Computer-Aided VLSI System Design

Final Project: Elliptic Curve Cryptographic Processor

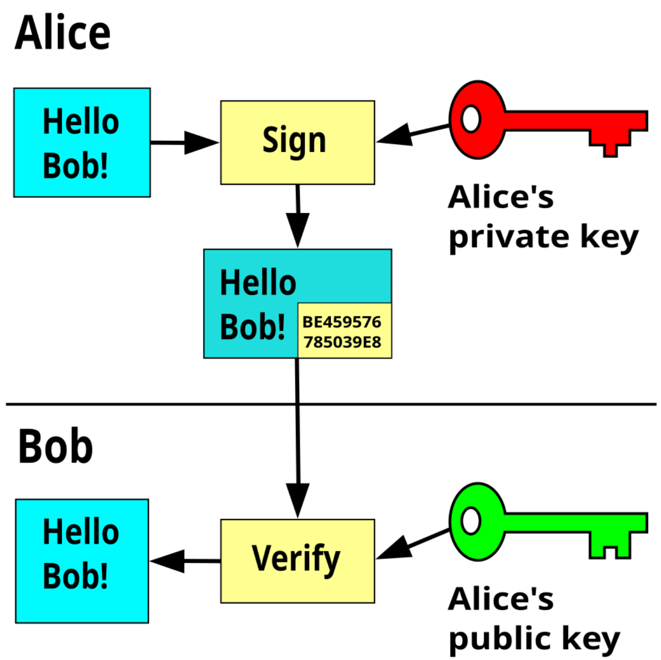
lecturer: Liang-Hsin Lin

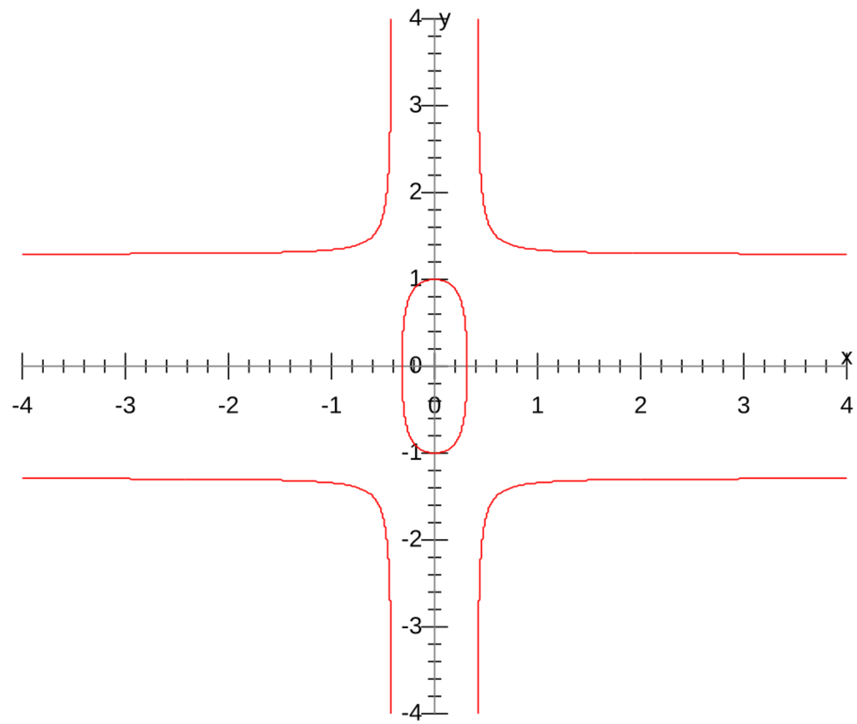
Graduate Institute of Electronics Engineering, National Taiwan University

NTU GIEE

Overview

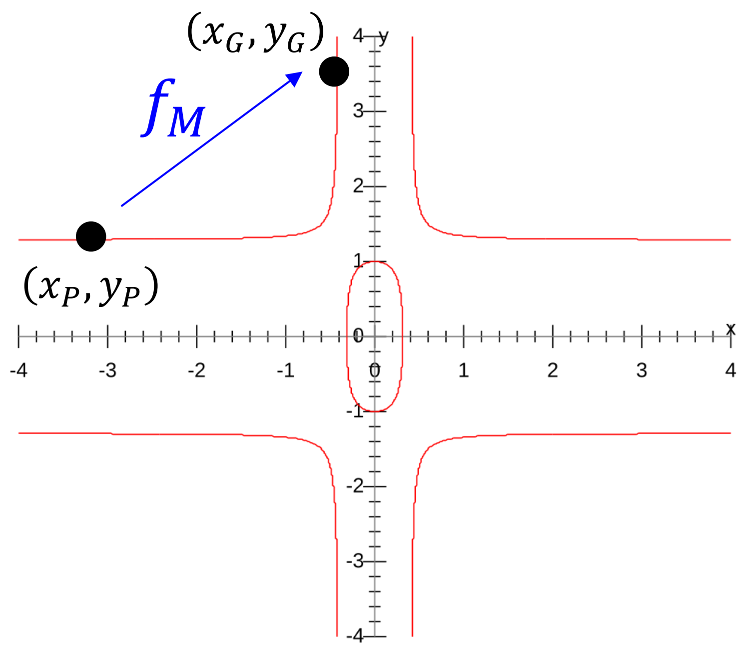
- Edwards-curve Digital Signature Algorithm (EdDSA) [1] is frequently used for data authentication nowadays
- Based on a mathematical structure: twisted Edward curve [2]
  - A special type of elliptic curve with strong security





