รายงานสหกิจศึกษาฉบับสมบูรณ์

กระบวนการบนเส้นโค้งเชิงวงรีในช่วงจำกัดP192โดยดำเนินงานบนฮาร์ดแวร์

Elliptic Curve Operation Finite Field P192 Hardware Implement

ชินภัทร์ ข้อหน่อ

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2562

ชื่อโครงงานสหกิจศึกษา กระบวนการบนเส้นโค้งเชิงวงรีในช่วงจำกัดP192โดยดำเนินงานบนฮาร์ดแวร์

ชื่อ-สกุล นักศึกษา นายชินภัทร์ ข้อหน่อ

คณะ วิศวกรรมศาสตร์ **ภาควิชา** อิเล็กทรอนิกส์

ชื่อ-สกุล อาจารย์นิเทศ ดร.สุเมฆ วิศยทักษิณ

ชื่อ-สกุล ผู้นิเทศงาน นายธนพล หงษ์ทรงเกียรติ

ชื่อสถานประกอบการ บริษัท ซิลิคอน คราฟท์ เทคโนโลยี จำกัด (มหาชน)

บทคัดย่อ

โครงงานนี้เป็นโครงงานสหกิจที่ร่วมกับบริษัท ซิลิคอน คราฟท์ เทคโนโลยี จำกัด (มหาชน) ซึ่งเป็นการศึกษาเกี่ยวกับวิทยาการเข้ารหัสบนเส้นโค้งเชิงวงรี โดยใช้การออกแบบ และการดำเนินงานเชิงฮาร์ดแวร์ ซึ่งโครงงานนี้จะใช้กระบวนการทางคณิตศาสตร์ในช่วงจำกัดP192 และมุ่งเน้นในส่วนของการคูณจุดบนเส้นกราฟ ที่เป็นพื้นฐานของกระบวนการบนเส้นโค้งเชิงวงรี และสามารถนำไปต่อยอดเพื่อใช้ในวิทยาการเข้ารหัสบนเส้นโค้งเชิงวงรีได้ โดยการทำงานของวงจรจะรับค่าสเกล่า และจะเริ่มทำงานเมื่อส่งสัญญาณเริ่มทำงาน จากนั้นจะได้ผลลัพธ์เป็นค่าการคูณกันระหว่างค่าสเกล่ากับจุดเริ่มต้นบนเส้นกราฟ ซึ่งอยู่ในรูปของจุด จุดหนึ่งบนกราฟที่เป็นค่าบนแกนตั้ง และแกนนอน

คำสำคัญ : เส้นโค้งเชิงวงรี, วิทยาการเข้ารหัสบนเส้นโค้งเชิงวงรี, กระบวนการบนเส้นโค้งเชิงวงรี,การคูณจุด

Co-operative Title: Elliptic Curve Operation Finite Field P192 Hardware Implement

Student Intern Name: Mr.Chinnapat Khornor

Faculty: Engineering **Department:** Electronics

Advisor Name: Dr.Sumek Wisayataksin

Mentor Name: Mr.Thanapol Hongsongkiat

Company: Silicon Craft Technology

ABSTRACT

This project co-operative with Silicon Craft Technology to study about Elliptic Curve Cryptography that design and implement in hardware in this project use arithmetic in finite field P192 and focus on point multiplication in elliptic curve that is basic in elliptic curve and can use to do many thing in cryptography. Point multiplication in this project started form input scalar value and sent enable signal to start process then output is multiplication between scalar value and base point in elliptic curve that shown in vector.

Keywords: elliptic curve, elliptic curve cryptography, elliptic curve operation, point multiplication

กิตติกรรมประกาศ

โครงงานนี้จะไม่สามารถเกิดขึ้นได้ถ้าไม่ได้รับความกรุณาจาก บริษัท ซิลิคอน คราฟท์ จำกัด(มหาชน) ซึ่งอนุญาตให้ร่วมปฏิบัติงานสหกิจ ตั้งแต่วันที่ 5 สิงหาคม พ.ศ.2562 ถึง วันที่ 29 พฤศจิกายน พ.ศ.2562 การที่ข้าพเจ้าได้ร่วมปฏิบัติงานสหกิจนั้น ส่งผลให้ข้าพเจ้าได้รับประสบการณ์การทำงาน และการออกแบบในส่วนของวงจรรวมดิจิตอล อีกทั้งความรู้และความร่วมมือในการทำโครงงาน ซึ่งสำเร็จได้ด้วยความช่วยเหลือและการสนับสนุนในด้านต่างๆ จากบุคลากรในแผนกการออกแบบวงจรรวมดิจิตอล อาจารย์นิเทศ และบุคลากรที่เกี่ยวข้อง ที่คอยติดต่อประสานงานสหกิจครั้งนี้

ข้าพเจ้าขอขอบพระคุณผู้มีส่วนร่วมทุกท่าน ที่ให้คำแนะนำและให้คำปรึกษาในการทำโครงงานนี้ตลอดจนให้การดูแล อีกทั้งคอยแนะนำประสบการณ์ในการทำงานจริง ข้าพเจ้าขอขอบคุณไว้ ณ ที่นี้ด้วย

ชินภัทร์ ข้อหน่อ

สารบัญ

หน้า

สารบัญภาพ

ภาพที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

ปัจจุบันโลกของดิจิทัลได้เข้ามามีบทบาทมากขึ้นเรื่อยๆ โดยสิ่งที่เห็นได้อย่างชัดเจนคือ สังคมออนไลน์ ซึ่งเป็นสังคมที่แทบจะพบเห็นผู้คนทุกช่วงอายุเข้ามามีส่วนร่วมเนื่องด้วยความที่เข้าถึงได้ง่าย และพื้นที่ส่วนใหญ่สามารถเข้าถึงอินเทอร์เน็ตได้ ทำให้เกิดกระแสของ IoT หรือ Internet of Thing ที่จะเชื่อมต่อสิ่งต่างๆในโลก ด้วยอินเทอร์เน็ตหรือสัญญาณไร้สาย อีกทั้งยังมีเรื่องของ Blockchain และ Cryptocurrency ที่เป็นการแลกเปลี่ยนข้อมูลดิจิทัล

ดังนั้นทำให้การพัฒนาของอินเทอร์เน็ต และการเชื่อมต่อไร้สาย ได้มีการพัฒนาอย่างรวดเร็ว ซึ่งสิ่งที่สำคัญไม่แพ้กันก็คือ ความปลอดภัย โดยในอดีตผู้ใช้งานน้อยคนนักที่จะสนใจในเรื่องนี้ แต่ในปัจจุบันผู้คนเริ่มคำนึงถึงความปลอดภัยในโลกดิจิทัลเพิ่มมากขึ้นเรื่อยๆ เนื่องจากการที่อินเทอร์เน็ตเข้าถึงได้ง่าย ส่งผลให้เกิดการโจรกรรมข้อมูลดิจิทัลมีเพิ่มมากขึ้น และถ้า Internet of Thing เกิดขึ้นจริง ทุกสิ่งสามารถควบคุมได้โดยข้อมูลดิจิทัล แต่ถ้าการรักษาความปลอดภัยทำได้ไม่ดีหรือมีช่องโหว่ ก็จะมีคนที่ประโยชน์จากช่องโหว่นั้นในการเข้าถึงข้อมูลดิจิทัล และสามารถควบคุมสิ่งต่างๆได้ ดังนั้นระบบรักษาความปลอดภัยนั้นจึงเป็นสิ่งที่สำคัญมากสิ่งหนึ่งในโลกดิจิทัล ส่งผลให้มีการพัฒนารูปแบบการเข้ารหัสด้วยวิธีต่างๆ เพื่อให้เกิดความปลอดภัยมากที่สุดจนเกิดเป็น Elliptic Curve Cryptography หรือ วิทยาการเข้ารหัสเส้นโค้งเชิงวงรี

บริษัท ซิลิคอน คราฟท์ เทคโนโลยี จำกัด(มหาชน) ได้เห็นถึงความสนใจ ในการที่จะเชื่อมระหว่างโลกอนาล็อกและดิจิทัล จึงได้เกิดการประยุกต์นำ Elliptic Curve Cryptography มาออกแบบและพัฒนาลงบนฮาร์ดแวร์ โดยส่วนหนึ่งใช้เป็นหัวข้อในการปฏิบัติงานสหกิจ

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 ศึกษาวิทยาการเข้ารหัส

1.2.2 เพื่อเรียนรู้วิธีการออกแบบวงจรรวมดิจิทัล

1.2.3 ศึกษาการทำงานของElliptic Curve Cryptography

1.3 ขอบเขตของการวิจัย

สามารถทำงานได้และให้ผลลัพธ์ที่ถูกต้องในการทำ Point Multiplication โดยอยู่ในพื้นฐานของ Finite field P192

1.4 วิธีการดำเนินการวิจัย

1.4.1 ศึกษาทฤษฎีการทำงานของหัวข้อที่สนใจ

1.4.2 ทดลองเขียนโปรแกรมลงบนซอฟท์แวร์เพื่อตรวจสอบว่าทฤษฎีนั้นใช้ได้จริง

1.4.3 ออกแบบและ วางโครงสร้างภายในวงจร

1.4.4 เขียนฮาร์ดแวร์ให้ทำงานตามที่ออกแบบเพื่อนำไปสังเคราะห์เป็นวงจรจริง

1.4.5 ทดสอบการทำงานของฮาร์ดแวร์ที่ออกแบบและ ตรวจสอบข้อผิดพลาด

1.4.6 นำฮาร์ดแวร์ที่เขียนไปสังเคราะห์เป็นวงจร

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 มีความรู้ ความเข้าใจ ในเรื่องวิทยาการเข้ารหัส

