



## รายงานสหกิจศึกษาฉบับสมบูรณ์

การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163

Hardware Implementation Elliptic Curve Operation Curve B-163

นายพสธร ไทรทอง

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2562

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า  
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ชื่อโครงการสหกิจศึกษา การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163

ชื่อ-สกุล นักศึกษา นายพชร ไทรทอง

คณะ วิศวกรรมศาสตร์ ภาควิชา อิเล็กทรอนิกส์

ชื่อ-สกุล อาจารย์นิเทศ ดร.สุเมฆ วิศยทัทธิณ

ชื่อ-สกุล ผู้นิเทศงาน นายธนพล หงษ์ทรงเกียรติ

ชื่อสถานประกอบการ บริษัท ซิลิคอนกราฟท์ เทคโนโลยี จำกัด (มหาชน)

### บทคัดย่อ

งานวิจัยฉบับนี้อธิบายหลักการออกแบบและสร้างวงจรการทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ซึ่งเป็นวงจรการควบคุมจุดของเส้นโค้งรูปไข่รูปแบบ B-163 โดยเส้นโค้งรูปไข่รูปแบบ B-163 อยู่ในสนามจำกัดแบบสนามฐานสองจึงต้องอาศัยการทำงานของสนามจำกัดแบบฐานสองเป็นพื้นฐานการคำนวณ โดยได้ใช้วงจรการคูณของสนามจำกัดแบบฐานสองเป็นแบบการคูณอนุกรมดิจิทัล และใช้วงจรการควบคุมจุดของเส้นโค้งรูปไข่รูปแบบ B-163 ใช้กระบวนการคำนวณแบบ X-Coordinate only Montgomery Ladder ผลลัพธ์ที่ได้คือ วงจรที่ได้ทำการออกแบบนั้นทำงานได้อย่างถูกต้องสมบูรณ์เมื่อนำไปทดสอบกับมาตรฐาน อีกทั้งวงจรมีระยะเวลาที่ใช้ในการคำนวณไม่มากนักเพียง 25,436 CLK และมีขนาดที่ค่อนข้างเล็ก คือ 26,122 GE หรือ  $261,222 \mu m^2$

คำสำคัญ : ออกแบบวงจรรวมดิจิทัล 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  $\mu m^2$

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

## กิตติกรรมประกาศ

การจัดทำงานวิจัยการสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 สำเร็จได้ ผู้จัดทำต้องขอขอบคุณทางบริษัท ซิลิคอนกราฟท์ เทคโนโลยี จำกัด (มหาชน) ที่ให้สถานที่ในการจัดทำ อุปกรณ์ในการทดลองและทำงานวิจัย และให้โอกาสไปศึกษาหาความรู้จากการทำงานจริงจากสถานประกอบการ ขอขอบคุณพี่ ๆ ที่บริษัท ซิลิคอนกราฟท์ เทคโนโลยี จำกัด (มหาชน) ทุกคนที่คอยให้คำปรึกษา คำแนะนำ ความรู้ และประสบการณ์ต่าง ๆ มากมาย

พสธร ไทรทอง  
ผู้จัดทำ



# สารบัญ

หน้า

บทคัดย่อภาษาไทย.....	i
บทคัดย่อภาษาอังกฤษ.....	ii
กิตติกรรมประกาศ.....	iii
สารบัญ.....	iv
สารบัญตาราง.....	vi
สารบัญภาพ.....	vii
<b>บทที่ 1 บทนำ.....</b>	<b>1</b>
1.1 ความเป็นมาและความสำคัญ.....	1
1.2 วัตถุประสงค์ของการวิจัย.....	1
1.3 ขอบเขตของการวิจัย.....	1
1.4 วิธีดำเนินการวิจัย.....	2
1.5 ประโยชน์ที่คาดว่าจะได้รับ.....	2
<b>บทที่ 2 แนวคิด ทฤษฎีและงานวิจัยที่เกี่ยวข้อง.....</b>	<b>3</b>
2.1 ทฤษฎีการออกแบบวงจรทางดิจิทัล.....	3
2.2 วิธีการออกแบบวงจรทางดิจิทัล.....	4
2.3 การเขียนภาษา HDL.....	5
2.4 ภาษา Verilog (Verilog HDL).....	6
2.5 Cryptography (วิทยาการเข้ารหัส).....	7
2.6 Elliptic-curve cryptography (ECC).....	8
2.7 Finite field (สนามจำกัด).....	9
2.8 Binary field Arithmetic (การดำเนินการของสนามจำกัดแบบสนามฐานสอง).....	10
2.9 Elliptic Curve operation (การดำเนินการของเส้นโค้งรูปไข่).....	11
2.10 งานวิจัยที่เกี่ยวข้อง.....	13
<b>บทที่ 3 วิธีดำเนินงานวิจัย.....</b>	<b>14</b>
3.1 ออกแบบวงจร.....	14

## สารบัญ(ต่อ)

	หน้า
3.2 วงจรการทำงานของเส้นโค้งรูปไข่รูปแบบ B-163.....	18
บทที่ 4 ผลการวิจัย .....	19
4.1 ผลการวิจัย .....	19
บทที่ 5 สรุปผลการวิจัยและข้อเสนอแนะ.....	21
5.1 สรุปผลการวิจัย .....	21
5.2 ข้อเสนอแนะ .....	21
เอกสารอ้างอิง .....	22
ภาคผนวก.....	23



## สารบัญตาราง

ตารางที่	หน้า
1.1 แสดงลำดับและวิธีการดำเนินงาน.....	2
4.1 แสดงขนาดของวงจรและโมดูลต่าง ๆ .....	20



## สารบัญภาพ

ภาพที่	หน้า
2.1 แผนผังขั้นตอนการออกแบบวงจรทางดิจิทัล.....	5
2.2 แผนผังขั้นตอนการออกแบบวงจรทางดิจิทัลด้วยภาษา HDL .....	6
2.3 symmetric-key cryptography .....	8
2.4 asymmetric-key cryptography .....	9
2.5 Elliptic Curve.....	10
2.6 Hierarchy of elliptic-curve implementations .....	12
2.7 Point Addition operation.....	13
2.8 Point doubling operation .....	14
2.9 Point multiplication operation.....	14
3.1 Flow Chart การทำงานของวงจรการคูณของสนามกำลังสอง.....	16
3.2 ภาพรวมวงจร Point multiplication operation.....	17
3.3 Flow Chart การทำงานของการ Inversion .....	18
3.4 Flow Chart การทำงานของวงจร Point multiplication operation ช่วงการคำนวณตามกำลัง ของจุดฐาน .....	19
3.5 วงจรการทำงานของเส้นโค้งรูปไข่รูปแบบ B-163.....	20
4.1 ผลการเปรียบเทียบความถูกต้องของการคำนวณ.....	22
4.2 ระยะเวลาทั้งหมดในการทำการคำนวณ.....	22

# บทที่ 1

## บทนำ

### 1.1 ความเป็นมาและความสำคัญ

ในปัจจุบันเทคโนโลยีมีความก้าวหน้า ก่อเกิดหลายสิ่งๆ ที่เข้ามาอย่างเปลี่ยนวิถีชีวิตของเราให้สะดวกสบายยิ่งขึ้น แต่ทุกอย่างย่อมต้องมีทั้งข้อดีและข้อเสีย ในอินเทอร์เน็ตมีข้อมูลเท็จและการปลอมแปลงข้อมูลมากมาย อีกทั้งข้อมูลความลับที่เราเก็บไว้ก็อาจถูกโจรกรรมไปได้ ความไม่ปลอดภัยของข้อมูลจึงเกิดขึ้นจึงทำให้เราไม่สามารถนำข้อมูลที่สำคัญของเราเข้าไปอยู่ในอินเทอร์เน็ตหรือส่งเป็นข้อมูลดิจิทัล ดังนั้นการที่ข้อมูลมีความปลอดภัยที่สูง ปลอมแปลงได้ยาก ข้อมูลที่สำคัญมีการเก็บรักษาที่ปลอดภัย และสามารถตรวจสอบได้ว่าถูกแก้ไขปลอมแปลงมาหรือไม่ จะทำให้เราสามารถทำธุรกรรมต่าง ๆ โดยมีความสบายใจและปลอดภัยมากกว่า

### 1.2 วัตถุประสงค์ของการวิจัย

- 1.2.1 เพื่อศึกษาหลักการออกแบบวงจรรวมอิเล็กทรอนิกส์ในทางดิจิทัล
- 1.2.2 เพื่อศึกษาหลักการเข้ารหัสข้อมูลโดยวิธีเส้นโค้งรูปไข่
- 1.2.3 เพื่อศึกษาวิธีการทำงานของเส้นโค้งรูปไข่ในขอบเขตจำกัด
- 1.2.4 เพื่อทำการออกแบบวงจรรวมดิจิทัลของการทำงานของเส้นโค้งรูปไข่

### 1.3 ขอบเขตของการวิจัย

- 1.3.1 ออกแบบวงจรรวมอิเล็กทรอนิกส์ในทางดิจิทัล
- 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											←			→		
แก้ไขและพัฒนางานวงจร												←			→	
ทำรูปเล่มงานวิจัย													←			→

## 1.5 ประโยชน์ที่คาดว่าจะได้รับ

- 1.5.1 ได้รับความรู้เกี่ยวกับหลักการออกแบบวงจรรวมอิเล็กทรอนิกส์ในทางดิจิทัล
- 1.5.2 ได้รับความรู้เกี่ยวกับหลักการเข้ารหัสข้อมูลโดยวิธีเส้นโค้งรูปไข่
- 1.5.3 ได้รับความรู้เกี่ยวกับวิธีการทำงานของเส้นโค้งรูปไข่ในขอบเขตจำกัด
- 1.5.4 ได้ทำการออกแบบวงจรรวมดิจิทัลของการทำงานของเส้นโค้งรูปไข่

## บทที่ 2

### แนวคิด ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ครั้งนี้ ผู้ดำเนินงาน ได้ทำการศึกษาและรวบรวมทฤษฎีและหลักการต่าง ๆ จากเอกสารและงานวิจัย โดยมีทฤษฎีและงานวิจัยที่เกี่ยวข้อง ดังนี้

#### 2.1 ทฤษฎีการออกแบบวงจรทางดิจิทัล

รูปแบบการออกแบบวงจรรวมทางดิจิทัลแบ่งออกได้ดังนี้

##### 2.1.1 การออกแบบด้วยสมการบูลีน (Designing with Boolean Equations)

วงจรลอจิกสร้างจากเกตและฟลิปฟล็อปจึงทำการการออกแบบด้วยสมการบูลีนได้ เนื่องจากเกตและฟลิปฟล็อปสามารถกำหนดได้จากสมการบูลีน วิธีการมีหลายเทคนิคที่ถูกออกแบบมา สำหรับการทำให้ออกแบบใช้ทรัพยากรคือเกตหรือฟลิปฟล็อปให้น้อยที่สุดประกอบด้วยการลดรูปของสมการบูลีนโดยใช้ Karnaugh map (K-Map) เป็นต้น และเนื่องด้วยการออกแบบด้วยสมการบูลีน ต้องการสมการสำหรับการกำหนดการใช้งานของแต่ละเกตดังนั้นจึงทำให้ไม่สามารถใช้งานได้จริงสำหรับระบบที่มีความซับซ้อนและมีขนาดใหญ่ที่ประกอบด้วยจำนวนเกตเป็นแสนๆตัวในปัจจุบันได้

##### 2.1.2 การออกแบบด้วย Schematic (Schematic-based Design)

การออกแบบด้วย Schematic ช่วยในการออกแบบด้วยสมการบูลีน เนื่องจากไม่ได้มีแค่เกตหรือฟลิปฟล็อปที่เป็นอุปกรณ์พื้นฐานเท่านั้นแต่ยังมีวงจรถูกจัดไว้ในไลบรารี (Library) ด้วย นอกจากนี้วิธีนี้ยังสามารถออกแบบระบบเป็นลักษณะของลำดับชั้น (Hierarchy) ได้อีกด้วย ซึ่งทำให้สามารถออกแบบระบบที่ซับซ้อนได้มากขึ้นและสามารถประหยัดเวลาในการออกแบบเนื่องจากสามารถมองเห็นเป็นรูปร่างของระบบที่ชัดเจนเพราะมีลักษณะการออกแบบที่มองเห็นเป็นภาพ (Graphical representation) แต่สำหรับระบบที่มีความซับซ้อนและมีขนาดใหญ่ที่ประกอบด้วยจำนวนเกตเป็นแสนๆตัวในปัจจุบันจะต้องเสียเวลาในการเชื่อมต่อแต่ละขาของอุปกรณ์รวมทั้งยากในการตรวจสอบความถูกต้องและการแก้ไข เพราะในวงจรมีการเชื่อมต่อกันมากมาย

