# DSP in VLSI

# Final Project

# Solving Linear Equations via QR Decomposition

*(Due by 06/11 11:59 AM)*

In the final project, we will learn to use QR decomposition for solving linear equations and implement high throughput hardware for it. You can use all kinds of design techniques that you have learned in this course to achieve the good area- and processing time product. Upload all your programs (C/Matlab/Python/Verilog) and reports to NTUCool. For students who use ADFP, please put all your Verilog file/Synthesis results in one folder "SubmittedFiles". TA will download it. For students who use FPGA flow, please put all your Verilog/synthesis results in the project folder and upload the compressed project folder.

## A. Algorithm

If $\mathbf{A}$ is an $N \times N$ real matrix, given the column vectors $\mathbf{y} = [y_1 \quad y_2 \quad \cdots \quad y_N]^{\boldsymbol{T}}$ and $\mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_N]^{\boldsymbol{T}}$, to solve the system of linear equations with real coefficients $\mathbf{y} = \mathbf{Ax}$, a common method is to triangularize the coefficient matrix $\mathbf{A}$. Using QR decomposition, we can write $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ is an orthonormal matrix, and $\mathbf{R}$ is an upper triangular matrix. Therefore

$$\mathbf{z} = \mathbf{Q}^T\mathbf{y} = \mathbf{Q}^T\mathbf{Ax} = \mathbf{Rx}. \tag{1}$$

Namely,

$$\mathbf{z} = \mathbf{Q}^T\mathbf{y} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_N \end{bmatrix} = \mathbf{Rx} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} & \cdots & r_{1,N} \\ 0 & r_{2,2} & r_{2,3} & r_{2,4} & \cdots & r_{2,N} \\ & \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & \cdots & r_{N-1,N-1} & r_{N-1,N} \\ 0 & \cdots & 0 & & \cdots & r_{N,N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} \tag{2}$$

Therefore, the unknowns $x_n$ can be solved sequentially by back-substitution starting from the $N$th row.

There are three methods to realize QR decomposition, including Gram-Schmidt algorithm, Householder transformation and Givens rotation. The pseudo code of Givens rotation can be described as follows.

$\mathbf{R} = \mathbf{A}$

$\mathbf{Q} = \mathbf{I}$

for $i = 1{:}N$

for $j = i + 1{:}N$

$$\alpha = \|[r_{i,i} \quad r_{j,i}]^T\|$$

$$c = r_{i,i}/\alpha \tag{3}$$

$$s = r_{j,i}/\alpha \tag{4}$$

$$r_{i,i:N} = cr_{i,i:N} + sr_{j,i:N} \tag{5}$$

$$r_{j,i:N} = -sr_{i,i:N} + cr_{j,i:N} \tag{6}$$

$$q_{i,1:N} = cq_{i,1:N} + sq_{j,1:N}$$

$$q_{j,1:N} = -sq_{i,1:N} + cq_{j,1:N}$$

end

end

Using a $4 \times 4$ matrix as an example, the operation of the Givens Rotation is as follows

(i) Consider $r_{1,1}$ and $r_{2,1}$. Use $r_{1,1}$ to eliminate $r_{2,1}$.

$$\begin{bmatrix} c^{(1)} & s^{(1)} & 0 & 0 \\ -s^{(1)} & c^{(1)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} \\ r_{2,1} & r_{2,2} & r_{2,3} & r_{2,4} \\ r_{3,1} & r_{3,2} & r_{3,3} & r_{3,4} \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix} = \begin{bmatrix} r_{1,1}^{(1)} & r_{1,2}^{(1)} & r_{1,3}^{(1)} & r_{1,4}^{(1)} \\ 0 & r_{2,2}^{(1)} & r_{2,3}^{(1)} & r_{2,4}^{(1)} \\ r_{3,1} & r_{3,2} & r_{3,3} & r_{3,4} \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix}$$

(ii) Consider $r_{1,1}$ and $r_{3,1}$. Use $r_{1,1}$to eliminate $r_{3,1}$.