1.5.2 ได้ความรู้ ประสบการณ์การออกแบบวงจรรวมดิจิทัลเพิ่มมากขึ้น

1.5.3 เข้าใจรูปแบบ การทำงาน และประโยชน์ของ Elliptic Curve

1.5.4 นำความรู้ที่ได้ไปประยุกต์ใช้กับการออกแบบวงจรดิจิทัลอื่นได้

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

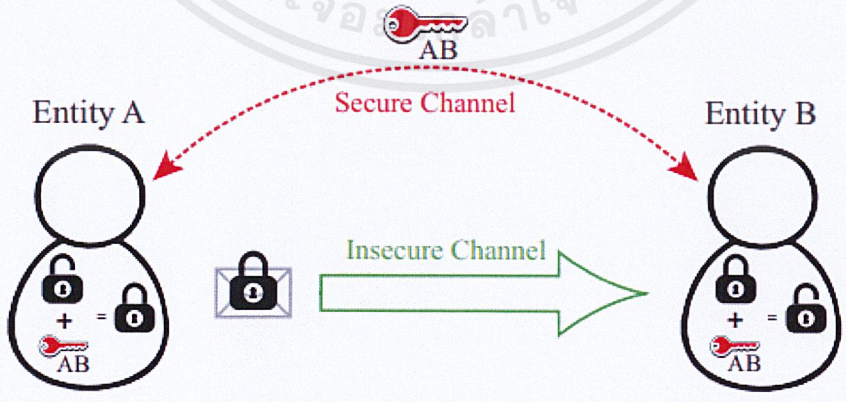
2.1 **วิทยาการเข้ารหัส** (Cryptography)

เป็นวิทยาการรักษาความปลอดภัยสารสนเทศ คำนี้มาจากภาษากรีก kryptos ซึ่งหมายความว่าซ่อน Cryptography สัมพันธ์ใกล้ชิดกับวินัยของ cryptology และCryptanalysis โดยรวมถึงเทคนิค เช่น microdots การควบรวมคำด้วยภาพ และวิธีการอื่นในการซ่อนสารสนเทศในพื้นที่จัดเก็บหรือ ส่งผ่าน อย่างไรก็ตามโลกที่คอมพิวเตอร์เป็นศูนย์กลางในวันนี้ ส่วนใหญ่ Cryptography มักจะ เกี่ยวข้องกับการผสมข้อความธรรมดา (plain text) ไปเป็น ciphertext กระบวนการเรียกว่า encryption จากนั้นย้อนกลับอีกครั้ง เพื่อให้ได้กลับมาเป็นข้อความธรรมดา ซึ่งจะเรียกว่า decryption

วิทยาการเข้ารหัสนั้นได้ถูกคิดค้นมาเป็นเวลามากกว่า 2000 ปีที่แล้ว โดยสิ่งประดิษฐ์ชิ้นแรกในด้านวิทยาการเข้ารหัสคือ The Scytale ถูกประดิษฐ์ขึ้นโดยชาวกรีก ซึ่งเป็นอุปกรณ์ที่ทำขึ้นจากไม้ที่มีด้านสมมาตรเท่ากันทุกด้าน โดยวิธีการเข้ารหัส จะนำกระดาษมาพันรอบ Scytale จากนั้นเขียนข้อความลับตามแนวนอนขนานกับไม้ และต้องใช้ Scytale ที่มีขนาดเดียวกันในการถอดรหัส

2.1.1 กุญแจเดี่ยว (Symmetric key)

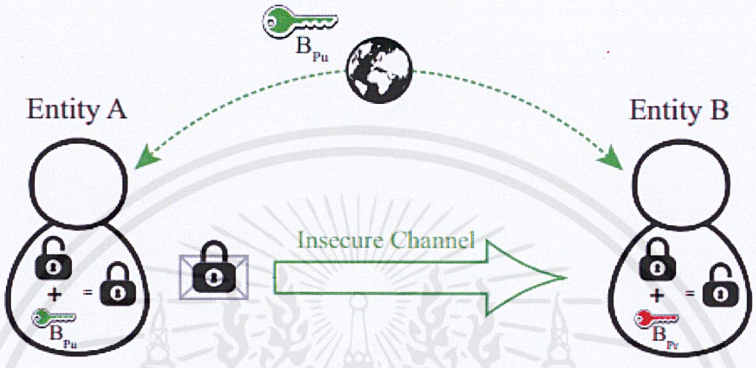
กระบวนการเข้ารหัสแบบที่ผู้ส่งกับผู้รับ โดยใช้กุญแจ (key) ร่วมกัน วิธีการนี้ผู้รับกับผู้ส่งต้องตกลงกันก่อนว่าจะใช้ กุญแจ (key) อะไรในการเข้ารหัสและถอดรหัส หรือ อาจจะเป็นชุดคีย์เดียวกันเลยก็ได้ เช่น ผู้ส่ง ส่งข้อความลับ (Cipher Text) ไปให้ ผู้รับ โดยมี กุญแจ (Key) เป็น “12345” ซึ่งผู้รับก็ต้องรู้ว่ากุญแจคืออะไรถึงจะสามารถแปลง ข้อความลับ (Cipher Text) เป็น ข้อความปกติได้ (Plain Text)



ภาพที่ 2.1 การสื่อสารโดยใช้กุญแจเดี่ยว

2.1.2 กุญแจคู่ (Asymmetric-key)

กระบวนการเข้ารหัสแบบที่ ผู้ส่ง และ ผู้รับ มีกุญแจกันคนละชุดที่เกี่ยวข้องกัน ซึ่งเรียกว่า public key และ private key ในการเข้ารหัสแบบนี้ public keyส่วนใหญ่จะใช้ในการ encryption ข้อความ (plain Text) เป็น ข้อความลับ (Cipher Text) ส่วน private key จะใช้ในการ decryption แปลงข้อความลับ กลับเป็น ข้อความปกติ



ภาพที่ 2.2 การสื่อสารโดยใช้กุญแจคู่

โดย Private key และ Public key จะมีความเกี่ยวข้องกันเนื่องจากในการสร้างกุญแจคู่จะใช้กระบวนการทางคณิตศาสตร์สร้างขึ้นมา เช่น the Integer-Factorization Problem (IFP), the Discrete Logarithm Problem (DLP) หรือ Elliptic Curve Discrete Logarithm Problem ทำให้กุญแจทั้งคู่นั้นมีความเกี่ยวข้องกัน

ในการสร้างกุญแจคู่ โดยการใช้ Elliptic Curve Discrete Logarithm Problem จะทำได้โดยการนำค่าสเกล่าค่าหนึ่งซึ่งจะใช้เป็นPrivate Key (d) มาคูณกับ จุดเริ่มต้นบนกราฟElliptic Curve(P) จะทำให้ได้จุดๆหนึ่งบนกราฟโดยจะใช้เป็นPublic Key(Q) จะได้เป็นสมการ

$$Q = d * P$$

Q = Public key

d = Private key

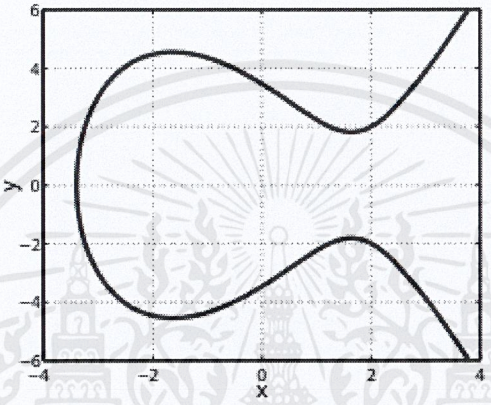
P = Starting point

2.2 เส้นโค้งเชิงวงรี (Elliptic Curve)

เส้นโค้งเชิงวงรีเป็นสมการทางคณิตศาสตร์รูปแบบหนึ่งที่มีรูปทั่วไป ดังนี้

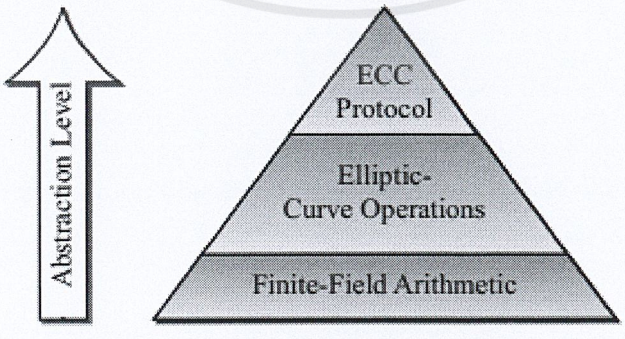
$$y^2 = x^3 + ax + b$$

เมื่อแทนค่าสมการแล้วนำไปเขียนเป็นกราฟจะได้ดังรูป



ภาพที่ 2.3 กราฟเส้นโค้งเชิงวงรี เมื่อ a = -8, b = 12

Elliptic curve cryptography สามารถแบ่งออกเป็นระดับชั้นต่างๆได้ โดยชั้นแรกที่สูงที่สุดจะเป็นตัว Elliptic curve ที่นำไปใช้งานในการเข้ารหัส ชั้นรองลงมาจะเป็นชั้นของOperation ที่เป็นการบวก การคูณ บนเส้นกราฟซึ่งทำให้เกิดเป็นชั้นบนสุด ส่วนชั้นสุดท้ายที่เป็นชั้นล่างสุดเป็นฐานที่ใช้กระบวนการทางคณิตศาสตร์บนพื้นที่จำกัดทำให้เกิดสมการต่างๆ



ภาพที่ 2.4 ระดับชั้นของเส้นโค้งเชิงวงรี

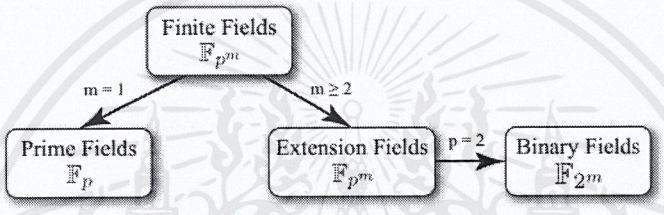
2.2.1 พื้นที่จำกัด (Finite Field)

Finite field คือ เซ็ตของจำนวนที่มีจำกัด แทนด้วย F ซึ่งจำนวนภายใน Finite filed จะถูกเรียกว่า order ถ้า Finite field (F) มีจำนวนสมาชิก (order) เท่ากับ p^m เมื่อ p เป็นจำนวนเฉพาะเรียกว่า characteristic ของF และถ้าm เป็นจำนวนเต็มมากกว่าหรือเท่ากับ 1 เรียกว่า dimension โดยเมื่อ field ถูกเขียนว่า F_{p^m} หรือ GF(p^m) จะหมายความว่าดังนี้

