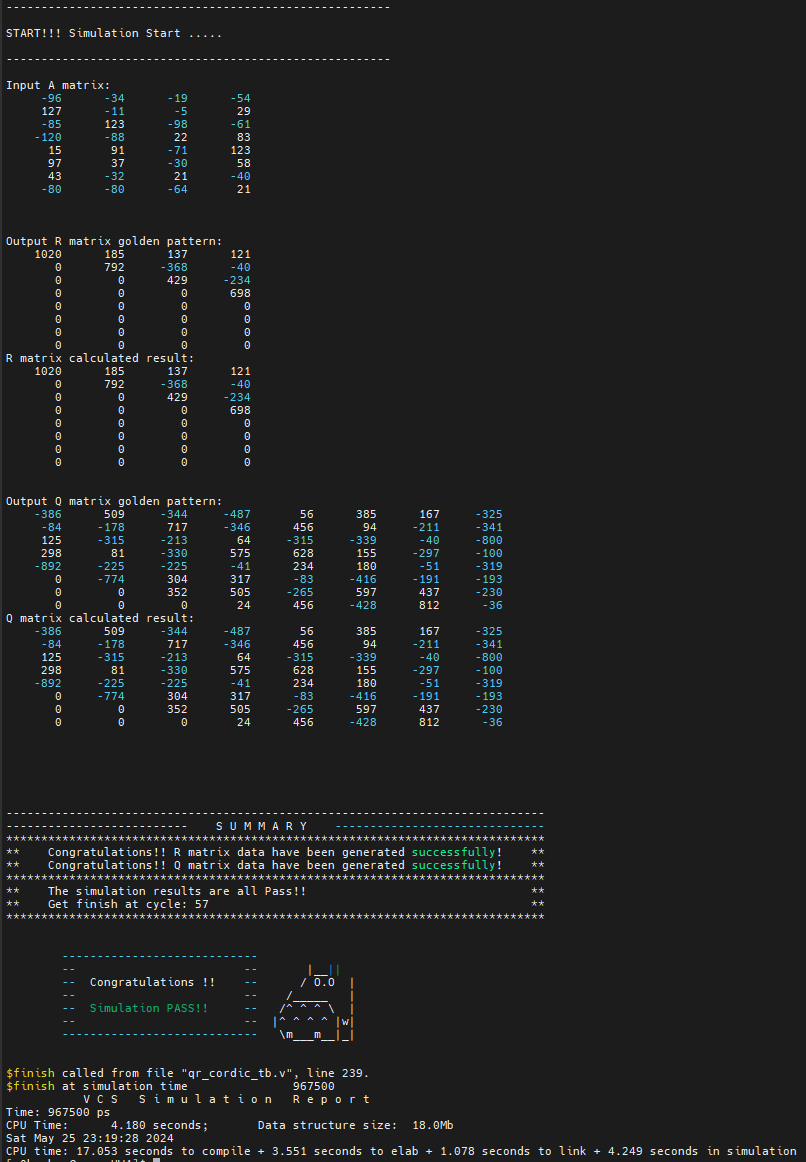
e. Power



* Total Power : 8.0899mW

f. Gate-level simulation (Run time : 1117.5ns)

Pattern 1: Pattern 2:

Pattern 3: Pattern4:

* Both R and Q are all pass in gate-level simulation
* Clock rate : 15ns
* Cycles : 57
* Run time : 967.5ns

Gate-level simulation Commands (VCS):

vcs -full64 -R -sverilog qr\_cordic\_tb.v qr\_cordic\_syn.v +define+SDF +access+r +neg\_tchk +vcs+fsdbon +fsdb+mda +fsdbfile+qr\_cordic.fsdb -v /home/cell\_library/CBDK\_IC\_Contest\_v2.5/Verilog/tsmc13\_neg.v +maxdelays

* The complete code and related data are placed on [GitHub](https://github.com/p810105b/VLSI-DPS-HW/tree/master/HW4)