การจำลองสมรรถนะของรหัสแอลดีพีซีด้วย FPGA สำหรับมาตรฐาน CCSDS

PERFORMANCE EVALUATION OF FPGA-BASED LDPC CODES FOR CCSDS STANDARD

โดย

นางสาวพนัชกร เงินเจริญดี

นางสาวศิริพรรษา สารคล่อง

นายสัญชัย ราชคำ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2564

การจำลองสมรรถนะของรหัสแอลดีพีซีด้วย FPGA สำหรับมาตรฐาน CCSDS

PERFORMANCE EVALUATION OF FPGA-BASED LDPC CODES FOR CCSDS STANDARD

โดย

นางสาวพนัชกร	เงินเจริญดี	61010685
นางสาวศิริพรรษา	สารคล่อง	61011019
นายสัญชัย	ราชคำ	61011101

อาจารย์ที่ปรึกษา

ผศ. ดร. เวธิต ภาคย์พิสุทธิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2564

ปริญญานิพนธ์ปีการศึกษา 2564

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การจำลองสมรรถนะของรหัสแอลดีพีซีด้วย FPGA สำหรับมาตรฐาน CCSDS
PERFORMANCE EVALUATION OF FPGA-BASED LDPC CODES FOR CCSDS STANDARD

ผู้จัดทำ

1. นางสาวพนัชกร เงินเจริญดี 61010685
2. นางสาวศิริพรรษา สารคล่อง 61011019
3. นายสัญชัย ราชคำ 61011101

เวธิต ภาคย์พิสุทธิ์

..... **อาจารย์ที่ปรึกษา**

(ผศ. ดร. เวธิต ภาคย์พิสุทธิ์)

กิตติกรรมประกาศ

การดำเนินปริญญานิพนธ์เรื่อง “การจำลองสมรรถนะของรหัสแอลดีพีซีด้วย FPGA สำหรับมาตรฐาน CCSDS” จะไม่สามารถสำเร็จลุล่วงไปได้ด้วยดีหากไม่ได้รับความช่วยเหลือและความอนุเคราะห์อย่างดียิ่งจาก ผศ. ดร.เวธิต ภาคย์พิสุทธิ์ รวมถึงนักศึกษาปริญญาโท และนักศึกษาปริญญาเอก ที่กรุณาให้คำแนะนำ คำปรึกษา และแนวทางการแก้ไขปัญหาที่เป็นประโยชน์ต่อการศึกษาค้นคว้าวิจัยให้ปริญญานิพนธ์นี้สำเร็จ สมบูรณ์ยิ่งขึ้น รวมถึงสนับสนุนสถานที่ เครื่องมือและอุปกรณ์ต่างๆ ที่ใช้ระหว่างการจัดทำปริญญานิพนธ์

ขอขอบคุณคณาจารย์และเจ้าหน้าที่ประจำภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่าน ที่ได้ให้ความรู้เอื้อเฟื้อสถานที่และประสบการณ์ให้แก่ผู้จัดทำ

ขอขอบคุณผู้ที่มีส่วนเกี่ยวข้องทุกท่านเป็นอย่างสูงที่คอยช่วยเหลือให้ความรู้เพิ่มเติมและขอขอบคุณครอบครัวและเพื่อนที่เป็นกำลังใจสำคัญให้แก่ผู้จัดทำ ให้สามารถทำงานลุล่วงไปได้ด้วยดี

นางสาวพนัชกร เงินเจริญดี
นางสาวศิริพรรษา สารคล่อง
นายสัญชัย ราชคำ
ผู้จัดทำ

การจำลองสมรรถนะของรหัสแอลดีพีซีด้วย FPGA สำหรับมาตรฐาน CCSDS

Performance Evaluation of FPGA-based LDPC codes for CCSDS standard

โดย นางสาวพนัชกร เงินเจริญดี 61010685

นางสาวศิริพรรษา สารคล่อง 61011019

นายสัญชัย ราชคำ 61011101

อาจารย์ที่ปรึกษา ผศ. ดร. เวธิต ภาคย์พิสุทธิ์

บทคัดย่อ

ปริญญานิพนธ์นี้มีวัตถุประสงค์เพื่อสร้างชุดจำลองสมรรถนะการเข้ารหัส และถอดรหัสตามมาตรฐาน Consultative Committee for Space Data Systems (CCSDS) โดยเริ่มต้นจากสร้างส่วนประสานงานผู้ใช้ (Graphical User Interface : GUI) ด้วยภาษา Python เพื่อรับพารามิเตอร์ที่ต้องการจากผู้ใช้งาน ได้แก่ ค่าความแปรปรวน ส่งพารามิเตอร์ดังกล่าวไปยังอุปกรณ์ลอจิกแบบโปรแกรมได้ (Field Programmable Gate Array : FPGA) ผ่านส่วนรับส่งข้อมูลด้วยโปรโตคอล Universal Asynchronous Receiver Transmitter (UART) เข้าสู่วงจรสร้างบิตข้อมูล ซึ่งจะส่งข้อมูลให้แก่วงจรเข้ารหัสเพื่อให้ได้เอาต์พุตออกมาเป็นคำรหัส จากนั้นส่งคำรหัสที่ได้ไปเข้าวงจรสร้างสัญญาณสุ่มเพื่อจำลองการสื่อสารผ่านช่องสัญญาณจริงจากภาคส่งไปยังภาครับ และส่งข้อมูลจากวงจรสร้างสัญญาณสุ่มไปยังวงจรถอดรหัสเพื่อให้ได้เอาต์พุตออกมาเป็นข้อมูลข่าวสาร และแสดงค่าอัตราบิตผิดพลาดบนส่วนประสานงานผู้ใช้