##### 2.1.3 การออกแบบโดยใช้ภาษาระดับสูง

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ลตรูปร่างไม่ต้องเสียเวลาลากเส้นวงจร ภาษา HDL มีหลายภาษาแต่ภาษาที่มีข้อดีละข้อเสียที่แตกต่างกันไป โดยที่นิยมใช้กันอยู่ได้แก่ ภาษา Abel HDL, ภาษา VHDL และ ภาษา Verilog

## 2.2 วิธีการออกแบบวงจรทางดิจิทัล

ในการออกแบบวงจรทางดิจิทัล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ ที่ใช้งานได้จะต้องผ่านขั้นตอนต่าง ๆ และผู้ออกแบบจะต้องตรวจสอบผลลัพธ์ในแต่ละขั้น ก่อนเข้าสู่กระบวนการออกแบบในขั้นต่อไป โดยขั้นตอนปกติที่ใช้ในการออกแบบวงจรทางดิจิทัลทั่วไป

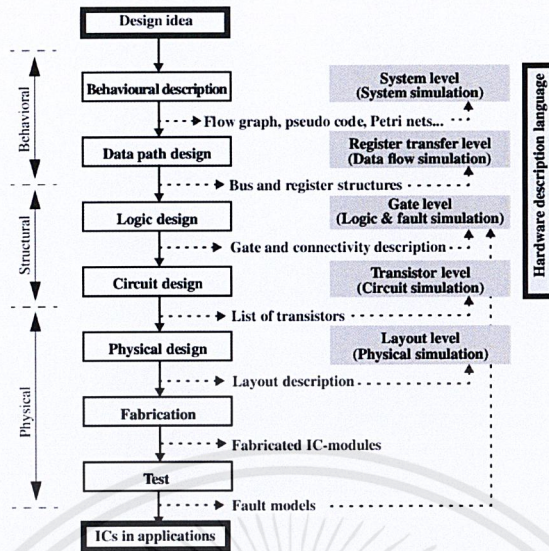
2.2.1 กำหนดแนวความคิดในการออกแบบแล้วทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ซึ่งภายในขั้นตอนนี้ผู้ออกแบบจำเป็นต้องสร้างรูปแบบในเชิงพฤติกรรมขึ้นมาตรวจสอบซึ่งอาจจะเป็นผังงานแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo code)

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

2.2.3 การออกแบบวงจรลอจิก ซึ่งจะเกี่ยวข้องกับการนำเกทดิจิทัลพื้นฐานและฟลิปฟลอป (flip-flop) มาประกอบเป็นอุปกรณ์ย่อยต่าง ๆ ผลลัพธ์ที่จะเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอป

2.2.4 เปลี่ยนเครือข่ายการโยงใยให้เป็นลำดับของทรานซิสเตอร์ (Transistor List) และ Layout ซึ่งจะเกี่ยวข้องโดยตรงกับการจัดวางทรานซิสเตอร์หรือไลบรารีเซลล์เพื่อแทนเกทและฟลิปฟลอปต่าง ๆ

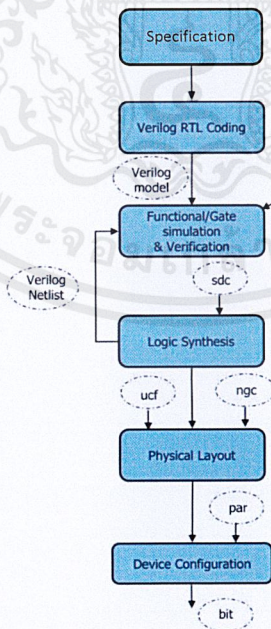
2.2.5 ส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด



ภาพที่ 2.1 แผนผังขั้นตอนการออกแบบวงจรทางดิจิทัล

ที่มา : [https://www.researchgate.net/figure/Design-flow-and-levels-of-abstractions\\_fig1\\_2675282](https://www.researchgate.net/figure/Design-flow-and-levels-of-abstractions_fig1_2675282)

### 2.3 การเขียนภาษา HDL



ภาพที่ 2.2 แผนผังขั้นตอนการออกแบบวงจรทางดิจิทัลด้วยภาษา 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 เพื่อให้ได้เนตลิสต์ในระดับเกตออกมา เครื่องมือที่ใช้ในการสังเคราะห์โค้ดภาษา HDL (HDL Synthesis Tool) ซึ่งสามารถสังเคราะห์วงจรสำหรับ FPGA ได้และสามารถเลือกไลบรารีที่เหมาะสมกับชิพที่เลือกใช้งานได้

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

2.3.5 การวางและการเชื่อมเส้นทาง (Place and Route) ผลที่ได้จากการสังเคราะห์นั้นจะอยู่ในรูปแบบเนตลิสต์ในระดับเกต โดยสามารถนำไปผ่าน ขั้นตอนการแปลงให้เป็นลอจิกและรวมถึงการวางและการเชื่อมเส้นทางภายในอุปกรณ์ FPGA

## 2.4 ภาษา Verilog (Verilog HDL)

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

การออกแบบทั่วไปในอุตสาหกรรมวงจรรวมโดยเฉพาะทางอเมริกาและญี่ปุ่นอย่างไรก็ตามภาษา Verilog ก่อนข้างจะด้อยในด้านความสามารถในการกำหนดการทำงานของระบบในระดับที่สูงขึ้น (System level specification)

#### 2.4.1 โครงสร้างของภาษา Verilog

ภาษา Verilog นั้น สามารถอธิบายพฤติกรรมของวงจรถติจติตอลได้หลายระดับ เช่น สามารถอธิบายวงจรในระดับการวาด (Layout) รีซิสเตอร์ ทรานซิสเตอร์และการเชื่อมต่อ(Wire) บนชิพ ไอซี (Integrated Circuit chip) ซึ่งเรียกว่า ระดับการสวิตช์ (Switch level) สามารถอธิบายการเชื่อมต่อของลอจิกเกตและฟลิปฟลอปในวงจรถติจติตอลเรียกว่า ระดับ เกต (Gate level) และสามารถอธิบายการส่งข้อมูลระหว่างรีจิสเตอร์เรียกว่า ระดับ RTL (Register Transfer Level) ซึ่งการออกแบบวงจรถติจติตอลด้วยภาษา Verilog ส่วนใหญ่จะใช้คำอธิบายพฤติกรรมของวงจรถติจติตอลในระดับ RTL นี้ ภาษา Verilog นั้น มีไวยากรณ์คล้ายภาษาซี เช่น จบประโยคด้วย ; หรือใช้ // เพื่อไม่ให้นำไปใช้ในการคอมไพล์ เป็นต้น โครงสร้างของภาษา Verilog นั้น ประกอบด้วยโมดูล (Module) สามารถรวมเข้าด้วยกันเป็นโมดูลขนาดใหญ่หรือเรียกว่า ท็อปโมดูล (Top module) โดยแต่ละโมดูลย่อยนั้น สามารถทำงานพร้อมกัน (Concurrent)

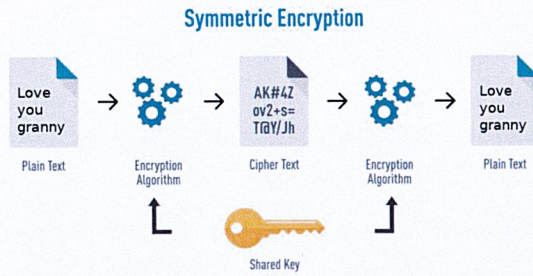
### 2.5 Cryptography (วิทยาการเข้ารหัส)

วิทยาการเข้ารหัสเป็นกระบวนการที่ทำให้การติดต่อสื่อสารระหว่างสองบุคคลมีความปลอดภัย โดยที่บุคคลที่สามไม่สามารถแอบฟังข้อมูลที่แลกเปลี่ยนกันได้

การเข้ารหัสจะทำก่อนการส่งข้อมูล โดยนำข้อมูลกับกุญแจ มาผ่านกระบวนการทางคณิตศาสตร์ จะได้เป็นข้อมูลที่เข้ารหัส และการถอดรหัสจะทำโดยนำเอาข้อมูลที่เข้ารหัสกับกุญแจมาผ่านกระบวนการทางคณิตศาสตร์ จะได้เป็นข้อมูลดั้งเดิม โดยในส่วนของกระบวนการไม่จำเป็นต้องเป็นความลับแต่กุญแจต้องเป็นความลับ ระบบเข้ารหัสสามารถแบ่งตามวิธีการใช้กุญแจได้เป็น 2 วิธีดังนี้

#### 2.5.1 symmetric-key cryptography

symmetric-key cryptography คือการเข้ารหัสข้อมูลด้วยกุญแจเดียว โดยที่ผู้ส่งจะใช้กุญแจนี้ในการเข้ารหัสและผู้รับจะใช้กุญแจที่เหมือนกันในการถอดรหัส ซึ่งวิธีการนี้ผู้รับกับผู้ส่งต้องตกลงกันก่อนว่าจะใช้รูปแบบกระบวนการทางคณิตศาสตร์ใดในการเข้ารหัสข้อมูลเพื่อเก็บเป็นความลับ

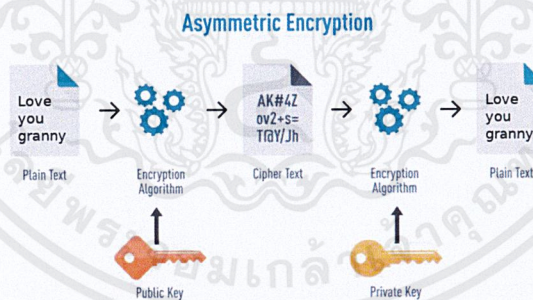


ภาพที่ 2.3 symmetric-key cryptography

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 ไม่สามารถนำกุญแจอื่นหรือของบุคคลอื่นมาถอดรหัสได้



ภาพที่ 2.4 asymmetric-key cryptography

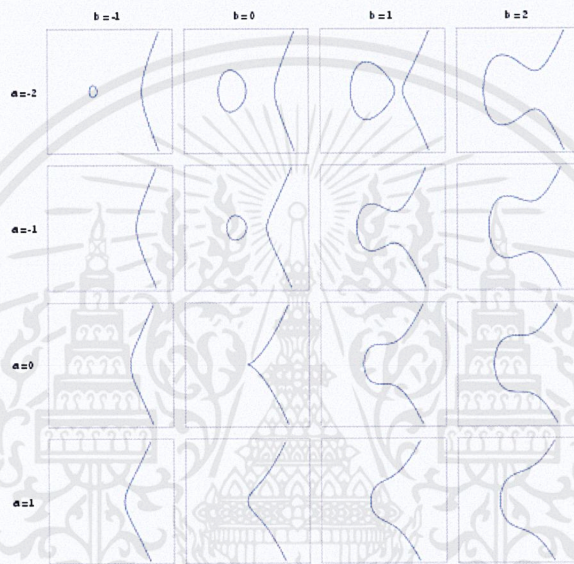
ที่มา : <https://medium.com/@kapilvermarbl/symmetric-asymmetric-and-hybrid-encryption-25d57c1c327b>

### 2.6 Elliptic-curve cryptography (ECC)

การเข้ารหัสแบบเส้นโค้งรูปไข่เป็นวิธีการที่มีประสิทธิภาพในการเข้ารหัส มันเป็นวิธีการที่ใช้สำหรับการเข้ารหัสกุญแจสาธารณะโดยใช้โครงสร้างพีชคณิตของเส้นโค้งรูปไข่บนสนามจำกัด เพื่อสร้างความปลอดภัยระหว่างคู่กุญแจ การเข้ารหัสแบบเส้นโค้งรูปไข่ได้รับความนิยมอย่างช้า ๆ ในช่วงไม่กี่ปีที่