System Model

- **Project Goal**: Accelerate the core function in EdDSA
  - Given a point $P = (x_P, y_P)$ on a twisted Edward curve
  - Compute another point $G = (x_G, y_G) = f_M(x_P, y_P)$



  - However, **integers** instead of real numbers are used

Finite Field F_p

- Twisted Edward curve is based on finite field
- Given prime q, the set $F_q = \{0,1,\dots,q-1\}$ is called a finite field
- The operations between arbitrary $x, y \in F_q$ are defined by
  - Addition: $x + y \ mod\ q$
  - Subtraction: $x - y \ mod\ q$
  - Multiplication: $x \times y \ mod\ q$
  - Division: $x \times y^{-1}\ where\ y^{-1} \in F_q$ s.t. $y \times y^{-1}\ mod\ q = 1$
- The "$mod\ q$" is neglected in the following context for elements in F_q

Twisted Edward Curve

- Given constant $a, d \in F_q$, the curve is defined by a set of points

$$E = \{(x \in F_q, y \in F_q) | ax^2 + y^2 = 1 + dx^2y^2\}$$

- The addition between two points $(x_1, y_1), (x_2, y_2)$ is defined by

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}\right)$$

Projective Coordinates

- Use (X_1, Y_1, Z_1) to represent $(x_1 = X_1/Z_1, y_1 = Y_1/Z_1)$
- The addition formula becomes

$$(X_1, Y_1, Z_1) + (X_2, Y_2, Z_2) = (X_3, Y_3, Z_3)$$
$$X_3 = Z_1 Z_2 (X_1 Y_2 + X_2 Y_1)(Z_1^2 Z_2^2 - d X_1 X_2 Y_1 Y_2)$$
$$Y_3 = Z_1 Z_2 (Y_1 Y_2 - a X_1 X_2)(Z_1^2 Z_2^2 + d X_1 X_2 Y_1 Y_2)$$
$$Z_3 = (Z_1^2 Z_2^2 - d X_1 X_2 Y_1 Y_2)(Z_1^2 Z_2^2 + d X_1 X_2 Y_1 Y_2)$$

Advantage of Projective Coordinates

- Less modular divisions
- Example: $(x_1, y_1) + (x_1, y_1) + (x_1, y_1)$
  - **Without** projective coordinate (4 divisions):
    - $(x_2, y_2) = (x_1, y_1) + (x_1, y_1)$: 2 divisions
    - $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$: 2 divisions
  - **With** projective coordinate (2 divisions)
    - $(X_2, Y_2, Z_2) = (x_1, y_1, 1) + (x_1, y_1, 1)$: 0 division
    - $(X_3, Y_3, Z_3) = (X_2, Y_2, Z_2) + (x_1, y_1, 1)$: 0 division
    - $(X_3, Y_3, Z_3) \rightarrow (x_3, y_3)$: 2 divisions

EdDSA

- The core operation in EdDSA involves the following procedure
  - Given an integer M and a point $P = (x_p, y_p)$ on a Twisted Edward Curve
  - Compute

$$f_M(x,y) = M \times P = \underbrace{(x_p, y_p) + (x_p, y_p) + \cdots + (x_p, y_p)}_{M}$$

- Ed25519 is a EdDSA scheme based on the twisted Edward curve

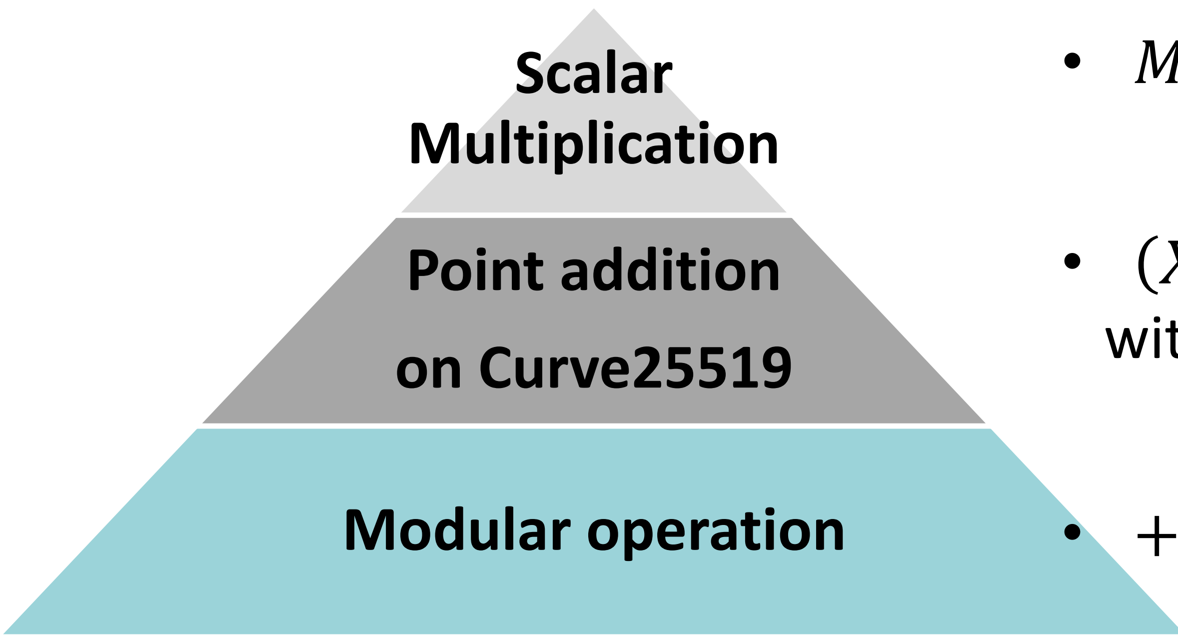
$$E_{25519} = \{(x \in F_q, y \in F_q) | ax^2 + y^2 = 1 + dx^2y^2\}$$