ABSTRACT

This thesis aims to design performance simulator of LDPC codes for Consultative Committee for Space Data Systems (CCSDS) standards, starting from designing a Graphical User Interface (GUI) with Python for receiving essential parameters including variance. These parameters are sent to a Field Programmable Gate Array (FPGA) logic device through the transmission interface with Universal Asynchronous Receiver Transmitter (UART) protocol to the bit-generating circuit, which sends the information to the encoder circuit to achieve a code word. The code words are then sent to a random signal generator circuit to simulate real world communication from transmitter to receiver, then send the data from a random signal generator circuit to a decoder circuit to achieve information and display the bit error rate on the graphic user interface.

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญตาราง

หน้า

ตารางที่

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการสื่อสารดิจิทัลมีความสำคัญอย่างมากในระบบการสื่อสารไร้สาย จึงได้มีการพัฒนาเทคนิคเพื่อตอบสนองความต้องการของผู้ใช้งานในการรับส่งข้อมูลโดยการนำข้อมูลผ่านกระบวนการเข้ารหัสเพื่อเพิ่มสมรรถนะในการแก้ไขความผิดพลาดจากการส่งข้อมูลผ่านช่องสัญญาณซึ่งมีสัญญาณรบกวน หนึ่งในวิธีการที่ถูกนำมาใช้คือการเข้ารหัสพาริตีเช็กความหนาแน่นต่ำ (Low-Density Parity-Check: LDPC) ซึ่งถูกนำมาใช้ในการเข้ารหัสและถอดรหัสช่องสัญญาณในมาตรฐาน Consultative Committee for Space Data Systems (CCSDS) เนื่องจากรหัส LDPC มีประสิทธิภาพในการแก้ไขบิตผิดพลาดที่สูง เพื่อจุดประสงค์ทางด้านการศึกษาของผู้ที่สนใจ ผู้จัดทำจึงสร้างชุดจำลองสมรรถนะการเข้ารหัส และถอดรหัสของมาตรฐาน CCSDS โดยทำการออกแบบและสร้างวงจรเข้ารหัส วงจรสร้างสัญญาณรบกวน และวงจรถอดรหัส เพื่อจำลองการสื่อสารผ่านช่องสัญญาณตามมาตรฐาน CCSDS

1.2 วัตถุประสงค์

1) เพื่อทดสอบวงจรเข้ารหัสและถอดรหัสตามมาตรฐาน CCSDS บน FPGA

2) เพื่อแสดงอัตราบิตผิดพลาด (Bit Error Rate : BER) ผ่านส่วนประสานงานผู้ใช้ (Graphic User Interface : GUI)

1.3 ขอบเขตของปริญญานิพนธ์

1) ออกแบบและสร้างวงจรเข้ารหัสและถอดรหัส LDPC ชนิด Accumulate Repeat 4 Jagged-Accumulate (AR4JA)

2) ออกแบบและสร้างวงจรเข้ารหัสและถอดรหัสบนอุปกรณ์ลอจิกแบบโปรแกรมได้ (Field Programmable Gate Array: FPGA)

บทที่ 2

ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 รหัสบล็อกเชิงเส้น

รหัสบล็อกเชิงเส้น สามารถเขียนแทนด้วยสัญลักษณ์ (N,K) ซึ่ง N คือความยาวคำรหัสและ K คือความยาวบิตข้อมูล ดังนั้นภายใต้คำรหัสจะประกอบไปด้วย 2 ส่วน โดยส่วนแรกคือบิตข้อมูลที่ต้องการส่งไปยังภาครับ (Message Bit) ซึ่งมีอยู่จำนวน K บิต และสามารถเขียนอยู่ในรูป $u=[u_0,u_1,u_2,...,u_{N-K}]$ โดยคำรหัสทั้งหมดที่ประกอบด้วยส่วนทั้งสองสามารถเขียนในรูป $c=[u_0,u_1,u_2,...,u_{K-1},p_0,p_1,p_2,...,p_{N-K}]$

ขีดความสามารถในการแก้ไขความผิดพลาดของรหัสบล็อกเชิงเส้นจะแปรผันตรงกับความยาวของคำรหัส ดังนั้นจึงมีการกำหนดอัตราส่วนของจำนวนบิตข้อมูลกับจำนวนบิตของคำรหัสเพื่อใช้สำหรับการอ้างอิงในกระบวนการต่างๆ ในการเข้ารหัสและถอดรหัส ซึ่งอัตราส่วนดังกล่าวถูกเรียกว่า อัตรารหัส R และนิยามได้ดังสมการที่ (2.1)

$$R=\frac{K}{N} \tag{2.1}$$

โดย R คืออัตรารหัส

K คือความยาวข้อความ

N คือความยาวคำรหัส

ค่าอัตรารหัสที่เลือกใช้ขึ้นอยู่กับปริมาณสัญญาณรบกวน (Noise) กล่าวคือในการสื่อสารที่มีสัญญาณรบกวนมากจะเลือกใช้อัตรารหัสที่มีค่าต่ำ หากการสื่อสารที่มีสัญญาณรบกวนน้อยจะเลือกใช้อัตรารหัสที่มีค่าสูง