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](https://en.wikipedia.org/wiki/Elliptic_curve)

## 2.7 Finite field (สนามจำกัด)

Finite field เป็นสนามที่มีองค์ประกอบจำนวนจำกัด จำนวนองค์ประกอบในสนามจำกัด เรียกว่า 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 เป็นจำนวนเต็มที่มีมากกว่าหรือเท่ากับ 1 เรียกว่ามิติ Finite field สามารถแยกแยะได้ด้วยวิธีการดังนี้

- ถ้ามิติ (m) มีค่าเท่ากับ 1 จะเรียกว่า Prime Fields (สนามเฉพาะ) หรือ  $F_p$
- ถ้ามิติ (m) มีค่ามากกว่า 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  $1101 \otimes 0111 \Rightarrow 1010$

### 2.8.2 การคูณ (Multiplication operation)

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

### 2.8.3 การยกกำลังสอง (Squaring operation)

จะเป็นการนำไปทำการคูณตัวเองก็ได้ แต่สามารถคูณดีกรีทั้งหมดด้วย 2 แทนได้ ซึ่งถ้ามองในรูป binary bit string แล้ว จะเป็นการนำ 0 ไปแทรกระหว่างบิตนั่นเอง แล้วจึงนำไปหารเอาเศษ เช่นตัวอย่างนี้  $1101^2 = 1010001 \text{ mod } 10011$

### 2.8.4 การหารเอาเศษ (Reduction Modulus operation)

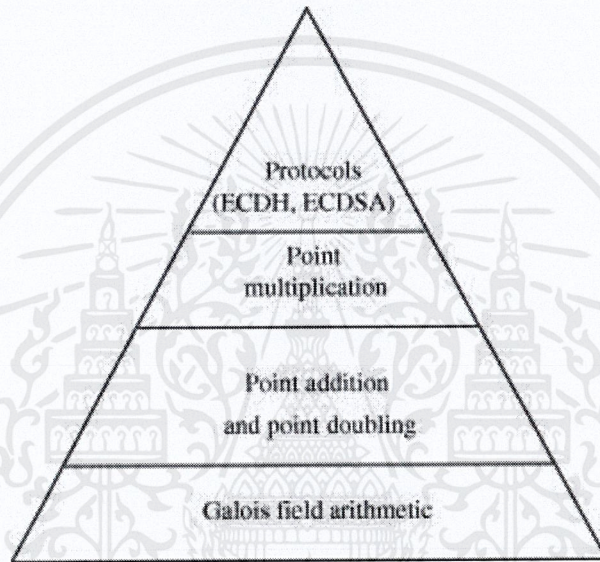
จะเป็นการนำไป XOR กันแบบบิตต่อบิตกับตัวลดการหารเอาเศษ (Reduce modulus) ที่ถูก shift บิตจนเท่ากันแล้ว จนมีดีกรีไม่เกิน  $m-1$  เช่นตัวอย่างนี้  $100011 \otimes 100110 = 0101$

## 2.8.5 การผกผัน (Inversion operation)

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

## 2.9 Elliptic Curve operation (การดำเนินการของเส้นโค้งรูปไข่)

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

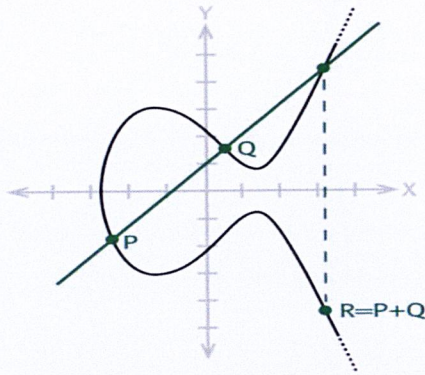


ภาพที่ 2.6 Hierarchy of elliptic-curve implementations

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

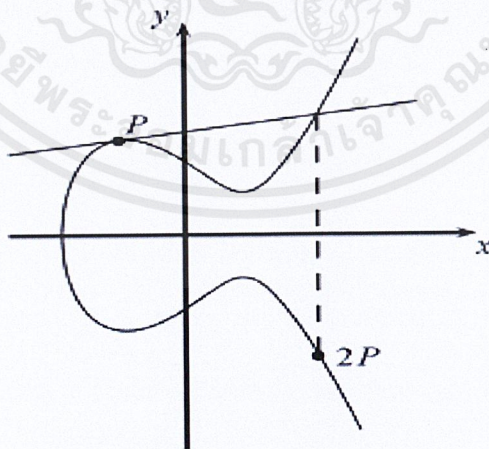
ที่มา : <https://sinhvientot.net/elliptic-curve-cryptography/>

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  $\lambda$  มีค่าเป็นอนันต์ จะทำให้  $R$  มีค่าเป็นอนันต์ด้วย

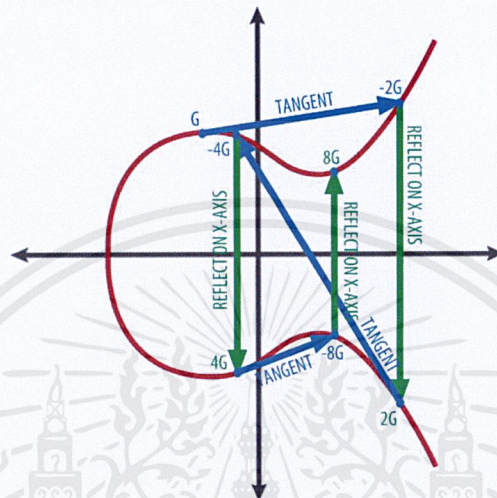


ภาพที่ 2.8 Point doubling operation

ที่มา : [https://www.researchgate.net/figure/Point-doubling-22-Elliptic-curve-Diffie-Hellman-ECDH-key-exchange\\_fig1\\_308548189](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  $P$  ใ้กับตัวเองโดยที่  $k$  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-ec- 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  $GF(2^m)$  without Precomputation โดยศึกษากระบวนการคูณที่เร็วในเส้นโค้งรูปไข่บนสนามจำกัดแบบสนามฐานสอง พบว่า ได้วิธีที่ดีและใช้เวลาทำงานน้อยกว่าวิธีอื่น ๆ ที่นำมาเปรียบเทียบ

## บทที่ 3

### วิธีดำเนินงานวิจัย

ในการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ผู้ดำเนินงานได้มีวิธีการดำเนินงานวิจัย ดังนี้

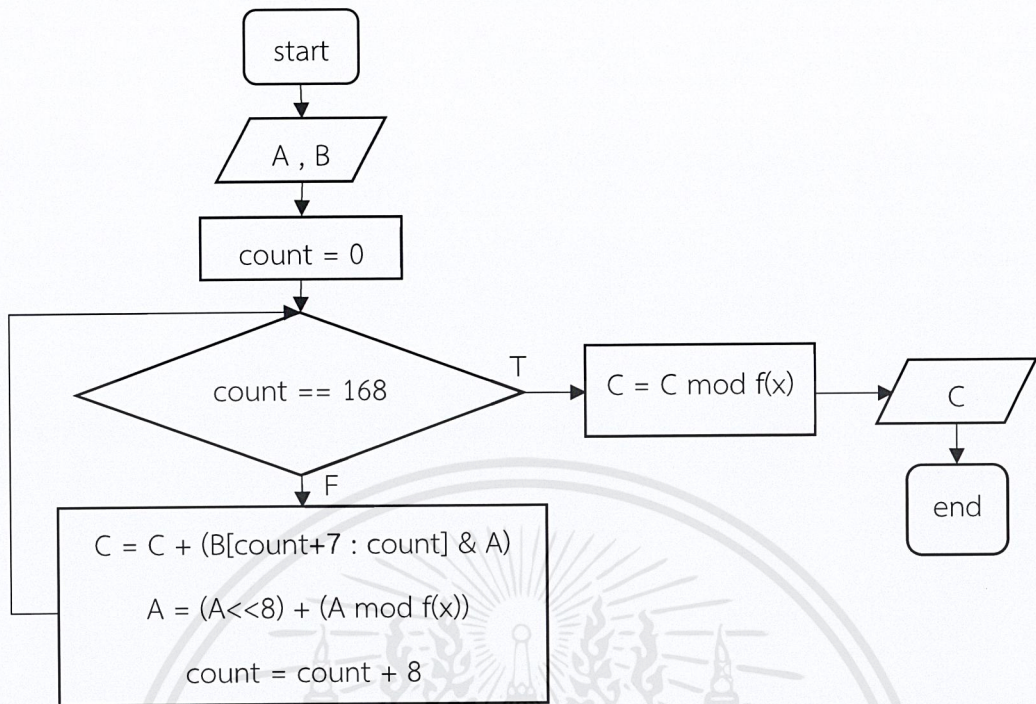
#### 3.1 ออกแบบวงจร

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  $n = 40000000000000000000292FE77E70C12A4234C33$
- ได้ทำการแยกขั้นตอนการออกแบบเป็น 2 ส่วนใหญ่ ได้แก่

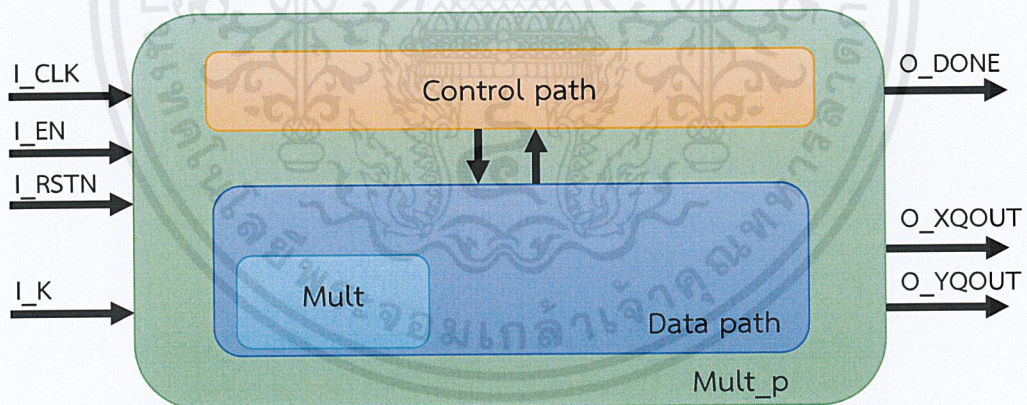
##### 3.1.1 วงจรการคูณของสนามกำจัดแบบสนามฐานสอง

วงจรการคูณของสนามกำจัดแบบสนามฐานสองจะประกอบไปด้วยการ + จะใช้การ XOR และ การ mod จะใช้การตั้งหารยาวจากสมการพหุนาม โดยหารเหลือเศษไว้ในรูปตัวแปรก่อนแล้วนำค่ามาแทนตามค่าที่นำมาคำนวณ วงจรการคูณของสนามกำจัดแบบสนามฐานสองใช้กระบวนการคำนวณแบบอนุกรมดิจิทัล มีขั้นตอนดังนี้