where $q = 2^{255} - 19, a = q - 1,$ and $d =$

0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3

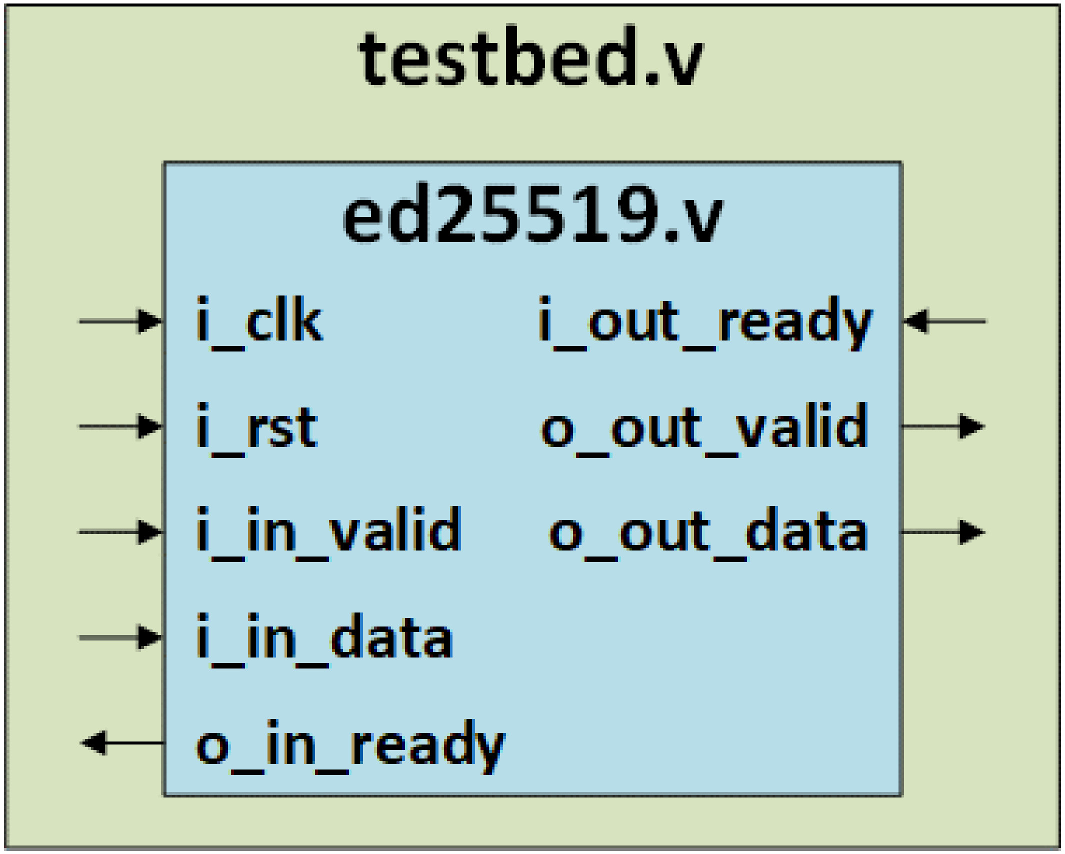
Ed25519 System Overview

- Input: scalar M, point $P = (x_P, y_P) \in E_{25519}$
- Output: another point $G = (x_G, y_G) = M \times P$
- Three operation levels for Ed25519



- $M \times P = M \times (x_P, y_P)$
- $(X_1, Y_1, Z_1) + (X_2, Y_2, Z_2)$ with projective coordinates
- $+, -, \times, \div$ on finite field F_q