ถ้า m = 1 จะหมายถึง Prime field

เมื่อ m > 1 จะหมายถึง Extension field

แต่ถ้า p = 2 และ m > 1 จะหมายถึง Binary field



ภาพที่ 2.5 โครงสร้างของFinite field

2.2.1.1 Prime field

Prime field คือ Finite field ที่มีorderเป็นจำนวนเฉพาะ GF(p) กระบวนการทางคณิตศาสตร์ของPrime fieldมีดังนี้

Addition : $(a + b) \bmod p$

Subtraction : $(a - b) \bmod p$

Multiplication : $(a*b) \bmod p$

Division : $a/b \bmod p = (a*b^{-1}) \bmod p$

Inversion : $a^{-1} \bmod p$, $(a*a^{-1}) \bmod p = 1$

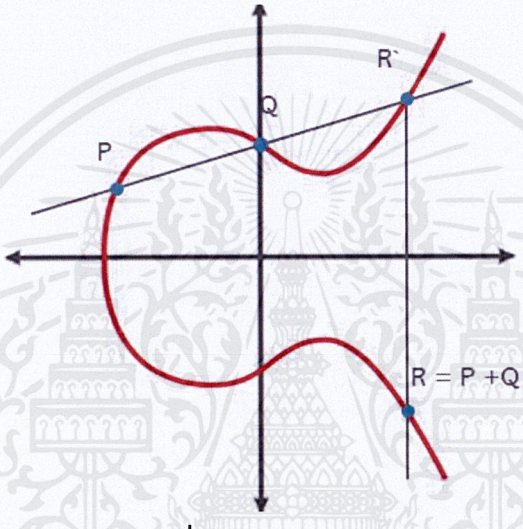
เมื่อ p คือ จำนวนสมาชิกบนfield เป็นจำนวนเฉพาะ

2.2.2 Elliptic Curve Operation

คือการกระทำของจุดต่างๆที่อยู่บนกราฟ elliptic curve มีอยู่ 3 แบบ คือ การบวกจุด(Point Addition), การคูณสอง(Point Doubling)และการคูณจุด(Point Multiplication)

2.2.2.1 การบวกจุดบนPrime field (Point Addition)

การบวกกันของจุดบนกราฟเส้นโค้งเชิงวงรีสามารถทำได้โดยการลากเส้นผ่านจุดสองจุดที่ต้องการบวกกัน ซึ่งจะไปตัดจุดอีกจุดบนเส้นกราฟจากนั้นสะท้อนจุดนั้นตามแกนx ซึ่งเป็นจุดที่เป็นผลลัพธ์



ภาพที่ 2.6 Point Addition

เมื่อนำวิธีการบวกข้างต้นมาทำเป็นสมการจะได้ ดังนี้

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$$

$$x_R = \lambda^2 - x_P - x_Q$$

$$y_R = \lambda(x_P - x_R) - y_P$$

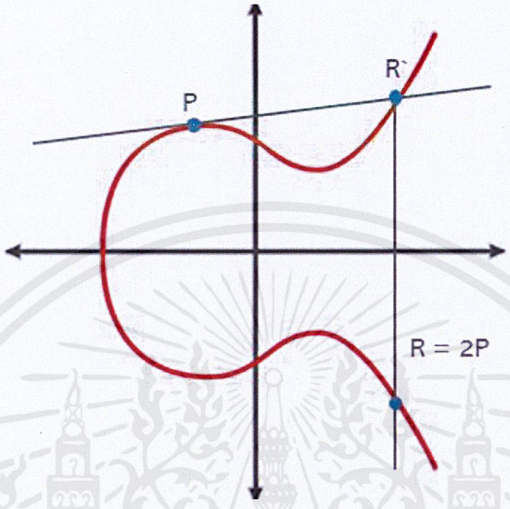
เมื่อ

R คือ ผลลัพธ์การบวก

P, Q คือ จุดใดๆ ที่ต้องการบวก

2.2.2.2 การคูณสองบนPrime field (Point Doubling)

การคูณสองของจุดบนเส้นกราฟเชิงวงรี สามารถทำได้โดยการลากเส้นตั้งฉากกับจุดที่ต้องการคูณ ซึ่งจะไปตัดจุดอีกจุดบนเส้นกราฟจากนั้นสะท้อนจุดนั้นตามแกนx ซึ่งเป็นจุดที่เป็นผลลัพธ์



ภาพที่ 2.7 Point Doubling

เมื่อเขียนเป็นสมการจะได้ว่า

$$\lambda = \frac{3x_P^2 + a}{2y_P}$$
$$x_R = \lambda^2 - 2x_P$$
$$y_R = \lambda(x_P - x_R) - y_P$$

เมื่อ

R คือ ผลลัพธ์การคูณสอง

P คือ จุดใดๆ ที่ต้องการคูณสอง

2.2.2.3 การคูณจุด(Point Multiplication)

เป็นการประยุกต์ใช้การบวกและการคูณสอง ซ้ำไปเรื่อยๆเพื่อให้ได้ผลลัพธ์เหมือนการคูณโดยมีหลายวิธีการในการคูณเช่น shift-add และ Montgomery Ladder เป็นต้น

2.3 ตัวแปรที่เกี่ยวข้องบน Finite field P192

ในการออกแบบระบบเข้ารหัสจะมีค่าของตัวแปรที่เกี่ยวข้อง ได้ถูกกำหนดไว้เป็นมาตรฐานเพื่อให้มีความเข้าใจที่ตรงกัน ซึ่งในECC ก็มีการกำหนดตัวแปรเช่นกัน โดยจะมีดังนี้

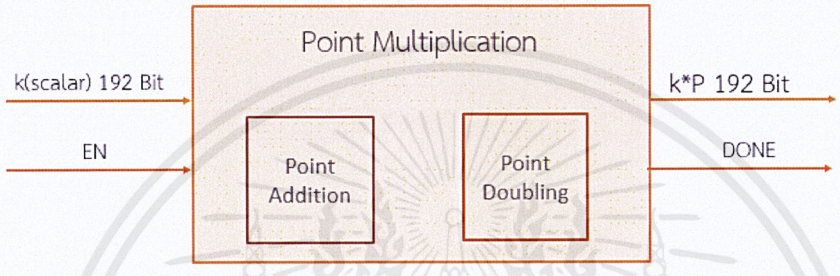
Domain Parameter Descriptions		
a, b	...	Coefficients describing the elliptic curve.
x	...	The x coordinate of the base point P.
y	...	The y coordinate of the base point P.
n	...	The order of the base point P.
p	...	Prime field order.

Domain Parameter Values		
a	=	−3
b	=	0x 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1
x	=	0x 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012
y	=	0x 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811
n	=	0x FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831
p	=	0x FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFF

ภาพที่ 2.8 แสดงตัวแปรที่เกี่ยวข้องใน P192

บทที่ 3
การดำเนินงาน

การทำงานของระบบจะเริ่มจากการรับค่าสเกล่าที่มีขนาด 192 บิต จากนั้นจะเริ่มทำงานก็ต่อเมื่อมีสัญญาณENถูกส่งมายังระบบ เมื่อได้รับสัญญาณดังกล่าวระบบจะเริ่มคำนวณ หลังจากคำนวณเสร็จสัญญาณ DONE จะขึ้นเป็น High เพื่อบอกว่าเอาท์พุทขนาด 192 บิต พร้อมใช้งานแล้ว



ภาพที่ 3.1 ภาพรวมการทำงานของระบบ

3.1 **การออกแบบส่วน** Finite field Arithmetic

ในการออกแบบส่วนนี้ การบวกลบและmod จะใช้เป็นวงจรสำเร็จที่ภายในโปรแกรมมีให้ ส่วนการคูณนั้นจะใช้เป็นวงจรshift – add ธรรมดา ส่วนการinvertจะใช้วิธีการดังนี้

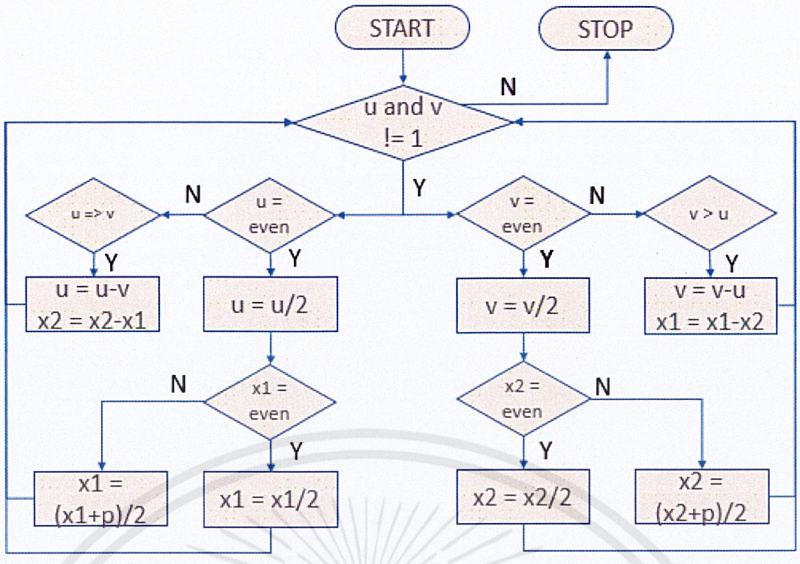
Algorithm A.3 Inversion in $\mathbb{F}_p$ using the binary GCD algorithm.

Input: Prime p, $a \in [1, p-1]$.
Output: $a^{-1} \mod p$.