ภาพที่ 3.1 Flow Chart การทำงานของวงจรการคูณของสนามกำลังจัดแบบสนามฐานสอง

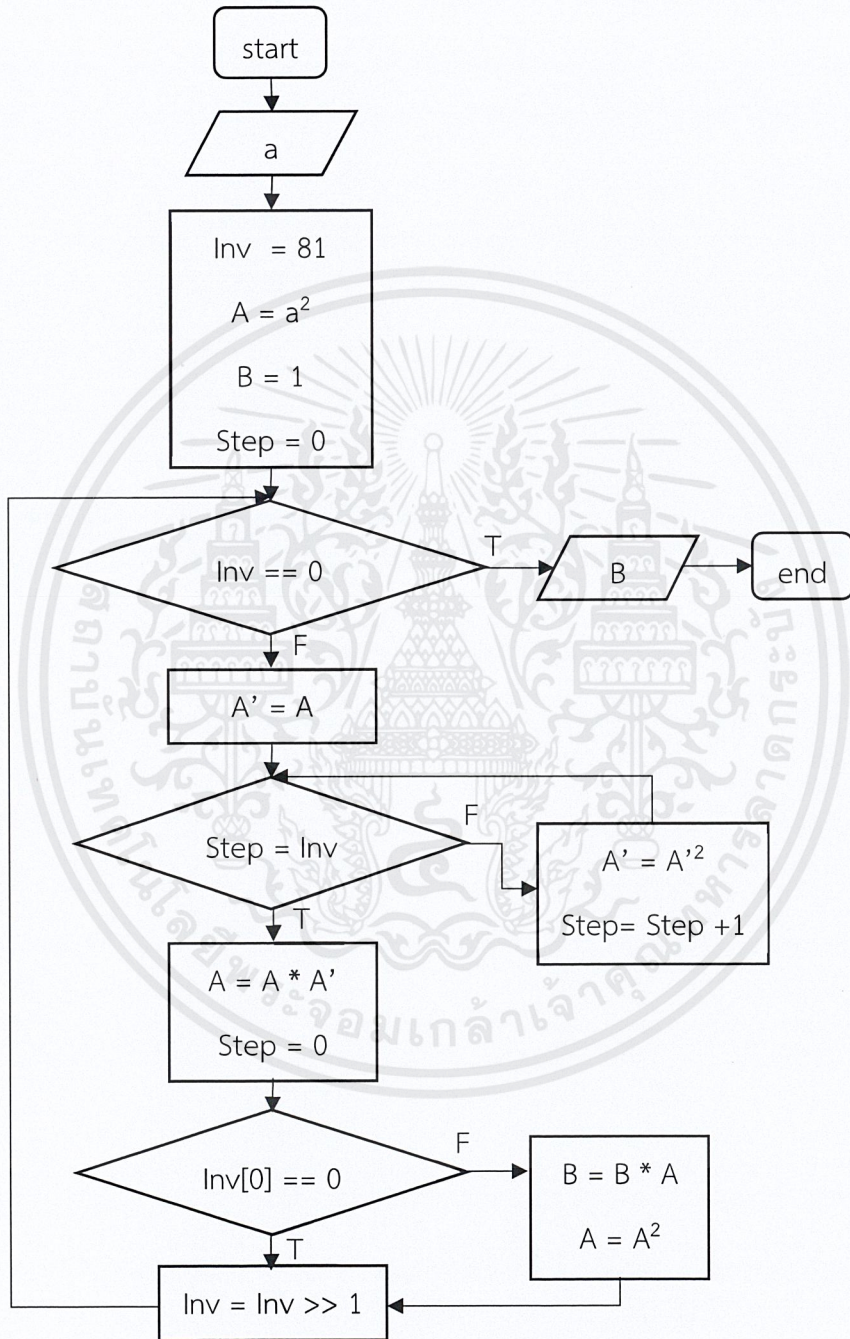
### 3.1.2 วงจร Point multiplication operation



ภาพที่ 3.2 ภาพรวมวงจร Point multiplication operation

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

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

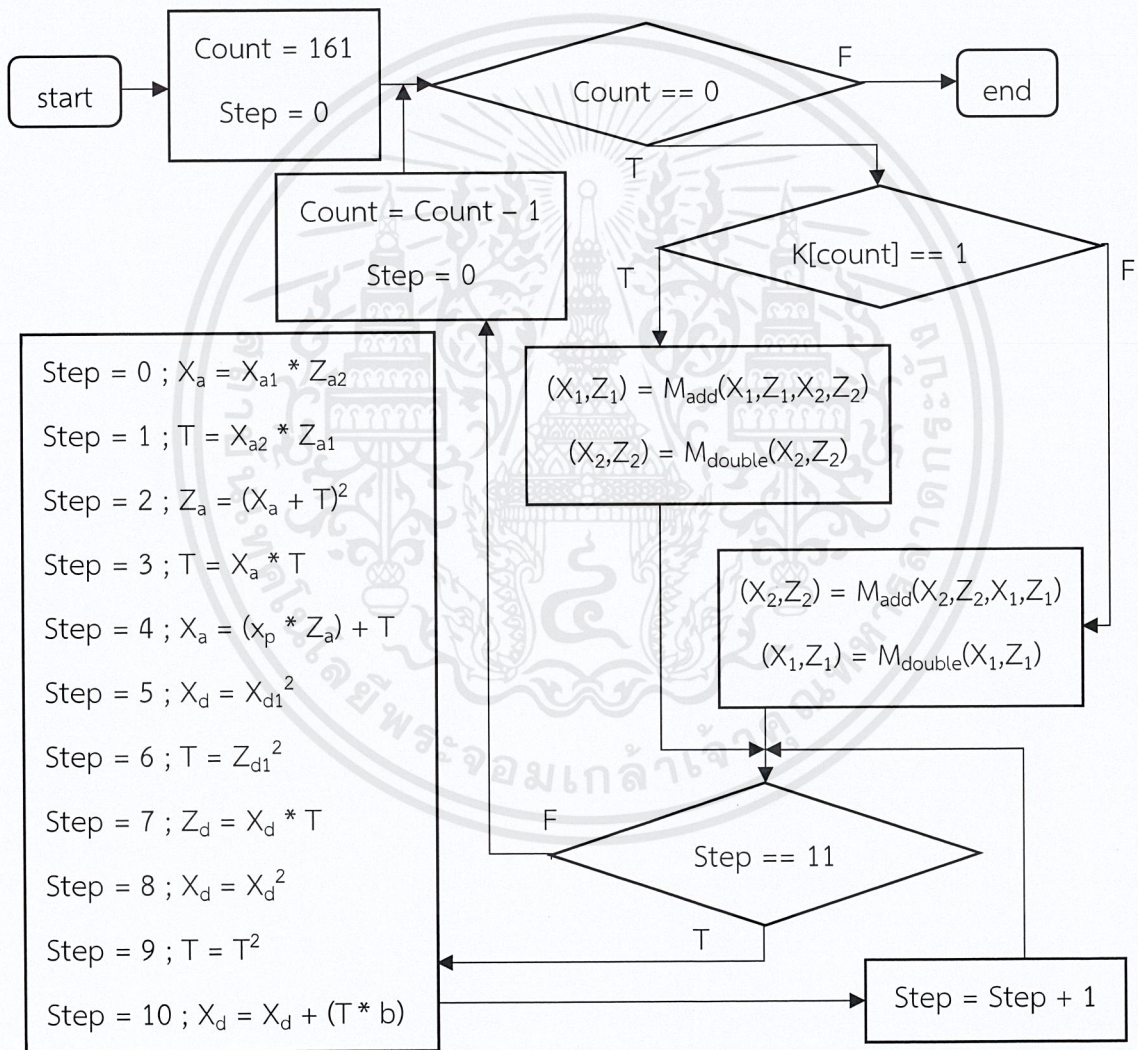


ภาพที่ 3.3 Flow Chart การทำงานของการ Inversion

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 และการยกกำลังสองจะทำการแทรก 0 ไประหว่างบิตของค่าแล้วนำมา mod  $f(x)$

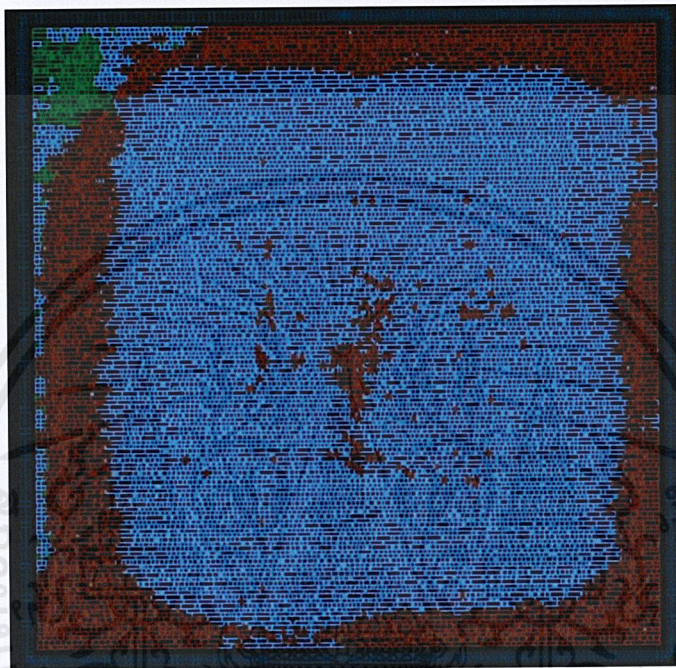
3.1.2.2 ช่วงการคำนวณตามกำลังของจุดฐาน โดยจะทำการนำค่าที่ตั้งค่าไว้ในช่วงแรกมาทำการคำนวณโดยการ Point addition แล้วตามด้วยการ Point doubling โดยการ + จะใช้การ XOR การยกกำลังสองจะทำการแทรก 0 ไประหว่างบิตของค่าแล้วนำมา mod  $f(x)$  และดังนี้



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 การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่แบบ 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

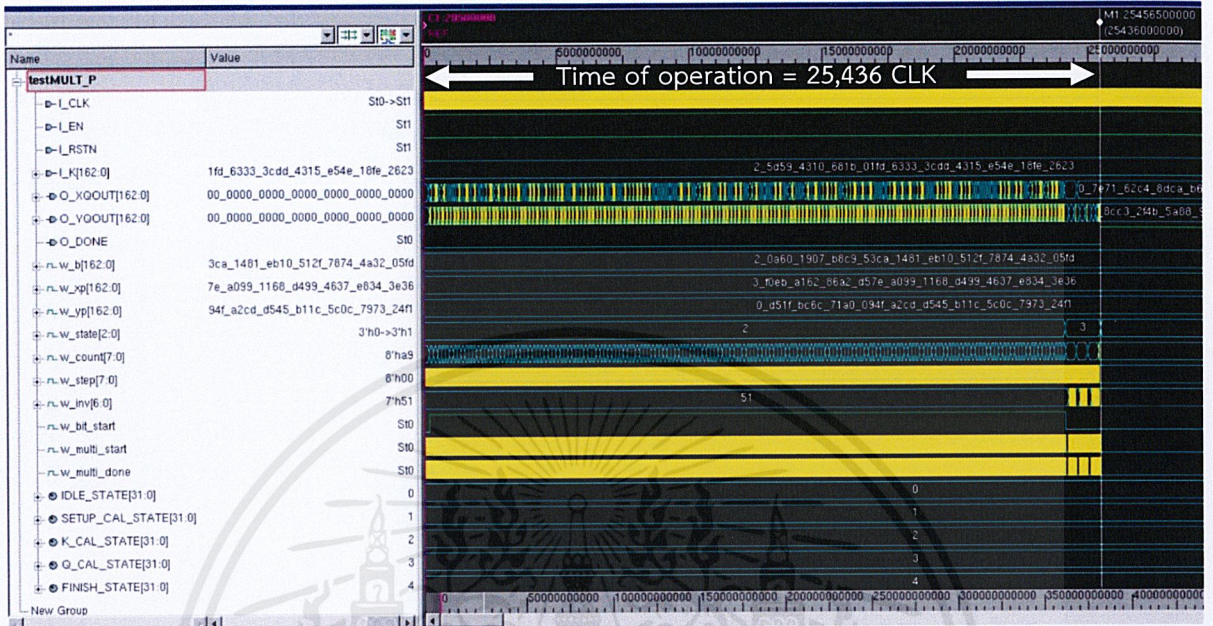
```
python verify.py
PASS >> I_K : 0x25d594310681b01fd63333cdd4315e54e18fe2623
PASS >> O_XQOUT : 0x07e7162c48dcab690aa9ef76d2ed066cedae33364
PASS >> O_YQOUT : 0x08cc32f4b5a88985c6e0c418e4abe988d5375371d
PASS >> I_K : 0x306a58722716e0013fc1b0400ad4a46b664d89288
PASS >> O_XQOUT : 0x269e6231a76ef19dfb51b2beb8d38f6a702b8fc16
PASS >> O_YQOUT : 0x2adc145f674f95c920962672aa00708a2c12f5461
PASS >> I_K : 0x04d6a11276237fbb1bd246fe7e6e1098d39b7cfe2
PASS >> O_XQOUT : 0x02f36f4d7e6b211bb93586b360ff84608d57e43e0
PASS >> O_YQOUT : 0x4a016289b33dc90dd97c1b63491c730cf456d9a16
PASS >> I_K : 0x2731e75563847c79674504c0dd1542aedf6536dd2
PASS >> O_XQOUT : 0x7166ea4ee46252358d53535d8bb51c1a6a4a0bf9
PASS >> O_YQOUT : 0x2854d98233b5b8c40db2862329e06a007027dc5d5
PASS >> I_K : 0x29a853fc117d1abcf84495e9136787fdf2c49d32d
PASS >> O_XQOUT : 0x6d5bea90d981377f8bb854834388ee71e10623c2b
PASS >> O_YQOUT : 0x4d754bacb6d40cc05acdbec83c1088b0e53961bfe
PASS >> I_K : 0x35c46e2607d17d8db84d7f7420141a43805e4a46c
PASS >> O_XQOUT : 0x50568c9c943e3d7239863950ae4c61dbc5b076cd7
PASS >> O_YQOUT : 0x7d0493c1e421ad2af13e1e97e18ee039168b2407e
PASS >> I_K : 0x27c588736162669dfc18bb7782b6c9f25fc995705
PASS >> O_XQOUT : 0x4ae976ef48717eb4b07ccc0b545b3e0fd0ee44b22
PASS >> O_YQOUT : 0x2aa634be4a57f1d783f6482c5224577a353c0c7a7
PASS >> I_K : 0x0156110c985db455e9740bcea7aed7a325707979b
PASS >> O_XQOUT : 0x1cd9ff77dde1712228b9e683526ccc1998cf11525
PASS >> O_YQOUT : 0x350aca0d4682c221704550b1da8f850403466d5f4
PASS >> I_K : 0x1c158eb2c071e9513ac7c864dbbc95394949978d5
PASS >> O_XQOUT : 0x46e9e9defed2a8301e4ab7929838f15f3ee4abb88
PASS >> O_YQOUT : 0x592743328b4dcae86ed77edc5a34902c1c7205d2a
PASS >> I_K : 0x21f43d47313dd20f1065113a4dc2ed4c607969c33
PASS >> O_XQOUT : 0x51ea3a8ef4e33eebfc2b0bb975fb6d38602c55a77
PASS >> O_YQOUT : 0x13108049c22f4126a1aee22f587d6452abc0e93d
PS D:\work\year 4-1\ECDSA\verify> |
```