Block Diagram

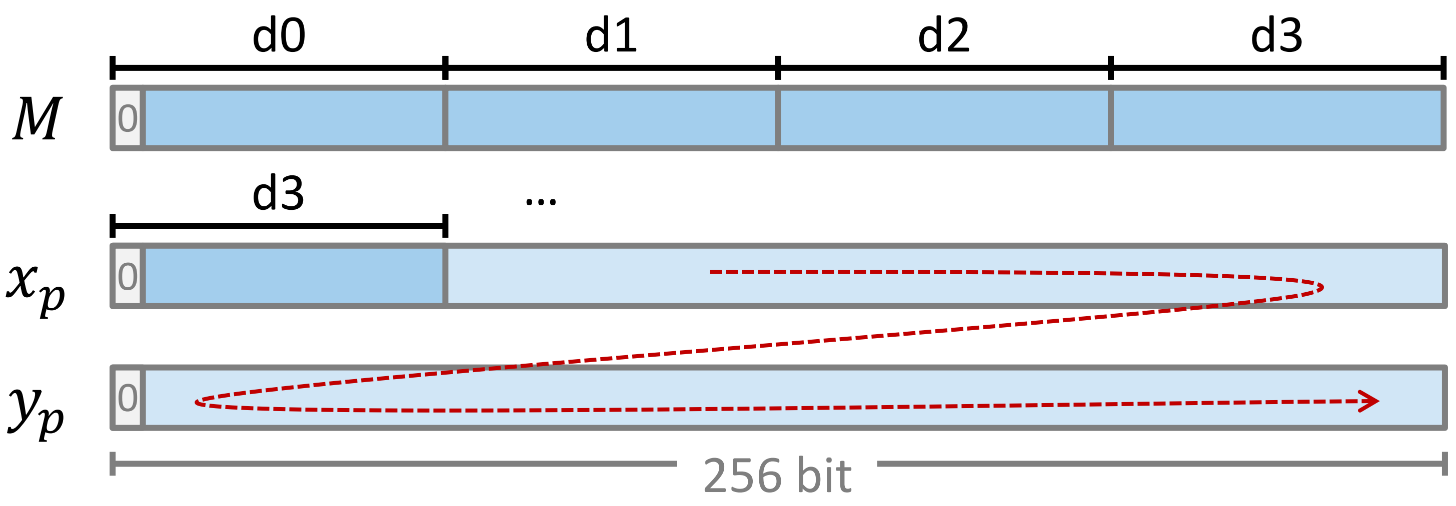


Input/Output

Signal Name	I/O	Width	Description
i_clk	I	1	System clock signal, synchronized on the rising edge
i_rst	I	1	Synchronous active high reset signal
i_in_valid	I	1	When high, indicates that **i_in_data** contains valid data
i_in_data	I	64	64-bit input data bus
i_out_ready	I	1	When high, indicates that testbench is ready to receive output data
o_in_ready	O	1	When high, indicates that the module is ready to accept input data
o_out_valid	O	1	When high, indicates that **o_out_data** contains valid output data
o_out_data	O	64	64-bit output data bus

I/O Data Sequence

- The I/O data port is 64-bit
- Input sequence order: $M \rightarrow x_P \rightarrow y_P$
- Output sequence order: $x_G \rightarrow y_G$
- Each 255-bit data is expanded to 256 bits by adding 0 as the MSB
- Each data unit is transmitted starting from the MSB down to the LSB in 64-bit chunks

