รายงานสหกิจศึกษาฉบับสมบูรณ์

การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163

Hardware Implementation Elliptic Curve Operation Curve B-163

นายพสธร ไทรทอง

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2562

ชื่อโครงงานสหกิจศึกษา การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163
ชื่อ-สกุล นักศึกษา นายพสธร ไทรทอง
คณะ วิศวกรรมศาสตร์ **ภาควิชา** อิเล็กทรอนิกส์
ชื่อ-สกุล อาจารย์นิเทศ ดร.สุเมฆ วิศยทักษิณ
ชื่อ-สกุล ผู้นิเทศงาน นายธนพล หงษ์ทรงเกียรติ
ชื่อสถานประกอบการ บริษัท ซิลิคอนคราฟท์ เทคโนโลยี จำกัด (มหาชน)

บทคัดย่อ

งานวิจัยฉบับนี้อธิบายหลักการออกแบบและสร้างวงจรการทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ซึ่งเป็นวงจรการคูณจุดของเส้นโค้งรูปไข่รูปแบบ B-163 โดยเส้นโค้งรูปไข่รูปแบบ B-163 อยู่ในสนามจำกัดแบบสนามฐานสองจึงต้องอาศัยการทำงานของสนามจำกัดแบบฐานสองเป็นพื้นฐานการคำนวณ โดยได้ใช้วงจรการคูณของสนามจำกัดแบบฐานสองเป็นแบบการคูณอนุกรมดิจิตอล และใช้วงจรการคูณจุดของเส้นโค้งรูปไข่รูปแบบ B-163 ใช้กระบวนการคำนวณแบบ X-Coordinate only Montgomery Ladder ผลลัพธ์ที่ได้คือ วงจรที่ได้ทำการออกแบบนั้นทำงานได้อย่างถูกต้องสมบูรณ์เมื่อนำไปทดสอบกับมาตรฐาน อีกทั้งวงจรยังมีระยะเวลาที่ใช้ในการคำนวณไม่มากนักเพียง 25,436 CLK และมีขนาดที่ค่อยข้างเล็ก คือ 26,122 GE หรือ 261,222 μm^2

คำสำคัญ : ออกแบบวงจรรวมดิจิตอล การทำงานของเส้นโค้งรูปไข่ วิทยาการเข้ารหัส เส้นโค้งรูปไข่รูปแบบ B-163

Co-operative Title: Hardware Implementation Elliptic Curve Operation Curve B-163

Student Intern Name: Mr. Potsatorn Saithong

Faculty: Engineering **Department:** Electronics Engineering

Advisor Name: Dr. Sumek Wisayataksin

Mentor Name: Mr. Thanapol Hongsongkiat

Company : Silicon Craft Technology Public Company Limited

ABSTRACT

This research describes the design and implement elliptic curve operation B-163 , which is the point multiplication operation of elliptic curve curve B-163. This of elliptic curve curve B-163 being in a binary finite field, so it depends on the work of a binary finite field as the basis By using the multiplication circuit of the binary finite field is a digital serial. And the point multiplication circuit of the elliptic curve curve B-163 uses the X-Coordinate only Montgomery Ladder algorithm. The result is The designed circuit works perfectly correctly when tested with standards. In addition, the circuit does not have much time to calculate, only 25,436 CLK and the smallest side is 26,122 GE or 261,222 μm^2

Keywords : Digital IC Design, Elliptic Curve Operation, Cryptography, Elliptic Curve B-163

กิตติกรรมประกาศ

การจัดทำงานวิจัยการสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 สำเร็จได้ ผู้จัดทำต้องขอขอบคุณทางบริษัท ซิลิคอนคราฟท์ เทคโนโลยี จำกัด (มหาชน) ที่ให้สถานที่ในการจัดทำ อุปกรณ์ในการทดลองและทำงานวิจัย และให้โอกาสไปศึกษาหาความรู้จากการทำงานจริงจากสถานประกอบการ

ขอขอบคุณพี่ ๆ ที่บริษัท ซิลิคอนคราฟท์ เทคโนโลยี จำกัด (มหาชน) ทุกคนที่คอยให้คำปรึกษา คำแนะนำ ความรู้ และประสบการณ์ต่าง ๆ มากมาย

พสธร ไทรทอง

ผู้จัดทำ

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

ภาพที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

ในปัจจุบันเทคโนโลยีมีความก้าวหน้า ก่อเกิดหลายสิ่งที่เข้ามาอย่างเปลี่ยนวิถีชีวิตของเราให้สะดวกสบายยิ่งขึ้น แต่ทุกอย่างย่อมต้องมีทั้งข้อดีและข้อเสีย ในอินเตอร์เน็ตมีข้อมูลเท็จและการปลอมแปลงข้อมูลมากมาย อีกทั้งข้อมูลความลับที่เราเก็บไว้ก็อาจถูกโจรกรรมไปได้ ความไม่ปลอดภัยของข้อมูลจึงเกิดขึ้นจึงทำให้เราไม่สามารถนำข้อมูลที่สำคัญของเราเข้าไปอยู่ในอินเตอร์เน็ตหรือส่งเป็นข้อมูลดิจิตอล ดังนั้นการที่ข้อมูลมีความปลอดภัยที่สูง ปลอมแปลงได้ยาก ข้อมูลที่สำคัญมีการเก็บรักษาที่ปลอดภัย และสามารถตรวจสอบได้ว่าถูกแก้ไขปลอมแปลงมาหรือไม่ จะทำให้เราสามารถทำธุรกรรมต่าง ๆ โดยมีความสบายใจและปลอดภัยมากกว่า

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อศึกษาหลักการออกแบบวงจรรวมอิเล็กทรอนิกส์ในทางดิจิตอล

1.2.2 เพื่อศึกษาหลักการเข้ารหัสข้อมูลโดยวิธีเส้นโค้งรูปไข่

1.2.3 เพื่อศึกษาวิธีการทำงานของเส้นโค้งรูปไข่ในขอบเขตจำกัด

1.2.4 เพื่อทำการออกแบบวงจรรวมดิจิตอลของการทำงานของเส้นโค้งรูปไข่

1.3 ขอบเขตของการวิจัย

1.3.1 ออกแบบวงจรรวมอิเล็กทรอนิกส์ในทางดิจิตอล

1.3.2 การเข้ารหัสข้อมูลโดยวิธีเส้นโค้งรูปไข่

1.3.3 การทำงานของเส้นโค้งรูปไข่ในขอบเขตจำกัด

1.4 วิธีดำเนินการวิจัย

ตารางที่ 1.1 แสดงลำดับและวิธีการดำเนินงาน

รายละเอียด	สิงหาคม				กันยายน				ตุลาคม				พฤศจิกายน			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
ศึกษาเกี่ยวกับทฤษฎีที่เกี่ยวข้อง	←	—	—	→												
ออกแบบวงจร					←	—	—	→								
เขียนโค้ดตามที่ออกแบบ								←	—	—	→					
ตรวจสอบการทำงานของวงจร											←	—	→			
แก้ไขและพัฒนาวงจร													←	—	→	
ทำรูปเล่มงานวิจัย														←	—	→

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 ได้รับความรู้เกี่ยวกับหลักการออกแบบวงจรรวมอิเล็กทรอนิกส์ในทางดิจิตอล

1.5.2 ได้รับความรู้เกี่ยวกับหลักการเข้ารหัสข้อมูลโดยวิธีเส้นโค้งรูปไข่

1.5.3 ได้รับความรู้เกี่ยวกับวิธีการทำงานของเส้นโค้งรูปไข่ในขอบเขตจำกัด

1.5.4 ได้ทำการออกแบบวงจรรวมดิจิตอลของการทำงานของเส้นโค้งรูปไข่

บทที่ 2
แนวคิด ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ครั้งนี้ ผู้ดำเนินงานได้ทำการศึกษาและรวบรวมทฤษฎีและหลักการต่าง ๆ จากเอกสารและงานวิจัย โดยมีทฤษฎีและงานวิจัยที่เกี่ยวข้อง ดังนี้

2.1 ทฤษฎีการออกแบบวงจรทางดิจิตอล

รูปแบบการออกแบบวงจรรวมทางดิจิตอลแบ่งออกได้ดังนี้

2.1.1 การออกแบบด้วยสมการบูลีน (Designing with Boolean Equations)

วงจรลอจิกสร้างจากเกตและฟลิบฟลอบจึงทำการการออกแบบด้วยสมการบูลีนได้ เนื่องจากเกตและฟลิบฟลอบสามารถกำหนดได้จากสมการบูลีน วิธีการมีหลายเทคนิคที่ถูกออกแบบมาสำหรับการทำให้ออกแบบใช้ทรัพยากรคือเกตหรือฟลิบฟลอบให้น้อยที่สุดประกอบด้วยการลดรูปของสมการบูลีนโดยใช้ Karnough map (K-Map) เป็นต้น และเนื่องด้วยการออกแบบด้วยสมการบูลีนต้องการสมการสำหรับการกำหนดการใช้งานของแต่ละเกตดังนั้นจึงทำให้ไม่สามารถใช้งานได้จริงสำหรับระบบที่มีความซับซ้อนและมีขนาดใหญ่ที่ประกอบด้วยจำนวนเกตเป็นแสนๆตัวในปัจจุบันได้

2.1.2 การออกแบบด้วย Schematic (Schematic-based Design)

การออกแบบด้วย Schematic ช่วยในการออกแบบด้วยสมการบูลีน เนื่องจากไม่ได้มีแค่เกตหรือฟลิบฟลอบที่เป็นอุปกรณ์พื้นฐานเท่านั้นแต่ยังมีวงจรหรือตัวไอซีที่อยู่ในไลบรารี (Library) ด้วย นอกจากนั้นวิธีนี้ยังสามารถออกแบบระบบเป็นลักษณะของลำดับชั้น (Hierarchy) ได้อีกด้วย ซึ่งทำให้สามารถออกแบบระบบที่ซับซ้อนได้มากขึ้นและสามารถประหยัดเวลาในการออกแบบเนื่องจากสามารถมองเห็นเป็นรูปร่างของระบบที่ชัดเจนเพราะมีลักษณะการออกแบบที่มองเห็นเป็นภาพ (Graphical representation) แต่สำหรับระบบที่มีความซับซ้อนและมีขนาดใหญ่ที่ประกอบด้วยจำนวนเกตเป็นแสนๆตัวในปัจจุบันจะต้องเสียเวลาในการเชื่อมต่อแต่ละขาของอุปกรณ์รวมทั้งยากในการตรวจสอบความถูกต้องและการแก้ไข เพราะในวงจรมีการเชื่อมต่อกันมากมาย

2.1.3 การออกแบบโดยใช้ภาษาระดับสูง

การออกแบบโดยใช้ภาษาระดับสูงเป็นการออกแบบด้วยวิธีเขียนโปรแกรมด้วยภาษา HDL โดยภาษา HDL (HDL: Hardware Description Language) เป็นภาษาโปรแกรมที่ใช้ออกแบบวงจรลอจิกแทนการออกแบบด้วยวิธีการเขียนเป็น Schematic Diagram การออกแบบวงจรลอจิกด้วยวิธีเขียนเป็นภาษา HDL มีข้อดีคือสามารถออกแบบวงจรที่มีความซับซ้อนมาก ๆ ได้ไม่ต้องใช้คณิตศาสตร์บูลีนในการ

ลดรูปวงจรไม่ต้องเสียเวลาลากเส้นวงจร ภาษา HDL มีหลายภาษาแต่ละภาษาก็มีข้อดีละข้อเสียที่แตกต่างกันไป โดยที่นิยมใช้กันอยู่ได้แก่ ภาษา Abel HDL, ภาษา VHDL และ ภาษา Verilog

2.2 วิธีการออกแบบวงจรทางดิจิตอล

ในการออกแบบวงจรทางดิจิตอล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ ที่ใช้งานได้จะต้องผ่านขั้นตอนต่าง ๆ และผู้ออกแบบจะต้องตรวจสอบผลลัพธ์ในแต่ละขั้น ก่อนเข้าสู่กระบวนการออกแบบในขั้นต่อไป โดยขั้นตอนปกติที่ใช้ในการออกแบบวงจรทางดิจิตอลทั่วไป

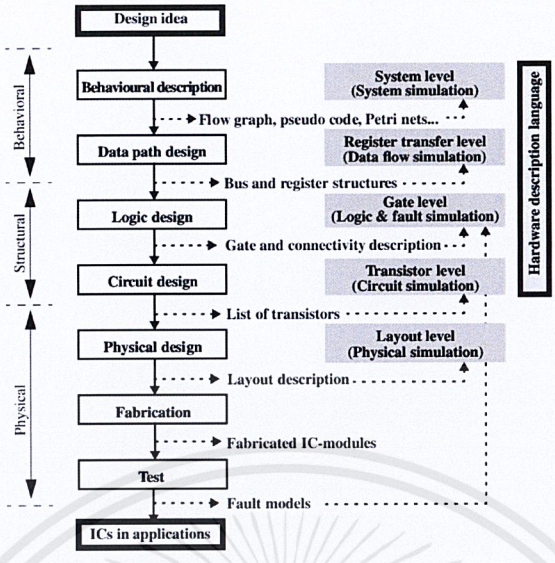
2.2.1 กำหนดแนวความคิดในการออกแบบแล้วทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ซึ่งภาย ในขั้นตอนนี้ผู้ออกแบบจำเป็นต้องสร้างรูปแบบในเชิงพฤติกรรมขึ้นมาตรวจสอบซึ่งอาจจะเป็นผังงานแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo code)

2.2.2 การออกแบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์และวงจรลอจิก ที่จำเป็นทั้งหมดเพื่อนำมาประกอบเป็นระบบที่สมบูรณ์ โดยแต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) ส่วนกระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่าง รีจิสเตอร์และวงจรลอจิกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้