ภาพที่ 4.1 ผลการเปรียบเทียบความถูกต้องของการคำนวณ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า  
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

#### 4.1.2 ข้อมูลจำเพาะของวงจร

- ระยะเวลาทั้งหมดในการทำการคำนวณ คือ 25,436 CLK



ภาพที่ 4.2 ระยะเวลาทั้งหมดในการทำการคำนวณ

- ขนาดทั้งหมดของวงจร คือ 26,122 GE หรือ  $261,222 \mu\text{m}^2$

ตารางที่ 4.1 แสดงขนาดของวงจรและโมดูลต่าง ๆ

Module	Area	
Multi_Point	26,122 GE	$261,222 \mu\text{m}^2$
Control_Path	627 GE	$6,265 \mu\text{m}^2$
Data_Path	25,496 GE	$254,957 \mu\text{m}^2$
Multiplication	6,593 GE	$65,928 \mu\text{m}^2$

- ระยะเวลาทั้งหมดในการทำการคำนวณของวงจรที่ใช้อ้างอิง คือ 41,764 CLK และขนาดทั้งหมดของวงจรของวงจรที่ใช้อ้างอิง คือ 34,582 GE หรือ  $324,185 \mu\text{m}^2$  ซึ่งวงจรที่ได้ทำการออกแบบมีทั้งขนาดเล็กกว่าและใช้เวลาทำงานน้อยกว่า

## บทที่ 5

### สรุปผลการวิจัยและข้อเสนอแนะ

ในการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 ผู้ดำเนินงานได้ทำการดำเนินงานวิจัย และนำผลการวิจัยที่ได้มาสรุปได้ดังนี้

#### 5.1 สรุปผลการวิจัย

จากที่ได้ทำการวิจัยเรื่อง การสร้างอุปกรณ์การทำงานของเส้นโค้งรูปไข่รูปแบบ B-163 โดยได้ทำการออกแบบวงจร และตรวจสอบวงจรที่ได้ทำการออกแบบนั้น พบว่า วงจรที่ได้ทำการออกแบบนั้นทำงานได้อย่างถูกต้องสมบูรณ์เมื่อนำไปทดสอบกับมาตรฐาน อีกทั้งวงจรยังมีระยะเวลาที่ใช้ในการคำนวณไม่มากนักเพียง 25,436 CLK และมีขนาดที่ค่อนข้างเล็ก คือ 26,122 GE หรือ  $261,222 \mu m^2$  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] ชีรยศ เวียงทอง (2548). “เรียนรู้การออกแบบระบบดิจิทัลด้วยภาษา 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  $GF(2^m)$  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  $GF(2^m)$  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

```
1 timescale 1ns / 1ps
2 module MULT_P(
3     input I_CLK,
4     input I_EN,
5     input I_RSTN,
6     input [162:0] I_K,
7     output [162:0] O_XQOUT,
8     output [162:0] O_YQOUT,
9     output O_DONE
10 );
11 parameter IDLE_STATE = 0;
12 parameter SETUP_CAL_STATE = 1;
13 parameter K_CAL_STATE = 2;
14 parameter Q_CAL_STATE = 3;
15 parameter FINISH_STATE = 4;
16
17 wire [162:0] w_b;
18 wire [162:0] w_xp;
19 wire [162:0] w_yp;
20 wire [2:0] w_state;
21 wire [7:0] w_count;
22 wire [7:0] w_step;
23 wire [6:0] w_inv;
24 wire w_bit_start;
25 wire w_multi_start;
26 wire w_multi_done;
27
28 assign w_b = 163'h20A60190788C9532A1481E10512F78744A2205FD;
29 assign w_xp = 163'h3F0EBA1628CA207EA05911604994639E8343E3C;
30 assign w_yp = 163'hD51FBC671A0094FA2CDD545B11C5C0C797324E1;
31
32 MULT_P_CONTROLPATH MULT_P_CONTROLPATH(
33     .I_CLK (I_CLK),
34     .I_EN (I_EN),
35     .I_RSTN (I_RSTN),
36     .I_K (I_K),
37     .I_MULT_DONE (w_multi_done),
38     .O_STATE (w_state),
39     .O_COUNT (w_count),
40     .O_STEP (w_step),
41     .O_INV (w_inv),
42     .O_BIT_START (w_bit_start),
43     .O_MULT_START (w_multi_start),
44     .O_DONE (O_DONE)
45 );
46
47 MULT_P_DATAPATH MULT_P_DATAPATH(
48     .I_CLK (I_CLK),
49     .I_RSTN (I_RSTN),
50     .I_STATE (w_state),
51     .I_COUNT (w_count),
52     .I_STEP (w_step),
53     .I_INV (w_inv),
54     .I_BIT_START (w_bit_start),
55     .I_MULT_START (w_multi_start),
56     .I_B (w_b),
57     .I_XP (w_xp),
58     .I_YP (w_yp),
59     .I_K (I_K),
60     .O_MULT_DONE (w_multi_done),
61     .O_XQOUT (O_XQOUT),
62     .O_YQOUT (O_YQOUT)
63 );
64 endmodule
```

## Code Module MULT\_P\_CONTROLPATH

```
1 timescale 1ns / 1ps
2 module MULT_P_CONTROLPATH(
3     input I_CLK,
4     input I_EN,
5     input I_RSTN,
6     input [162:0] I_K,
7     input I_MULT_DONE,
8     output [2:0] O_STATE,
9     output [7:0] O_COUNT,
10    output [7:0] O_STEP,
11    output [6:0] O_INV,
12    output O_BIT_START,
13    output O_MULT_START,
14    output O_DONE
15 );
16 parameter IDLE_STATE = 0;
17 parameter SETUP_CAL_STATE = 1;
18 parameter K_CAL_STATE = 2;
19 parameter Q_CAL_STATE = 3;
20 parameter FINISH_STATE = 4;
21
22 reg [2:0] r_state;
23 reg [7:0] r_count;
24 reg [7:0] r_step;
25 reg r_bit_start;
26 reg r_multi_start;
27 reg [6:0] r_inv;
28
29 assign O_STATE = r_state;
30 assign O_COUNT = r_count;
31 assign O_STEP = r_step;
32 assign O_INV = r_inv;
33 assign O_BIT_START = r_bit_start;
34 assign O_MULT_START = r_multi_start;
35 assign O_DONE = r_state[4];
36
37 //-----r_state
38 always@(posedge I_CLK or negedge I_RSTN) begin
39     if(!I_RSTN) begin
40         r_state <= IDLE_STATE;
41     end
42     else if(!I_EN) begin
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า  
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

43     r_state <= IDLE_STATE;
44 end
45 else if(r_state == IDLE_STATE) begin
46     r_state <= SETUP_CAL_STATE;
47 end
48 else if((r_state == SETUP_CAL_STATE) && (r_count == 168)) begin
49     r_state <= K_CAL_STATE;
50 end
51 else if((r_state == K_CAL_STATE) && (r_count == 6) && (r_step == 10) && I_MULT_DONE && r_multi_start) begin
52     r_state <= Q_CAL_STATE;
53 end
54 else if((r_state == Q_CAL_STATE) && (r_count == 0) && I_MULT_DONE && r_multi_start) begin
55     r_state <= FINISH_STATE;
56 end
57 end
58 //-----r_count-----
59 always@(posedge I_CLK or negedge I_RSTN) begin
60     if(!I_RSTN) begin
61         r_count <= 168;
62     end
63     else if(r_state == IDLE_STATE) begin
64         r_count <= 168;
65     end
66     else if((r_state == SETUP_CAL_STATE) || (r_state == K_CAL_STATE) || (r_state == Q_CAL_STATE)) begin
67         if(r_count >= 168) begin
68             r_count <= r_count - 1;
69         end
70         else if((r_step == 10) && I_MULT_DONE && r_multi_start && (r_count <= 167) && (r_count >= 6)) begin
71             r_count <= r_count - 1;
72         end
73         else if((r_inv == 0) && (r_count <= 5) && (r_count >= 3)) begin
74             r_count <= r_count - 1;
75         end
76         else if(I_MULT_DONE && r_multi_start && (r_count <= 3) && (r_count != 0)) begin
77             r_count <= r_count - 1;
78         end
79     end
80 end
81 //-----r_step-----
82 always@(posedge I_CLK or negedge I_RSTN) begin
83     if(!I_RSTN) begin
84         r_step <= 0;
85     end
86     else if(r_state == IDLE_STATE) begin
87         r_step <= 0;
88     end
89     else if(r_state == K_CAL_STATE) begin
90         if((r_step == 10) && I_MULT_DONE && r_multi_start) begin
91             r_step <= 0;
92         end
93         else if((r_step != 10) && I_MULT_DONE && r_multi_start) begin
94             r_step <= r_step + 1;
95         end
96         else if((r_step == 3) || (r_step == 5) || (r_step == 7) || (r_step == 9) || (r_step == 11)) begin
97             r_step <= r_step + 1;
98         end
99     end
100     else if(r_state == Q_CAL_STATE) begin
101         if((r_count <= 1) && (r_count >= 0)) begin
102             if(r_inv != 0) begin
103                 if(r_step <= r_inv) begin
104                     r_step <= r_step + 1;
105                 end
106                 else if(I_MULT_DONE && r_multi_start && (r_step == r_inv + 1)) begin
107                     r_step <= r_step + 1;
108                 end
109                 else if((r_step == r_inv + 1) && ~r_inv) begin
110                     r_step <= 1;
111                 end
112             end
113             else if(I_MULT_DONE && r_multi_start && (r_step == r_inv + 1) && r_inv) begin
114                 r_step <= 1;
115             end
116         end
117         else begin
118             r_step <= 1;
119         end
120     end
121 end
122 //-----r_bit_start-----
123 always@(posedge I_CLK or negedge I_RSTN) begin
124     if(!I_RSTN) begin
125         r_bit_start <= 0;
126     end
127     else if(r_state == IDLE_STATE) begin
128         r_bit_start <= 0;
129     end
130     else if(r_state == SETUP_CAL_STATE) begin
131         if((r_count-1) && (r_count == 168)) begin
132             r_bit_start <= 1;
133         end
134     end
135     else if(r_state == K_CAL_STATE) begin
136         if((r_count == 1) && (r_step == 1) && I_MULT_DONE && r_multi_start) begin
137             r_bit_start <= 0;
138         end
139         else if(!K[r_count-1] && (r_step == 10) && I_MULT_DONE && r_multi_start) begin
140             r_bit_start <= 1;
141         end
142     end
143 end
144 //-----r_multi_start-----
145 always@(posedge I_CLK or negedge I_RSTN) begin
146     if(!I_RSTN) begin
147         r_multi_start <= 0;
148     end
149     else if(r_state == IDLE_STATE) begin
150         r_multi_start <= 0;
151     end
152     else if(r_state == K_CAL_STATE) begin
153         if(I_MULT_DONE && r_multi_start) begin
154             r_multi_start <= 0;
155         end

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า  
 ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

156     else if(r_step != 2) && (r_step != 3) && (r_step != 4) && (r_step != 5) && (r_step != 6) && (r_step != 7) && (r_step != 8) && (r_step != 9) begin
157         r_multi_start <= 1;
158     end
159 end
160 else if(r_state == Q_CAL_STATE) begin
161     if(r_count <= 0) && (r_count >= 3) begin
162         if(r_inv != 0) begin
163             if(!MULT_DONE && (r_step == 0) && (r_count != 3)) begin
164                 r_multi_start <= 1;
165             end
166             else if(!MULT_DONE && (r_step == r_inv + 1)) begin
167                 r_multi_start <= 1;
168             end
169             else if(!MULT_DONE && (r_step == r_inv + 2) && r_inv[1]) begin
170                 r_multi_start <= 1;
171             end
172             else if(!MULT_DONE) begin
173                 r_multi_start <= 0;
174             end
175         end
176     else begin
177         r_multi_start <= 0;
178     end
179 end
180 else begin
181     if(!MULT_DONE && r_multi_start) begin
182         r_multi_start <= 0;
183     end
184     else begin
185         r_multi_start <= 1;
186     end
187 end
188 end
189 end
190 //-----r_inv-----
191 always@(posedge I_CLK or negedge I_RSTN) begin
192     if(!I_RSTN) begin
193         r_inv <= 0;
194     end
195     else if(r_state == IDLE_STATE) begin
196         r_inv <= 0;
197     end
198     else if(r_state == Q_CAL_STATE) begin
199         if(r_count <= 2) && (r_count >= 3) begin
200             if(r_inv != 0) begin
201                 if(!r_inv[1] && (r_step == r_inv+1)) begin
202                     r_inv <= r_inv >> 1;
203                 end
204                 else if(!MULT_DONE && r_multi_start && (r_step == r_inv+1)) begin
205                     r_inv <= r_inv >> 1;
206                 end
207             end
208             else begin
209                 r_inv <= 0;
210             end
211         end
212     end
213 end
214 endmodule