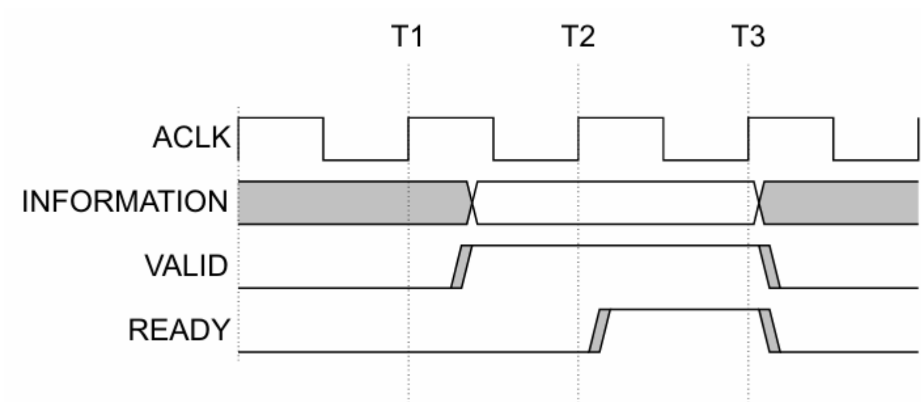


*Sequence: d0, d1, d2, d3, …

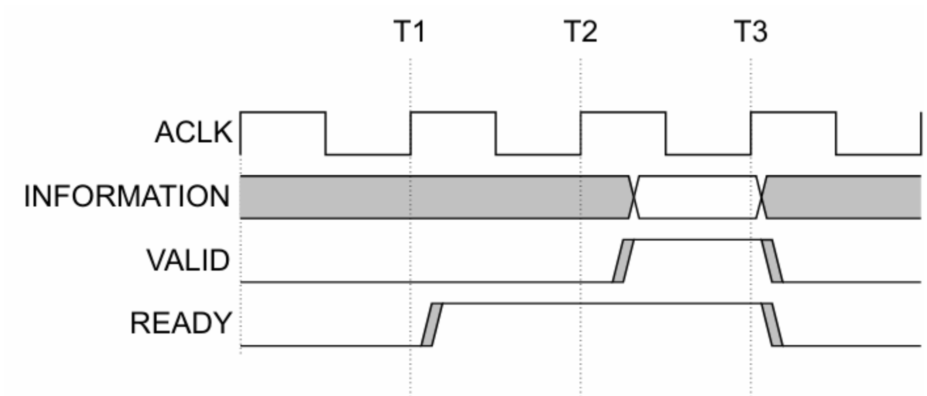
I/O Handshake

- The source generates the **VALID** signal to indicate when the data is available
- The destination generates the **READY** signal to indicate that it can accept the information
- Transfer occurs only when both the **VALID** and **READY** signals are HIGH
- There must be no combinatorial paths between input and output signals

VALID before **READY** handshake



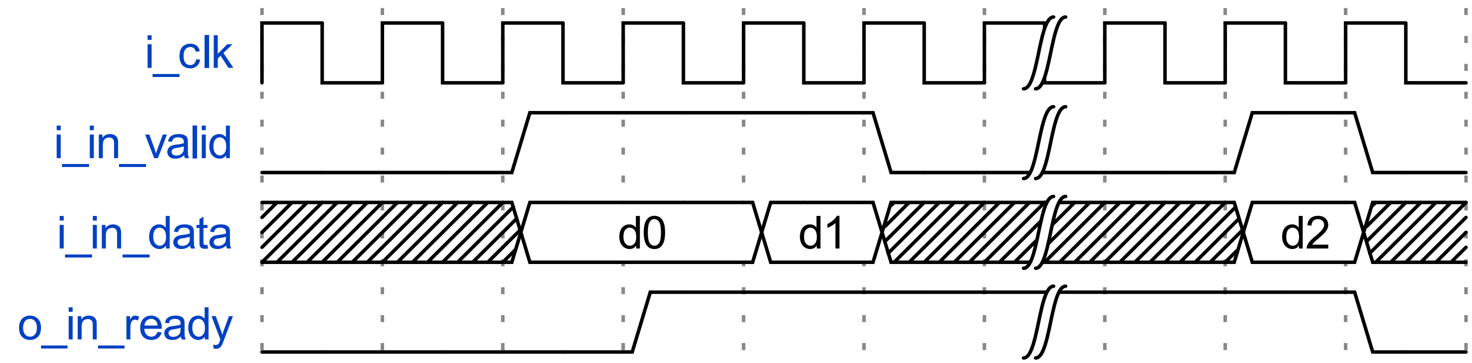
READY before **VALID** handshake



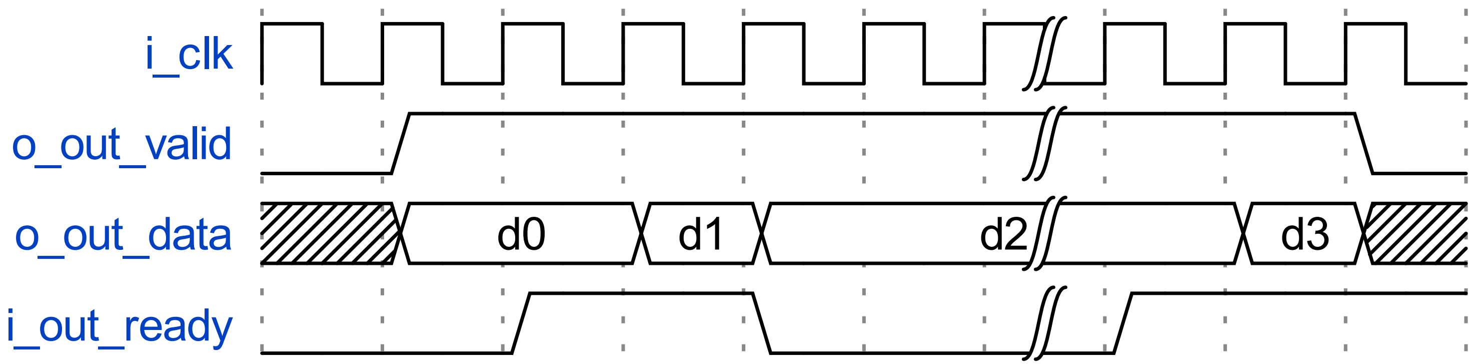
Reference: AMBA AXI and ACE Protocol Specification [5]

I/O Waveform

Input Interface



Output Interface



HW Implementation: Mod. Op. (Add & Sub)

- Let $x, y \in F_q$
- Modular addition:

$$x + y \bmod q = \begin{cases} x + y, if\ x + y < q \\ x + y - q, otherwise \end{cases}$$

- Modular subtraction:

$$x - y \bmod q = \begin{cases} x - y, if\ x \geq y \\ x + (q - y), otherwise \end{cases}$$

HW Implementation: Mod. Op. (Mul)

- Let $x, y \in F_q, R = 2^{255}, q^{-1}$ be the inverse of q modular R $(q \times q^{-1}\ mod\ R = 1)$