2.2.3 การออกแบบวงจรลอจิก ซึ่งจะเกี่ยวข้องกับการนำเกทดิจิตอลพื้นฐานและฟลิบฟลอบ (flip-flop) มาประกอบเป็นอุปกรณ์ย่อยต่าง ๆ ผลลัพธ์ที่จะเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิบฟลอบ

2.2.4 เปลี่ยนเครือข่ายการโยงใยให้เป็นลำดับของทรานซิสเตอร์ (Transistor List) และ Layout ซึ่งจะเกี่ยวข้องโดยตรงกับการจัดวางทรานซิสเตอร์หรือไลบรารีเซลล์เพื่อแทนเกทและฟลิบฟลอบต่าง ๆ

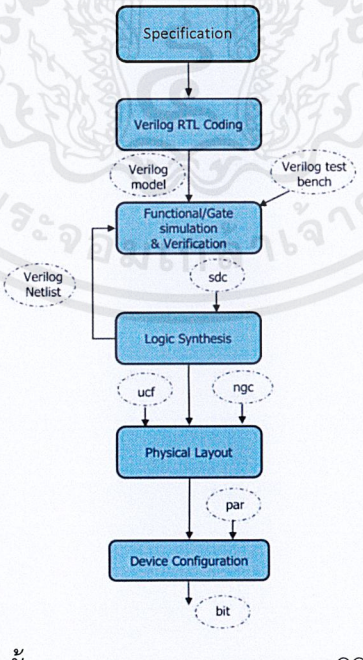
2.2.5 ส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด



ภาพที่ 2.1 แผนผังขั้นตอนการออกแบบวงจรทางดิจิตอล

ที่มา : https://www.researchgate.net/figure/Design-flow-and-levels-of-abstractions_fig1_2675282

2.3 การเขียนภาษา HDL



ภาพที่ 2.2 แผนผังขั้นตอนการออกแบบวงจรทางดิจิตอลด้วยภาษา HDL

ที่มา : https://www.slideserve.com/minty/fpga-design-flow

2.3.1 การกำหนดลักษณะเฉพาะในการออกแบบ (Design Specifications) หลังจากที่ได้มีการออกแบบโครงสร้างไว้เป็นที่เรียบร้อยแล้ว เช่น การแบ่งวงจรทั้งหมดออกเป็นบล็อกย่อย ๆ แล้วทำการเขียนภาษาอธิบายพฤติกรรมของบล็อกเหล่านั้น (Behavioral Description) ด้วยภาษา HDL แล้วจึงนำมารวมเข้าด้วยกัน โดยการเรียกใช้ส่วนประกอบ และทำการเชื่อมต่อสัญญาณระหว่างบล็อกย่อย ๆ เข้าด้วยกัน จะเห็นได้ว่าการอธิบายวงจรในระดับนี้จึงเป็นการอธิบายเชิงโครงสร้าง (Structural Description) การเขียนโค้ดภาษา HDL จะมีลักษณะคล้ายกับภาษาระดับสูงอื่น ๆ

2.3.2 การจำลองการทำงานของฟังก์ชัน (Functional simulation) ขั้นตอนที่สำคัญในการออกแบบวงจรรวมก็คือ การตรวจสอบว่าแบบจำลอง (Model) ทำหน้าที่ตามที่ได้กำหนดไว้หรือไม่ ซึ่งตรวจสอบได้โดยการจำลองการทำงานเพื่อดูการทำงานของวงจรที่สร้างขึ้น (Behavioral Simulation) จึงต้องเขียนโมดูลพิเศษขึ้นมา เพื่อใช้ในการตรวจสอบพฤติกรรมการทำงานของโมดูลที่จะใช้งาน (Design Under Test) ซึ่งโมดูลที่ใช้ทดสอบนี้ เรียกว่า HDL Test bench โดยจะเป็นการกำหนดรูปแบบของสัญญาณขาเข้า (Input Signal) แล้วไปเปรียบเทียบกับสัญญาณที่ป้อนให้กับโมดูลว่าการทำงานนั้นเป็นไปตามที่คาดหวังไว้หรือไม่

2.3.3 การสังเคราะห์วงจร (Synthesis circuit) เมื่อแน่ใจแล้วว่า วงจรที่สร้างมานั้นสามารถทำงานได้ตามต้องการ ขั้นตอนต่อไปก็คือการนำ เอาโค้ดต้นแบบดังกล่าวไปทำการสังเคราะห์วงจร เพื่อให้ได้เน็ตลิสต์ในระดับเกตออกมา เครื่องมือที่ใช้ในการสังเคราะห์โค้ดภาษา HDL (HDL Synthesis Tool) ซึ่งสามารถสังเคราะห์วงจรสำหรับ FPGA ได้และสามารถเลือกไลบรารีให้เหมาะสมกับชิพที่เลือกใช้งานได้

2.3.4 การจำลองทางเวลา (Timing simulation) การจำลองการทำงานในระดับพฤติกรรมหรือหลังจากการสังเคราะห์วงจรแต่ยังไม่ได้ผ่าน การเชื่อมวงจรเข้ากับโมดูลชุดอื่น ๆ จะเป็นการตรวจสอบความถูกต้องในระดับสัญญาณลอจิกเท่านั้น แต่ยังไม่ได้ให้รายละเอียดที่แน่นอนเกี่ยวกับเรื่องเวลา (Timing) มากนัก เพราะจะต้องผ่านขั้นตอนการเชื่อมเส้นทางก่อน ดังนั้นการจำลองการทำงาน ทางเวลาจึงเป็นการตรวจสอบการทำงานของวงจรอีกครั้ง

2.3.5 การวางและการเชื่อมเส้นทาง (Place and Route) ผลที่ได้จากการสังเคราะห์นั้นจะอยู่ในรูปแบบเน็ตลิสต์ในระดับเกต โดยสามารถนำไปผ่าน ขั้นตอนการแปลงให้เป็นลอจิกและรวมถึงการวางและการเชื่อมเส้นทางภายในอุปกรณ์ FPGA

2.4 ภาษา Verilog (Verilog HDL)

ภาษา Verilog เป็นภาษาที่ใช้ในการอธิบายลักษณะการทำงานของฮาร์ดแวร์ (Hardware description language) ความง่ายในการทำความเข้าใจและใช้งานโครงสร้างไวยากรณ์เหมือนภาษา C และไม่จุกจิกถือว่ามีความยืดหยุ่นในการเขียนมากกว่า VHDL ดังนั้นจึงค่อนข้างเป็นที่นิยมในการใช้งานใน

การออกแบบทั่วไปในอุตสาหกรรมวงจรรวมโดยเฉพาะทางอเมริกาและญี่ปุ่นอย่างไรก็ตามภาษา Verilog ค่อนข้างจะด้อยในด้านความสามารถในการกำหนดการทำงานของระบบในระดับที่สูงขึ้น (System level specification)

2.4.1 โครงสร้างของภาษา Verilog

ภาษา Verilog นั้น สามารถอธิบายพฤติกรรมของวงจรดิจิตอลได้หลายระดับ เช่น สามารถอธิบายวงจรในระดับการวาด (Layout) รีซิสเตอร์ ทรานซิสเตอร์และการเชื่อมต่อ(Wire) บนชิพไอซี (Integrated Circuit chip) ซึ่งเรียกว่า ระดับการสวิตช์ (Switch level) สามารถอธิบายการเชื่อมต่อของลอจิกเกตและฟลิบฟลอบในวงจรดิจิตอลเรียกว่า ระดับ เกต (Gate level) และสามารถอธิบายการส่งข้อมูลระหว่างรีจิสเตอร์เรียกว่า ระดับ RTL (Register Transfer Level) ซึ่งการออกแบบวงจรดิจิตอลด้วยภาษาภาษา Verilog ส่วนใหญ่จะใช้อธิบายพฤติกรรมของวงจรดิจิตอลในระดับ RTL นี้ ภาษา Verilog นั้นมีไวยกรณ์คล้ายภาษาซี เช่น จบประโยคด้วย ; หรือใช้ // เพื่อไม่ให้นำไปใช้ในการคอมไพล์ เป็นต้น โครงสร้างของภาษา Verilog นั้น ประกอบด้วยโมดูล (Module) สามารถรวมเข้าด้วยกันเป็นโมดูลขนาดใหญ่หรือเรียกว่า ท็อปโมดูล (Top module) โดยแต่ละโมดูลย่อยนั้น สามารถทำงานพร้อมกัน (Concurrent)

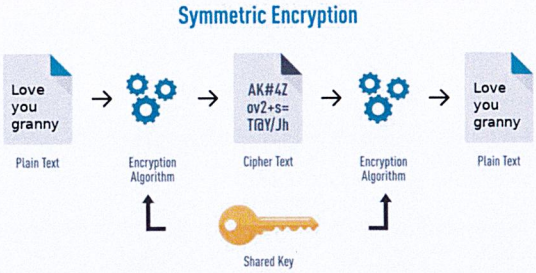
2.5 Cryptography (วิทยาการเข้ารหัส)

วิทยาการเข้ารหัสเป็นกระบวนที่ทำให้การติดต่อสื่อสารระหว่างสองบุคคลมีความปลอดภัย โดยที่บุคคลที่สามไม่สามารถแอบฟังข้อมูลที่แลกเปลี่ยนกันได้

การเข้ารหัสจะทำก่อนการส่งข้อมูล โดยนำข้อมูลกับกุญแจ มาผ่านกระบวนการทางคณิตศาสตร์ จะได้เป็นข้อมูลที่เข้ารหัส และการถอดรหัสจะทำโดยนำเอาข้อมูลที่เข้ารหัสกับกุญแจมาผ่านกระบวนการทางคณิตศาสตร์ จะได้เป็นข้อมูลดั้งเดิม โดยในส่วนของกระบวนการไม่จำเป็นต้องเป็นความลับแต่กุญแจต้องเป็นความลับ ระบบเข้ารหัสสามารถแบ่งตามวิธีการใช้กุญแจได้เป็น 2 วิธีดังนี้

2.5.1 symmetric-key cryptography

symmetric-key cryptography คือการเข้ารหัสข้อมูลด้วยกุญแจเดี่ยว โดยที่ผู้ส่งจะใช้กุญแจนี้ในการเข้ารหัสและผู้รับจะใช้กุญแจที่เหมือนกันในการถอดรหัส ซึ่งวิธีการนี้ผู้รับกับผู้ส่งต้องตกลงกันก่อนว่าจะใช้รูปแบบกระบวนการทางคณิตศาสตร์ใดในการเข้ารหัสข้อมูลเพื่อเก็บเป็นความลับ

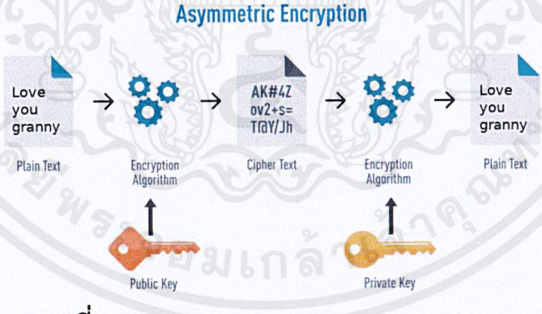


ภาพที่ 2.3 symmetric-key cryptography

ที่มา : https://medium.com/@kapilvermarbl/symmetric-asymmetric-and-hybrid-encryption-25d57c1c327b

2.5.2 asymmetric-key cryptography

asymmetric-key cryptography คือการเข้ารหัสข้อมูลด้วยกุญแจคู่ โดยจะใช้หลักกุญแจคู่ทำการเข้ารหัสและถอดรหัส โดยประกอบด้วย private key และ public key โดยถ้าผู้ส่งต้องการส่งข้อมูลลับให้ผู้รับ ผู้ส่งจะต้องใช้ public key ของผู้รับคนนั้นในการเข้ารหัส แล้วทางผู้รับจะใช้ private key ของตัวเองในการถอดรหัส ซึ่งถ้าใช้ public key ของใครเข้ารหัสก็ต้องใช้ private key ของคนนั้นถอดรหัส ไม่สามารถนำกุญแจคู่อื่นหรือของบุคคลอื่นมาถอดรหัสได้



ภาพที่ 2.4 asymmetric-key cryptography

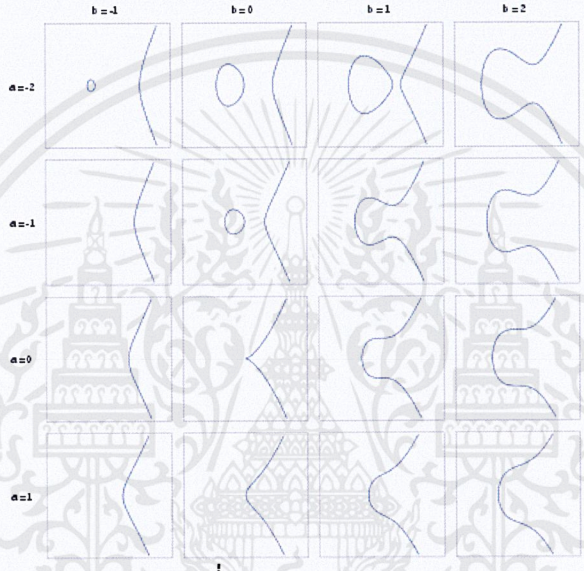
ที่มา : https://medium.com/@kapilvermarbl/symmetric-asymmetric-and-hybrid-encryption-25d57c1c327b

2.6 Elliptic-curve cryptography (ECC)

การเข้ารหัสแบบเส้นโค้งรูปไข่เป็นวิธีการที่มีประสิทธิภาพในการเข้ารหัส มันเป็นวิธีการที่ใช้สำหรับการเข้ารหัสกุญแจสาธารณะโดยใช้โครงสร้างพีชคณิตของเส้นโค้งรูปไข่บนสนามจำกัด เพื่อสร้างความปลอดภัยระหว่างคู่กุญแจ การเข้ารหัสแบบเส้นโค้งรูปไข่ได้รับความนิยมอย่างช้า ๆในช่วงไม่กี่ปีที่