1: $u = a$, $v = p$.
2: $x_1 = 1$, $x_2 = 0$.
3: **while** $u \neq 1$ **and** $v \neq 1$ **do**
4: **while** u is even **do**
5: $u = u/2$.
6: **if** x_1 is even **then**
7: $x_1 = x_1/2$.
8: **else**
9: $x_1 = (x_1 + p)/2$.
10: **end if**
11: **end while**
12: **while** v is even **do**
13: $v = v/2$.
14: **if** x_2 is even **then**
15: $x_2 = x_2/2$.
16: **else**
17: $x_2 = (x_2 + p)/2$.
18: **end if**
19: **end while**

ภาพที่ 3.2 การ invert โดยใช้ binary GCD

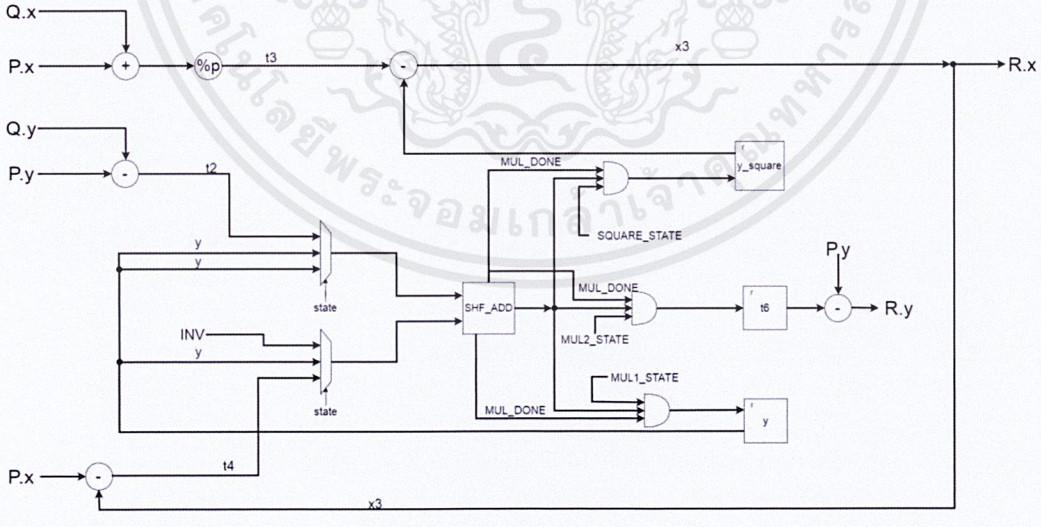
เมื่อนำ algorithm ข้างต้นมาออกแบบเป็นFlow chart จะได้ว่า



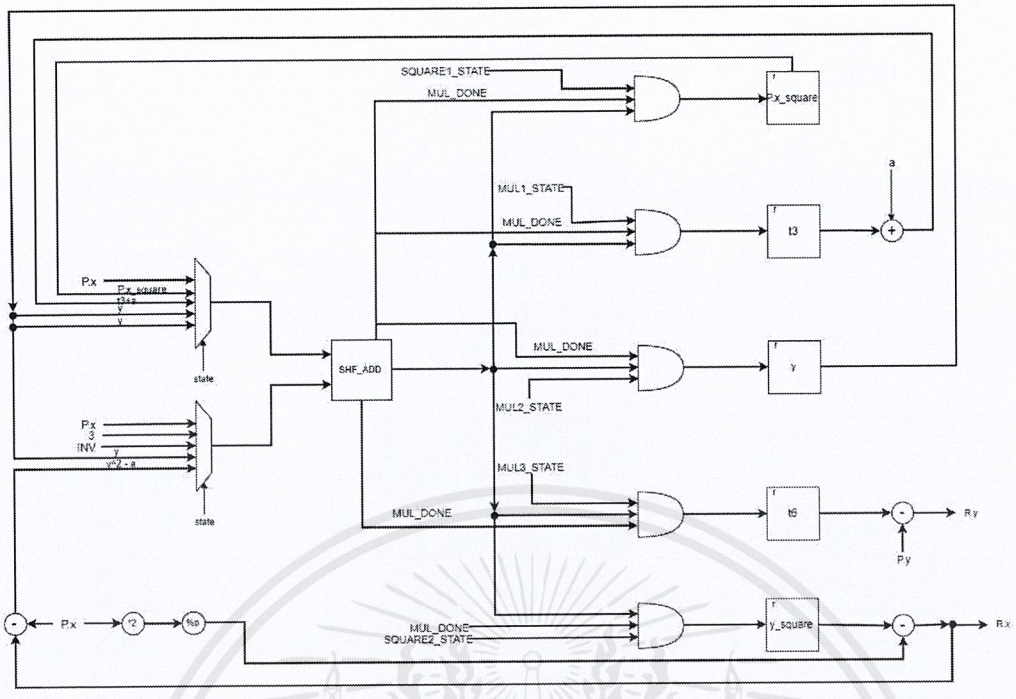
ภาพที่ 3.3 Flow Chart การ Invert

3.2 การออกแบบส่วน Point Addition **และ** Point Doubling

จากสมการการบวก และการคูณสองในบทที่ผ่านมา ทำให้เราสามารถออกแบบเป็นวงจรได้ โดยในส่วนนี้จะรับค่ามาจากการ Invert ทำให้ภายในส่วนนี้จะมีเพียงวงจรบวก Modและคูณ ซึ่งจะใช้วงจรคูณเพียงชุดเดียวเพื่อลดขนาดของวงจรลง จะทำให้เหลือวงจรดังรูป

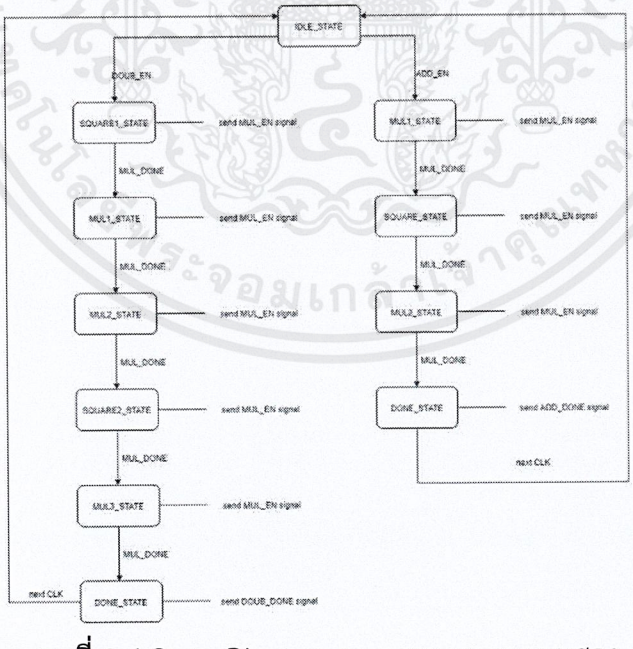


ภาพที่ 3.4 วงจรPoint Addition



ภาพที่ 3.5 วงจรPoint Doubling

เนื่องจากวงจรทั้งสองมีลักษณะที่คล้ายกันมาก ทำให้สามารถยุบรวมวงจรเป็นวงจรเดียวและใช้สัญญาณควบคุมมาเป็นตัวควบคุมว่าจะใช้การบวกหรือ การคูณสอง ซึ่งจะลดขนาดของวงจรได้อย่างมาก



ภาพที่ 3.6 State Diagram ของวงจรบวกและคูณสอง

3.3 การออกแบบ Point Multiplication

ในการออกแบบส่วนการคูณจุด นั้นเลือกใช้เป็นวิธีการMontgomery Ladderเนื่องจากเป็นวิธีการที่ในทุกรอบจะทำเหมือนกันคือ Doubling และ Addition ทำให้มีความปลอดภัยค่อนข้างสูง โดยการทำงานของวิธีการMontgomery Ladderจะเป็นดังนี้

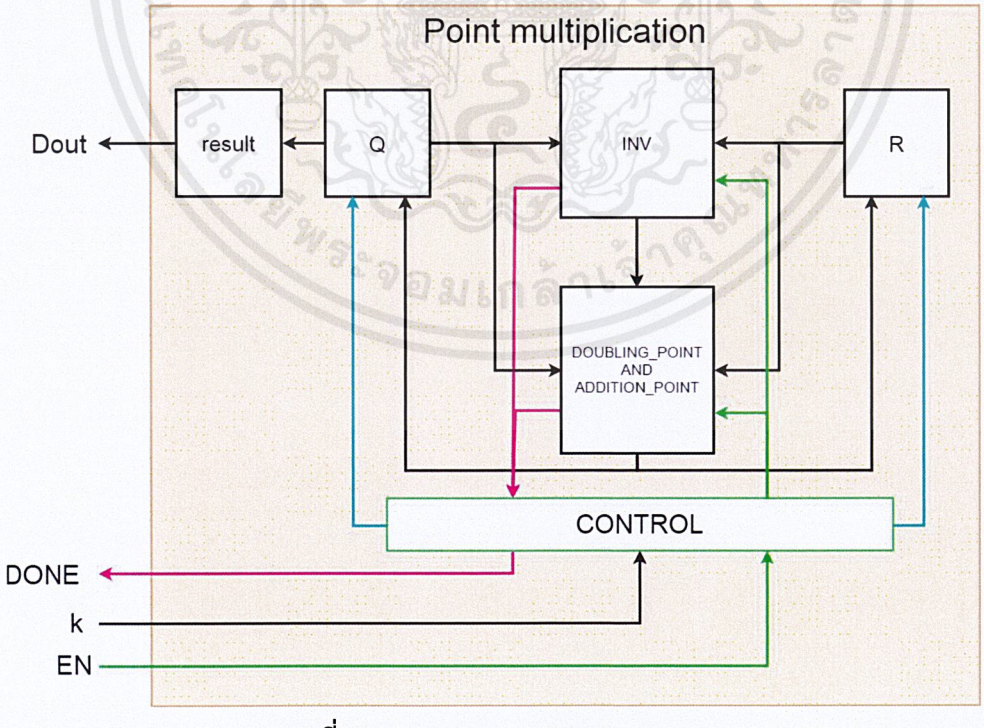
Montgomery Ladder algorithm

Input : scalar $k = (k_{t-1}, \ldots, k_0)_2$
Output : $Q = k \times P$

```
1: Q = 0, R = P
2: for i = t - 1 downto 0 do
3:         if k_i = 1 then
4:                 Q = Q + R
5:                 R = 2R
6:         else
7:                 R = R + Q
8:                 Q = 2Q
9:         end if
10: end for
11: return Q
```