- Montgomery multiplication (MM):

512 -> 255

$$MM(x, y) = \frac{xy}{R} mod\ q = \begin{cases} t - q, if\ t \geq q \\ t, otherwise \end{cases}$$

$$t = \frac{xy - (xyq^{-1}\ mod\ R)q}{R}$$

255

- Modular multiplication:

$$x \times y\ mod\ q = MM(MM(x, y), R^2\ mod\ q)$$

HW Implementation: Mod. Op. (Div)

- Let $a, b \in F_q$, $q = 2^{255} - 19$
- Modular division: modular multiplication + modular inversion

$$a/b \; mod \; q = a \otimes b^{-1} mod \; q$$

- Modular inversion (by Fermat's little theorem)

$$b^{-1} mod \; q = b^{q-2} \; mod \; q$$

Algorithm 1: Modular Inversion

Parameter: $q = 2^{255} - 19$
Input : $b \in F_q$
Output : $b^{-1} \mod q \in F_q$
$r = 1$
for $i = 255$ to 1 **do**
  $r = r^2 \mod q$
  if ith bit of $q - 2$ is 1 **then**
    $r = rb \mod q$
return r

HW Implementation: Scalar Multiplication

- Using addition formula in projective coordinate

$$M \times (x_P, y_P) = M \times (x_P, y_P, 1) = \underbrace{(x_P, y_P, 1) + \cdots + (x_P, y_P, 1)}_{M}$$

Algorithm 2: Scalar Multiplication

Parameter: Curve E_{25519}
Input : 255-bit scalar M and point $P = (x_p, y_p) \in E_{25519}$
Output : point $G = (x_G, y_G) = M \times P$

```
r = (0, 1, 1)              // zero point in projective coordinate
P = (x_P, y_P, 1)          // point P in projective coordinate
for i = 255 to 1 do
    r = r + r              // point addition
    if ith bit of M is 1 then
        r = r + P          // point addition
return r
```

Coordinate Reduction

- Reduce extra coordinate Z in projective coordinates
  - Let $MP = (X_{MP}, Y_{MP}, Z_{MP})$ be the result of scalar multiplication (Algorithm 2)
  - Find the point in normal coordinate

$$(x_{MP}, y_{MP}) = (X_{MP}/Z_{MP}, Y_{MP}/Z_{MP})$$

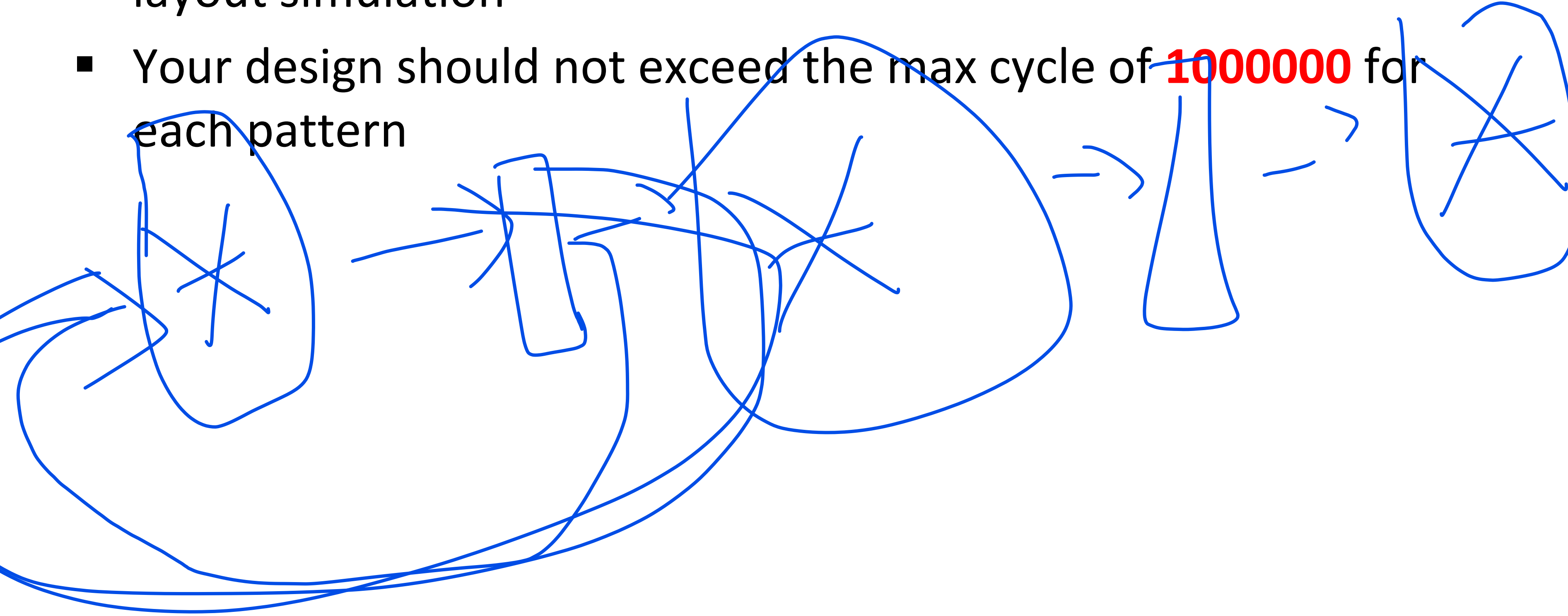
  - Make sure x_{MP} and y_{MP} are both even