2.1.1 รหัสตรวจสอบพาริตีความหนาแน่นต่ำ

รหัสตรวจสอบพาริตีความหนาแน่นต่ำ (Low-Density Parity-Check : LDPC) เป็นรหัสบล็อกเชิงเส้นที่ถูกนำเสนอโดย R. Gallager และพัฒนาต่อยอดโดย D. Mackay และ R. Neal ซึ่งนักวิจัยทั้งสองได้มีการพิสูจน์ว่ารหัส LDPC มีสมรรถนะเข้าใกล้ความจุช่องสัญญาณหรือลิมิตของแชนนอน (Shannon’s Limit) ภายใต้การถอดรหัสแบบวนซ้ำด้วยการแพร่กระจายความเชื่อมั่น (Belief Propagation: BP)

2.1.1.1 เมทริกซ์กำเนิด

เมทริกซ์กำเนิดเป็นเมทริกซ์ขนาด $K \times N$ สำหรับการสร้างรหัสบล็อกเชิงเส้น สมาชิกแต่ละตัวในเมทริกซ์ดังกล่าวมีค่าได้เป็น 2 แบบคือ 0 หรือ 1 ซึ่งรูปแบบของเมทริกซ์กำเนิดเป็นดังสมการที่ (2.2)

$$G_{K\times N} = \left[P_{K\times(N-K)} \quad \middle| I_{K\times K} \right] \tag{2.2}$$

โดย G คือเมทริกซ์กำเนิด

I คือเมทริกซ์เอกลักษณ์

K คือความยาวข้อความ

N คือความยาวคำรหัส

P คือเมทริกซ์ย่อยส่วนพาริตีดังสมการที่ (2.3)

$$P_{K\times(N-K)} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,(N-K)} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,(N-K)} \\ \vdots & \vdots & \ddots & \vdots \\ p_{K,1} & p_{K,2} & \cdots & p_{K,(N-K)} \end{bmatrix} \tag{2.3}$$

โดย P คือเมทริกซ์ย่อยส่วนพาริตี

K คือความยาวข้อความ

N คือความยาวคำรหัส

p คือสมาชิกในเมทริกซ์ P

2.1.1.2 เมทริกซ์ตรวจสอบพาริตี

เมทริกซ์ตรวจสอบพาริตีเป็นเมทริกซ์ขนาด $(N-K)\times N$ สำหรับตรวจสอบความผิดพลาดอย่างง่าย โดยสมาชิกแต่ละตัวในเมทริกซ์ดังกล่าวมีค่าได้เป็น 2 แบบคือ 0 หรือ 1 ซึ่งรูปแบบของเมทริกซ์ตรวจสอบพาริตีเป็นดังสมการที่ (2.4)

$$H_{(N-K)\times N} = \left[I_{(N-K)\times(N-K)} \quad \middle| P^t \right] \tag{2.4}$$

โดย K คือความยาวข้อความ

N คือความยาวคำรหัส

H คือเมทริกซ์ตรวจสอบพาริตี

P^t คือทรานสโพสของเมทริกซ์ P จากสมการที่ (2.3)

เมทริกซ์ตรวจสอบพาริตี H_{QC} ขนาด $I \times J$ แบบ Quasi-Cyclic จะประกอบด้วยเมทริกซ์ Q ซึ่งเป็นเมทริกซ์หมุนวนย่อยในเมทริกซ์ตรวจสอบพาริตี ซึ่งเมทริกซ์ย่อยดังกล่าวมีขนาด $M \times M$ และ P จะแสดงถึงจำรอบการหมุนในเมทริกซ์ย่อยดังสมการที่ (2.5)

$$H_{QC} = \begin{bmatrix} Q_{1,1}(P) & Q_{1,2}(P) & \cdots & Q_{1,j}(P) \\ Q_{2,1}(P) & Q_{2,2}(P) & \cdots & Q_{2,j}(P) \\ \vdots & \vdots & \ddots & \vdots \\ Q_{i,1}(P) & Q_{i,2}(P) & \cdots & Q_{i,j}(P) \end{bmatrix} \quad (2.5)$$

โดย H_{QC} คือเมทริกซ์ตรวจสอบพาริตี

Q คือเมทริกซ์ย่อยของเมทริกซ์ตรวจสอบพาริตีขนาด $M \times M$

M ค่าการขยายเมทริกซ์ (Lifting factor)

P คือจำนวนครั้งของการหมุนวนในเมทริกซ์ย่อย

2.1.2 มาตรฐาน Consultative Committee for Space Data Systems

Consultative Committee for Space Data Systems (CCSDS) คือคณะกรรมการที่กำหนดมาตรฐานสำหรับการสื่อสารบนอวกาศซึ่งอาศัยการเข้ารหัสและถอดรหัส LDPC โดยมีคุณสมบัติต่าง ๆ ดังนี้