ผ่านมาเนื่องจากความสามารถในการรักษาความปลอดภัยโดยที่มีขนาดของคีย์ที่เล็กกว่ามากเมื่อเทียบกับการเข้ารหัสที่อื่น

โดย Elliptic Curve หรือเส้นโค้งรูปไข่มีคุณสมบัติเป็นโค้งพีชคณิตระนาบ ไม่ตัดตัวมันเอง และสมมาตรกับแกน x มีสมการรูปทั่วไป $y^2 = x^3 + ax + b$ ซึ่งเส้นโค้งรูปไข่มีทั้งแบบบนจำนวนจริง บนจำนวนเชิงซ้อน บนสนามทั่วไป และบนสนามจำกัด โดยที่การเข้ารหัสแบบเส้นโค้งรูปไข่จะใช้เส้นโค้งรูปไข่บนสนามจำกัด



ภาพที่ 2.5 Elliptic Curve

ที่มา : https://en.wikipedia.org/wiki/Elliptic_curve

2.7 Finite field (สนามกำจัด)

Finite field เป็นสนามที่มีองค์ประกอบจำนวนจำกัด จำนวนองค์ประกอบในสนามจำกัดเรียกว่า order สำหรับ Finite field ที่ order นี้คือ p^m เขียนแทนด้วย F_{p^m} หรือเรียกว่าสนาม Galois GF(p^m) โดยที่ p แทนจำนวนเฉพาะเรียกว่าคุณสมบัติของ Finite field และ m เป็นจำนวนเต็มที่มากกว่าหรือเท่ากับ 1 เรียกว่ามิติ Finite field สามารถแยกแยะได้ด้วยวิธีการดังนี้

- ถ้ามิติ (m) มีค่าเท่ากับ 1 จะเรียกว่า Prime Fields (สนามเฉพาะ) หรือ F_p
- ถ้ามิติ (m) มีค่ามากกกว่า 1 จะเรียกว่า Extension Fields (สนามส่วนขยาย)
- ถ้า Extension Fields มีคุณสมบัติ (p) มีค่าเท่ากับ 2 จะเรียกว่า Binary Fields (สนามฐานสอง) หรือ F_{2^m}

Finite field มีการดำเนินการแบบไบนารี 2 รายการคือ “+” และการคูณ “.” จากมุมมองทางคณิตศาสตร์กลุ่มที่แสดงโดย S คือชุด S ขององค์ประกอบยกกำลัง 2 พร้อมกับการดำเนินการ * คือการดำเนินการแบบไบนารีที่ตอบสนองดังต่อไปนี้

- คุณสมบัติปิด (Closure) : x, y $\in$ S ; z = x * y , z $\in$ S

- คุณสมบัติเชื่อมโยง (Associativity) : x, y, z $\in$ S ; (x * y) * z = x * (y * z)

- คุณสมบัติองค์ประกอบเอกลักษณ์ (Identity element) : ที่ e อยู่ใน S ; x * e = e * x = x สำหรับทุก x ที่อยู่ใน S

- คุณสมบัติองค์ประกอบผกผัน (Inverse element) : สำหรับทุก x ที่อยู่ใน S, x’ จะทำให้ x * x’ = e โดยที่ e คือองค์ประกอบเริ่มต้น

2.8 Binary field Arithmetic (การดำเนินการของสนามกำจัดแบบสนามฐานสอง)

เนื่องจากองค์ประกอบของ Finite field binary fields เป็นพหุนามฐานสอง (binary polynomial) ที่มีค่าสัมประสิทธิ์อยู่ในสนาม GF(2) = {0,1} ซึ่งมี 2^m เป็นองค์ประกอบอยู่ในสนามและดีกรีของพหุนามแต่ละอันไม่เกิน m-1 ซึ่งสามารถเขียนอยู่ในรูป binary bit string เช่น $x^4 + x + 1$ (10011) ดังนั้นการดำเนินการของสนามกำจัดแบบสนามฐานสองจะประกอบด้วย

2.8.1 การบวกและการลบ (Addition and Subtraction operation)

จะเป็นการนำไป XOR กันแบบบิตต่อบิต เช่นตัวอย่างนี้ 1101 $\otimes$ 0111=> 1010

2.8.2 การคูณ (Multiplication operation)

จะเป็นการนำไป shift บิตแล้ว XOR กันแบบบิตต่อบิต จากนั้นนำไปหารเอาเศษ เช่นตัวอย่างนี้ 1101 $\cdot$ 0111 = 110100 $\otimes$ 11010 $\otimes$ 1101 = 100011 mod 10011

2.8.3 การยกกำลังสอง (Squaring operation)

จะเป็นการนำไปทำการคูณตัวเองก็ได้ แต่สามารถคูณดีกรีทั้งหมดด้วย 2 แทนได้ ซึ่งถ้ามองในรูป binary bit string แล้ว จะเป็นการนำ 0 ไปแทรกระหว่างบิตนั้นเอง แล้วจึงนำไปหารเอาเศษ เช่นตัวอย่างนี้ 1101^2 = 1010001 mod 10011

2.8.4 การหารเอาเศษ (Reduction Modulus operation)

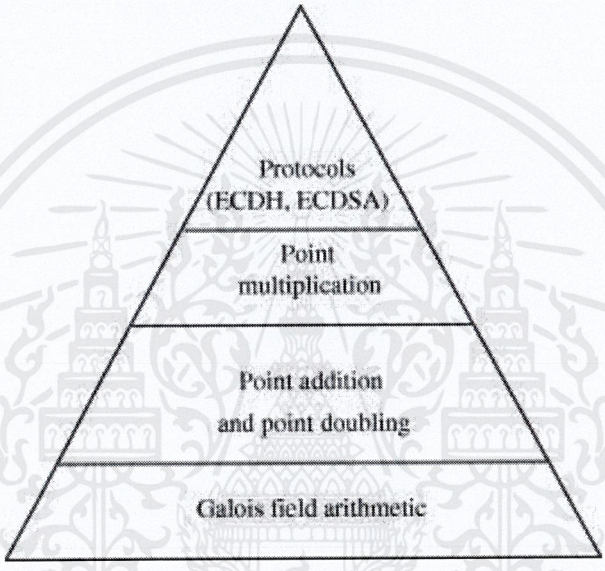
จะเป็นการนำไป XOR กันแบบบิตต่อบิตกับตัวลดการหารเอาเศษ (Reduce modulus) ที่ถูก shift บิตจนเท่ากันแล้ว จนมีดีกรีไม่เกิน m-1 เช่นตัวอย่างนี้ 100011 $\otimes$ 100110 = 0101

2.8.5 การผกผัน (Inversion operation)

จะเป็นการนำไปเข้าสมการ $\mathrm{c} = a^{-1} mod\ f(z) = a^{2^m - 2} mod\ f(z)$

2.9 Elliptic Curve operation (การดำเนินการของเส้นโค้งรูปไข่)

การดำเนินการของ Elliptic Curve เป็นกระบวนการทางคณิตศาสตร์ของ Elliptic Curve การดำเนินการนี้กำหนดจุดผลลัพธ์บนเส้นโค้ง เพื่อที่จะดำเนินการ Elliptic Curve เหล่านี้จำเป็นต้องมีการคำนวณทางคณิตศาสตร์ finite field ซึ่งจะประกอบด้วย

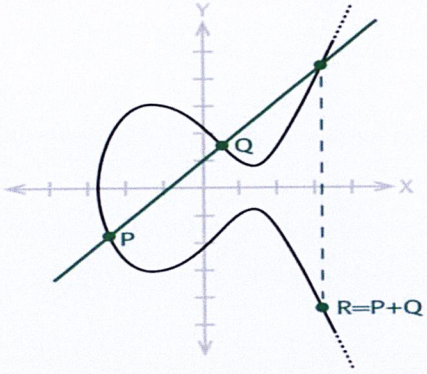


ภาพที่ 2.6 Hierarchy of elliptic-curve implementations

ที่มา : https://www.sciencedirect.com/science/article/pii/S1383762108000684

2.9.1 Point addition operation

จะเป็นการลากเส้นผ่านระหว่างจุด 2 จุดแล้วทำการหาจุดตัดกราฟจากนั้นลากขนานแกน y ให้มาตัดกราฟอีกครั้ง สามารถทำการคำนวณตามสมการนี้ ถ้า $P = (x_1 , y_1)$ และ $Q = (x_2 , y_2)$ ทำให้ $R = P + Q = (x_3 , y_3)$ โดยที่ $x_3 = \lambda^2 - x_1 - x_2$ และ $y_3 = \lambda(x_1 - x_3) - y_1$ เมื่อ $\lambda = (y_2 - y_1) / (x_2 - x_1)$



ภาพที่ 2.7 Point Addition operation

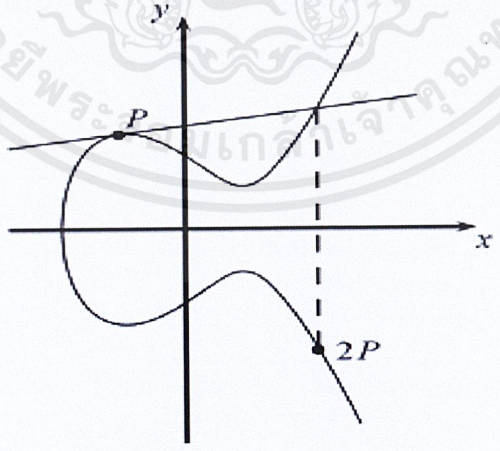
ที่มา : https://sinhvientot.net/elliptic-curve-cryptography/

เนื่องจากการดำเนินการของ Elliptic Curve มีการคำนวณทางคณิตศาสตร์ finite field เป็นรากฐานทำให้มีคุณสมบัติของ finite field ด้วย คือ

- คุณสมบัติองค์ประกอบเอกลักษณ์ : P + e = e + P = P ; P อยู่ใน E(K)
- คุณสมบัติองค์ประกอบผกผัน : P + (-P) = e ; P อยู่ใน E(K)

2.9.2 Point doubling operation

จะเป็นการลากเส้นสัมผัสจุดนั้นแล้วทำการหาจุดตัดกราฟจากนั้นลากขนานแกน y ให้มาตัดกราฟอีกครั้ง สามารถทำการคำนวณตามสมการนี้ ถ้า P = (x_1 , y_1) ทำให้ R = 2P = (x_3 , y_3) โดยที่ $x_3 = \lambda^2 - 2x_1$ และ $y_3 = \lambda(x_1 - x_3) - y_1$ เมื่อ $\lambda = (3x_1^2 + a) / 2y_1$ จากสมการจะสังเกตได้ว่าถ้า y_1 มีค่าเท่ากับ 0 จะทำให้ λ มีค่าเป็นอนันต์ จะทำให้ R มีค่าเป็นอนันต์ด้วย

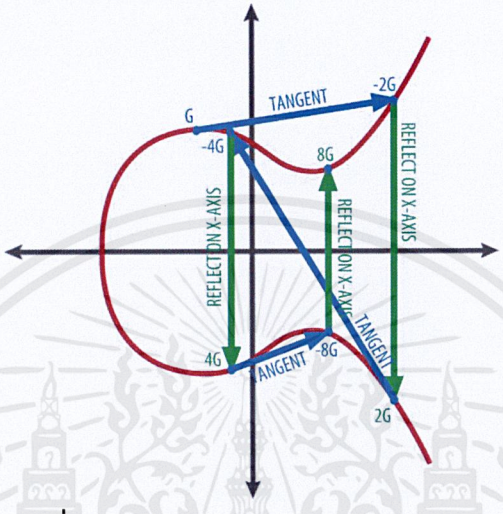


ภาพที่ 2.8 Point doubling operation

ที่มา : https://www.researchgate.net/figure/Point-doubling-22-Elliptic-curve-Diffie-Hellman-ECDH-key-exchange_fig1_308548189

2.9.3 Point multiplication operation

จะเป็นการคูณสเกลาร์ kP = (k-1)P + P สำหรับ k > 0 จากนั้นเพิ่มสำเนา k-1 ของ P ให้กับตัวเองโดยที่ k เป็นจำนวนเต็มบวกและ P เป็นจุดบนเส้นโค้งรูปไข่ ถ้า k = 0; kP จะชี้ไปที่อนันต์ซึ่งเป็นองค์ประกอบเอกลักษณ์ และถ้า k < 0 ; (-a)P = a(-P)



ภาพที่ 2.9 Point multiplication operation

ที่มา : https://crypto.stackexchange.com/questions/48657/how-does-ecc-go-from-decimals-to-integers

2.10 งานวิจัยที่เกี่ยวข้อง

Michael Meuhlberghuber (2011 : 68-75) ได้ทำการวิจัยเรื่อง Comparing ECDSA Hardware Implementations based on Binary and Prime Fields โดยศึกษา Elliptic Curve Digital Signature Algorithm เปรียบเทียบระหว่างการใช้เส้นโค้งรูปไข่บนสนามจำกัดแบบสนามฐานสองกับเส้นโค้งรูปไข่บนสนามจำกัดแบบสนามเฉพาะ พบว่า Elliptic Curve Digital Signature Algorithm ที่คำนวณบนเส้นโค้งรูปไข่บนสนามจำกัดแบบสนามฐานสอง มีขนาดที่เล็กกว่าและใช้เวลาในการคำนวณน้อยกว่า