ภาพที่ 3.7 แสดงการทำงานของMontgomery Ladder

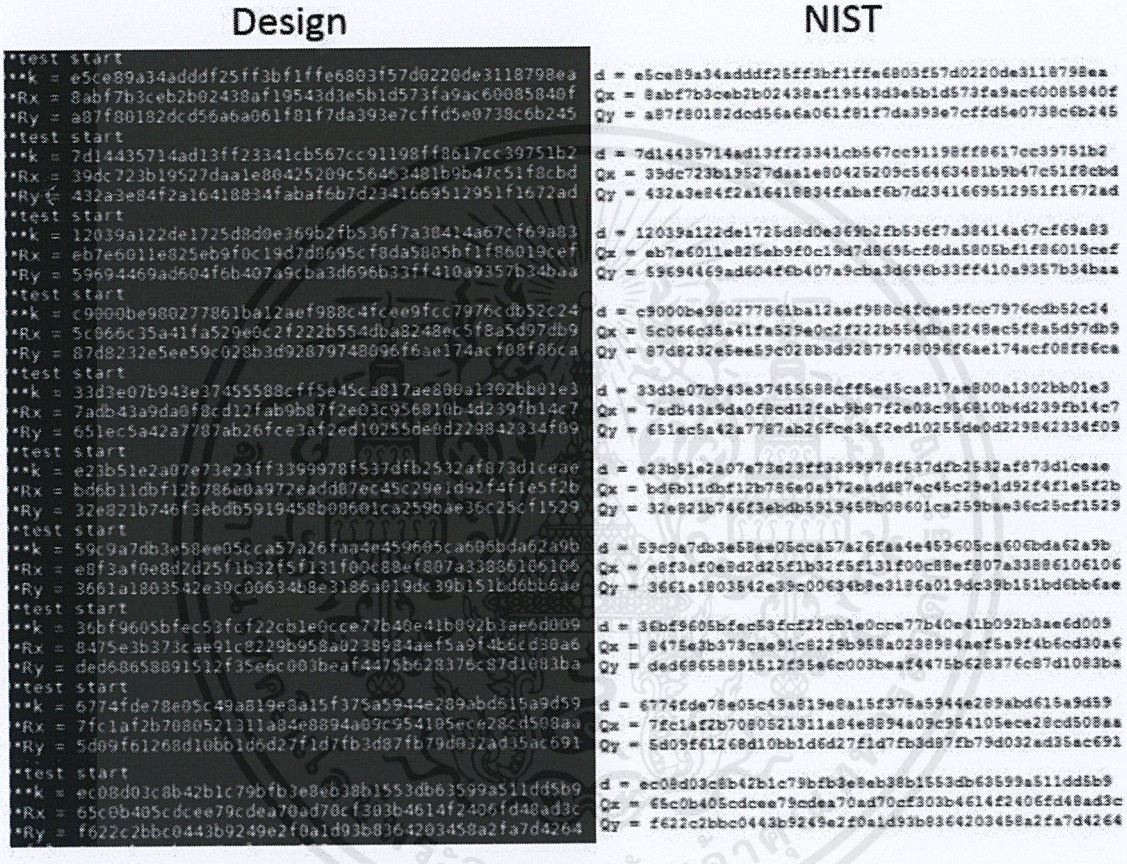
เมื่อนำวิธีการนี้ไปใช้จะได้วงจรดังนี้



ภาพที่ 3.8 วงจรPoint Multiplication

บทที่ 4
ผลการวิจัย

ในการทดสอบการทำงานของวงจรพบว่ามีผลลัพธ์ถูกต้องตามมาตรฐานของ National Institute of Standards and Technology หรือ NIST

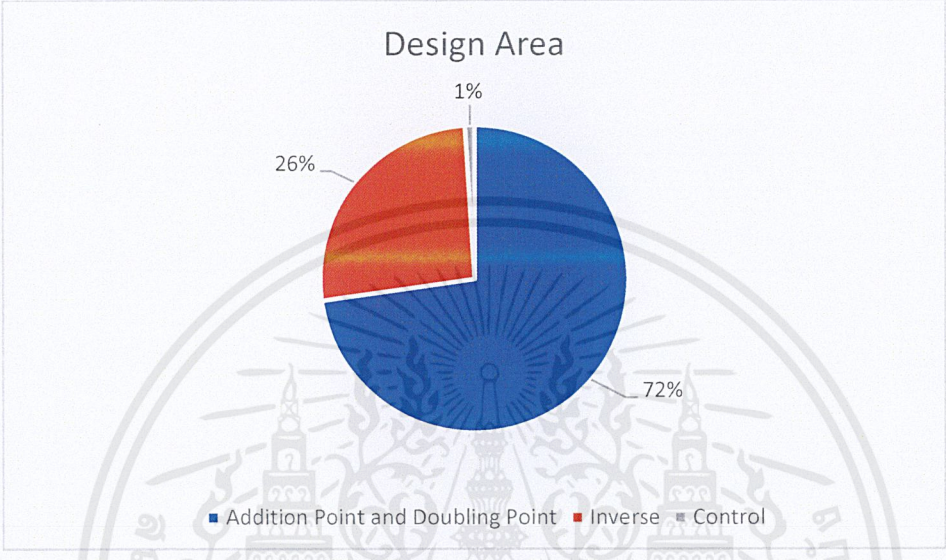


ภาพที่ 4.1 การเปรียบเทียบผลลัพธ์ระหว่างDesign กับ NIST

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

การวิจัยครั้งนี้ได้ใช้พื้นที่ไปทั้งหมด 69436 GE โดยแบ่งเป็นแต่ละส่วนได้ดังนี้



ภาพที่ 5.1 กราฟพื้นที่ที่ใช้

ใช้เวลาการทำงานทั้งหมดประมาณ 506432 CLK แบ่งเป็น

Addition Point ต่อครั้ง 584 CLK

Doubling Point ต่อครั้ง 972 CLK

Inversion ต่อครั้งไม่เกิน 615 CLK

5.2 ข้อเสนอแนะ

ในการออกแบบวงจรคูณจุด(Point Multiplication) สามารถลดขนาดได้อีก โดยการออกแบบเสมือน CPU ซึ่งจะใช้วงจรบวก คูณ Mod เพียงอย่างละหนึ่งวงจรเท่านั้น แต่จะเพิ่มความซับซ้อนในการออกแบบวงจรอย่างมาก นอกจากนี้อาจจะต้องคำนึงถึงเวลาการทำงานที่อาจจะเพิ่มขึ้นมา ต้องวิเคราะห์ดูว่าเวลาที่เสียไปนั้นคุ้มกับขนาดที่ลดลงหรือไม่

เอกสารอ้างอิง

[1] Michael Muhlberghuber. (2557). ComparingECDSAHardwareImplementations based on Binary and Prime Fields, สืบค้นเมื่อ 29 สิงหาคม,2562, จากเว็บไซต์ https://www.fields.utoronto.ca/programs/scientific/0708/cryptography/dana_neustadter.pdf

[2] NIST. (ม.ป.ป). Cryptographic Algorithm Validation Program, สืบค้นเมื่อ 23 ตุลาคม,(2562),https://csrc.nist.gov/Projects/Cryptographic-Algorithm-Validation-Program/Component-Testing#ECCCDH

[3] Avi Kak. (2562).Finite field, สืบค้นเมื่อ 13 สิงหาคม, 2562, จากเว็บไซต์ https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture7.pdf

[4] ANDREA CORBELLINI. (2558), Elliptic Curve Cryptography: finite fields and discrete logarithms, สืบค้นเมื่อ 28 สิงหาคม, 2562, จากเว็บไซต์ https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/

[5] Abdulah Abdulah Zadeh. (2012). Division and Inversion Over Finite Fields, สืบค้นเมื่อ 10 กันยายน, 2562,http://cdn.intechopen.com/pdfs/29704/InTech-Division_and_inversion_over_finite_fields.pdf

[6] NECTEC. (มปป).IC Fabrication, สืบค้นเมื่อ 30 สิงหาคม, 2562, จากเว็บไซต์ http://tmec.nectec.or.th/public/uploaded/Knowledge/IC%20design.pdf