2.1.2.1 คุณสมบัติที่ต้องการ

1) ตัวเข้ารหัสที่เป็นระบบแบบแผน

2) อัตรารหัส 0.5, 0.66, 0.75, 0.8

3) ความยาวของบล็อกข้อมูล k มีขนาดตั้งแต่ 1000 ถึง 16000

2.1.2.2 เกณฑ์การประเมิน

1) การคำนวณการถอดรหัสมีความซับซ้อนต่ำ

2) ประสิทธิภาพของโค้ด โค้ดต้องมี SNR น้อยที่ Word Error Rate (WER) น้อยกว่า 10^{-4} และมากกว่า 10^{-6}

อัตรารหัสที่ทำการเลือกคือ 1/2, 2/3, และ 4/5 มีค่า Signal to Noise Ratio (SNR) เท่ากับ 1 dB บนกราฟความจุอัตราช่องสัญญาณ Additive White Gaussian Noise (AWGN)

เลือกความยาวบล็อกข้อมูล k เท่ากับ 1024, 4096, 16384 ค่า k ทั้งสามค่ามีค่า SNR เท่ากับ 0.6 dB ที่ WER = 10^{-8}

การเข้ารหัสที่มีความซับซ้อนต่ำ ซึ่งมีเมทริกซ์ตรวจสอบพาริตีหลากหลายที่ช่วยอำนวยความสะดวกในการถอดรหัส รหัสดังกล่างมีชื่อว่า Accumulate-Repeat-4-Jagged-Accumulate (AR4JA)

2.1.2.3 ข้อมูลจำเพาะ

ความสัมพันธ์ของความยาวบล็อกรหัสและอัตรารหัสเป็นดังตารางที่ 2.1

ตารางที่ 2.1 ความสัมพันธ์ของความยาวบล็อกรหัสและอัตรารหัส

อัตรารหัส (R)	ความยาวคำรหัส (N)		
1/2	2048	8192	32768
2/3	1536	6144	24576
4/5	1280	5120	20480

2.1.2.4 เมทริกซ์ตรวจสอบพาริตีของมาตรฐาน CCSDS

เมทริกซ์ตรวจสอบพาริตี H มีโครงสร้างจากเมทริกซ์ย่อยขนาด $M \times M$ เมทริกซ์ H สำหรับอัตรารหัส 1/2, 2/3, 3/4, 4/5 เป็นดังสมการที่ (2.6) ถึง (2.9) ตามลำดับ

$$H_{1/2} = \begin{bmatrix} 0_M & 0_M & I_M & 0_M & I_M \oplus \Pi_1 \\ I_M & I_M & 0_M & I_M & \Pi_2 \oplus \Pi_3 \oplus \Pi_4 \\ I_M & \Pi_5 \oplus \Pi_6 & 0_M & \Pi_7 \oplus \Pi_8 & I_M \end{bmatrix} \quad (2.6)$$

$$H_{2/3} = \left[\begin{array}{cc} 0_M & 0_M \\ \Pi_9 \oplus \Pi_{10} \oplus \Pi_{11} & I_M \\ I_M & \Pi_{12} \oplus \Pi_{13} \oplus \Pi_{14} \end{array}\right| H_{1/2} \Bigg] \quad (2.7)$$

$$H_{3/4} = \left[\begin{array}{cc} 0_M & 0_M \\ \Pi_{15} \oplus \Pi_{16} \oplus \Pi_{17} & I_M \\ I_M & \Pi_{18} \oplus \Pi_{19} \oplus \Pi_{20} \end{array}\right| H_{2/3} \Bigg] \quad (2.8)$$

$$H_{4/5} = \left[\begin{array}{cc|c} 0_M & 0_M & \\ \Pi_{21} \oplus \Pi_{22} \oplus \Pi_{23} & I_M & H_{3/4} \\ I_M & \Pi_{24} \oplus \Pi_{25} \oplus \Pi_{26} & \end{array}\right] \tag{2.9}$$

โดย $H_{1/2}$ คือเมทริกซ์ตรวจสอบพาริตีอัตรารหัส 1/2

$H_{2/3}$ คือเมทริกซ์ตรวจสอบพาริตีอัตรารหัส 2/3

$H_{3/4}$ คือเมทริกซ์ตรวจสอบพาริตีอัตรารหัส 3/4

$H_{4/5}$ คือเมทริกซ์ตรวจสอบพาริตีอัตรารหัส 4/5

M คือค่าการขยายเมทริกซ์ (Lifting factor)

Π_k คือเมทริกซ์หมุนวน

การสร้างเมทริกซ์หมุนวน Π_k จะทำโดยการหาหลักในเมทริกซ์ย่อย $\pi_k(i)$ ดังสมการที่ (2.10) ซึ่ง $\pi_k(i)$ บ่งบอกถึงหลักในเมทริกซ์ย่อย ณ แถว i ซึ่งมีสมาชิกเป็น 1 หลักอื่นที่นอกเหนือจากที่ทำการคำนวณได้ บ่งบอกถึงสมาชิกที่เป็น 0 ณ หลักและแถวนั้น

$$\pi_k(i) = \frac{M}{4}\left(\left(\theta_k + \lfloor 4i/M \rfloor\right) \bmod 4\right) + \left(\phi_k\left(\lfloor 4i/M \rfloor, M\right) + i\right) \bmod \frac{M}{4} \tag{2.10}$$