```

## Code Module MULT\_P\_DATAPATH

```

1 timescale 1ns / 1fs
2 module MULT_P_DATAPATH(
3     input I_CLK,
4     input I_RSTN,
5     input [2:0] I_STATE,
6     input [7:0] I_COUNT,
7     input [7:0] I_STEP,
8     input [6:0] I_INV,
9     input I_RIP_START,
10    input I_MULT_START,
11    input [162:0] I_B,
12    input [162:0] I_XP,
13    input [162:0] I_YP,
14    input [162:0] I_X,
15    output O_MULT_DONE,
16    output [162:0] O_XQOUT,
17    output [162:0] O_YQOUT
18);
19 parameter IDLE_STATE = 0;
20 parameter SETUP_CAL_STATE = 1;
21 parameter K_CAL_STATE = 2;
22 parameter Q_CAL_STATE = 3;
23 parameter FINISH_STATE = 4;
24
25 reg [162:0] r_x1;
26 reg [162:0] r_x2;
27 reg [162:0] r_z1;
28 reg [162:0] r_z2;
29 reg [162:0] r_xp_sq;
30 reg [162:0] r_xp_fourth;
31 reg [162:0] r_ybuff;
32
33 wire [162:0] w_sq;
34 wire [162:0] w_dsqr;
35 wire [162:0] w_multi_din1;
36 wire [162:0] w_multi_din2;
37 wire [162:0] w_multi_dout;
38
39 assign w_dsqr = {w_sq[0]^w_sq[161]^w_sq[159],w_sq[162]^w_sq[158],w_sq[160]^w_sq[157],w_sq[161]^w_sq[156],w_sq[162]^w_sq[155],w_sq[160]^w_sq[154],w_sq[161]^w_sq[153],w_sq[162]^w_sq[152],w_sq[160]^w_sq[151],w_sq[161]^w_sq[150],w_sq[162]^w_sq[149],w_sq[160]^w_sq[148],w_sq[161]^w_sq[147],w_sq[162]^w_sq[146],w_sq[160]^w_sq[145],w_sq[161]^w_sq[144],w_sq[162]^w_sq[143],w_sq[160]^w_sq[142],w_sq[161]^w_sq[141],w_sq[162]^w_sq[140],w_sq[160]^w_sq[139],w_sq[161]^w_sq[138],w_sq[162]^w_sq[137],w_sq[160]^w_sq[136],w_sq[161]^w_sq[135],w_sq[162]^w_sq[134],w_sq[160]^w_sq[133],w_sq[161]^w_sq[132],w_sq[162]^w_sq[131],w_sq[160]^w_sq[130],w_sq[161]^w_sq[129],w_sq[162]^w_sq[128],w_sq[160]^w_sq[127],w_sq[161]^w_sq[126],w_sq[162]^w_sq[125],w_sq[160]^w_sq[124],w_sq[161]^w_sq[123],w_sq[162]^w_sq[122],w_sq[160]^w_sq[121],w_sq[161]^w_sq[120],w_sq[162]^w_sq[119],w_sq[160]^w_sq[118],w_sq[161]^w_sq[117],w_sq[162]^w_sq[116],w_sq[160]^w_sq[115],w_sq[161]^w_sq[114],w_sq[162]^w_sq[113],w_sq[160]^w_sq[112],w_sq[161]^w_sq[111],w_sq[162]^w_sq[110],w_sq[160]^w_sq[109],w_sq[161]^w_sq[108],w_sq[162]^w_sq[107],w_sq[160]^w_sq[106],w_sq[161]^w_sq[105],w_sq[162]^w_sq[104],w_sq[160]^w_sq[103],w_sq[161]^w_sq[102],w_sq[162]^w_sq[101],w_sq[160]^w_sq[100],w_sq[161]^w_sq[99],w_sq[162]^w_sq[98],w_sq[160]^w_sq[97],w_sq[161]^w_sq[96],w_sq[162]^w_sq[95],w_sq[160]^w_sq[94],w_sq[161]^w_sq[93],w_sq[162]^w_sq[92],w_sq[160]^w_sq[91],w_sq[161]^w_sq[90],w_sq[162]^w_sq[89],w_sq[160]^w_sq[88],w_sq[161]^w_sq[87],w_sq[162]^w_sq[86],w_sq[160]^w_sq[85],w_sq[161]^w_sq[84],w_sq[162]^w_sq[83],w_sq[160]^w_sq[82],w_sq[161]^w_sq[81],w_sq[162]^w_sq[80],w_sq[160]^w_sq[79],w_sq[161]^w_sq[78],w_sq[162]^w_sq[77],w_sq[160]^w_sq[76],w_sq[161]^w_sq[75],w_sq[162]^w_sq[74],w_sq[160]^w_sq[73],w_sq[161]^w_sq[72],w_sq[162]^w_sq[71],w_sq[160]^w_sq[70],w_sq[161]^w_sq[69],w_sq[162]^w_sq[68],w_sq[160]^w_sq[67],w_sq[161]^w_sq[66],w_sq[162]^w_sq[65],w_sq[160]^w_sq[64],w_sq[161]^w_sq[63],w_sq[162]^w_sq[62],w_sq[160]^w_sq[61],w_sq[161]^w_sq[60],w_sq[162]^w_sq[59],w_sq[160]^w_sq[58],w_sq[161]^w_sq[57],w_sq[162]^w_sq[56],w_sq[160]^w_sq[55],w_sq[161]^w_sq[54],w_sq[162]^w_sq[53],w_sq[160]^w_sq[52],w_sq[161]^w_sq[51],w_sq[162]^w_sq[50],w_sq[160]^w_sq[49],w_sq[161]^w_sq[48],w_sq[162]^w_sq[47],w_sq[160]^w_sq[46],w_sq[161]^w_sq[45],w_sq[162]^w_sq[44],w_sq[160]^w_sq[43],w_sq[161]^w_sq[42],w_sq[162]^w_sq[41],w_sq[160]^w_sq[40],w_sq[161]^w_sq[39],w_sq[162]^w_sq[38],w_sq[160]^w_sq[37],w_sq[161]^w_sq[36],w_sq[162]^w_sq[35],w_sq[160]^w_sq[34],w_sq[161]^w_sq[33],w_sq[162]^w_sq[32],w_sq[160]^w_sq[31],w_sq[161]^w_sq[30],w_sq[162]^w_sq[29],w_sq[160]^w_sq[28],w_sq[161]^w_sq[27],w_sq[162]^w_sq[26],w_sq[160]^w_sq[25],w_sq[161]^w_sq[24],w_sq[162]^w_sq[23],w_sq[160]^w_sq[22],w_sq[161]^w_sq[21],w_sq[162]^w_sq[20],w_sq[160]^w_sq[19],w_sq[161]^w_sq[18],w_sq[162]^w_sq[17],w_sq[160]^w_sq[16],w_sq[161]^w_sq[15],w_sq[162]^w_sq[14],w_sq[160]^w_sq[13],w_sq[161]^w_sq[12],w_sq[162]^w_sq[11],w_sq[160]^w_sq[10],w_sq[161]^w_sq[9],w_sq[162]^w_sq[8],w_sq[160]^w_sq[7],w_sq[161]^w_sq[6],w_sq[162]^w_sq[5],w_sq[160]^w_sq[4],w_sq[161]^w_sq[3],w_sq[162]^w_sq[2],w_sq[160]^w_sq[1],w_sq[161]^w_sq[0]};
40