ภาคผนวก

ภาคผนวก ก

วงจรAddition and Doubling

```
module  AD_POINT    #(  parameter [191:0] p = 192'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF,
                        parameter [191:0] a = -3,
                        parameter   NUM_BIT      = 192
                    )
                    (   input   CLK,
                        input   RSTn,
                        input   ADD_EN,
                        input   DOUB_EN,

                        output  ADD_DONE,
                        output  DOUB_DONE,
                        // R=P+Q
                        input   [NUM_BIT-1 : 0]Px,
                        input   [NUM_BIT-1 : 0]Py,
                        input   [NUM_BIT-1 : 0]Qx,
                        input   [NUM_BIT-1 : 0]Qy,
                        input   [NUM_BIT-1 : 0]INV,

                        output  [NUM_BIT-1 : 0]Rx,
                        output  [NUM_BIT-1 : 0]Ry
                    );

        localparam  IDLE_STATE   = 0;
        localparam  AMUL1_STATE  = 1;
        localparam  ASQUARE_STATE = 2;
        localparam  AMUL2_STATE  = 3;

        localparam  DSQUARE1_STATE = 4;
        localparam  DMUL1_STATE  = 5;
        localparam  DMUL2_STATE  = 6;
        localparam  DSQUARE2_STATE = 7;
        localparam  DMUL3_STATE  = 8;

        localparam  ADONE_STATE  = 9;
        localparam  DDONE_STATE  = 10;
        localparam  AW_STATE     = 11;
        localparam  DW_STATE     = 12;
        localparam  STATE_BIT    = 4;

        reg     [STATE_BIT-1 : 0]state;
        wire    MUL_DONE;
        wire    [NUM_BIT-1 : 0] DOUT_MUL;
        ////CAL + -
        reg     [NUM_BIT-1 : 0] sq1;
        reg     [NUM_BIT-1 : 0] sq2;
        reg     [NUM_BIT-1 : 0] mul1 ;
        reg     [NUM_BIT-1 : 0] mul2 ;
        reg     [NUM_BIT-1 : 0] mul3 ;
        reg     [NUM_BIT-1 : 0] resultx ;
        reg     [NUM_BIT-1 : 0] resulty ;


        wire    [NUM_BIT-1 : 0] t1 = INV; ///                          inv(Xq - Xp)

        wire    [NUM_BIT : 0]  px2 =  {Px[NUM_BIT-1:0],1'b0} ;
        wire    [NUM_BIT-1 : 0] t2 = (px2 % p); ///

        wire    [NUM_BIT   : 0] a1m = Qx + Px;
        wire    [NUM_BIT-1 : 0] Da1 = mul1 + a;
        wire    [NUM_BIT-1 : 0] a1 = a1m % p;////Qx + Px

        wire    [NUM_BIT-1 : 0] m11= (ADD_EN && DOUB_EN == 0) ? Qy: sq2;
        wire    [NUM_BIT-1 : 0] m12= (ADD_EN && DOUB_EN == 0) ? Py: t2;
        wire    [NUM_BIT-1 : 0] m1 = (m11 >= m12) ? (m11 - m12): p - (m12 - m11);  /// Rx in DOUB

        wire    [NUM_BIT-1 : 0] m31= (ADD_EN && DOUB_EN == 0) ? sq1: mul3;
        wire    [NUM_BIT-1 : 0] m32= (ADD_EN && DOUB_EN == 0) ? a1: Py;
        wire    [NUM_BIT-1 : 0] m3 = (m31 >= m32) ? (m31 - m32) : p - (m32 - m31);///   lampda^2 - (Qx + Px) in add

        wire    [NUM_BIT-1 : 0] m21= (ADD_EN && DOUB_EN == 0) ? Px: Px;
        wire    [NUM_BIT-1 : 0] m22= (ADD_EN && DOUB_EN == 0) ? m3: m1;
        wire    [NUM_BIT-1 : 0] m2 = (m21 >= m22) ? (m21 - m22): p - (m22 - m21); ////Px - Rx in ADD Px-Rx

        wire    [NUM_BIT-1 : 0] m41= mul2;
        wire    [NUM_BIT-1 : 0] m42= Py;
        wire    [NUM_BIT-1 : 0] m4 = (m41 >= m42) ? (m41 - m42): p - (m42 - m41); ////lampda*(Px - Rx) - Py in ADD

        ///CONNECT SIGNAL TO MUL&INV

        wire    MUL_EN = (state == AMUL1_STATE  ) ? 1 : ///  ENABLE MULTIPLY
                         (state == ASQUARE_STATE) ? 1 :
                         (state == AMUL2_STATE  ) ? 1 :
                         (state == DMUL1_STATE   )  ? 1 : ///  ENABLE MULTIPLY
                         (state == DMUL2_STATE   )  ? 1 :
                         (state == DSQUARE1_STATE )  ? 1 :
                         (state == DSQUARE2_STATE )  ? 1 :
```

```
                    (state == DMUL3_STATE    ) ? 1 : 0;

wire    [NUM_BIT-1 : 0] DIN_MUL_A = (state == AMUL1_STATE)  ? t1 : //// DATA A TO MUL
                                    (state == ASQUARE_STATE)    ? mul1  :
                                    (state == AMUL2_STATE)      ? mul1  :
                                    (state == DSQUARE1_STATE)   ? Px : //// DATA A TO MUL
                                    (state == DMUL1_STATE)      ? 3  :
                                    (state == DMUL2_STATE)      ? Da1  :
                                    (state == DSQUARE2_STATE)   ? mul2  :
                                    (state == DMUL3_STATE)      ? mul2 :
                                    0;
wire    [NUM_BIT-1 : 0] DIN_MUL_B = (state == AMUL1_STATE)  ? m1 ://// DATA B TO MUL
                                    (state == ASQUARE_STATE)    ? mul1  :
                                    (state == AMUL2_STATE)      ? m2 :
                                    (state == DSQUARE1_STATE)   ? Px ://// DATA B TO MUL
                                    (state == DMUL1_STATE)      ? sq1  : //// px_square*3
                                    (state == DMUL2_STATE)      ? t1 :
                                    (state == DSQUARE2_STATE)   ? mul2  :
                                    (state == DMUL3_STATE)      ? m2 :
                                    0;

assign  Rx = resultx;
assign  Ry = resulty;

assign  ADD_DONE = (state == ADONE_STATE)? 1 : 0;
assign  DOUB_DONE = (state == DDONE_STATE)? 1 : 0;
SHF_ADD     mul(    .CLK    (CLK),
                    .RSTn   (RSTn),
                    .EN     (MUL_EN),
                    .A      (DIN_MUL_A),
                    .B      (DIN_MUL_B),
                    .C      (DOUT_MUL),
                    .DONE   (MUL_DONE)
                );

always@(posedge CLK or negedge RSTn)
    begin
    if(~RSTn)
        state <= IDLE_STATE;
    else begin
        case(state)
                IDLE_STATE      :   begin
                                        if(ADD_EN == 1 && DOUB_EN == 0)
                                            state <= AMUL1_STATE;
                                        else if(DOUB_EN == 1 && ADD_EN == 0)
                                            state <= DSQUARE1_STATE;
                                        else
                                            state <= state;
                                    end
                AMUL1_STATE     :   begin
                                        if(MUL_DONE)
                                            state <= ASQUARE_STATE;
                                        else
                                            state <= state;
                                    end
                ASQUARE_STATE   :   begin
                                        if(MUL_DONE)
                                            state <= AMUL2_STATE;
                                        else
                                            state <= state;
                                    end
                AMUL2_STATE     :   begin
                                        if(MUL_DONE)
                                            state <= AW_STATE;
                                        else
                                            state <= state;
                                    end
                AW_STATE        :   begin
                                        state <= ADONE_STATE;
                                    end
                ADONE_STATE     :   begin
                                        state <= IDLE_STATE;
                                    end
                DSQUARE1_STATE  :   begin
                                        if(MUL_DONE)
                                            state <= DMUL1_STATE;
                                        else
                                            state <= state;
                                    end
                DMUL1_STATE     :   begin
                                        if(MUL_DONE)
                                            state <= DMUL2_STATE;
                                        else
                                            state <= state;
```

```
                                    end
                    DMUL2_STATE     :   begin
                                        if(MUL_DONE)
                                            state <= DSQUARE2_STATE;
                                        else
                                            state <= state;
                                    end
                    DSQUARE2_STATE  :   begin
                                        if(MUL_DONE)
                                            state <= DMUL3_STATE;
                                        else
                                            state <= state;
                                    end
                    DMUL3_STATE     :   begin
                                        if(MUL_DONE)
                                            state <= DW_STATE;
                                        else
                                            state <= state;
                                    end
                    DW_STATE        :   begin
                                        state <= DDONE_STATE;
                                    end
                    DDONE_STATE     :   begin
                                        state <= IDLE_STATE;
                                    end
                    default: state <= IDLE_STATE;
                endcase
            end
        end
    always@(posedge CLK or negedge RSTn) begin
        if(~RSTn)
            mul1 <= 0;          /// y in ADD px_square*3
        else begin
            if(state == AMUL1_STATE && MUL_DONE)
                mul1 <= DOUT_MUL;
            else if(state == DMUL1_STATE && MUL_DONE)
                mul1 <= DOUT_MUL;
            else begin
                mul1 <= mul1;
            end
        end
    end
```

```
    always@(posedge CLK or negedge RSTn) begin
        if(~RSTn)
            sq2 <= 0;   /// y_square in ADD
        else begin
            if(state == DSQUARE2_STATE && MUL_DONE)
                sq2 <= DOUT_MUL;
            else begin
                sq2 <= sq2;
            end
        end
    end
    always@(posedge CLK or negedge RSTn) begin
        if(~RSTn)
            sq1 <= 0;   /// y_square in ADD px_square in D
        else begin
            if(state == ASQUARE_STATE && MUL_DONE)
                sq1 <= DOUT_MUL;
            else if(state == DSQUARE1_STATE && MUL_DONE)
                sq1 <= DOUT_MUL;
            else begin
                sq1 <= sq1;
            end
        end
    end
    always@(posedge CLK or negedge RSTn) begin
        if(~RSTn)
            mul2 <= 0;  //// t5 in ADD y
        else begin
            if(state == AMUL2_STATE && MUL_DONE)
                mul2 <= DOUT_MUL;
            else if(state == DMUL2_STATE && MUL_DONE)
                mul2 <= DOUT_MUL;
            else begin
                mul2 <= mul2;
            end
        end
    end
    always@(posedge CLK or negedge RSTn) begin
        if(~RSTn)
            mul3 <= 0;
        else begin
            if(state == DMUL3_STATE && MUL_DONE)
```

```
                    mul3 <= DOUT_MUL;
                else begin
                    mul3 <= mul3;
                end
            end
        end
        always@(posedge CLK or negedge RSTn) begin
            if(~RSTn)
                resultx <= 0;
            else begin
                if(ADD_EN == 1 && DOUB_EN == 0)
                    if(Qx == Px)
                        resultx <= 0;
                    else if(Py == 0 && Px == 0)
                        resultx <= Qx;
                    else if(Qy == 0 && Qx == 0)
                        resultx <= Px;
                    else begin
                        resultx <= m3;
                    end
                else if(DOUB_EN == 1 && ADD_EN == 0)
                    if(Py == 0)
                        resultx <= 0;
                    else begin
                        resultx <= m1;
                    end
                else begin
                    resultx <= resultx;
                end
            end
        end
        always@(posedge CLK or negedge RSTn) begin
            if(~RSTn)
                resulty <= 0;
            else begin
                if(ADD_EN == 1 && DOUB_EN == 0)
                    if(Qx == Px)
                        resulty <= 0;
                    else if(Py == 0 && Px == 0)
                        resulty <= Qy;
                    else if(Qy == 0 && Qx == 0)
                        resulty <= Py;

                    else begin
                        resulty <= m4;
                    end
                else if(DOUB_EN == 1 && ADD_EN == 0)
                    if(Py == 0)
                        resulty <= 0;
                    else begin
                        resulty <= m3;
                    end
                else begin
                    resulty <= resulty;
                end
            end
        end
    endmodule
```