Julio Lopez and Ricardo Dahab (1999 : 324-327) ได้ทำการวิจัยเรื่อง Fast Multiplication on Elliptic Curves over GF(2m) without Precomputation โดยศึกษากระบวนการคูณที่เร็วในเส้นโค้งรูปไข่บนสนามจำกัดแบบสนามฐานสอง พบว่า ได้วิธีที่ดีและใช่เวลาทำงานน้อยกว่าวิธีอื่น ๆ ที่นำมาเปรียบเทียบ

บทที่ 3
วิธีดำเนินงานวิจัย

ในการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ผู้ดำเนินงานได้มีวิธีการดำเนินงานวิจัย ดังนี้

3.1 ออกแบบวงจร

ในการออกแบบวงจรการทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ซึ่งมีคุณสมบัติ คือ

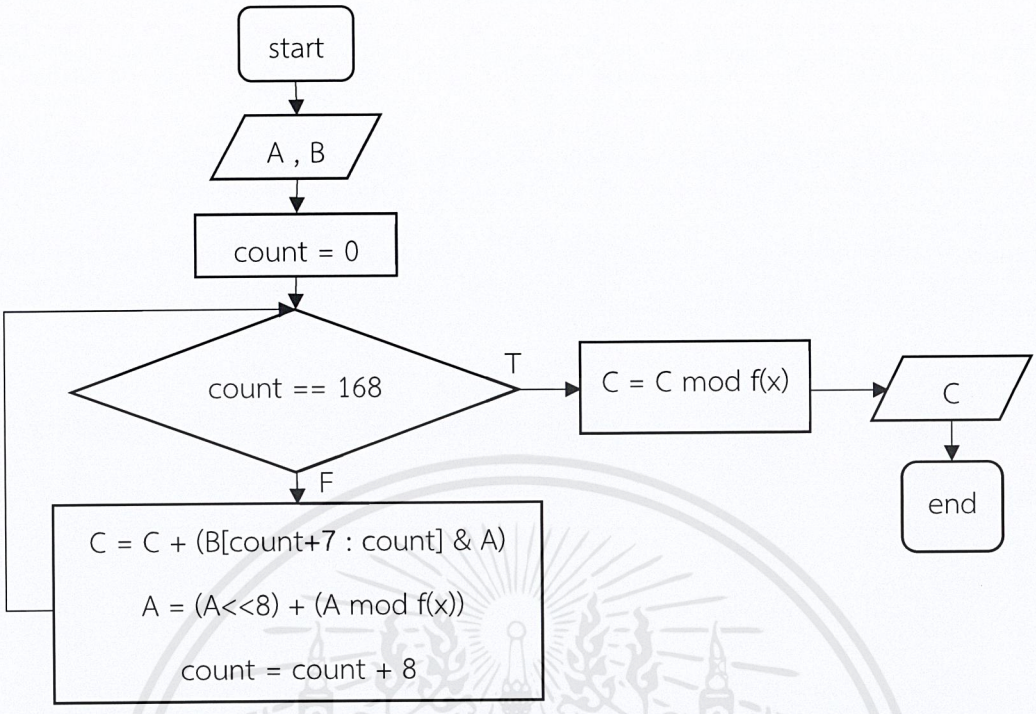
- สมการเส้นโค้งรูปไข่รูปแบบ B-163 $y^2 + xy = x^3 + ax + b\,; a = 1$
- ขนาดกุญแจและผลลัพธ์ 163 บิต
- สมการตัวลดการหารหาเศษ $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
- สัมประสิทธิ์โค้งรูปไข่ a = 1 , b = 0x20A601907B8C953CA1481EB10512F78744A3205FD
- จุดฐาน P x = 0x3F0EbA16286A2D57EA0991168D4994637E8343E36

  y = 0xD51FBC6C71A0094FA2CDD545B11C5C0C797324F1
- กำลังของจุดฐาน P n = 4000000000000000000000292FE77E70C12A4234C33

ได้ทำการแยกขั้นตอนการออกแบบเป็น 2 ส่วนใหญ่ ได้แก่

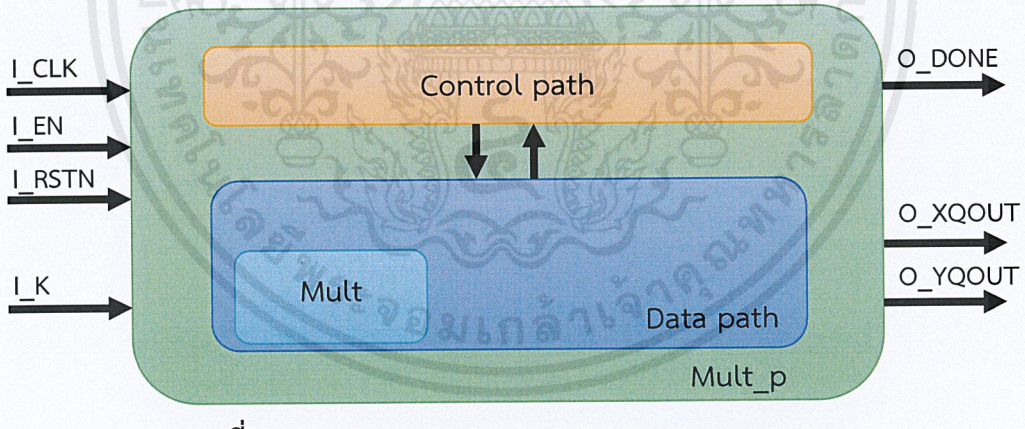
3.1.1 วงจรการคูณของสนามกำจัดแบบสนามฐานสอง

วงจรการคูณของสนามกำจัดแบบสนามฐานสองจะประกอบไปด้วยการ + จะใช้การ XOR และ การ mod จะใช้การตั้งหารยาวจากสมการพหุนาม โดยหารเหลือเศษไว้ในรูปตัวแปรก่อนแล้วนำค่ามาแทนตามค่าที่นำมาคำนวณ วงจรการคูณของสนามกำจัดแบบสนามฐานสองใช้กระบวนการคำนวณแบบอนุกรมดิจิตอล มีขั้นตอนดังนี้



ภาพที่ 3.1 Flow Chart การทำงานของวงจรการคูณของสนามกำจัดแบบสนามฐานสอง

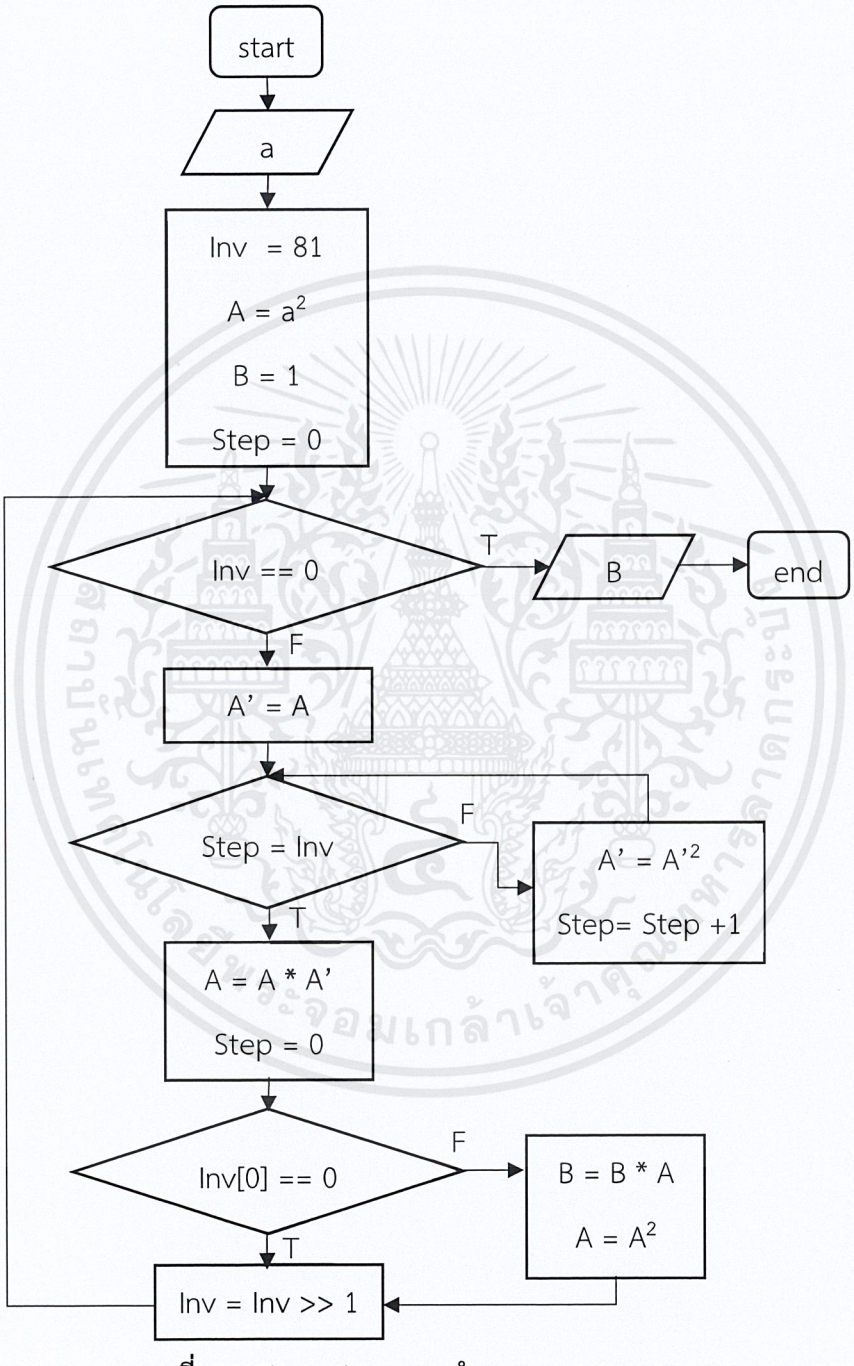
3.1.2 วงจร Point multiplication operation



ภาพที่ 3.2 ภาพรวมวงจร Point multiplication operation

วงจร Point multiplication operation จะแบ่งเป็น 2 ส่วนซึ่งประกอบด้วย data path กับ control path ซึ่งเป็นการแบ่งเพื่อให้สามารถดูสัญญาณได้ชัดเจนและสะดวกเรียบร้อยเท่านั้น ไม่ได้มีผลต่อกระบวนการคำนวณแต่อย่างใด และใน data path ก็จะมีวงจรการคูณของสนามกำจัดแบบสนามฐานสอง ประกอบการคำนวณอยู่ภายใน ซึ่งวงจร Point multiplication operation จะประกอบไปด้วย

การ + จะใช้การ XOR, การ mod จะใช้การตั้งหารยาวจากสมการพหุนาม โดยหารเหลือเศษไว้ในรูปตัวแปรก่อนแล้วนำค่ามาแทนตามค่าที่นำมาคำนวณ, การยกกำลังสองจะทำโดยการแทรก 0 ไประหว่างบิตของค่าแล้วนำมา mod f(x) และการ Inversion ที่ทำได้โดยขั้นตอนดังนี้

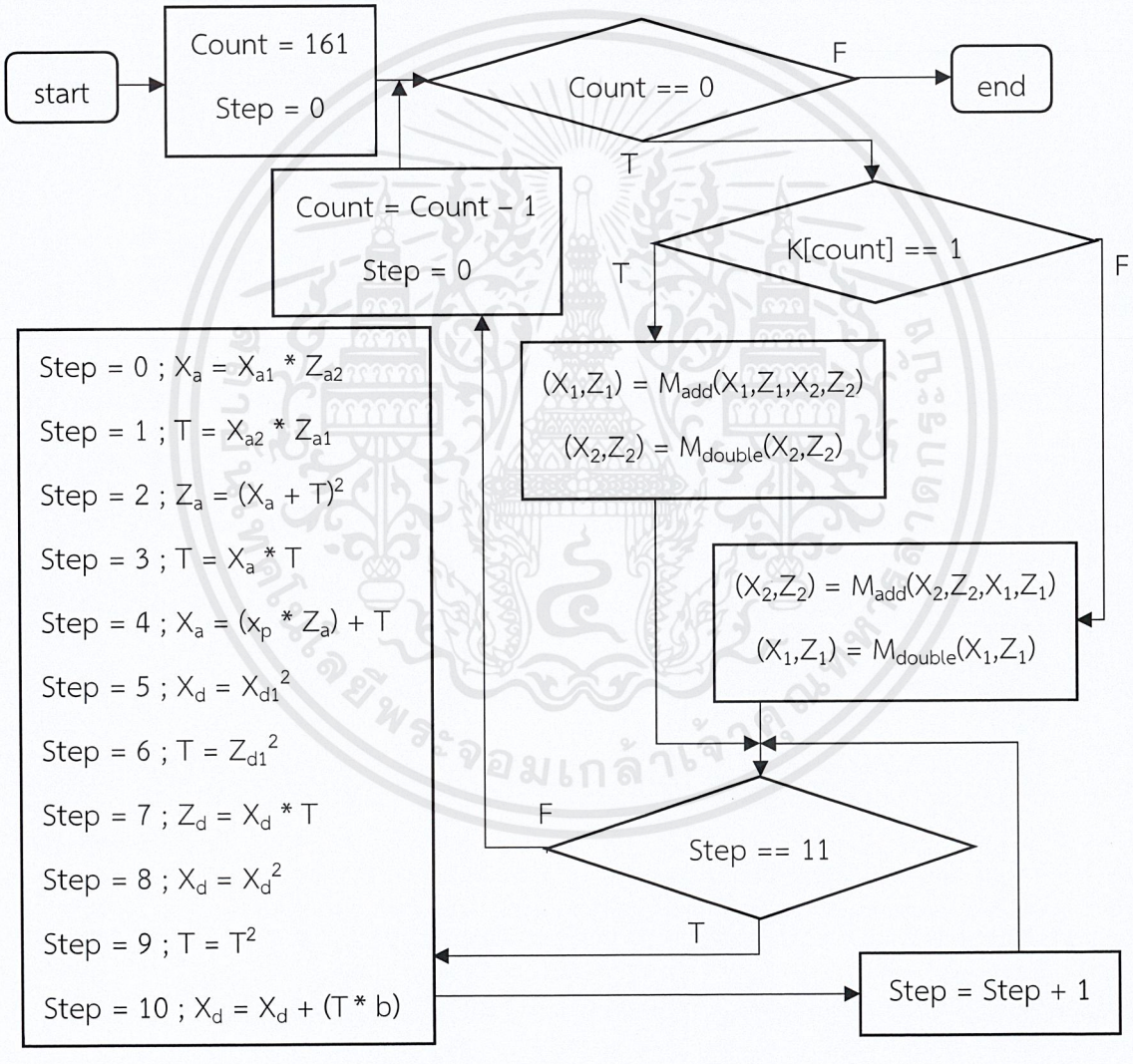


ภาพที่ 3.3 Flow Chart การทำงานของการ Inversion

วงจร Point multiplication operation ใช้กระบวนการคำนวณแบบ X-Coordinate only Montgomery Ladder มีขั้นตอนแบ่งเป็นช่วงการทำงานได้ 3 ช่วงดังนี้