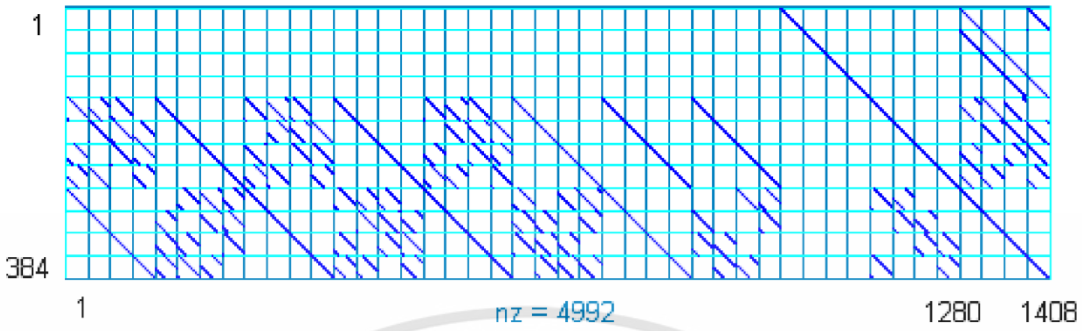
โดย π_k คือค่าหลักในเมทริกซ์ย่อย

i คือแถวในเมทริกซ์ย่อย

M คือค่าการขยายเมทริกซ์ (Lifting factor)

ฟังก์ชัน θ_k และ $\phi_k(j,M)$ หาได้จากตารางที่ 2.2 และตารางที่ 2.3 $\phi_k(j,M)$ โดยที่ $M = \{128, 256, 512, 1024, 2048, 4096, 8192\}$

สำหรับเมทริกซ์ H ใด ๆ ที่สร้างขึ้น ค่า M ตัวสุดท้ายจะถูกทำการตัดออก (Punctured) ตัวอย่างเช่น เมทริกซ์ตรวจสอบพาริตีที่มีขนาด $n = 1280$ และ $k = 1024$ อัตรารหัสเท่ากับ 4/5 โดยมีเส้นสีน้ำเงินแสดงการหมุนที่ไม่เท่ากับ 0 และโครงสร้างจะระบุด้วยเส้นตาราง โดยเส้นตารางรองจะเว้นระยะเป็นช่วง m และเส้นตารางหลักที่ (ไม่แสดง) $M = 4m$ ดังรูปที่ 2.1



รูปที่ 2.1 เมทริกซ์ H ตัวอย่างขนาด $n=1280$ และ $k=1024$ ที่อัตรารหัสเท่ากับ 4/5

ตารางที่ 2.2 ตารางแสดงค่า $\phi_k(0,M)$ และ $\phi_k(1,M)$

k	ϕ_k	$\phi_k(0,M)$ $M=2^7...2^{13}$							$\phi_k(1,M)$ $M=2^7...2^{13}$						
1	3	1	59	16	160	108	226	1148	0	0	0	0	0	0	0
2	0	22	18	103	241	126	618	2032	27	32	53	182	375	767	1822
3	1	0	52	105	185	238	404	249	30	21	74	249	436	227	203
4	2	26	23	0	251	481	32	1807	28	36	45	65	350	247	882
5	2	0	11	50	209	96	912	485	7	30	47	70	260	284	1989
6	3	10	7	29	103	28	950	1044	1	29	0	141	84	370	957
7	0	5	22	115	90	59	534	717	8	44	59	237	318	482	1705
8	1	18	25	30	184	225	63	873	20	29	102	77	382	273	1083
9	0	3	27	92	248	323	971	364	26	39	25	55	169	886	1072
10	1	22	30	78	12	28	304	1926	24	14	3	12	213	634	354
11	2	3	43	70	111	386	409	1241	4	22	88	227	67	762	1942
12	0	8	14	66	66	305	708	1769	12	15	65	42	313	184	446
13	2	25	46	39	173	34	719	532	23	48	62	52	242	696	1456
14	3	25	62	84	42	510	176	768	15	55	68	243	188	413	1940
15	0	2	44	79	157	147	743	1138	15	39	91	179	1	854	1660
16	1	27	12	70	174	199	759	365	22	11	70	250	306	544	1661
17	2	7	38	29	104	347	674	141	31	1	115	247	397	864	587
18	0	7	47	32	144	391	958	1527	3	50	31	164	80	82	708
19	1	15	1	45	43	165	984	505	29	40	121	17	33	1009	1466
20	2	10	52	113	181	414	11	1312	21	62	45	31	7	437	433
21	0	4	61	86	250	97	413	1840	2	27	56	149	447	36	1345
22	1	19	10	1	202	158	925	709	5	38	54	105	336	562	867
23	2	7	55	42	68	86	687	1427	11	40	108	183	424	816	1551
24	1	9	7	118	177	168	752	989	26	15	14	153	134	452	2041
25	2	26	12	33	170	506	867	1925	9	11	30	177	152	290	1383
26	3	17	2	126	89	489	323	270	17	18	116	19	492	778	1790