วงจร Inversion

```
module MUL_INV  #(
                        parameter   NUM_BIT  = 192,
                        parameter [191:0] p = 192'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFF
                )
                (
                    input   CLK,
                    input   RSTn,
                    input   EN,
                    input   [NUM_BIT  - 1 : 0]  DIN,
                    output  [NUM_BIT  - 1 : 0]  INV,
                    output   DONE
                );

            localparam  IDLE_STATE   = 0;
            localparam  INIT_STATE   = 1;
            localparam  UX1_STATE    = 2;
            localparam  VX2_STATE    = 3;
            localparam  COMP_STATE   = 4;
            localparam  RESULT_STATE = 5;
            localparam  DONE_STATE   = 6;
            localparam   STATE_BIT    = 3;
                /// P+Q
            //variable to inv

            reg  [STATE_BIT - 1 : 0]   state;
            reg  [NUM_BIT-1 : 0]       u;
            reg  [NUM_BIT-1 : 0]       v;
            reg  [NUM_BIT-1 : 0]       x1;
            reg  [NUM_BIT-1 : 0]       x2;
            reg  [NUM_BIT - 1 : 0]     inv;

            wire [NUM_BIT : 0]   x1p;
            wire [NUM_BIT : 0]   x2p;
            wire [NUM_BIT-1:0] uv1 = (u>=v)?u:v;
            wire [NUM_BIT-1:0] uv2 = (u>=v)?v:u;
            wire [NUM_BIT-1:0] deluv = uv1 - uv2;

            wire    [NUM_BIT-1 : 0] m11= (u>=v) ? x1: x2;
            wire    [NUM_BIT-1 : 0] m12= (u>=v) ? x2: x1;
            wire    [NUM_BIT-1 : 0] delx12 = (m11 >= m12) ? (m11 - m12): p - (m12 - m11);

            wire [NUM_BIT-1:0] div = (state == UX1_STATE) ? u :
                                     (state == VX2_STATE) ? v :0;

            wire [NUM_BIT-1:0] divuv = (div[0] == 1) ? div : {1'b0,div[(NUM_BIT-1) : 1]};

            wire [NUM_BIT-1:0] div2 = (state == UX1_STATE && x1[0] == 0) ? x1 :
                                      (state == VX2_STATE && x2[0] == 0) ? x2 :0;
            wire [NUM_BIT-1:0] divx12 = (div[0] == 1) ? div2 : {1'b0,div2[(NUM_BIT-1) : 1]};

            wire [NUM_BIT:0] divxp2 =(state == UX1_STATE && x1[0] == 1) ? x1p :
                                     (state == VX2_STATE && x2[0] == 1) ? x2p :0;
            wire [NUM_BIT-1:0] divxp12 = (div[0] == 1) ? divxp2 : {1'b0,divxp2[(NUM_BIT) : 1]};

            assign  INV = inv;
            assign  x1p = (x1+p);
            assign  x2p = (x2+p);
            assign  DONE = (state == DONE_STATE)? 1: 0;

            ////////////////////////////////INVERSE/////////////////////////////////////////
            always@(posedge CLK or negedge RSTn)
                begin
                    if(~RSTn)
                        inv <= 0;
                    else begin
                        if(state != RESULT_STATE)
                            inv <= inv;
                        else begin
                            if(u == 1)
                                inv <= x1;
                            else begin
                                inv <= x2;
                            end
                        end
                    end
                end
            always@(posedge CLK or negedge RSTn)
                begin
                if(~RSTn)
                    state <= IDLE_STATE;
                else begin
                    case(state)
                            IDLE_STATE      :   begin
                                                    if(EN)
                                                        state <= INIT_STATE;
                                                    else
```

```
86                                              state <= state;
87                                          end
88
89                      INIT_STATE      :   begin
90                                          if(DIN == 0)
91                                              state <= RESULT_STATE;
92                                          else if(( u!=1 )&&(v != 1))
93                                              state <= UX1_STATE;
94                                          else
95                                              state <= RESULT_STATE;
96                                          end
97
98                      UX1_STATE       :   begin
99                                          if(u[0]==1)
100                                             state <= VX2_STATE;
101                                         else
102                                             state <= state;
103                                         end
104                     VX2_STATE       :   begin
105                                         if(v[0]==1)
106                                             state <= COMP_STATE;
107                                         else
108                                             state <= state;
109                                         end
110                     COMP_STATE      :   begin
111                                         if(( u!=1 )&&(v != 1))
112                                             state <= UX1_STATE;
113                                         else
114                                             state <= RESULT_STATE;
115                                         end
116                     RESULT_STATE    :   begin
117                                             state <= DONE_STATE;
118                                         end
119                     DONE_STATE      :   begin
120                                             state <= IDLE_STATE;
121                                         end
122                     default: state <= IDLE_STATE;
123                 endcase
124             end
125         end
126     always@(posedge CLK or negedge RSTn)
127         begin
```

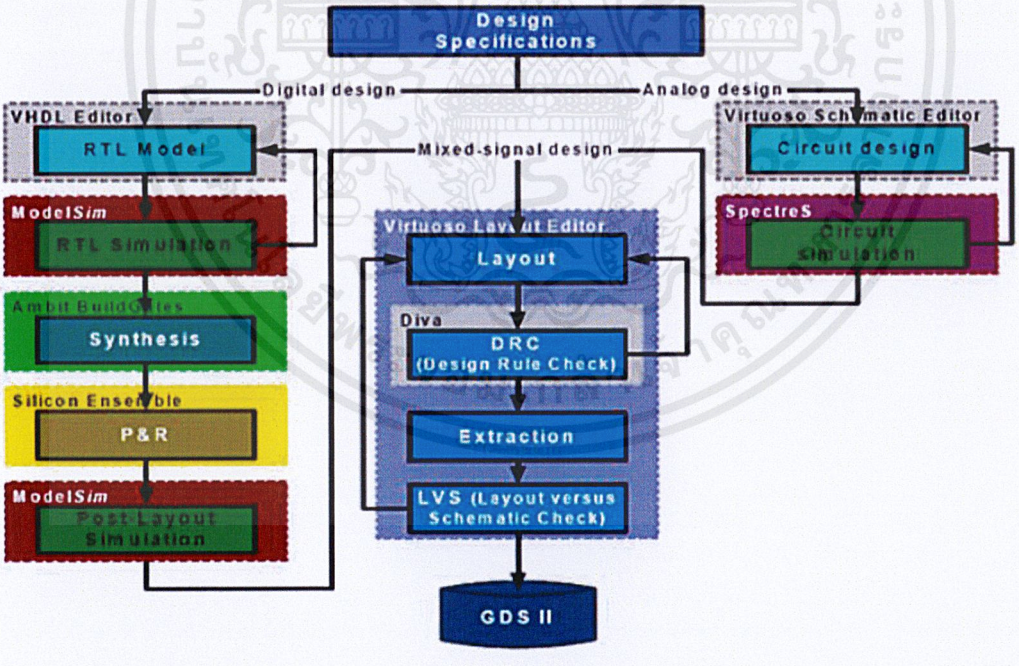
```
128             if(~RSTn)
129                 u <= 0;
130             else begin
131                 if(state == IDLE_STATE)
132                 u <= 0;
133                 else if(state == INIT_STATE)
134                     u   <=  DIN;
135                 else if(state == UX1_STATE)
136                     u <= divuv;
137                 else if(state == COMP_STATE && u >= v)
138                     u <= deluv;
139                 else begin
140                     u <= u;
141                 end
142             end
143         end
144     always@(posedge CLK or negedge RSTn)
145         begin
146             if(~RSTn)
147                 v <= 0;
148             else begin
149                 if(state == IDLE_STATE)
150                     v <= 0;
151                 else if(state == INIT_STATE)
152                     v   <=  p;
153                 else if(state == VX2_STATE)
154                     v <= divuv;
155                 else if(state == COMP_STATE && u < v)
156                     v <= deluv;
157                 else begin
158                     v <= v;
159                 end
160             end
161         end
162     always@(posedge CLK or negedge RSTn)
163         begin
164             if(~RSTn)
165                 x1 <= 1;
166             else begin
167                 if(state == IDLE_STATE)
168                     x1 <= 1;
169                 //else if(state == EXEC_STATE && x1[0] == 0 && u[0] == 0 && x1[NUM_BIT+2] == 1)
```

```
170                     //    x1 <= {1'b1,x1[(NUM_BIT+2) : 1]};
171                     else if(state == UX1_STATE && x1[0] == 0 && u[0] == 0)
172                         x1 <= divx12;
173                     else if(state == UX1_STATE && x1[0] == 1 && u[0] == 0)
174                         x1 <= divxp12;
175                     else if(state == COMP_STATE && u >= v)
176                         x1 <= delx12;
177                     else begin
178                         x1 <= x1;
179                     end
180                 end
181             end
182         always@(posedge CLK or negedge RSTn)
183             begin
184                 if(~RSTn)
185                     x2 <= 0;
186                 else begin
187                     if(state == IDLE_STATE)
188                         x2 <= 0;
189                     //else if(state == EXEC_STATE && x2[0] == 0 && v[0] == 0 && x2[NUM_BIT+2] == 1)
190                     //    x2 <= {1'b1,x2[(NUM_BIT+2) : 1]};
191                     else if(state == VX2_STATE && x2[0] == 0 && v[0] == 0)
192                         x2 <= divx12;
193                     else if(state == VX2_STATE && x2[0] == 1 && v[0] == 0)
194                         x2 <= divxp12;
195                     else if(state == COMP_STATE && u < v)
196                         x2 <= delx12;
197                     else begin
198                         x2 <= x2;
199                     end
200                 end
201             end
202         ////////////////////////END INVERSE/////////////////////////////////////////
203 endmodule
```

ภาคผนวก ข

การออกแบบวงจรรวม (IC Fabrication)

วงจรรวม (IC) นั้นมีขั้นตอนการพัฒนาอยู่ 2 ส่วน ได้แก่ส่วนของการออกแบบ และส่วนของการผลิต ตามที่กล่าวข้างต้น ในด้านการออกแบบนั้นดำเนินการโดยงานวิจัยออกแบบวงจรรวม และในส่วนของการผลิตนั้นดำเนินการโดยศูนย์ไมโครอิเล็กทรอนิกส์ (TMEC) และโรงงานผลิตในต่างประเทศ วิธีการออกแบบวงจรรวมจะมีทั้งแบบดิจตอลล้วนๆ (Digital IC design) หรือแบบอะนาล็อกล้วนๆ (Analog IC design) หรือรวมทั้งสองแบบในเวลาเดียวกัน (Mixed signal IC design) ในกรณีที่ระบบมีความซับซ้อนสูงการออกแบบอาจใช้เทคนิค system-on-chip design ทั้งนี้ในขั้นตอนการออกแบบไอซีโดยทั่วไปสามารถแสดงได้ดังรูป