$$\begin{bmatrix} c^{(2)} & 0 & s^{(2)} & 0 \\ 0 & 1 & 0 & 0 \\ -s^{(2)} & 0 & c^{(2)} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1}^{(1)} & r_{1,2}^{(1)} & r_{1,3}^{(1)} & r_{1,4}^{(1)} \\ 0 & r_{2,2}^{(1)} & r_{2,3}^{(1)} & r_{2,4}^{(1)} \\ r_{3,1} & r_{3,2} & r_{3,3} & r_{3,4} \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix} = \begin{bmatrix} r_{1,1}^{(2)} & r_{1,2}^{(2)} & r_{1,3}^{(2)} & r_{1,4}^{(2)} \\ 0 & r_{2,2}^{(1)} & r_{2,3}^{(1)} & r_{2,4}^{(1)} \\ 0 & r_{3,2}^{(1)} & r_{3,3}^{(1)} & r_{3,4}^{(1)} \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix}$$

(iii) Consider $r_{1,1}$ and $r_{4,1}$. Use $r_{1,1}$to eliminate $r_{4,1}$.

$$\begin{bmatrix} c^{(3)} & 0 & 0 & s^{(3)} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s^{(3)} & 0 & 0 & c^{(3)} \end{bmatrix} \begin{bmatrix} r_{1,1}^{(2)} & r_{1,2}^{(2)} & r_{1,3}^{(2)} & r_{1,4}^{(2)} \\ 0 & r_{2,2}^{(1)} & r_{2,3}^{(1)} & r_{2,4}^{(1)} \\ 0 & r_{3,2}^{(1)} & r_{3,3}^{(1)} & r_{3,4}^{(1)} \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix} = \begin{bmatrix} r_{1,1}^{(3)} & r_{1,2}^{(3)} & r_{1,3}^{(3)} & r_{1,4}^{(3)} \\ 0 & r_{2,2}^{(1)} & r_{2,3}^{(1)} & r_{2,4}^{(1)} \\ 0 & r_{3,2}^{(1)} & r_{3,3}^{(1)} & r_{3,4}^{(1)} \\ 0 & r_{4,2}^{(1)} & r_{4,3}^{(1)} & r_{4,4}^{(1)} \end{bmatrix}$$

(iv) Consider $r_{2,2}$ and $r_{3,2}$. Use $r_{2,2}$to eliminate $r_{3,2}$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c^{(4)} & s^{(4)} & 0 \\ 0 & -s^{(4)} & c^{(4)} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1}^{(3)} & r_{1,2}^{(3)} & r_{1,3}^{(3)} & r_{1,4}^{(3)} \\ 0 & r_{2,2}^{(1)} & r_{2,3}^{(1)} & r_{2,4}^{(1)} \\ 0 & r_{3,2}^{(1)} & r_{3,3}^{(1)} & r_{3,4}^{(1)} \\ 0 & r_{4,2}^{(1)} & r_{4,3}^{(1)} & r_{4,4}^{(1)} \end{bmatrix} = \begin{bmatrix} r_{1,1}^{(3)} & r_{1,2}^{(3)} & r_{1,3}^{(3)} & r_{1,4}^{(3)} \\ 0 & r_{2,2}^{(2)} & r_{2,3}^{(2)} & r_{2,4}^{(2)} \\ 0 & 0 & r_{3,3}^{(2)} & r_{3,4}^{(2)} \\ 0 & r_{4,2}^{(1)} & r_{4,3}^{(1)} & r_{4,4}^{(1)} \end{bmatrix}$$

(v) Consider $r_{2,2}$ and $r_{4,2}$. Use $r_{2,2}$to eliminate $r_{4,2}$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c^{(5)} & 0 & s^{(5)} \\ 0 & 0 & 1 & 0 \\ 0 & -s^{(5)} & 0 & c^{(5)} \end{bmatrix} \begin{bmatrix} r_{1,1}^{(3)} & r_{1,2}^{(3)} & r_{1,3}^{(3)} & r_{1,4}^{(3)} \\ 0 & r_{2,2}^{(2)} & r_{2,3}^{(2)} & r_{2,4}^{(2)} \\ 0 & 0 & r_{3,3}^{(2)} & r_{3,4}^{(2)} \\ 0 & r_{4,2}^{(1)} & r_{4,3}^{(1)} & r_{4,4}^{(1)} \end{bmatrix} = \begin{bmatrix} r_{1,1}^{(3)} & r_{1,2}^{(3)} & r_{1,3}^{(3)} & r_{1,4}^{(3)} \\ 0 & r_{2,2}^{(3)} & r_{2,3}^{(3)} & r_{2,4}^{(3)} \\ 0 & 0 & r_{3,3}^{(2)} & r_{3,4}^{(2)} \\ 0 & 0 & r_{4,3}^{(2)} & r_{4,4}^{(2)} \end{bmatrix}$$