3.1.2.1 ช่วงตั้งค่าเริ่มต้นที่จะนำมาใช้ในการคำนวณต่อไป โดยจะทำการจัดตามสมการนี้ $X_1 = x_P$, $Z_1 = 1$, $X_2 = x_P^4 + b$, $Z_2 = x_P^2$ โดยการ + จะใช้การ XOR และการยกกำลังสองจะทำโดยการแทรก 0 ไประหว่างบิตของค่าแล้วนำมา mod f(x)

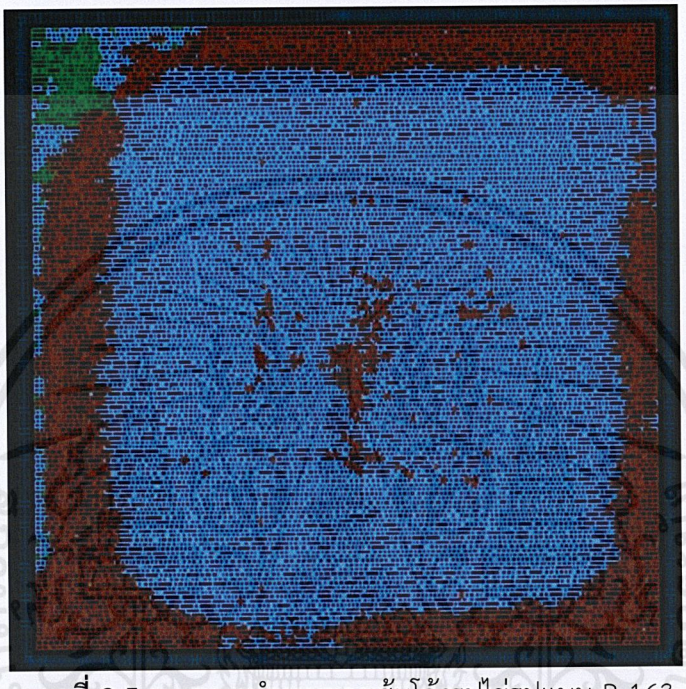
3.1.2.2 ช่วงการคำนวณตามกำลังของจุดฐาน โดยจะทำการนำค่าที่ตั้งค่าไว้ในช่วงแรกมาทำการคำนวณโดยการ Point addition แล้วตามด้วยการ Point doubling โดยการ + จะใช้การ XOR การยกกำลังสองจะทำโดยการแทรก 0 ไประหว่างบิตของค่าแล้วนำมา mod f(x) และดังนี้



ภาพที่ 3.4 Flow Chart การทำงานของวงจร Point multiplication operation ช่วงการคำนวณตามกำลังของจุดฐาน

3.1.2.3 ช่วงการคำนวณค่า x_Q และ y_Q โดยจะทำการจัดตามสมการนี้ $x_Q = X_1 / Z_1$ และ $y_Q = (x_1 + x_P) * \{ (x_1 + x_P) * (x_2 + x_P) + x_P^2 + y_P \} / x_P + y_P$ โดยการ + จะใช้การ XOR การยกกำลังสองจะทำโดยการแทรก 0 ไประหว่างบิตของค่าแล้วนำมา mod f(x) และการ Inversion

3.2 วงจรการทำงานของเส้นโค้งรูปไข่รูปแบบ B-163



ภาพที่ 3.5 วงจรการทำงานของเส้นโค้งรูปไข่รูปแบบ B-163

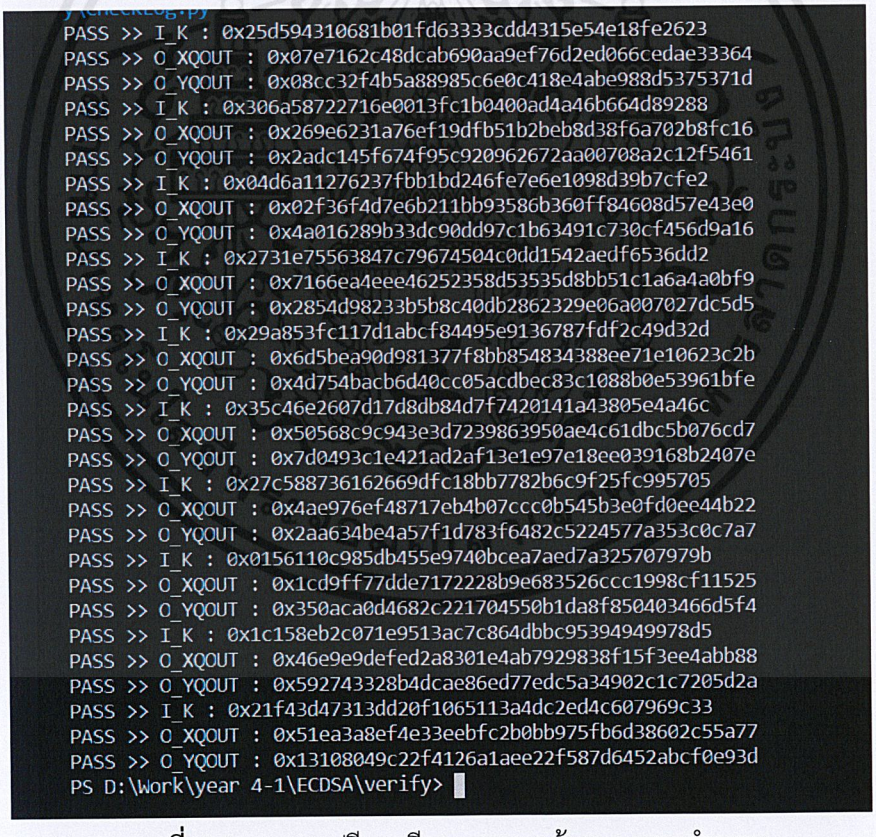
บทที่ 4
ผลการวิจัย

ในการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ผู้ดำเนินงานได้มีวิธีการดำเนินงานวิจัย และได้ผลการวิจัยดังนี้

4.1 ผลการวิจัย

4.1.1 ความถูกต้องของการคำนวณ

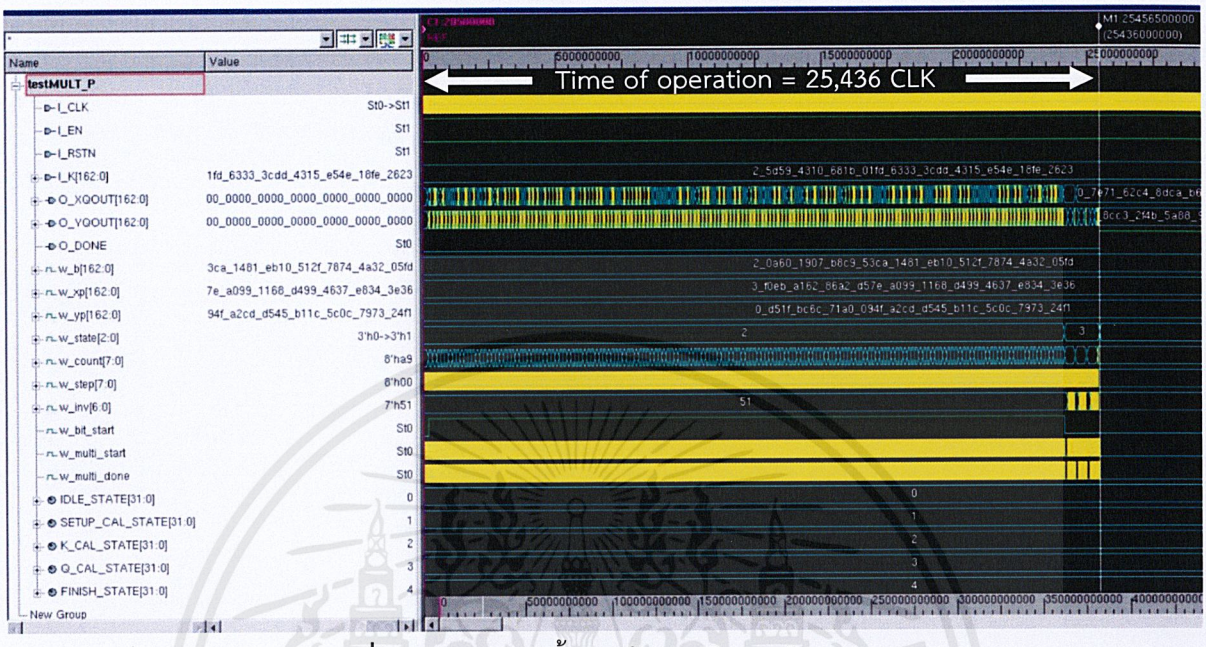
ตรวจสอบความถูกต้องของการคำนวณโดยทำการใส่ค่า I_K .ให้ตรงกับ Key Pair Generation by Testing Candidates ใน Algorithm Test Vectors ของ FIPS 186-4 ซึ่งเป็นมาตรฐาน แล้วนำไฟล์ log มาเปรียบเทียบโดยการเขียน python ได้ผลออกมาตรงกันทั้งหมด



ภาพที่ 4.1 ผลการเปรียบเทียบความถูกต้องของการคำนวณ

4.1.2 ข้อมูลจำเพาะของวงจร

- ระยะเวลาทั้งหมดในการทำการคำนวณ คือ 25,436 CLK



ภาพที่ 4.2 ระยะเวลาทั้งหมดในการทำการคำนวณ

- ขนาดทั้งหมดของวงจร คือ 26,122 GE หรือ 261,222 μm^2

ตารางที่ 4.1 แสดงขนาดของวงจรและโมดูลต่าง ๆ

Module	Area	
Multi_Point	26,122 GE	261,222 μm^2
Control_Path	627 GE	6,265 μm^2
Data_Path	25,496 GE	254,957 μm^2
Multiplication	6,593 GE	65,928 μm^2

- ระยะเวลาทั้งหมดในการทำการคำนวณของวงจรที่ใช้อ้างอิง คือ 41,764 CLK และขนาดทั้งหมดของวงจรของวงจรที่ใช้อ้างอิง คือ 34,582 GE หรือ 324,185 μm^2 ซึ่งวงจรที่ได้ทำการออกแบบมีทั้งขนาดที่เล็กกว่าและใช้เวลาทำงานน้อยกว่า

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

ในการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ผู้ดำเนินงานได้ทำการดำเนินงานวิจัย และนำผลการวิจัยที่ได้มาสรุปได้ดังนี้

5.1 สรุปผลการวิจัย

จากที่ได้ทำการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 โดยได้ทำการออกแบบวงจร และตรวจสอบวงจรที่ได้ทำการออกแบบนั้น พบว่า วงจรที่ได้ทำการออกแบบนั้นทำงานได้อย่างถูกต้องสมบูรณ์เมื่อนำไปทดสอบกับมาตรฐาน อีกทั้งวงจรยังมีระยะเวลาที่ใช้ในการคำนวณไม่มากนักเพียง 25,436 CLK และมีขนาดที่ค่อยข้างเล็ก คือ 26,122 GE หรือ 261,222 μm^2 จึงถือได้ว่าเป็นวงจรที่มีประสิทธิภาพที่ดี

5.2 ข้อเสนอแนะ

ในการทำการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 โดยได้ทำการออกแบบวงจร และตรวจสอบวงจรที่ได้ทำการออกแบบนั้น พบว่า วงจรเป็นวงจรการทำงานที่มีกระบวนการคิดทางคณิตศาสตร์ที่หลากหลายรูปแบบ ควรทำการค้นหาและประเมินความคุ้มค่าของกระบวนการคิดทางคณิตศาสตร์ที่หลากหลายตั้งแต่ในกระบวนการออกแบบ

เอกสารอ้างอิง

[1] somboon. (2549). “การออกแบบลอจิกโมดูลด้วยวิธีเขียนโปรแกรมภาษา HDL.” Available at : http://www.bpcd.net

[2] ธีรยศ เวียงทอง (2548). “เรียนรู้การออกแบบระบบดิจิตอลด้วยภาษา Verilog เบื้องต้น”. Available at : http://dusithost.dusit.ac.th

[3] สุธีร์ กิจเจริญการกุล (2558). “การเข้ารหัสเชิงควอนตัม ตอนที่ 1 [บทความวิชาการ]” Available at : https://www.techtalkthai.com

[4] Hans Knutson (2561). “What is the math behind elliptic curve cryptography?” Available at : https://hackernoon.com

[5] Johan Dams (2555). “An introduction to elliptic curve cryptography” Available at : https://www.embedded.com

[6] Kerman Kohli (2562). “Learning Cryptography, Part 1: Finite Fields” Available at : https://medium.com