$$x_G = \begin{cases} x_{MP}, x_{MP} \text{ is even} \\ -x_{MP} \bmod q, x_{MP} \text{ is odd} \end{cases} \quad y_G = \begin{cases} y_{MP}, y_{MP} \text{ is even} \\ -y_{MP} \bmod q, y_{MP} \text{ is odd} \end{cases}$$

  - Derive point $G = (x_G, y_G)$

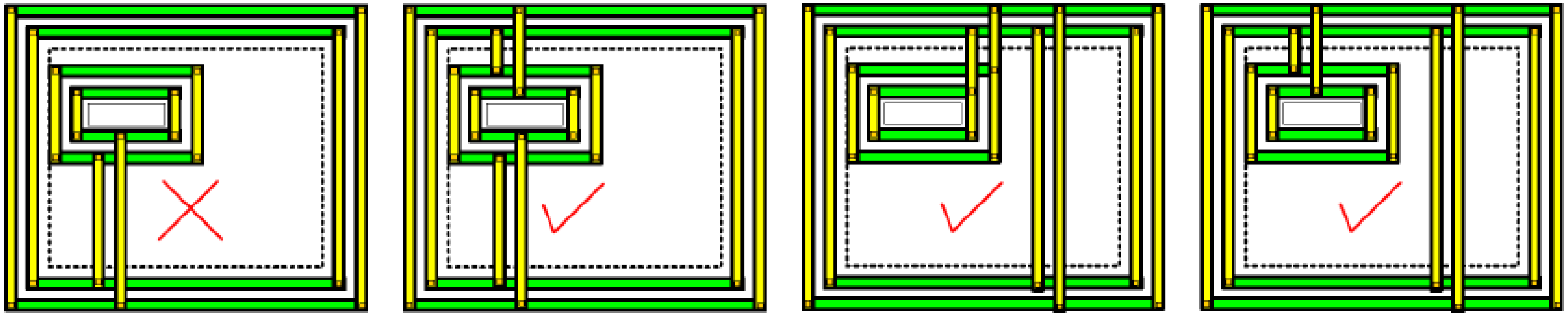
Timing Specification

- Only the worst-case library is used for synthesis
- The slack for setup time should be non-negative
- **No timing violation** for the gate level simulation and post-layout simulation
- Your design should not exceed the max cycle of **1000000** for each pattern



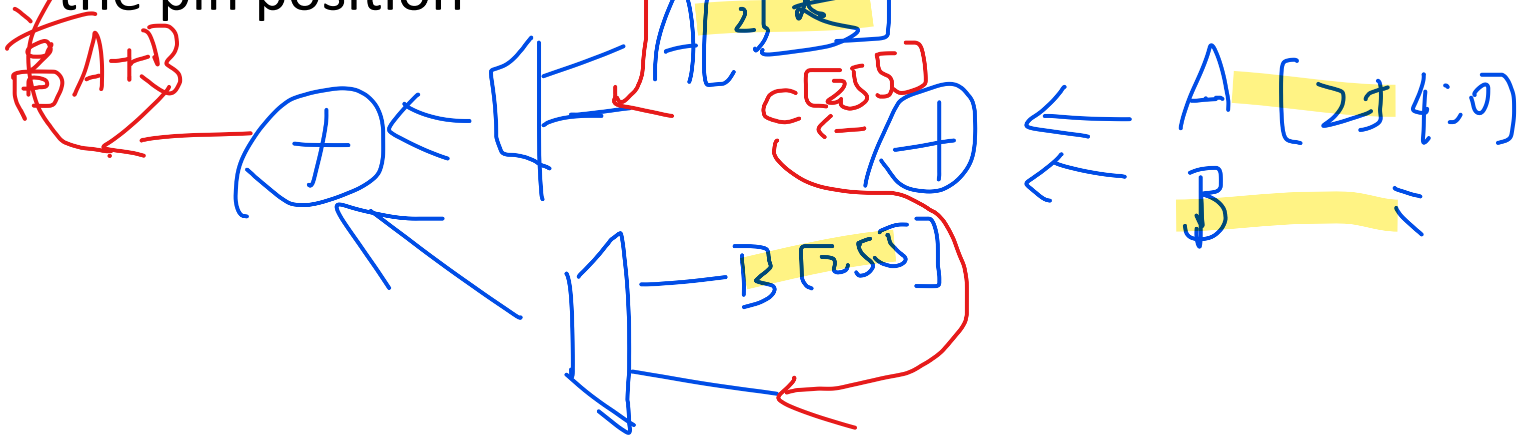
APR Specifications (1)