(vi) Consider $r_{3,3}$ and $r_{4,3}$. Use $r_{3,3}$ to eliminate $r_{4,3}$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c^{(6)} & s^{(6)} \\ 0 & 0 & -s^{(6)} & c^{(6)} \end{bmatrix} \begin{bmatrix} r_{1,1}^{(3)} & r_{1,2}^{(3)} & r_{1,3}^{(3)} & r_{1,4}^{(3)} \\ 0 & r_{2,2}^{(3)} & r_{2,3}^{(3)} & r_{2,4}^{(3)} \\ 0 & 0 & r_{3,3}^{(2)} & r_{3,4}^{(2)} \\ 0 & 0 & r_{4,3}^{(2)} & r_{4,4}^{(2)} \end{bmatrix} = \begin{bmatrix} r_{1,1}^{(3)} & r_{1,2}^{(3)} & r_{1,3}^{(3)} & r_{1,4}^{(3)} \\ 0 & r_{2,2}^{(3)} & r_{2,3}^{(3)} & r_{2,4}^{(3)} \\ 0 & 0 & r_{3,3}^{(3)} & r_{3,4}^{(3)} \\ 0 & 0 & 0 & r_{4,4}^{(3)} \end{bmatrix}$$

After six steps, six elements can be eliminated (set to zero), resulting in the upper triangular matrix $\mathbf{R}$. If the rotation matrix in each step is denoted as $\mathbf{G_i}$, for $i = 1{:}\,6$, then $\mathbf{Q}^T = \mathbf{G}_6\mathbf{G}_5\mathbf{G}_4\mathbf{G}_3\mathbf{G}_2\mathbf{G}_1$.

Givens rotation can be implemented by CORDIC. Taking the first step as an example, as illustrated in Figure 1. The values $r_{1,1}$, $r_{1,2}$, $r_{1,3}$, $r_{1,4}$ are sequentially fed into the X input of the CORDIC module, while the corresponding values $r_{2,1}$, $r_{2,2}$, $r_{2,3}$, $r_{2,4}$ are fed into the Y input. According to the Givens rotation algorithm (Equations (3) to (6)), the rotation angle must satisfy the condition,

$$-s^{(1)}r_{1,1} + c^{(1)}r_{2,1} = -\sin\left(\theta^{(1)}\right)r_{1,1} + \cos\left(\theta^{(1)}\right)r_{2,1} = 0 \text{ ,}$$

Namely, $\tan\left(\theta^{(1)}\right) = r_{2,1}/r_{1,1}$. Hence, the CORDIC module is programmed into vectoring mode when $r_{1,1}$ and $r_{2,1}$ enter so that $r_{2,1}$ can become 0 at the output. We can simply use 1 bit to record the clockwise or counter clockwise direction at each micro rotation stage. When $r_{1,m}$ and $r_{2,m}$ enter, for $m = 2,3,4$, the CORDIC module is program into rotation mode to complete the Givens rotations in (5) and (6), which can be further described by the following equation.

$$\begin{bmatrix} r_{1,m}^{(1)} \\ r_{2,m}^{(1)} \end{bmatrix} = \begin{bmatrix} \cos\left(\theta^{(1)}\right) & \sin\left(\theta^{(1)}\right) \\ -\sin\left(\theta^{(1)}\right) & \cos\left(\theta^{(1)}\right) \end{bmatrix} \begin{bmatrix} r_{1,m} \\ r_{2,m} \end{bmatrix} \tag{7}$$