[7] Darrel Hankerson (2547). “Guide to Elliptic Curve Cryptography” New York: Springer-Verlag pp 240-260

[8] Julio Lopez and Ricardo Dahab. (1999) Fast Multiplication on Elliptic Curves over GF(2m) without Precomputation. Department of Combinatorics and Optimization University of Waterloo Waterloo, Ontario N2L 3G1, Canada

[8] Julio Lopez and Ricardo Dahab. (1999) Fast Multiplication on Elliptic Curves over GF(2m) without Precomputation. Department of Combinatorics and Optimization University of Waterloo Waterloo, Ontario N2L 3G1, Canada

[9] Michael Meuhlberghuber. (2011) Comparing ECDSA Hardware Implementations based on Binary and Prime Fields Master Thesis Institute for Applied Information Processing and Communications Graz University of Technology Austria

ภาคผนวก

Code Module MULT_P

```
`timescale 1ns / 1ps
module MULT_P(
    input           I_CLK,
    input           I_EN,
    input           I_RSTN,
    input  [162:0] I_K,
    output [162:0] O_XQOUT,
    output [162:0] O_YQOUT,
    output          O_DONE
);
    parameter IDLE_STATE      = 0;
    parameter SETUP_CAL_STATE = 1;
    parameter K_CAL_STATE     = 2;
    parameter Q_CAL_STATE     = 3;
    parameter FINISH_STATE    = 4;

    wire [162:0] w_b;
    wire [162:0] w_xp;
    wire [162:0] w_yp;
    wire [2:0]   w_state;
    wire [7:0]   w_count;
    wire [7:0]   w_step;
    wire [6:0]   w_inv;
    wire         w_bit_start;
    wire         w_multi_start;
    wire         w_multi_done;

    assign w_b = 163'h20A601907B8C953CA1481EB10512F78744A3205FD;
    assign w_xp = 163'h3F0EBA16286A2D57EA0991168D4994637E8343E36;
    assign w_yp = 163'hD51FBC6C71A0094FA2CDD545B11C5C0C797324F1;

    MULT_P_CONTROLPATH MULT_P_CONTROLPATH(
        .I_CLK          (I_CLK),
        .I_EN           (I_EN),
        .I_RSTN         (I_RSTN),
        .I_K            (I_K),
        .I_MULT_DONE    (w_multi_done),
        .O_STATE        (w_state),
        .O_COUNT        (w_count),
        .O_STEP         (w_step),
        .O_INV          (w_inv),
        .O_BIT_START    (w_bit_start),
        .O_MULT_START   (w_multi_start),
        .O_DONE         (O_DONE)
    );

    MULT_P_DATAPATH MULT_P_DATAPATH(
        .I_CLK          (I_CLK),
        .I_RSTN         (I_RSTN),
        .I_STATE        (w_state),
        .I_COUNT        (w_count),
        .I_STEP         (w_step),
        .I_INV          (w_inv),
        .I_BIT_START    (w_bit_start),
        .I_MULT_START   (w_multi_start),
        .I_B            (w_b),
        .I_XP           (w_xp),
        .I_YP           (w_yp),
        .I_K            (I_K),
        .O_MULT_DONE    (w_multi_done),
        .O_XQOUT        (O_XQOUT),
        .O_YQOUT        (O_YQOUT)
    );
endmodule
```

Code Module MULT_P_CONTROLPATH

```
`timescale 1ns / 1ps
module MULT_P_CONTROLPATH(
    input           I_CLK,
    input           I_EN,
    input           I_RSTN,
    input  [162:0] I_K,
    input           I_MULT_DONE,
    output [2:0]    O_STATE,
    output [7:0]    O_COUNT,
    output [7:0]    O_STEP,
    output [6:0]    O_INV,
    output          O_BIT_START,
    output          O_MULT_START,
    output          O_DONE
);
    parameter IDLE_STATE      = 0;
    parameter SETUP_CAL_STATE = 1;
    parameter K_CAL_STATE     = 2;
    parameter Q_CAL_STATE     = 3;
    parameter FINISH_STATE    = 4;

    reg [2:0] r_state;
    reg [7:0] r_count;
    reg [7:0] r_step;
    reg       r_bit_start;
    reg       r_multi_start;
    reg [6:0] r_inv;

    assign O_STATE = r_state;
    assign O_COUNT = r_count;
    assign O_STEP = r_step;
    assign O_INV = r_inv;
    assign O_BIT_START = r_bit_start;
    assign O_MULT_START = r_multi_start;
    assign O_DONE = r_state[2];

    //--------------------r_state--------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_state <= IDLE_STATE;
        end
        else if(~I_EN) begin
```

```
            r_state <= IDLE_STATE;
        end
        else if(r_state == IDLE_STATE) begin
            r_state <= SETUP_CAL_STATE;
        end
        else if((r_state == SETUP_CAL_STATE) && (r_count == 168)) begin
            r_state <= K_CAL_STATE;
        end
        else if((r_state == K_CAL_STATE) && (r_count == 0) && (r_step == 10) && I_MULT_DONE && r_multi_start) begin
            r_state <= Q_CAL_STATE;
        end
        else if((r_state == Q_CAL_STATE) && (r_count == 0) && I_MULT_DONE && r_multi_start) begin
            r_state <= FINISH_STATE;
        end
    end
    //------------------r_count------------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_count <= 169;
        end
        else if(r_state == IDLE_STATE) begin
            r_count <= 169;
        end
        else if((r_state == SETUP_CAL_STATE) || (r_state == K_CAL_STATE) || (r_state == Q_CAL_STATE)) begin
            if(r_count >= 168) begin
                r_count <= r_count - 1;
            end
            else if((r_step == 10) && I_MULT_DONE && r_multi_start && (r_count <= 167) && (r_count >= 6)) begin
                r_count <= r_count - 1;
            end
            else if((r_inv == 0) && (r_count <= 5) && (r_count >= 3)) begin
                r_count <= r_count - 1;
            end
            else if(I_MULT_DONE && r_multi_start && (r_count <= 2) && (r_count != 0)) begin
                r_count <= r_count - 1;
            end
        end
    end
    //------------------r_step------------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_step <= 0;
        end
        else if(r_state == IDLE_STATE) begin
            r_step <= 0;
        end
        else if(r_state == K_CAL_STATE) begin
            if((r_step == 10) && I_MULT_DONE && r_multi_start) begin
                r_step <= 0;
            end
            else if((r_step != 10) && I_MULT_DONE && r_multi_start) begin
                r_step <= r_step + 1;
            end
            else if((r_step == 2) || (r_step == 4) || (r_step == 6) || (r_step == 8) || (r_step == 9)) begin
                r_step <= r_step + 1;
            end
        end
        else if(r_state == Q_CAL_STATE) begin
            if((r_count <= 5) && (r_count >= 3)) begin
                if(r_inv != 0) begin
                    if(r_step <= r_inv) begin
                        r_step <= r_step + 1;
                    end
                    else if(I_MULT_DONE && r_multi_start && (r_step == r_inv + 1)) begin
                        r_step <= r_step + 1;
                    end
                    else if((r_step == r_inv + 2) && ~r_inv[ ]) begin
                        r_step <= 1;
                    end
                    else if(I_MULT_DONE && r_multi_start && (r_step == r_inv + 2) && r_inv[ ]) begin
                        r_step <= 1;
                    end
                end
                else begin
                    r_step <= 0;
                end
            end
        end
    end
    //------------------r_bit_start------------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_bit_start <= 0;
        end
        else if(r_state == IDLE_STATE) begin
            r_bit_start <= 0;
        end
        else if(r_state == SETUP_CAL_STATE) begin
            if(I_K[r_count-6] && (r_count == 168)) begin
                r_bit_start <= 1;
            end
        end
        else if(r_state == K_CAL_STATE) begin
            if((r_count == 0) && (r_step == 10) && I_MULT_DONE && r_multi_start) begin
                r_bit_start <= 0;
            end
            else if(I_K[r_count-6] && (r_step == 10) && I_MULT_DONE && r_multi_start) begin
                r_bit_start <= 1;
            end
        end
    end
    //------------------r_multi_start------------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_multi_start <= 0;
        end
        else if(r_state == IDLE_STATE) begin
            r_multi_start <= 0;
        end
        else if(r_state == K_CAL_STATE) begin
            if(I_MULT_DONE && r_multi_start) begin
                r_multi_start <= 0;
            end
```

```
            else if((r_step != 2) && (r_step != 3) && (r_step != 6) && (r_step != 8) && (r_step != 9)) begin
                r_multi_start <= 1;
            end
        end
        else if(r_state == Q_CAL_STATE) begin
            if((r_count <= 5) && (r_count >= 3)) begin
                if(r_inv != 0) begin
                    if(~I_MULT_DONE && (r_step == 0) && (r_count != 5)) begin
                        r_multi_start <= 1;
                    end
                    else if(~I_MULT_DONE && (r_step == r_inv + 1)) begin
                        r_multi_start <= 1;
                    end
                    else if(~I_MULT_DONE && (r_step == r_inv + 2) && r_inv[0]) begin
                        r_multi_start <= 1;
                    end
                    else if(I_MULT_DONE) begin
                        r_multi_start <= 0;
                    end
                end
                else begin
                    r_multi_start <= 0;
                end
            end
            else begin
                if(I_MULT_DONE && r_multi_start) begin
                    r_multi_start <= 0;
                end
                else begin
                    r_multi_start <= 1;
                end
            end
        end
    end
    //-------------------r_inv-------------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_inv <= 81;
        end
        else if(r_state == IDLE_STATE) begin
            r_inv <= 81;
        end
        else if(r_state == Q_CAL_STATE) begin
            if((r_count <= 5) && (r_count >= 3)) begin
                if(r_inv != 0) begin
                    if(~r_inv[0] && (r_step == r_inv+1)) begin
                        r_inv <= r_inv >> 1;
                    end
                    else if(I_MULT_DONE && r_multi_start && (r_step == r_inv+2)) begin
                        r_inv <= r_inv >> 1;
                    end
                end
                else begin
                    r_inv <= 81;
                end
            end
        end
    end
endmodule
```