ตารางที่ 2.3 ตารางแสดงค่า $\phi_k(2,M)$ และ $\phi_k(3,M)$

k	ϕ_k	$\phi_k(2,M)$ $M=2^7...2^{13}$							$\phi_k(3,M)$ $M=2^7...2^{13}$						
1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	12	46	8	35	219	254	318	13	44	35	162	312	285	1189
3	1	30	45	119	167	16	790	494	19	51	97	7	503	554	458
4	2	18	27	89	214	263	642	1467	14	12	112	31	388	809	460
5	2	10	48	31	84	415	248	757	15	15	64	164	48	185	1039
6	3	16	37	122	206	403	899	1085	20	12	93	11	7	49	1000
7	0	13	41	1	122	184	328	1630	17	4	99	237	185	101	1265
8	1	9	13	69	67	279	518	64	4	7	94	125	328	82	1223
9	0	7	9	92	147	198	477	689	4	2	103	133	254	898	874
10	1	15	49	47	54	307	404	1300	11	30	91	99	202	627	1292
11	2	16	36	11	23	432	698	148	17	53	3	105	285	154	1491
12	0	18	10	31	93	240	160	777	20	23	6	17	11	65	631
13	2	4	11	19	20	454	497	1431	8	29	39	97	168	81	464
14	3	23	18	66	197	294	100	659	22	37	113	91	127	823	461
15	0	5	54	49	46	479	518	352	19	42	92	211	8	50	844
16	1	3	40	81	162	289	92	117	15	48	119	128	437	413	392
17	2	29	27	96	101	373	464	836	5	4	74	82	475	462	922
18	0	11	35	38	76	104	592	1572	21	10	73	115	85	175	256
19	1	4	25	83	78	141	198	348	17	18	116	248	419	715	1986
20	2	8	46	42	253	270	856	1040	9	56	31	62	459	537	19
21	0	2	24	58	124	439	235	779	20	9	127	26	468	722	266
22	1	11	33	24	143	333	134	476	18	11	98	140	209	37	471
23	2	11	18	25	63	399	542	191	31	23	23	121	311	488	1166
24	1	3	37	92	41	14	545	1393	13	8	38	12	211	179	1300
25	2	15	35	38	214	277	777	1752	2	7	18	41	510	430	1033
26	3	13	21	120	70	412	483	1627	18	24	62	249	320	264	1606

2.1.3 การเข้ารหัส LDPC

2.1.3.1 การเข้ารหัส LDPC โดยวิธี Gaussian Jordan

การเข้ารหัส LDPC โดยใช้วิธี Gaussian Jordan ทำโดยการนำเมทริกซ์ตรวจสอบพาริตี H มาทำให้อยู่ในรูปแบบดังสมการที่ (2.4) จากนั้นจึงสามารถแปลงให้เป็นเมทริกซ์กำเนิด G ดังสมการที่ (2.2) ด้วยการกำจัดแบบเกาส์ (Gaussian Elimination) และขั้นตอนสุดท้ายคือการนำข้อมูลข่าวสารมาทำการคูณกับเมทริกซ์กำเนิด G จึงได้คำรหัสออกมาดังสมการที่ (2.11)

$$c = mG \tag{2.11}$$

โดย c คือคำรหัส

m คือบิตข้อมูล

G คือเมทริกซ์กำเนิด

2.1.3.2 การเข้ารหัส LDPC โดยวิธี Quasi-cyclic (QC LDPC)

การเข้ารหัส LDPC โดยใช้วิธี Quasi-cyclic ทำโดยการนำเมทริกซ์ตรวจสอบพาริตี H ตามมาตรฐาน CCSDS ดังสมการที่ (2.6) ถึง (2.9) มาแปลงให้เป็นเมทริกซ์กำเนิด G ซึ่งยังคงคุณสมบัติ QC ด้วยการแบ่งเมทริกซ์ตรวจสอบพาริตีให้เป็นดังสมการที่ (2.12)

$$H = \begin{bmatrix} U_1 & U_2 & \cdots & U_i & D \end{bmatrix} \tag{2.12}$$

$$U_i = H(1:row, (i-1)*M/4+1 : i*M/4) \tag{2.13}$$

โดย H คือเมทริกซ์ตรวจสอบพาริตี

U_i คือเมทริกซ์ที่มีจำนวนหลักเท่ากับค่า $M/4$

row คือจำนวนแถวของเมทริกซ์ตรวจสอบพาริตี

D คือเมทริกซ์ที่เหลือจากการแบ่งเมทริกซ์ตรวจสอบพาริตี

จากนั้นนำเมทริกซ์ D ที่ได้มาทำการอินเวอร์สเพื่อหาค่า g_i^T ดังสมการที่ (2.14) และนำไปใช้ในการประกอบเมทริกซ์กำเนิด G ดังสมการที่ (2.15) เพื่อที่จะนำไปคำนวณหาคำรหัสดังสมการที่ (2.11) โดยการเลื่อนข้อมูลข่าวสารแทนการคูณข้อมูลข่าวสารกับเมทริกซ์กำเนิด G