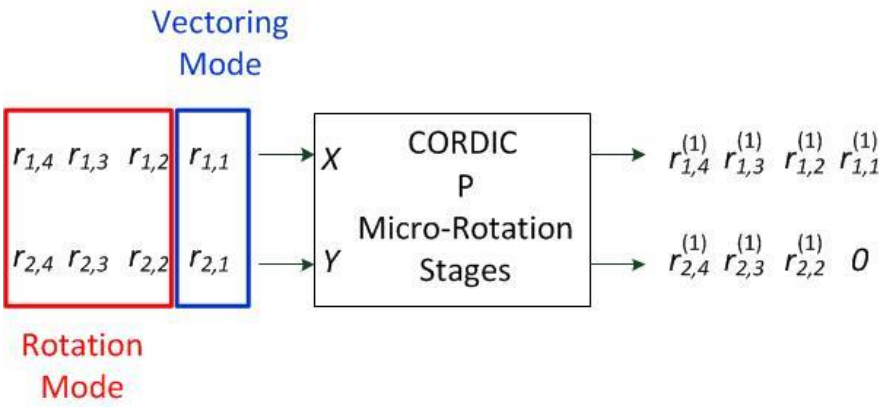



Fig. 1 Use CORDIC to realize the Givens rotation

B. Architecture

For high throughput applications, we can use systolic array to implement QR decomposition. The triangular systolic array for QR decomposition of $3 \times 3$ matrix is shown in Fig. 2. The west input of CORDIC is **Xin** and north input is **Yin**. The east output is **Xout** and the south output is **Yout.** When the input signal marked by the red rectangle enters from the west input, the CORDIC is switched into vectoring mode. Otherwise, the CORDIC is switched into rotation mode. Hence, the top left one executes step (i). The top right one compute step (ii). And the lower right one performs step (iv). Finally, the upper triangular matrix R appears at the output.

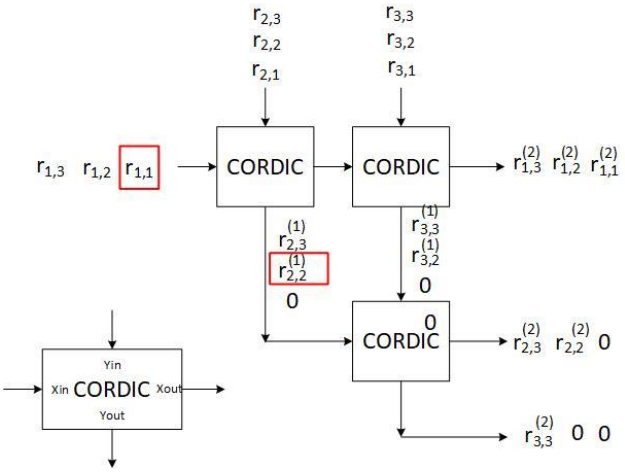
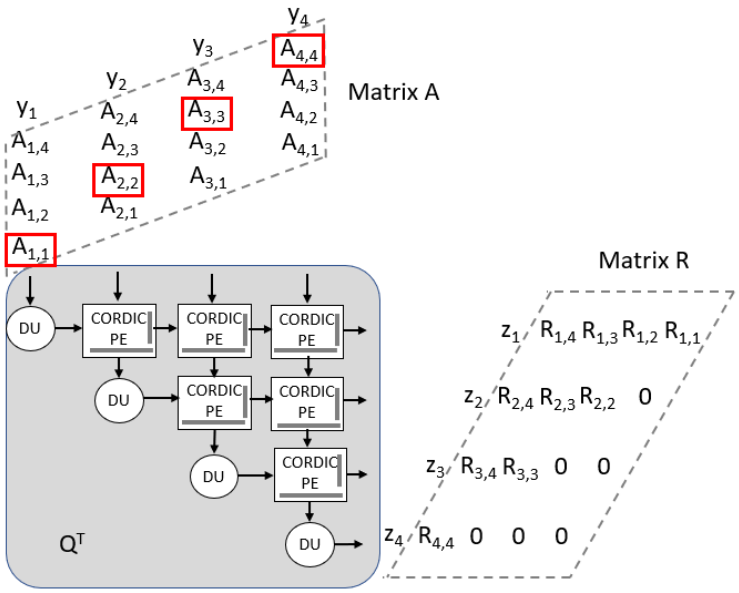