```



```

157     r_xp_sqf <= r_r2;
158     end
159     else if((i_COUNT == 3) && (i_STEP == 0)) begin
160         r_xp_sqf <= r_xp_fourth;
161     end
162     else if(i_COUNT == 2) begin
163         r_xp_sqf <= w_dsqr;
164     end
165     end
166 end
167 //-----r_xp_fourth-----
168 always@(posedge i_CLK or negedge i_RSTN) begin
169     if(!i_RSTN) begin
170         r_xp_fourth <= 0;
171     end
172     else if(i_STATE == IDLE_STATE) begin
173         r_xp_fourth <= 0;
174     end
175     else if(i_STATE == SETUP_CAL_STATE) begin
176         if(i_COUNT == 166) begin
177             r_xp_fourth <= w_dsqr;
178         end
179     end
180     else if(i_STATE == Q_CAL_STATE) begin
181         if((i_COUNT == 4) && (i_INV == 0)) begin
182             r_xp_fourth <= i_XP;
183         end
184         if((i_COUNT == 3) && (i_INV == 0)) begin
185             r_xp_fourth <= r_ybuff;
186         end
187         else if((i_COUNT == 3) && (i_STEP <= i_INV)) begin
188             r_xp_fourth <= w_dsqr;
189         end
190         else if((i_COUNT == 3) && (i_STEP == i_INV + 1) && O_MULT_DONE && i_MULT_START) begin
191             r_xp_fourth <= w_multi_dout;
192         end
193         else if((i_COUNT == 3) && (i_STEP == i_INV + 2) && i_INV[0] && O_MULT_DONE && i_MULT_START) begin
194             r_xp_fourth <= w_dsqr;
195         end
196     end
197 end
198 //-----r_x1-----
199 always@(posedge i_CLK or negedge i_RSTN) begin
200     if(!i_RSTN) begin
201         r_x1 <= 0;
202     end
203     else if(i_STATE == IDLE_STATE) begin
204         r_x1 <= 0;
205     end
206     else if(i_STATE == SETUP_CAL_STATE) begin
207         if(i_R[i_COUNT-1] && (i_COUNT == 166)) begin
208             r_x1 <= i_XP;
209         end
210     end
211     else if(i_STATE == R_CAL_STATE) begin
212         if(i_R[i_COUNT-1] && (i_STEP == 99) && O_MULT_DONE && i_MULT_START && ~i_BIT_START) begin
213             r_x1 <= i_XP;
214         end
215         else if(i_R[i_COUNT-1] && (i_STEP == 0) && O_MULT_DONE && i_MULT_START && i_BIT_START) begin
216             r_x1 <= w_multi_dout;
217         end
218         else if(i_R[i_COUNT-1] && (i_STEP == 0) && O_MULT_DONE && i_MULT_START && i_BIT_START) begin
219             r_x1 <= w_multi_dout ^ r_ybuff;
220         end
221         else if(~i_R[i_COUNT-1] && (i_STEP == 0) && i_BIT_START) begin
222             r_x1 <= w_dsqr;
223         end
224         else if(~i_R[i_COUNT-1] && (i_STEP == 0) && i_BIT_START) begin
225             r_x1 <= w_dsqr;
226         end
227         else if(~i_R[i_COUNT-1] && (i_STEP == 1) && O_MULT_DONE && i_MULT_START && i_BIT_START) begin
228             r_x1 <= r_x1 ^ w_multi_dout;
229         end
230     end
231     else if(i_STATE == Q_CAL_STATE) begin
232         if((i_COUNT == 3) && (i_STEP <= i_INV) && O_MULT_DONE && i_MULT_START) begin
233             r_x1 <= w_multi_dout;
234         end
235     end
236 end
237 //-----r_x2-----
238 always@(posedge i_CLK or negedge i_RSTN) begin
239     if(!i_RSTN) begin
240         r_x2 <= 0;
241     end
242     else if(i_STATE == IDLE_STATE) begin
243         r_x2 <= 0;
244     end
245     else if(i_STATE == SETUP_CAL_STATE) begin
246         if(i_R[i_COUNT-1] && (i_COUNT == 166)) begin
247             r_x2 <= w_dsqr ^ i_B;
248         end
249     end
250     else if(i_STATE == R_CAL_STATE) begin
251         if(i_R[i_COUNT-1] && (i_STEP == 1) && O_MULT_DONE && i_MULT_START && ~i_BIT_START) begin
252             r_x2 <= r_xp_fourth ^ i_B;
253         end
254         else if(i_R[i_COUNT-1] && (i_STEP == 0) && i_BIT_START) begin
255             r_x2 <= w_dsqr;
256         end
257         else if(i_R[i_COUNT-1] && (i_STEP == 0) && i_BIT_START) begin
258             r_x2 <= w_dsqr;
259         end
260         else if(i_R[i_COUNT-1] && (i_STEP == 10) && O_MULT_DONE && i_MULT_START && i_BIT_START) begin
261             r_x2 <= r_x2 ^ w_multi_dout;
262         end
263         else if(~i_R[i_COUNT-1] && (i_STEP == 0) && O_MULT_DONE && i_MULT_START && i_BIT_START) begin
264             r_x2 <= w_multi_dout;
265         end
266         else if(~i_R[i_COUNT-1] && (i_STEP == 4) && O_MULT_DONE && i_MULT_START && i_BIT_START) begin
267             r_x2 <= w_multi_dout ^ r_ybuff;
268         end
269     end
270     else if(i_STATE == Q_CAL_STATE) begin
271         if((i_COUNT == 3) && (i_STEP <= i_INV) && O_MULT_DONE && i_MULT_START) begin
272             r_x2 <= w_multi_dout;
273         end

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า  
 ไม่ว่าจะกรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

274     end
275     //-----r_z1-----
276     always@(posedge I_CLK or negedge I_RSTN) begin
277         if(!I_RSTN) begin
278             r_z1 <= 0;
279         end
280         else if(I_STATE == IDLE_STATE) begin
281             r_z1 <= 0;
282         end
283         else if(I_STATE == SETUP_CAL_STATE) begin
284             if(I_K[I_COUNT-1] && (I_COUNT == 168)) begin
285                 r_z1 <= 168'd1;
286             end
287         end
288         else if(I_STATE == K_CAL_STATE) begin
289             if(I_K[I_COUNT-1] && (I_STEP == 10) && O_MULT_DONE && I_MULT_START && ~I_BIT_START) begin
290                 r_z1 <= 168'd1;
291             end
292             else if(I_K[I_COUNT-1] && (I_STEP == 2) && I_BIT_START) begin
293                 r_z1 <= w_dsqr;
294             end
295             else if(~I_K[I_COUNT-1] && (I_STEP == 1) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
296                 r_z1 <= w_multi_dout;
297             end
298         end
299         else if(I_STATE == Q_CAL_STATE) begin
300             if((I_COUNT == 3) && (I_INV == 3)) begin
301                 r_z1 <= r_ybuff;
302             end
303             else if((I_COUNT == 5) && (I_STEP <= I_INV)) begin
304                 r_z1 <= w_dsqr;
305             end
306             else if((I_COUNT == 5) && (I_STEP == I_INV + 1) && O_MULT_DONE && I_MULT_START) begin
307                 r_z1 <= w_multi_dout;
308             end
309             else if((I_COUNT == 5) && (I_STEP == I_INV + 2) && I_INV[1] && O_MULT_DONE && I_MULT_START) begin
310                 r_z1 <= w_dsqr;
311             end
312         end
313     end
314     //-----r_z2-----
315     always@(posedge I_CLK or negedge I_RSTN) begin
316         if(!I_RSTN) begin
317             r_z2 <= 0;
318         end
319         else if(I_STATE == IDLE_STATE) begin
320             r_z2 <= 0;
321         end
322         else if(I_STATE == SETUP_CAL_STATE) begin
323             if(I_K[I_COUNT-1] && (I_COUNT == 168)) begin
324                 r_z2 <= r_xp_sqr;
325             end
326         end
327         else if(I_STATE == K_CAL_STATE) begin
328             if(I_K[I_COUNT-1] && (I_STEP == 10) && O_MULT_DONE && I_MULT_START && ~I_BIT_START) begin
329                 r_z2 <= r_xp_sqr;
330             end
331             else if(I_K[I_COUNT-1] && (I_STEP == 1) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
332                 r_z2 <= w_multi_dout;
333             end
334             else if(~I_K[I_COUNT-1] && (I_STEP == 1) && I_BIT_START) begin
335                 r_z2 <= w_dsqr;
336             end
337         end
338         else if(I_STATE == Q_CAL_STATE) begin
339             if((I_COUNT == 3) && (I_INV == 1)) begin
340                 r_z2 <= r_ybuff;
341             end
342             else if((I_COUNT == 4) && (I_STEP <= I_INV)) begin
343                 r_z2 <= w_dsqr;
344             end
345             else if((I_COUNT == 4) && (I_STEP == I_INV + 1) && O_MULT_DONE && I_MULT_START) begin
346                 r_z2 <= w_multi_dout;
347             end
348             else if((I_COUNT == 4) && (I_STEP == I_INV + 2) && I_INV[1] && O_MULT_DONE && I_MULT_START) begin
349                 r_z2 <= w_dsqr;
350             end
351         end
352     end
353     //-----r_ybuff-----
354     always@(posedge I_CLK or negedge I_RSTN) begin
355         if(!I_RSTN) begin
356             r_ybuff <= 0;
357         end
358         else if(I_STATE == IDLE_STATE) begin
359             r_ybuff <= 0;
360         end
361         else if(I_STATE == K_CAL_STATE) begin
362             if((I_STEP == 1) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
363                 r_ybuff <= w_multi_dout;
364             end
365             else if((I_STEP == 3) && O_MULT_DONE && I_MULT_START && I_BIT_START) begin
366                 r_ybuff <= w_multi_dout;
367             end
368             else if((I_STEP == 4) && I_BIT_START) begin
369                 r_ybuff <= w_dsqr;
370             end
371             else if((I_STEP == 5) && I_BIT_START) begin
372                 r_ybuff <= w_dsqr;
373             end
374         end
375         else if(I_STATE == Q_CAL_STATE) begin
376             if((I_COUNT <= 1) && (I_COUNT >= 3) && (I_STEP == 0)) begin
377                 r_ybuff <= 1;
378             end
379             else if((I_COUNT <= 5) && (I_COUNT >= 3) && (I_STEP == I_INV + 0) && I_INV[1] && O_MULT_DONE && I_MULT_START) begin
380                 r_ybuff <= w_multi_dout;
381             end
382             else if((I_COUNT <= 1) && (I_COUNT >= 3) && O_MULT_DONE && I_MULT_START) begin
383                 r_ybuff <= w_multi_dout;
384             end
385             else if((I_COUNT == 0) && O_MULT_DONE && I_MULT_START) begin
386                 if(I_K == 0) begin
387                     r_ybuff <= 0;
388                 end
389                 else if(I_K == 1) begin
390                     r_ybuff <= I_YF;
391                 end
392             end
393             else begin
394                 r_ybuff <= w_multi_dout ^ I_YF;
395             end
396         end
397     end
398 end
399 endmodule