$$g_i^T = D^{-1}U_i I \tag{2.14}$$

โดย g_i^T คือทรานสโพสเมทริกซ์ย่อยของเมทริกซ์กำเนิด G

D^{-1} คืออินเวอร์สของเมทริกซ์ D

U คือเมทริกซ์ที่มีจำนวนหลักเท่ากับค่า $M/4$

I คือเมทริกซ์เอกลักษณ์

$$G = \left[\begin{array}{c|c} g_1 & \\ g_2 & \\ \vdots & I \\ g_i & \end{array}\right] \tag{2.15}$$

โดย G คือเมทริกซ์กำเนิด G

g คือเมทริกซ์ย่อยของเมทริกซ์กำเนิด G

I คือเมทริกซ์เอกลักษณ์

2.1.3.3 การเข้ารหัส LDPC จากเมทริกซ์ตรวจสอบพาริตีโดยตรง

การเข้ารหัสโดยใช้วิธีจากมเทริกซ์ตรวจสอบพาริตีโดยตรง ทำได้โดยจัดรูปเมทริกซ์ตรวจสอบพาริตี ให้อยู่ในรูปแบบดังสมการที่ (2.16) และกำหนดค่าคำรหัสให้มีรูปแบบดังสมการที่ (2.17)

$$H = \left[\begin{array}{cccc|ccccc} a_{1,1} & a_{1,2} & \cdots & a_{1,y} & b_{1,1} & \cdots & b_{1,k} & 1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,y} & b_{2,1} & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 1 & \ddots & \vdots \\ a_{j,1} & a_{j,2} & \cdots & a_{j,y} & 1 & 0 & \cdots & 0 \end{array}\right] \tag{2.16}$$

$$c = [u_1 \quad u_2 \quad \cdots \quad u_y \quad p_1 \quad p_2 \quad \cdots \quad p_{k+1}] \tag{2.17}$$

โดย H คือเมทริกซ์ตรวจสอบพาริตี

c คือคำรหัส

a คือส่วนประกอบเมทริกซ์ตรวจสอบพาริตีที่เกี่ยวข้องกับบิตข้อความ

b คือส่วนประกอบเมทริกซ์ตรวจสอบพาริตีที่เกี่ยวข้องกับบิตพาริตี

u คือบิตข้อความ

p คือบิตพาริตี

จากนั้นนำเมทริกซ์คำรหัสมาทรานสโพส เพื่อที่จะได้นำมาคูณกับเมทริกซ์ตรวจสอบพาริตีดังสมการที่ (2.18) ซึ่งเป็นคุณสมบัติของรหัสบล็อกเชิงเส้น

กระบวนการหาบิตพาริตีเป็นดังสมการที่ (2.19) ถึง (2.21) จากนั้นนำบิตพาริตีมาประกอบในคำรหัส จะได้คำรหัสที่สมบูรณ์ดังสมการที่ (2.17)

$$Hc^T = 0 \tag{2.18}$$

โดย H คือเมทริกซ์ตรวจสอบพาริตี

c^T คือทรานสโพสของคำรหัส

$$Hc^T = \left[\begin{array}{cccc|cccc} a_{1,1} & a_{1,2} & \cdots & a_{1,y} & b_{1,1} & \cdots & b_{1,k} & 1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,y} & b_{2,1} & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 1 & \ddots & \vdots \\ a_{j,1} & a_{j,2} & \cdots & a_{j,y} & 1 & 0 & \cdots & 0 \end{array}\right] \left[\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_y \\ p_1 \\ p_2 \\ \vdots \\ p_{k+1} \end{array}\right] = 0 \quad (2.19)$$

$$a_{j,1}u_1 \oplus a_{j,2}u_2 \oplus \ldots \oplus a_{j,y}u_y \oplus p_1 = 0 \quad (2.20)$$

$$a_{j-1,1}u_1 \oplus a_{j-1,2}u_2 \oplus \ldots \oplus a_{j-1,y}u_y \oplus p_1 b_{j-1} \oplus p_2 = 0 \quad (2.21)$$

โดย H คือเมทริกซ์ตรวจสอบพาริตี

c^T คือทรานสโพสของคำรหัส

a คือส่วนประกอบเมทริกซ์ตรวจสอบพาริตีที่เกี่ยวข้องกับบิตข้อความ

b คือส่วนประกอบเมทริกซ์ตรวจสอบพาริตีที่เกี่ยวข้องกับบิตพาริตี

u คือบิตข้อความ

p คือบิตพาริตี

จากสมการที่ (2.19) ถึง (2.21) มีกระบวนการคือ นำเมทริกซ์ตรวจสอบพาริตีมาคูณกับคำรหัสที่ถูกทรานสโพส โดยจะเริ่มคูณจากแถวล่างสุดของเมทริกซ์ตรวจสอบพาริตี เมื่อได้ค่า p_1 จะสามารถนำไปหาค่า p_2 และพาริตีตัวอื่นๆได้ เมื่อได้บิตพาริตีครบ ก็จะได้คำรหัสที่สมบูรณ์