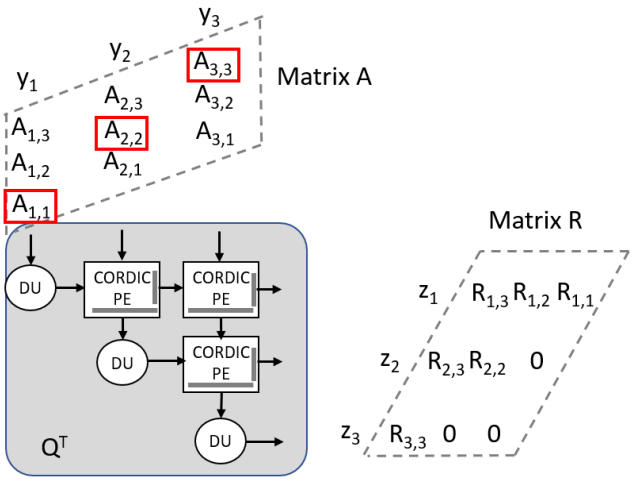

Fig. 2 The triangular systolic array for QR decomposition of $3 \times 3$ matrix.

Since the whole systolic array can be regarded as performing $\mathbf{Q}^T$, the vector y can be sent together and we can obtain vector $\mathbf{z}$ for solving the linear equations as shown in Fig. 3. Note that if you use more pipeline stages inside the CORDIC, the outputs become skewer and more D flip-flops are required to align the signals. A good tradeoff

should be determined.




(a)




(b)

Fig. 3 Triangular systolic array for (a) 4x4 QR decomposition, and (b) 3x3 QR decomposition.

The whole hardware for solving 4 linear equations can be described by Fig. 4. Once matrix **R** and vector **z** are obtained, we can use back substitution for solving vector **x**, which can be given as follows.

$$\hat{x}_4 = \frac{z_4}{R_{4,4}} \tag{8}$$

$$\hat{x}_3 = \frac{z_3 - R_{4,4}\hat{x}_4}{R_{3,3}} \tag{9}$$

$$\hat{x}_2 = \frac{z_2 - R_{2,3}\hat{x}_3 - R_{2,4}\hat{x}_4}{R_{2,2}} \tag{10}$$

$$\hat{x}_1 = \frac{z_1 - R_{1,2}\hat{x}_2 - R_{1,3}\hat{x}_3 - R_{1,4}\hat{x}_4}{R_{1,1}} \ldots\ldots \qquad \ldots\ldots\ldots\ldots\ldots\ldots (10)$$

Note that for solving only three linear equations, QR decomposition may not be adopted in some applications. However, for a fair comparison, all the students must use CORDIC systolic array to implement QR decomposition. Since the input is regular, your hardware should have a constant throughput.

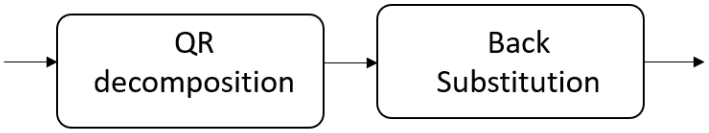



Fig. 4 Whole hardware for solving linear equations.

C. Test Patterns

Two kinds of test patterns are used.

(a) Generate the test patterns by yourselves to evaluate the word-length settings for QR decomposition and back substitution. Please use Gaussian random variable with zero mean and unit variance to generate vector $\mathbf{x}$ first as the answer. Then, generate matrix $\mathbf{A}$. Finally, calculate $\mathbf{y} = \mathbf{Ax}$ to obtain the other input. You can generate 100 sets by yourselves for evaluate word-lengths. To avoid the numerical instability problem, please discard those testing sets with $R_{4,4} < 0.05$.

(b) Four sets of equations, $\mathbf{y} = \mathbf{Ax}$, will be provided on NTUCool for evaluating the performance.

D. I/O Ports

The I/O of the hardware is defined as follows for fair comparison of area and throughput.

(a) **InValid**: If it is 1, the matrix $\mathbf{A}$ and vector $\mathbf{y}$ is sent through the **InData** ports. If it is 0, the **InData** ports are not processed. Note that this input signal is essential to control the switch between vectoring mode and rotation mode.

(b) **InData** $(4 \times W_I)$: To accommodate the highest throughput, this port is mainly used for feeding matrix $\mathbf{A}$ **and** vector $\mathbf{y}$. The word-length $W_I$ is determined by yourselves.

(c) **OutData** $(4 \times W_O)$: The computed $\tilde{\mathbf{x}}$..

(d) **Clk:** clock signal

(e) **Reset:** reset signal

(f) **Ctr:** Other I/O control signals that you need (For example, Valid signal for output $\tilde{\mathbf{x}}$).