Code Module MULT_P_DATAPATH

```
`timescale 1ns / 1ps
module MULT_P_DATAPATH(
    input           I_CLK,
    input           I_RSTN,
    input  [2:0]    I_STATE,
    input  [7:0]    I_COUNT,
    input  [7:0]    I_STEP,
    input  [6:0]    I_INV,
    input           I_BIT_START,
    input           I_MULT_START,
    input  [162:0] I_B,
    input  [162:0] I_XP,
    input  [162:0] I_YP,
    input  [162:0] I_K,
    output          O_MULT_DONE,
    output [162:0] O_XQOUT,
    output [162:0] O_YQOUT
);
    parameter IDLE_STATE      = 0;
    parameter SETUP_CAL_STATE = 1;
    parameter K_CAL_STATE     = 2;
    parameter Q_CAL_STATE     = 3;
    parameter FINISH_STATE    = 4;

    reg [162:0] r_x1;
    reg [162:0] r_x2;
    reg [162:0] r_z1;
    reg [162:0] r_z2;
    reg [162:0] r_xp_sqr;
    reg [162:0] r_xp_fourth;
    reg [162:0] r_ybuff;

    wire [162:0] w_sqr;
    wire [162:0] w_dsqr;
    wire [162:0] w_multi_din1;
    wire [162:0] w_multi_din2;
    wire [162:0] w_multi_dout;

    assign w_dsqr = {w_sqr[81]^w_sqr[161]^w_sqr[159],w_sqr[162]^w_sqr[159],w_sqr[80]^w_sqr[160]^w_sqr[158],w_sqr[161]^w_sqr[158],w_sqr[79]^w_sqr[159]^w_sqr[157],w_sq
                     w_sqr[78]^w_sqr[158]^w_sqr[156],w_sqr[159]^w_sqr[156],w_sqr[77]^w_sqr[157]^w_sqr[155],w_sqr[158]^w_sqr[155],w_sqr[76]^w_sqr[156]^w_sqr[154],w_sq
```

```
                    w_sqr[75]^w_sqr[155]^w_sqr[153],w_sqr[154]^w_sqr[153],w_sqr[74]^w_sqr[154]^w_sqr[152],w_sqr[153]^w_sqr[152],w_sq
                    w_sqr[73]^w_sqr[152]^w_sqr[150],w_sqr[153]^w_sqr[150],w_sqr[72]^w_sqr[151]^w_sqr[149],w_sqr[152]^w_sqr[149],w_sqr[70]^w_sqr[150]^w_sqr[148],w_sq
                    w_sqr[69]^w_sqr[149]^w_sqr[147],w_sqr[150]^w_sqr[147],w_sqr[68]^w_sqr[148]^w_sqr[146],w_sqr[149]^w_sqr[146],w_sqr[67]^w_sqr[147]^w_sqr[145],w_sq
                    w_sqr[66]^w_sqr[146]^w_sqr[144],w_sqr[147]^w_sqr[144],w_sqr[65]^w_sqr[145]^w_sqr[143],w_sqr[146]^w_sqr[143],w_sqr[64]^w_sqr[144]^w_sqr[142],w_sq
                    w_sqr[63]^w_sqr[143]^w_sqr[141],w_sqr[144]^w_sqr[141],w_sqr[62]^w_sqr[142]^w_sqr[140],w_sqr[143]^w_sqr[140],w_sqr[61]^w_sqr[141]^w_sqr[139],w_sq
                    w_sqr[60]^w_sqr[140]^w_sqr[138],w_sqr[141]^w_sqr[138],w_sqr[59]^w_sqr[139]^w_sqr[137],w_sqr[140]^w_sqr[137],w_sqr[58]^w_sqr[138]^w_sqr[136],w_sq
                    w_sqr[57]^w_sqr[137]^w_sqr[135],w_sqr[138]^w_sqr[135],w_sqr[56]^w_sqr[136]^w_sqr[134],w_sqr[137]^w_sqr[134],w_sqr[55]^w_sqr[135]^w_sqr[133],w_sq
                    w_sqr[54]^w_sqr[134]^w_sqr[132],w_sqr[135]^w_sqr[132],w_sqr[53]^w_sqr[133]^w_sqr[131],w_sqr[134]^w_sqr[131],w_sqr[52]^w_sqr[132]^w_sqr[130],w_sq
                    w_sqr[51]^w_sqr[131]^w_sqr[129],w_sqr[132]^w_sqr[129],w_sqr[50]^w_sqr[130]^w_sqr[128],w_sqr[131]^w_sqr[128],w_sqr[49]^w_sqr[129]^w_sqr[127],w_sq
                    w_sqr[48]^w_sqr[128]^w_sqr[126],w_sqr[129]^w_sqr[126],w_sqr[47]^w_sqr[127]^w_sqr[125],w_sqr[128]^w_sqr[125],w_sqr[46]^w_sqr[126]^w_sqr[124],w_sq
                    w_sqr[45]^w_sqr[125]^w_sqr[123],w_sqr[126]^w_sqr[123],w_sqr[44]^w_sqr[124]^w_sqr[122],w_sqr[125]^w_sqr[122],w_sqr[43]^w_sqr[123]^w_sqr[121],w_sq
                    w_sqr[42]^w_sqr[122]^w_sqr[120],w_sqr[123]^w_sqr[120],w_sqr[41]^w_sqr[121]^w_sqr[119],w_sqr[122]^w_sqr[119],w_sqr[40]^w_sqr[120]^w_sqr[118],w_sq
                    w_sqr[39]^w_sqr[119]^w_sqr[117],w_sqr[120]^w_sqr[117],w_sqr[38]^w_sqr[118]^w_sqr[116],w_sqr[119]^w_sqr[116],w_sqr[37]^w_sqr[117]^w_sqr[115],w_sq
                    w_sqr[36]^w_sqr[116]^w_sqr[114],w_sqr[117]^w_sqr[114],w_sqr[35]^w_sqr[115]^w_sqr[113],w_sqr[116]^w_sqr[113],w_sqr[34]^w_sqr[114]^w_sqr[112],w_sq
                    w_sqr[33]^w_sqr[113]^w_sqr[111],w_sqr[114]^w_sqr[111],w_sqr[32]^w_sqr[112]^w_sqr[110],w_sqr[113]^w_sqr[110],w_sqr[31]^w_sqr[111]^w_sqr[109],w_sq
                    w_sqr[30]^w_sqr[110]^w_sqr[108],w_sqr[111]^w_sqr[108],w_sqr[29]^w_sqr[109]^w_sqr[107],w_sqr[110]^w_sqr[107],w_sqr[28]^w_sqr[108]^w_sqr[106],w_sq
                    w_sqr[27]^w_sqr[107]^w_sqr[105],w_sqr[108]^w_sqr[105],w_sqr[26]^w_sqr[106]^w_sqr[104],w_sqr[107]^w_sqr[104],w_sqr[25]^w_sqr[105]^w_sqr[103],w_sq
                    w_sqr[24]^w_sqr[104]^w_sqr[102],w_sqr[105]^w_sqr[102],w_sqr[23]^w_sqr[103]^w_sqr[101],w_sqr[104]^w_sqr[101],w_sqr[22]^w_sqr[102]^w_sqr[100],w_sq
                    w_sqr[21]^w_sqr[101]^w_sqr[ 99],w_sqr[102]^w_sqr[ 99],w_sqr[20]^w_sqr[100]^w_sqr[ 98],w_sqr[101]^w_sqr[ 98],w_sqr[19]^w_sqr[ 99]^w_sqr[ 97],w_sq
                    w_sqr[18]^w_sqr[ 98]^w_sqr[ 96],w_sqr[ 99]^w_sqr[ 96],w_sqr[17]^w_sqr[ 97]^w_sqr[ 95],w_sqr[ 98]^w_sqr[ 95],w_sqr[16]^w_sqr[ 96]^w_sqr[ 94],w_sq
                    w_sqr[15]^w_sqr[ 95]^w_sqr[ 93],w_sqr[ 96]^w_sqr[ 93],w_sqr[14]^w_sqr[ 94]^w_sqr[ 92],w_sqr[ 95]^w_sqr[ 92],w_sqr[13]^w_sqr[ 93]^w_sqr[ 91],w_sq
                    w_sqr[12]^w_sqr[ 92]^w_sqr[ 90],w_sqr[ 93]^w_sqr[ 90],w_sqr[11]^w_sqr[ 91]^w_sqr[ 89],w_sqr[ 92]^w_sqr[ 89],w_sqr[10]^w_sqr[ 90]^w_sqr[ 88],w_sq
                    w_sqr[ 9]^w_sqr[ 89]^w_sqr[ 87],w_sqr[ 90]^w_sqr[ 87],w_sqr[ 8]^w_sqr[ 88]^w_sqr[ 86],w_sqr[ 89]^w_sqr[ 86],w_sqr[ 7]^w_sqr[ 87]^w_sqr[ 85],w_sq
                    w_sqr[6]^w_sqr[86]^w_sqr[84]^w_sqr[162],w_sqr[87]^w_sqr[84]^w_sqr[162]^w_sqr[162],
                    w_sqr[5]^w_sqr[85]^w_sqr[83]^w_sqr[161]^w_sqr[162],w_sqr[86]^w_sqr[83]^w_sqr[161]^w_sqr[161],
                    w_sqr[4]^w_sqr[84]^w_sqr[82]^w_sqr[162]^w_sqr[162]^w_sqr[161]^w_sqr[161],w_sqr[85]^w_sqr[82]^w_sqr[162]^w_sqr[160]^w_sqr[162]^w_sqr[160],
                    w_sqr[3]^w_sqr[83]^w_sqr[161]^w_sqr[160],w_sqr[84]^w_sqr[161],
                    w_sqr[2]^w_sqr[82]^w_sqr[162]^w_sqr[160],w_sqr[83]^w_sqr[162]^w_sqr[160],
                    w_sqr[1]^w_sqr[161],w_sqr[82]^w_sqr[162]^w_sqr[160],w_sqr[0]^w_sqr[160]};

assign w_sqr = (I_COUNT == 168)                                                 ? I_XP          :
               (I_COUNT == 169)                                                 ? r_xp_sqr      :
               ((I_STATE == K_CAL_STATE) && (I_STEP ==  2) &&  I_K[I_COUNT-6]) ? r_x1 ^ r_ybuff :
               ((I_STATE == K_CAL_STATE) && (I_STEP ==  2) && ~I_K[I_COUNT-6]) ? r_x2 ^ r_ybuff :
               ((I_STATE == K_CAL_STATE) && (I_STEP ==  5) &&  I_K[I_COUNT-6]) ? r_x2          :
               ((I_STATE == K_CAL_STATE) && (I_STEP ==  5) && ~I_K[I_COUNT-6]) ? r_x1          :
               ((I_STATE == K_CAL_STATE) && (I_STEP ==  6) &&  I_K[I_COUNT-6]) ? r_z2          :
               ((I_STATE == K_CAL_STATE) && (I_STEP ==  6) && ~I_K[I_COUNT-6]) ? r_z1          :
               ((I_STATE == K_CAL_STATE) && (I_STEP ==  7) &&  I_K[I_COUNT-6]) ? r_x2          :

               ((I_STATE == K_CAL_STATE) && (I_STEP ==  7) && ~I_K[I_COUNT-6]) ? r_x1          :
               ((I_STATE == K_CAL_STATE) && (I_STEP ==  9))                    ? r_ybuff       :
               (I_COUNT == 0)                                                  ? r_z1          :
               (I_COUNT == 4)                                                  ? r_z2          :
               (I_COUNT == 2)                                                  ? r_xp_fourth   :
               I_XP;
assign w_multi_din1 = ((I_STATE == K_CAL_STATE) && (I_STEP ==  0) &&  I_K[I_COUNT-6]) ? r_x1          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  0) && ~I_K[I_COUNT-6]) ? r_x2          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  3) &&  I_K[I_COUNT-6]) ? r_x2          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  3) && ~I_K[I_COUNT-6]) ? r_x1          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  4) &&  I_K[I_COUNT-6]) ? r_x1          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  4) && ~I_K[I_COUNT-6]) ? r_x2          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  8))                    ? I_XP          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  8) &&  I_K[I_COUNT-6]) ? r_x2          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  8) && ~I_K[I_COUNT-6]) ? r_x1          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  9))                    ? r_ybuff       :
                      ((I_COUNT == 1) && (I_STEP == I_INV + 2))                       ? r_xp_sqr      :
                      ((I_COUNT == 3) && (I_STEP == I_INV + 2) && I_INV[ 1])          ? r_ybuff       :
                      ((I_COUNT == 4) && (I_STEP < I_INV))                            ? r_x1          :
                      ((I_COUNT == 4) && (I_STEP == I_INV + 1))                       ? r_xp_sqr      :
                      ((I_COUNT == 4) && (I_STEP == I_INV + 2) && I_INV[ 1])          ? r_ybuff       :
                      ((I_COUNT == 3) && (I_STEP < I_INV))                            ? r_x2          :
                      ((I_COUNT == 3) && (I_STEP == I_INV + 1))                       ? r_xp_sqr      :
                      ((I_COUNT == 3) && (I_STEP == I_INV + 2) && I_INV[ 1])          ? r_ybuff       :
                      (I_COUNT == 2)                                                  ? r_x1 ^ I_XP   :
                      (I_COUNT == 1)                                                  ? r_x1 ^ I_XP   :
                      r_ybuff;
assign w_multi_din2 = ((I_STATE == K_CAL_STATE) && (I_STEP ==  0) &&  I_K[I_COUNT-6]) ? r_z2          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  0) && ~I_K[I_COUNT-6]) ? r_z1          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  3) &&  I_K[I_COUNT-6]) ? r_z1          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  3) && ~I_K[I_COUNT-6]) ? r_z2          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  8))                    ? r_ybuff       :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  4) &&  I_K[I_COUNT-6]) ? r_z1          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  4) && ~I_K[I_COUNT-6]) ? r_z2          :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  8))                    ? r_ybuff       :
                      ((I_STATE == K_CAL_STATE) && (I_STEP ==  9))                    ? I_B           :
                      ((I_COUNT == 1) && (I_STEP == I_INV + 2))                       ? r_z1          :
                      ((I_COUNT == 3) && (I_STEP == I_INV + 2) && I_INV[ 1])          ? r_z1          :
                      ((I_COUNT == 4) && (I_STEP < I_INV))                            ? r_z1          :

                      ((I_COUNT == 4) && (I_STEP == I_INV + 1))                       ? r_z2          :
                      ((I_COUNT == 4) && (I_STEP == I_INV + 2) && I_INV[ 1])          ? r_z2          :
                      ((I_COUNT == 3) && (I_STEP < I_INV))                            ? r_z2          :
                      ((I_COUNT == 3) && (I_STEP == I_INV + 1))                       ? r_xp_fourth   :
                      ((I_COUNT == 3) && (I_STEP == I_INV + 2) && I_INV[ 1])          ? r_xp_fourth   :
                      (I_COUNT == 2)                                                  ? r_x2 ^ I_XP   :
                      (I_COUNT == 1)                                                  ? r_ybuff ^ r_xp_sqr ^ I_YP :
                      r_xp_fourth;
assign O_XQOUT = r_x1;
assign O_YQOUT = r_ybuff;

MULT MULT(
    .I_CLK  (I_CLK),
    .I_EN   (I_MULT_START),
    .I_RSTN (I_RSTN),
    .I_DIN1 (w_multi_din1),
    .I_DIN2 (w_multi_din2),
    .O_DOUT (w_multi_dout),
    .O_DONE (O_MULT_DONE)
);

//------------------r_xp_sqr-----------------------
always@(posedge I_CLK or negedge I_RSTN) begin
    if(~I_RSTN) begin
        r_xp_sqr <= 0;
    end
    else if(I_STATE == IDLE_STATE) begin
        r_xp_sqr <= 0;
    end
    else if(I_STATE == SETUP_CAL_STATE) begin
        if(I_COUNT == 168) begin
            r_xp_sqr <= w_dsqr;
        end
    end
    else if(I_STATE == Q_CAL_STATE) begin
        if((I_COUNT == 5) && (I_STEP == 1)) begin
            r_xp_sqr <= r_z1;
        end
        else if((I_COUNT == 4) && (I_STEP == 1)) begin
```

```
            r_xp_sqr <= r_z2;
        end
        else if((I_COUNT == 3) && (I_STEP == 1)) begin
            r_xp_sqr <= r_xp_fourth;
        end
        else if(I_COUNT == 2) begin
            r_xp_sqr <= w_dsqr;
        end
    end
end
//------------------r_xp_fourth-----------------------
always@(posedge I_CLK or negedge I_RSTN) begin
    if(~I_RSTN) begin
        r_xp_fourth <= 0;
    end
    else if(I_STATE == IDLE_STATE) begin
        r_xp_fourth <= 0;
    end
    else if(I_STATE == SETUP_CAL_STATE) begin
        if(I_COUNT == 168) begin
            r_xp_fourth <= w_dsqr;
        end
    end
    else if(I_STATE == Q_CAL_STATE) begin
        if((I_COUNT == 4) && (I_INV == 0)) begin
            r_xp_fourth <= I_XP;
        end
        if((I_COUNT == 3) && (I_INV == 0)) begin
            r_xp_fourth <= r_ybuff;
        end
        else if((I_COUNT == 3) && (I_STEP <= I_INV)) begin
            r_xp_fourth <= w_dsqr;
        end
        else if((I_COUNT == 3) && (I_STEP == I_INV + 1) && O_MULT_DONE && I_MULT_START) begin
            r_xp_fourth <= w_multi_dout;
        end
        else if((I_COUNT == 3) && (I_STEP == I_INV + 2) && I_INV[0] && O_MULT_DONE && I_MULT_START) begin
            r_xp_fourth <= w_dsqr;
        end
    end
end
//------------------r_x1-----------------------
always@(posedge I_CLK or negedge I_RSTN) begin
    if(~I_RSTN) begin
        r_x1 <= 0;
    end
    else if(I_STATE == IDLE_STATE) begin
        r_x1 <= 0;
    end
    else if(I_STATE == SETUP_CAL_STATE) begin
        if(I_K[I_COUNT-1] && (I_COUNT == 168)) begin
            r_x1 <= I_XP;
        end
    end
    else if(I_STATE == K_CAL_STATE) begin
        if(I_K[I_COUNT-1] && (I_STEP == 10) && O_MULT_DONE && I_MULT_START && ~I_BIT_START) begin
            r_x1 <= I_XP;
        end
        else if(I_K[I_COUNT-1] && (I_STEP == 0) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
            r_x1 <= w_multi_dout;
        end
        else if(I_K[I_COUNT-1] && (I_STEP == 4) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
            r_x1 <= w_multi_dout ^ r_ybuff;
        end
        else if(~I_K[I_COUNT-1] && (I_STEP == 5) && I_BIT_START) begin
            r_x1 <= w_dsqr;
        end
        else if(~I_K[I_COUNT-1] && (I_STEP == 8) && I_BIT_START) begin
            r_x1 <= w_dsqr;
        end
        else if(~I_K[I_COUNT-1] && (I_STEP == 10) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
            r_x1 <= r_x1 ^ w_multi_dout;
        end
    end
    else if(I_STATE == Q_CAL_STATE) begin
        if((I_COUNT == 3) && (I_STEP <= I_INV) && O_MULT_DONE && I_MULT_START) begin
            r_x1 <= w_multi_dout;
        end
    end
end
//------------------r_x2-----------------------
always@(posedge I_CLK or negedge I_RSTN) begin
    if(~I_RSTN) begin
        r_x2 <= 0;
    end
    else if(I_STATE == IDLE_STATE) begin
        r_x2 <= 0;
    end
    else if(I_STATE == SETUP_CAL_STATE) begin
        if(I_K[I_COUNT-1] && (I_COUNT == 168)) begin
            r_x2 <= w_dsqr ^ I_B;
        end
    end
    else if(I_STATE == K_CAL_STATE) begin
        if(I_K[I_COUNT-1] && (I_STEP == 10) && O_MULT_DONE && I_MULT_START && ~I_BIT_START) begin
            r_x2 <= r_xp_fourth ^ I_B;
        end
        else if(I_K[I_COUNT-1] && (I_STEP == 5) && I_BIT_START) begin
            r_x2 <= w_dsqr;
        end
        else if(I_K[I_COUNT-1] && (I_STEP == 8) && I_BIT_START) begin
            r_x2 <= w_dsqr;
        end
        else if(I_K[I_COUNT-1] && (I_STEP == 10) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
            r_x2 <= r_x2 ^ w_multi_dout;
        end
        else if(~I_K[I_COUNT-1] && (I_STEP == 0) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
            r_x2 <= w_multi_dout;
        end
        else if(~I_K[I_COUNT-1] && (I_STEP == 4) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
            r_x2 <= w_multi_dout ^ r_ybuff;
        end
    end
    else if(I_STATE == Q_CAL_STATE) begin
        if((I_COUNT == 3) && (I_STEP <= I_INV) && O_MULT_DONE && I_MULT_START) begin
            r_x2 <= w_multi_dout;
        end
```

```
            end
        end
        //--------------------r_z1-----------------------
        always@(posedge I_CLK or negedge I_RSTN) begin
            if(~I_RSTN) begin
                r_z1 <= 0;
            end
            else if(I_STATE == IDLE_STATE) begin
                r_z1 <= 0;
            end
            else if(I_STATE == SETUP_CAL_STATE) begin
                if(I_K[I_COUNT-6] && (I_COUNT == 168)) begin
                    r_z1 <= 163'd1;
                end
            end
            else if(I_STATE == K_CAL_STATE) begin
                if(I_K[I_COUNT-6] && (I_STEP == 10) && O_MULT_DONE && I_MULT_START && ~I_BIT_START) begin
                    r_z1 <= 163'd1;
                end
                else if(I_K[I_COUNT-6] && (I_STEP == 2) && I_BIT_START) begin
                    r_z1 <= w_dsqr;
                end
                else if(~I_K[I_COUNT-6] && (I_STEP == 7) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
                    r_z1 <= w_multi_dout;
                end
            end
            else if(I_STATE == Q_CAL_STATE) begin
                if((I_COUNT == 5) && (I_INV == 0)) begin
                    r_z1 <= r_ybuff;
                end
                else if((I_COUNT == 5) && (I_STEP <= I_INV)) begin
                    r_z1 <= w_dsqr;
                end
                else if((I_COUNT == 5) && (I_STEP == I_INV + 1) && O_MULT_DONE && I_MULT_START) begin
                    r_z1 <= w_multi_dout;
                end
                else if((I_COUNT == 5) && (I_STEP == I_INV + 2) && I_INV[0] && O_MULT_DONE && I_MULT_START) begin
                    r_z1 <= w_dsqr;
                end

            end
        end
        //--------------------r_z2-----------------------
        always@(posedge I_CLK or negedge I_RSTN) begin
            if(~I_RSTN) begin
                r_z2 <= 0;
            end
            else if(I_STATE == IDLE_STATE) begin
                r_z2 <= 0;
            end
            else if(I_STATE == SETUP_CAL_STATE) begin
                if(I_K[I_COUNT-6] && (I_COUNT == 168)) begin
                    r_z2 <= r_xp_sqr;
                end
            end
            else if(I_STATE == K_CAL_STATE) begin
                if(I_K[I_COUNT-6] && (I_STEP == 10) && O_MULT_DONE && I_MULT_START && ~I_BIT_START) begin
                    r_z2 <= r_xp_sqr;
                end
                else if(I_K[I_COUNT-6] && (I_STEP == 7) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
                    r_z2 <= w_multi_dout;
                end
                else if(~I_K[I_COUNT-6] && (I_STEP == 2) && I_BIT_START) begin
                    r_z2 <= w_dsqr;
                end
            end
            else if(I_STATE == Q_CAL_STATE) begin
                if((I_COUNT == 4) && (I_INV == 0)) begin
                    r_z2 <= r_ybuff;
                end
                else if((I_COUNT == 4) && (I_STEP <= I_INV)) begin
                    r_z2 <= w_dsqr;
                end
                else if((I_COUNT == 4) && (I_STEP == I_INV + 1) && O_MULT_DONE && I_MULT_START) begin
                    r_z2 <= w_multi_dout;
                end
                else if((I_COUNT == 4) && (I_STEP == I_INV + 2) && I_INV[0] && O_MULT_DONE && I_MULT_START) begin
                    r_z2 <= w_dsqr;
                end

            end
        end
        //--------------------r_ybuff-----------------------
        always@(posedge I_CLK or negedge I_RSTN) begin
            if(~I_RSTN) begin
                r_ybuff <= 0;
            end
            else if(I_STATE == IDLE_STATE) begin
                r_ybuff <= 0;
            end
            else if(I_STATE == K_CAL_STATE) begin
                if((I_STEP == 1) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
                    r_ybuff <= w_multi_dout;
                end
                else if((I_STEP == 3) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
                    r_ybuff <= w_multi_dout;
                end
                else if((I_STEP == 6) && I_BIT_START) begin
                    r_ybuff <= w_dsqr;
                end
                else if((I_STEP == 9) && I_BIT_START) begin
                    r_ybuff <= w_dsqr;
                end
            end
            else if(I_STATE == Q_CAL_STATE) begin
                if((I_COUNT <= 5) && (I_COUNT >= 3) && (I_STEP == 0)) begin
                    r_ybuff <= 1;
                end
                else if((I_COUNT <= 5) && (I_COUNT >= 3) && (I_STEP == I_INV + 2) && I_INV[0] && O_MULT_DONE && I_MULT_START) begin
                    r_ybuff <= w_multi_dout;
                end
                else if((I_COUNT <= 2) && (I_COUNT >= 1) && O_MULT_DONE && I_MULT_START) begin
                    r_ybuff <= w_multi_dout;
                end
                else if((I_COUNT == 0) && O_MULT_DONE && I_MULT_START) begin
                    if(I_K == 0) begin
                        r_ybuff <= 0;
                    end
                    else if(I_K == 1) begin
                        r_ybuff <= I_YP;
                    end
                    else begin
                        r_ybuff <= w_multi_dout ^ I_YP;
                    end
                end
            end
        end
    endmodule
```