1. Design Entry

ผู้ออกแบบเริ่มต้นด้วยการกำหนดรายละเอียดหน้าที่การทำงานของวงจรรวมที่ตนต้องการ แล้วป้อนรายละเอียดนี้เข้าสู่คอมพิวเตอร์ ซึ่งสามารถทำได้โดยการวาดแผนภาพเค้าร่าง (schematic) ของวงจรโดยตรง หรือโดยการสร้างโปรแกรมในภาษาพรรณนาฮาร์ดแวร์

(Hardware Description Language: HDL) เช่นภาษา VHDL และภาษา Verilog แล้วให้คอมพิวเตอร์ทำการสังเคราะห์ (synthesis) แผนภาพเค้าร่างให้ การออกแบบในขั้นนี้เราสนใจเพียงพฤติกรรมของระบบที่เราออกแบบเท่านั้น

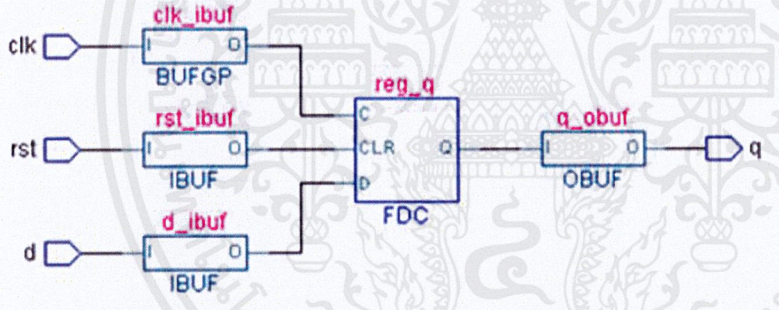
```
library ieee;
    use ieee.std_logic_1164.all;
    entity ff_reset is
      port (
        d     : in  std_logic;  -- data input
        clk   : in  std_logic;  -- clock
        rst   : in  std_logic;  -- reset, active hi
        q     : out std_logic);  -- output
    end entity ff_reset;

    architecture rtl of ff_reset is

    begin  -- architecture rtl

      FF_Proc : process (clk, rst) is
      begin  -- process FF_Proc
        if (rst = '1') then
          q <= '0';
        elsif (clk'event and clk = '1')  then
            q <= d;
         end if;
      end process FF_Proc;
    end architecture rtl;
```

รูปตัวอย่างการออกแบบโดยใช้ภาษา VHDL แล้วสังเคราะห์เป็นลอจิกเกต



2. Simulation

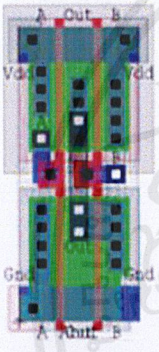
ผู้ออกแบบนำ schematic มาทำการจำลองการทำงาน (simulation) เพื่อตรวจสอบความถูกต้องของวงจรที่ออกแบบตามข้อมูลในเวกเตอร์ทดสอบ (test vector) ที่ผู้ออกแบบกำหนดไว้ โดยพิจารณาจากไดอะแกรมทางเวลา (timing diagram) และการจำลองความผิดพลาดที่เกิดขึ้น ผลจาก simulation จะถูกใช้ในการปรับปรุงแก้ไขวงจรให้ถูกต้อง ก่อนการออกแบบผังวงจรในขั้นตอนสุดท้าย

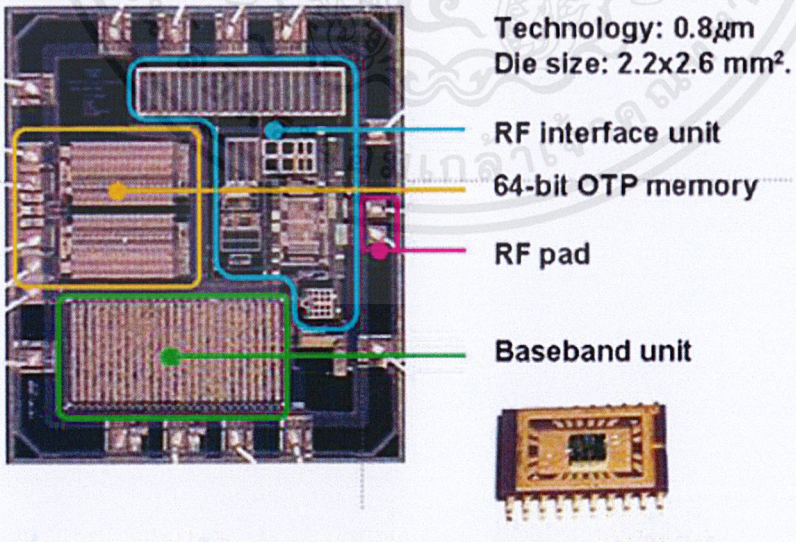
3. Physical Layout

ผู้ออกแบบนำแผนภาพเค้าร่าง (schematic) มาแปลงให้เป็นผังภูมิวงจร (layout) ระดับกายภาพ ซึ่งจะใช้เป็นแบบที่จะถูกถ่ายทอดลงบนแผ่นเวเฟอร์ (แว่นผลึกของสารกึ่งตัวนำซิลิกอน) ที่เตรียมเข้ากระบวนการเจือสารให้เป็นแผ่นวงจรรวมที่จะถูกตัดแบ่งเป็นชิปหลายตัว

ต่อไป การออกแบบผังวงจรมี 2 วิธีหลักๆ คือ

1. ระบบ Pre-Treatment เป็นระบบการทำความสะอาดน้ำเบื้องต้น เพื่อปรับน้ำดิบที่มีความกระด้างปะปนอยู่ให้เป็นน้ำอ่อน (Soft Water) โดยการกำจัดพวกของแข็งแขวนลอย ความขุ่น ตะกอน สารอินทรีย์ละลายและคลอรีน รวมทั้งอิออนต่างๆ
2. ระบบ Ultra Deionized (UDI) Water เป็นระบบการทำความสะอาดน้ำอ่อน (Soft Water) ให้เป็นน้ำที่มีความบริสุทธิ์สูง โดยทำการกำจัดอิออนต่างๆ ที่เหลือจากการทำความสะอาดน้ำเบื้องต้น อีกทั้งยังฆ่าจุลินทรีย์ที่ปะปนมากับน้ำ เพื่อให้ได้น้ำความบริสุทธิ์สูงตรงตามมาตรฐานในการผลิตวงจรรวม ซึ่งโดยปกติ ศูนย์ฯ สามารถผลิตน้ำสะอาดหรือ Ultra Deionized (UDI) Water ได้ถึง 3.5 ลูกบาศก์เมตรต่อชั่วโมง และมีค่าความต้านทานไม่ต่ำกว่า 18 เมกะโอห์ม-เซนติเมตร





Thailand IC Design Incubator (TIDI)

หลังการออกแบบผังวงจรเสร็จสิ้น ผู้ออกแบบจะใช้คอมพิวเตอร์ช่วยตรวจสอบว่าผังภูมิวงจร (layout) นั้นถูกต้องตามกฎการออกแบบ กฎทางไฟฟ้า และมีความผิดเพี้ยนจากแผนภาพเค้าร่าง (schematic) หรือไม่ หากมีความผิดพลาดหรือผิดเพี้ยน จุดผิดเหล่านั้นต้องได้รับการแก้ไข ก่อนส่งแฟ้มข้อมูลผังวงจรรวมไปให้โรงงานทำหน้ากาก (mask) สำหรับใช้ผลิตเป็นไมโครชิปต่อไป

เทคโนโลยีหลักที่ใช้ในการออกแบบและผลิตวงจรรวมนั้นได้แก่ เทคโนโลยี CMOS ซึ่งจะมีขนาดเล็กลงเรื่อยๆ ในส่วนของ TMEC นั้นแนวโน้มจะเน้นไปทาง low power/low voltage ซึ่งจะใช้งานทางด้านเซนเซอร์

ในส่วนอื่นของเทคโนโลยีที่ใช้พัฒนาวงจรรวมนั้นนอกจากการผลิตโดยใช้เทคโนโลยี CMOS แล้วยังสามารถพัฒนาวงจรโดยใช้เทคโนโลยีเอฟพีจีเอ (Field Programmable Gate Array) โดยการทดสอบต้นแบบวงจรบนบอร์ดพัฒนาก่อนที่จะออกแบบในระดับผังภูมิต่อไป เช่นบอร์ด Xilinx Spartan-3 รองรับวงจรได้ถึง 1.6 ล้านเกต และบอร์ดพัฒนาเอฟพีจีเอ Virtex-5

1. เครื่องมือและอุปกรณ์

   เครื่องมือหลักที่จำเป็นในการดำเนินการวิจัยและพัฒนาได้แก่

   1. คอมพิวเตอร์ความเร็วสูงสำหรับการออกแบบและจำลองวงจร ซึ่งทำงานอยู่บนระบบปฏิบัติการ Linux และ Solaris
   2. ซอฟต์แวร์สำหรับออกแบบวงจรรวม (Electronic Design Automation: EDA) ประกอบด้วย Cadence IC Design Software, Xilinx FPGA Design, Tanner Tools
   3. เครื่องมือวัดและทดสอบ
   4. ดิทเทคโนโลยีสำหรับการออกแบบ (Design kit) เช่นเทคโนโลยี CMOS 0.8/0.5/0.35/0.25/0.18um

ประวัติผู้เขียน

ชื่อ ชินภัทร์ ข้อหน่อ

ที่อยู่ เลขที่ 333 หมู่ 1 ตำบล พรหมพิราม อำเภอ พรหมพิราม จังหวัด พิษณุโลก 65150

เบอร์โทรศัพท์ 082 771 4223

อีเมลล์ chinnapatkh@gmail.com

ประวัติการศึกษา กำลังศึกษาอยู่ ชั้นปีที่ 4 คณะวิศวกรรมศาสตร์ ภาควิชาอิเล็กทรอนิกส์
สาขาวิชา อิเล็กทรอนิกส์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ประสบการณ์ ฝึกงานและปฏิบัติงานสหกิจ แผนกดิจิตอลไอซีดีไซน์
ระยะเวลา 6 เดือน ณ บริษัท ซิลิคอน คราฟท์ จำกัด มหาชน