Requirements:

1. The word-length must be set so that the root mean square error (RMSE) $\sqrt{\frac{1}{4}\sum_{i=1}^{4}(\hat{x}_i - x_i)^2}$ $<10^{-3}$ for **your 100 sets of linear equations**. Show how you decide the wordlengths of CORDIC modules, dividers, and multipliers. Draw the figure of RMSE versus wordlength.

2. Draw the RMSE versus set index (1~100) to construct your bit-true model. If RMSEs of $P$ sets are larger than $10^{-3}$, a deduction of $0.5P$ points will be applied. If more than 6 sets can not satisfy the criterion, the evaluation of AT product (area and processing time) is put in the last level.

3. Explain your hardware design concept in the report. Draw the block diagram of back substitution in your design.

4. Please show the hardware simulation results **with the input patterns provided on the NTUCool.** If the number of clock cycles in the simulation is large, then cut the whole timing diagram into several segments and paste them. Also, you need to keep and show the time stamp in your simulation figure for verifying the setting of the clock period. The timing diagram should be given like this.

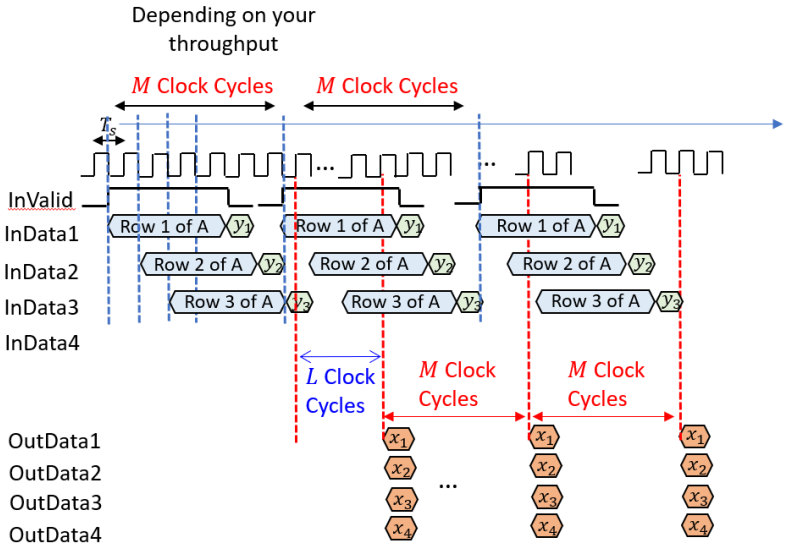



Fig. 5 Example of timing diagram

(a) Please provide the timing diagram of behavior simulation and calculate the RMSE of the 4 sets of solutions. If RMSEs of $S$ sets are larger than $10^{-3}$, a deduction of S points will be applied. If more than 3 sets can not satisfy the criterion, the evaluation of AT product (area and processing time) is put in the last level.

(b) Please provide the timing diagram of the post-synthesis simulation. You need to indicate the clock period setting ($T_s$) in your post-synthesis simulation and the values of $M$. The processing time is calculated by $MT_s$ Explain your latency $L$ in your design.

(c) Show that your post-synthesis simulation results are the same as the behavior simulation results.

5. Show that your implementation is consistent with your fixed-point simulation by drawing the figure indicating the errors of the results using linear equations on NTUCool.

6. Show your synthesis report. If you use VIVADO, please use the following equation to combine all the resource consumptions into one metric [1]:
   NA = LUTs + DSPs × 280 + FFs
   If you use ADFP, please simply show the total area after synthesis.

7. Now calculate your AT product by
   Area (or NA) × processing time ($MT_s$).
   We will classify the AT product into 3 levels and evaluate the implementation results of your AT product
   **(Note: 15% of the score is decided by the level of the AT product. The remaining 85% is decided by the simulations, functionality, descriptions, and timing diagram….)**

8. Highlight any of your design innovations. List the working items and weightings of two members.

9. If you have problems with solving 4 linear equations, you can choose to solve 3 linear equations and the evaluation of AT product will be put into level 4.

Reference:

[1] J. Chen, Z. Zhang, H. Lu, J. Hu and G. E. Sobelman, "An Intra-Iterative Interference Cancellation Detector for Large-Scale MIMO Communications Based on Convex Optimization," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 11, pp. 2062-2072, Nov. 2016.