- Only the **macro layout** is needed
  - IO pads and bonding pads are not required
- **VDD** and **VSS power rings** should each be **2 µm wide**, with only one ring required for each
- **Power stripes**
  - At least one set, with **VDD** and **VSS** stripes each **2 µm wide**
  - Vertical power stripes require at least one set (horizontal power stripes are optional)



APR Specifications (2)

- Remember to add the **Power Rail** (follow pin)
- **Dummy metal layers** are not needed
- **Core Filler** must be added
- The **GDSII** file after APR must be generated
- Ensure that the APR DRC/LVS is completely **error-free**
- You can generate the ioc file first, and then reload the file to set the pin position



Simulation Settings

- Run the simulations using `01_run`, `03_run`, and `05_run`
- You can only edit the bottom section of each `*_sim.f` file

```
// ============================================================================
//                  Your Can Only Modify The Below Part
// ============================================================================

// Your Design Files
// ----------------------------------------------------------------------------
./ed25519.sv

// Define Flags
// ----------------------------------------------------------------------------
+define+RANDOM_IO_HANDSHAKE
```

- There is a define flag RANDOM_IO_HANDSHAKE
  - Randomized valid/ready signal for testing in the testbench
  - Each pattern will be tested with this flag enabled
  - Performance (time) is evaluated without this flag

Submission (1)

- Create a folder named **teamID_final** with the structure below:

```
team01_final/
├── 01_RTL
│   ├── ed25519.v (and other Verilog files)
│   └── rtl_sim.f (include all your Verilog files)
├── 02_SYN
│   ├── ed25519_syn.area
│   ├── ed25519_syn.timing_min
│   └── ed25519_syn.timing_max
├── 03_GATE
│   ├── ed25519_syn.v
│   ├── ed25519_syn.sdf
│   └── gate_sim.f
├── 04_APR
│   ├── final (saved APR design)
│   ├── final.dat (design database)
│   └── ed25519.gds
├── 05_POST
│   ├── ed25519_pr.v
│   ├── ed25519_pr.sdf
│   └── post_sim.f
└── reports
    ├── design.spec
    └── team01_report.pdf
```

Submission (2)

- **Deadline:** 2024/12/17 13:59:59 (UTC+8)

- Compress the folder **teamID_final** **(all lowercase)** in a tar file named **teamID_final_vk.tar** (k: version number, e.g., 1,2,…)
- Ensure all required files are submitted
  - **10-point deduction** for each missing file
- Submit to **NTU Cool**

Report Requirements

1. **APR Results**
   – Show the screenshot of the layout and DRC/LVS results
2. **Algorithm Design**
   – Clearly describe the algorithm analysis and the optimization techniques employed
3. **Hardware Implementation**
   – Provide specifics about the hardware architecture, including the design of key modules and optimization techniques
4. **Performance Evaluation**
   – Assess the efficiency of your design, including metrics such as speed and area.

Grading Policy

- Baseline 50% + Performance 40% + Report 10%

Item	%	Description
RTL Simulation	20	Pass **all** pattern simulations (3 public + 1 hidden)
Synthesis	10	Pass gate-level simulation
APR	20	Finish APR with no DRC/LVS errors Pass post-layout simulation
Performance	40	Area x Time
Report	10	See Page 19 for more details

- **Performance = 0 if**
  - DRC/LVS errors occur
  - Post-layout simulation fails

Grading Policy

- **No late submission is allowed**
  - Any submissions after the deadline will receive 0 points
- **5-point deduction** for incorrect naming or format
  - Pack all files into a single folder and compress the folder
  - Ensure that the files submitted can be decompressed and executed without issues
- **No plagiarism**
  - Plagiarism in any form, including copying from online sources, is strictly prohibited

Discussion

- **NTU Cool Discussion Forum**
  - For any questions not related to assignment answers or privacy concerns, please use the NTU Cool discussion forum
  - TAs will prioritize answering questions on the NTU Cool discussion forum

- **Email:** d10943004@ntu.edu.tw
  - Title should start with [CVSD 2024 Fall Final Project]
  - Email with wrong title will be moved to trash automatically

Final Project Presentation

- **Date:** December 24, 2024
- **Time:** 14:20 - 17:20

- Top-performing teams will be invited to present their design optimization strategies
- Bonus points will be awarded for presentations
- Additional information will be provided later

Hints for HW Optimizations

- Point doubling $P + P$ is faster than point addition $P + P'$ ($P \neq P'$) [3]
- Minimize the # of modular operations in point doubling/addition [3]
- Try different coordinate representation [3]
  - Projective, Extended, Inverted, …
- Try different algorithms for faster scalar multiplication [4]
  - Double-and-Add (Algorithm 2), Windowed, Sliding-window, …
- Reduce # of Montgomery multiplications with factor $R^2\ mod\ q$
- Design a Montgomery multiplier with low hardware complexity

Reference

- [1] https://zh.wikipedia.org/zh-tw/EdDSA
- [2] https://en.wikipedia.org/wiki/Twisted_Edwards_curve
- [3] https://www.hyperelliptic.org/EFD/g1p/auto-twisted.html
- [4] https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication
- [5] AMBA AXI and ACE Protocol Specification, Arm.