Code Module MULT

```
`timescale 1ns / 1ps
module MULT(
    input           I_CLK,
    input           I_EN,
    input           I_RSTN,
    input  [162:0] I_DIN1,
    input  [162:0] I_DIN2,
    output [162:0] O_DOUT,
    output          O_DONE
);
    parameter   IDLE_STATE   = 0;
    parameter   CAL_STATE    = 1;
    parameter   FINISH_STATE = 2;

    reg [1:0]   r_state;
    reg [7:0]   r_count;
    reg [162:0] r_a;
    reg [169:0] r_c;

    wire [167:0] w_b;
    wire [162:0] w_b0;
    wire [162:0] w_b1;
    wire [162:0] w_b2;
    wire [162:0] w_b3;
    wire [162:0] w_b4;
    wire [162:0] w_b5;
    wire [162:0] w_b6;
    wire [162:0] w_b7;

    assign w_b = {5'b0,I_DIN2};
    assign w_b0 = (w_b[r_count])    ? 163'h7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
    assign w_b1 = (w_b[r_count+1]) ? 163'h7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
    assign w_b2 = (w_b[r_count+2]) ? 163'h7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
    assign w_b3 = (w_b[r_count+3]) ? 163'h7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
    assign w_b4 = (w_b[r_count+4]) ? 163'h7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
    assign w_b5 = (w_b[r_count+5]) ? 163'h7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
    assign w_b6 = (w_b[r_count+6]) ? 163'h7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
    assign w_b7 = (w_b[r_count+7]) ? 163'h7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
    assign O_DOUT = r_c[162:0] ^ {r_c[168], r_c[169]^r_c[168], r_c[168]^r_c[167], r_c[167]^r_c[166], r_c[166]^r_c[165]^r_c[169], r_c[165]^r_c[164]^r_c[168],
                                  r_c[164]^r_c[163]^r_c[167], r_c[163]^r_c[169]^r_c[166], r_c[169]^r_c[168]^r_c[165], r_c[168]^r_c[167]^r_c[164], r_c[167]^r_c[166]^r_c[163], r_c[166]^r_c[165],
                                  r_c[165], r_c[164], r_c[163]};
    assign O_DONE = r_state[1];

    //----------------------r_state----------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_state <= IDLE_STATE;
        end
        else if(~I_EN) begin
            r_state <= IDLE_STATE;
        end
        else if(r_state == IDLE_STATE) begin
            r_state <= CAL_STATE;
        end
        else if (r_count == 160) begin
            r_state <= FINISH_STATE;
        end
    end
    //----------------------r_count----------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_count <= 0;
        end
        else if(r_state == IDLE_STATE) begin
            r_count <= 0;
        end
        else if(r_state == CAL_STATE) begin
            if(r_count < 160) begin
                r_count <= r_count + 8;
            end
        end
    end
    //----------------------r_a----------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_a <= 0;
        end
        else if(r_state == IDLE_STATE) begin
            r_a <= I_DIN1;
        end
        else if(r_state == CAL_STATE) begin
            r_a <= (r_a << 8) ^ {r_a[162], r_a[162]^r_a[161], r_a[161]^r_a[160], r_a[160]^r_a[159], r_a[159]^r_a[158]^r_a[162], r_a[158]^r_a[157]^r_a[161],
                                 r_a[157]^r_a[156]^r_a[160], r_a[156]^r_a[155]^r_a[162]^r_a[159], r_a[155]^r_a[161]^r_a[158], r_a[160]^r_a[157], r_a[159]^r_a[156],
                                 r_a[156]^r_a[155], r_a[157], r_a[158], r_a[159]};
        end
    end
    //----------------------r_c----------------------
    always@(posedge I_CLK or negedge I_RSTN) begin
        if(~I_RSTN) begin
            r_c <= 0;
        end
        else if(r_state == IDLE_STATE) begin
            r_c <= 0;
        end
        else if(r_state == CAL_STATE) begin
            r_c <= r_c ^ ((r_a & w_b7) << 7) ^ ((r_a & w_b6) << 6) ^ ((r_a & w_b5) << 5) ^ ((r_a & w_b4) << 4)
                       ^ ((r_a & w_b3) << 3) ^ ((r_a & w_b2) << 2) ^ ((r_a & w_b1) << 1) ^  (r_a & w_b0);
        end
    end
endmodule
```

ประวัติผู้เขียน

ชื่อ-สกุล นายพสธร ไทรทอง

วัน เดือน ปีเกิด 16 มิถุนายน 2540

ที่อยู่ 25/320 หมู่ 3 ต.บางหญ้าแพรก อ.เมืองสมุทรสาคร จ.สมุทรสาคร 74000

อาชีพปัจจุบัน นักศึกษา

รหัสนักศึกษา 59010928

ชั้นปีที่ 4

ภาควิชา อิเล็กทรอนิกส์

สาขา วิศวกรรมอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์

มหาวิทยาลัย สถาบันพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ประวัติการทำงาน ฝึกงานกับทำสหกิจศึกษากับบริษัท ซิลิคอนคราฟท์ เทคโนโลยี จำกัด (มหาชน)