```

# Code Module MULT

```

1 timescale 1ns / 1p
2 module MULT(
3     input      I_CLK,
4     input      I_EN,
5     input      I_RSTN,
6     input [162:0] I_DIN1,
7     input [162:0] I_DIN2,
8     output [162:0] O_DOUT,
9     output      O_DONE
10 );
11 parameter IDLE_STATE = 0;
12 parameter CAL_STATE  = 1;
13 parameter FINISH_STATE = 2;
14
15 reg [1:0] r_state;
16 reg [8:0] r_count;
17 reg [162:0] r_a;
18 reg [162:0] r_c;
19
20 wire [162:0] w_b;
21 wire [162:0] w_b0;
22 wire [162:0] w_b1;
23 wire [162:0] w_b2;
24 wire [162:0] w_b3;
25 wire [162:0] w_b4;
26 wire [162:0] w_b5;
27 wire [162:0] w_b6;
28 wire [162:0] w_b7;
29
30 assign w_b = {w_b0,I_DIN2};
31 assign w_b0 = (w_b[r_count]) ? 163'h"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
32 assign w_b1 = (w_b[r_count+1]) ? 163'h"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
33 assign w_b2 = (w_b[r_count+2]) ? 163'h"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
34 assign w_b3 = (w_b[r_count+3]) ? 163'h"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
35 assign w_b4 = (w_b[r_count+4]) ? 163'h"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
36 assign w_b5 = (w_b[r_count+5]) ? 163'h"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
37 assign w_b6 = (w_b[r_count+6]) ? 163'h"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
38 assign w_b7 = (w_b[r_count+7]) ? 163'h"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF : 163'h0;
39 assign O_DOUT = r_c[162:0] * (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[165]*r_c[164], r_c[164]*r_c[163],
40 r_c[163]*r_c[162], r_c[162]*r_c[161], r_c[161]*r_c[160], r_c[160]*r_c[159], r_c[159]*r_c[158], r_c[158]*r_c[157], r_c[157]*r_c[156],
41 r_c[156]*r_c[155], r_c[155]*r_c[154], r_c[154]*r_c[153], r_c[153]*r_c[152], r_c[152]*r_c[151], r_c[151]*r_c[150], r_c[150]*r_c[149],
42 r_c[149]*r_c[148], r_c[148]*r_c[147], r_c[147]*r_c[146], r_c[146]*r_c[145], r_c[145]*r_c[144], r_c[144]*r_c[143], r_c[143]*r_c[142],
43 r_c[142]*r_c[141], r_c[141]*r_c[140], r_c[140]*r_c[139], r_c[139]*r_c[138], r_c[138]*r_c[137], r_c[137]*r_c[136], r_c[136]*r_c[135],
44 r_c[135]*r_c[134], r_c[134]*r_c[133], r_c[133]*r_c[132], r_c[132]*r_c[131], r_c[131]*r_c[130], r_c[130]*r_c[129], r_c[129]*r_c[128],
45 r_c[128]*r_c[127], r_c[127]*r_c[126], r_c[126]*r_c[125], r_c[125]*r_c[124], r_c[124]*r_c[123], r_c[123]*r_c[122], r_c[122]*r_c[121],
46 r_c[121]*r_c[120], r_c[120]*r_c[119], r_c[119]*r_c[118], r_c[118]*r_c[117], r_c[117]*r_c[116], r_c[116]*r_c[115], r_c[115]*r_c[114],
47 r_c[114]*r_c[113], r_c[113]*r_c[112], r_c[112]*r_c[111], r_c[111]*r_c[110], r_c[110]*r_c[109], r_c[109]*r_c[108], r_c[108]*r_c[107],
48 r_c[107]*r_c[106], r_c[106]*r_c[105], r_c[105]*r_c[104], r_c[104]*r_c[103], r_c[103]*r_c[102], r_c[102]*r_c[101], r_c[101]*r_c[100],
49 r_c[100]*r_c[99], r_c[99]*r_c[98], r_c[98]*r_c[97], r_c[97]*r_c[96], r_c[96]*r_c[95], r_c[95]*r_c[94], r_c[94]*r_c[93], r_c[93]*r_c[92],
50 r_c[92]*r_c[91], r_c[91]*r_c[90], r_c[90]*r_c[89], r_c[89]*r_c[88], r_c[88]*r_c[87], r_c[87]*r_c[86], r_c[86]*r_c[85], r_c[85]*r_c[84],
51 r_c[84]*r_c[83], r_c[83]*r_c[82], r_c[82]*r_c[81], r_c[81]*r_c[80], r_c[80]*r_c[79], r_c[79]*r_c[78], r_c[78]*r_c[77], r_c[77]*r_c[76],
52 r_c[76]*r_c[75], r_c[75]*r_c[74], r_c[74]*r_c[73], r_c[73]*r_c[72], r_c[72]*r_c[71], r_c[71]*r_c[70], r_c[70]*r_c[69], r_c[69]*r_c[68],
53 r_c[68]*r_c[67], r_c[67]*r_c[66], r_c[66]*r_c[65], r_c[65]*r_c[64], r_c[64]*r_c[63], r_c[63]*r_c[62], r_c[62]*r_c[61], r_c[61]*r_c[60],
54 r_c[60]*r_c[59], r_c[59]*r_c[58], r_c[58]*r_c[57], r_c[57]*r_c[56], r_c[56]*r_c[55], r_c[55]*r_c[54], r_c[54]*r_c[53], r_c[53]*r_c[52],
55 r_c[52]*r_c[51], r_c[51]*r_c[50], r_c[50]*r_c[49], r_c[49]*r_c[48], r_c[48]*r_c[47], r_c[47]*r_c[46], r_c[46]*r_c[45], r_c[45]*r_c[44],
56 r_c[44]*r_c[43], r_c[43]*r_c[42], r_c[42]*r_c[41], r_c[41]*r_c[40], r_c[40]*r_c[39], r_c[39]*r_c[38], r_c[38]*r_c[37], r_c[37]*r_c[36],
57 r_c[36]*r_c[35], r_c[35]*r_c[34], r_c[34]*r_c[33], r_c[33]*r_c[32], r_c[32]*r_c[31], r_c[31]*r_c[30], r_c[30]*r_c[29], r_c[29]*r_c[28],
58 r_c[28]*r_c[27], r_c[27]*r_c[26], r_c[26]*r_c[25], r_c[25]*r_c[24], r_c[24]*r_c[23], r_c[23]*r_c[22], r_c[22]*r_c[21], r_c[21]*r_c[20],
59 r_c[20]*r_c[19], r_c[19]*r_c[18], r_c[18]*r_c[17], r_c[17]*r_c[16], r_c[16]*r_c[15], r_c[15]*r_c[14], r_c[14]*r_c[13], r_c[13]*r_c[12],
60 r_c[12]*r_c[11], r_c[11]*r_c[10], r_c[10]*r_c[9], r_c[9]*r_c[8], r_c[8]*r_c[7], r_c[7]*r_c[6], r_c[6]*r_c[5], r_c[5]*r_c[4], r_c[4]*r_c[3],
61 r_c[3]*r_c[2], r_c[2]*r_c[1], r_c[1]*r_c[0]);
62
63 assign O_DONE = r_state[1];
64
65 //-----r_state-----
66 always@(posedge I_CLK or negedge I_RSTN) begin
67     if(!I_RSTN) begin
68         r_state <= IDLE_STATE;
69     end
70     else if(!I_EN) begin
71         r_state <= IDLE_STATE;
72     end
73     else if(r_state == IDLE_STATE) begin
74         r_state <= CAL_STATE;
75     end
76     else if(r_count == 160) begin
77         r_state <= FINISH_STATE;
78     end
79 end
80
81 //-----r_count-----
82 always@(posedge I_CLK or negedge I_RSTN) begin
83     if(!I_RSTN) begin
84         r_count <= 0;
85     end
86     else if(r_state == IDLE_STATE) begin
87         r_count <= 0;
88     end
89     else if(r_state == CAL_STATE) begin
90         if(r_count < 160) begin
91             r_count <= r_count + 8;
92         end
93     end
94 end
95
96 //-----r_a-----
97 always@(posedge I_CLK or negedge I_RSTN) begin
98     if(!I_RSTN) begin
99         r_a <= 0;
100    end
101    else if(r_state == IDLE_STATE) begin
102        r_a <= I_DIN1;
103    end
104    else if(r_state == CAL_STATE) begin
105        r_a <= (r_a << 1) * (r_a[161], r_a[160]*r_a[159], r_a[159]*r_a[158], r_a[158]*r_a[157], r_a[157]*r_a[156], r_a[156]*r_a[155], r_a[155]*r_a[154],
106        r_a[154]*r_a[153], r_a[153]*r_a[152], r_a[152]*r_a[151], r_a[151]*r_a[150], r_a[150]*r_a[149], r_a[149]*r_a[148], r_a[148]*r_a[147], r_a[147]*r_a[146],
107        r_a[146]*r_a[145], r_a[145]*r_a[144], r_a[144]*r_a[143], r_a[143]*r_a[142], r_a[142]*r_a[141], r_a[141]*r_a[140], r_a[140]*r_a[139], r_a[139]*r_a[138],
108        r_a[138]*r_a[137], r_a[137]*r_a[136], r_a[136]*r_a[135], r_a[135]*r_a[134], r_a[134]*r_a[133], r_a[133]*r_a[132], r_a[132]*r_a[131], r_a[131]*r_a[130],
109        r_a[130]*r_a[129], r_a[129]*r_a[128], r_a[128]*r_a[127], r_a[127]*r_a[126], r_a[126]*r_a[125], r_a[125]*r_a[124], r_a[124]*r_a[123], r_a[123]*r_a[122],
110        r_a[122]*r_a[121], r_a[121]*r_a[120], r_a[120]*r_a[119], r_a[119]*r_a[118], r_a[118]*r_a[117], r_a[117]*r_a[116], r_a[116]*r_a[115], r_a[115]*r_a[114],
111        r_a[114]*r_a[113], r_a[113]*r_a[112], r_a[112]*r_a[111], r_a[111]*r_a[110], r_a[110]*r_a[109], r_a[109]*r_a[108], r_a[108]*r_a[107], r_a[107]*r_a[106],
112        r_a[106]*r_a[105], r_a[105]*r_a[104], r_a[104]*r_a[103], r_a[103]*r_a[102], r_a[102]*r_a[101], r_a[101]*r_a[100], r_a[100]*r_a[99], r_a[99]*r_a[98],
113        r_a[98]*r_a[97], r_a[97]*r_a[96], r_a[96]*r_a[95], r_a[95]*r_a[94], r_a[94]*r_a[93], r_a[93]*r_a[92], r_a[92]*r_a[91], r_a[91]*r_a[90], r_a[90]*r_a[89],
114        r_a[89]*r_a[88], r_a[88]*r_a[87], r_a[87]*r_a[86], r_a[86]*r_a[85], r_a[85]*r_a[84], r_a[84]*r_a[83], r_a[83]*r_a[82], r_a[82]*r_a[81], r_a[81]*r_a[80],
115        r_a[80]*r_a[79], r_a[79]*r_a[78], r_a[78]*r_a[77], r_a[77]*r_a[76], r_a[76]*r_a[75], r_a[75]*r_a[74], r_a[74]*r_a[73], r_a[73]*r_a[72], r_a[72]*r_a[71],
116        r_a[71]*r_a[70], r_a[70]*r_a[69], r_a[69]*r_a[68], r_a[68]*r_a[67], r_a[67]*r_a[66], r_a[66]*r_a[65], r_a[65]*r_a[64], r_a[64]*r_a[63], r_a[63]*r_a[62],
117        r_a[62]*r_a[61], r_a[61]*r_a[60], r_a[60]*r_a[59], r_a[59]*r_a[58], r_a[58]*r_a[57], r_a[57]*r_a[56], r_a[56]*r_a[55], r_a[55]*r_a[54], r_a[54]*r_a[53],
118        r_a[53]*r_a[52], r_a[52]*r_a[51], r_a[51]*r_a[50], r_a[50]*r_a[49], r_a[49]*r_a[48], r_a[48]*r_a[47], r_a[47]*r_a[46], r_a[46]*r_a[45], r_a[45]*r_a[44],
119        r_a[44]*r_a[43], r_a[43]*r_a[42], r_a[42]*r_a[41], r_a[41]*r_a[40], r_a[40]*r_a[39], r_a[39]*r_a[38], r_a[38]*r_a[37], r_a[37]*r_a[36], r_a[36]*r_a[35],
120        r_a[35]*r_a[34], r_a[34]*r_a[33], r_a[33]*r_a[32], r_a[32]*r_a[31], r_a[31]*r_a[30], r_a[30]*r_a[29], r_a[29]*r_a[28], r_a[28]*r_a[27], r_a[27]*r_a[26],
121        r_a[26]*r_a[25], r_a[25]*r_a[24], r_a[24]*r_a[23], r_a[23]*r_a[22], r_a[22]*r_a[21], r_a[21]*r_a[20], r_a[20]*r_a[19], r_a[19]*r_a[18], r_a[18]*r_a[17],
122        r_a[17]*r_a[16], r_a[16]*r_a[15], r_a[15]*r_a[14], r_a[14]*r_a[13], r_a[13]*r_a[12], r_a[12]*r_a[11], r_a[11]*r_a[10], r_a[10]*r_a[9], r_a[9]*r_a[8],
123        r_a[8]*r_a[7], r_a[7]*r_a[6], r_a[6]*r_a[5], r_a[5]*r_a[4], r_a[4]*r_a[3], r_a[3]*r_a[2], r_a[2]*r_a[1], r_a[1]*r_a[0]);
124
125 //-----r_c-----
126 always@(posedge I_CLK or negedge I_RSTN) begin
127     if(!I_RSTN) begin
128         r_c <= 0;
129     end
130     else if(r_state == IDLE_STATE) begin
131         r_c <= 0;
132     end
133     else if(r_state == CAL_STATE) begin
134         r_c <= r_c ^ ((r_a & w_b7) << 7) ^ ((r_a & w_b5) << 5) ^ ((r_a & w_b4) << 4)
135         ^ ((r_a & w_b3) << 3) ^ ((r_a & w_b2) << 2) ^ ((r_a & w_b1) << 1) ^ (r_a & w_b0);
136     end
137 end
138
139 endmodule
140

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า  
 ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

## ประวัติผู้เขียน

ชื่อ-สกุล นายพชร 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 ซิลิคอนกราฟท์ เทคโนโลยี จำกัด

(มหาชน)