2.1.3.4 การเข้ารหัส LDPC โดยวิธี AR4JA

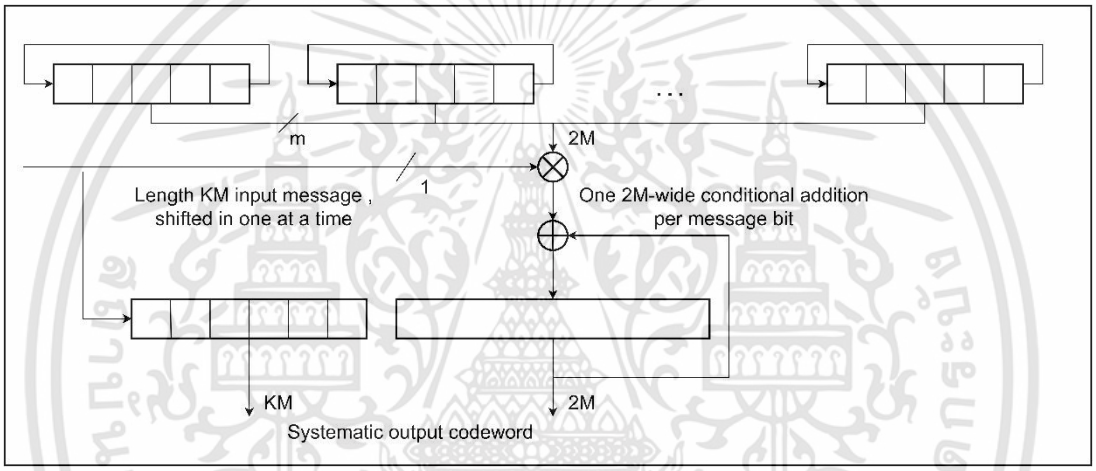
วิธีการสร้างบล็อกให้สัมพันธ์กับเมทริกซ์ตรวจสอบพาริตี AR4JA คือ การดำเนินการคูณเมทริกซ์โดยใช้เมทริกซ์กำเนิดบล็อกวนซ้ำ ซึ่งมีกระบวนการดังนี้

1) ให้ P เป็นเมทริกซ์ย่อยขนาด $3M \times 3M$ ของเมทริกซ์ H ประกอบด้วยคอลัมน์ $3M$ คอลัมน์สุดท้าย ให้ Q เป็นเมทริกซ์ย่อยขนาด $3M \times MK$ ของเมทริกซ์ H ประกอบด้วยคอลัมน์ MK คอลัมน์แรก

2) คำนวณ $W = (P^{-1}Q)^T$ โดยการคำนวณทางคณิตศาสตร์มอดูโล 2

3) สร้างเมทริกซ์ $G_0 = [I_{MK} \quad W]$ โดยที่ I_{MK} เป็นเมทริกซ์เอกลักษณ์ของเมทริกซ์ $MK \times MK$ และ W คือความหนาแน่นของเมทริกซ์หมุนวนขนาด $MK \times M(N-K)$

เมทริกซ์ G_0 คือ บล็อกหมุนวนที่มีขนาดของการซ้อนทับกันเท่ากับ $m = M/4$ บล็อกหมุนวนมีการอ้างอิงจากผลรวมของโมโนเมียลจาก $x^i (i \in \{0,\ldots,m-1\})$ โดยที่ x^i แทน เมทริกซ์หมุนวนเป็นหนึ่งในคอลัมน์เมทริกซ์แถวแรก $(j=0)$ คอลัมน์ที่ $i+1$ ของแถวที่ 2 $(j=1)$ ดังรูปที่ 2.2



รูปที่ 2.2 การเข้ารหัสแบบวนซ้ำและป้อนค่ากลับ

2.1.4 การพังก์เจอร์ (Punctured)

การพังก์เจอร์ คือกระบวนการที่นำพาริตีบิตบางส่วนออกจากคำรหัส ซึ่งทำหลังจากกระบวนการเข้ารหัส โดยการตัดคำรหัส ณ บิตท้ายสุดของคำรหัสโดยมีจำนวนเท่ากับค่าการขยายเมทริกซ์ตามที่เราได้กำหนดตั้งแต่ก่อนเริ่มทำการเข้ารหัส

2.1.5 การถอดรหัส LDPC ด้วยอัลกอริทึม Belief Propagation

ที่เครื่องรับสัญญาณจะได้รับสัญญาณคำรหัสที่ถูกรบกวนโดยสัญญาณรบกวนและทำการถอดรหัสข่าวสารแบบซอฟต์ (Soft Information) คือการถอดรหัสด้วยค่าอัตราค่าความน่าจะเป็นแบบลอการิทึม (Log-likelihood Ratio : LLR) L_{ch} ของสัญญาณที่ได้รับ y_i สามารถคำนวณ

ได้ด้วยสมการที่ (2.22) และจะถูกเปลี่ยนแปลงเรื่อย ๆ เมื่อเข้าสู่กระบวนการเปรียบเทียบกับ LUT ของ Check Node และ Variable Node ดังรูปที่

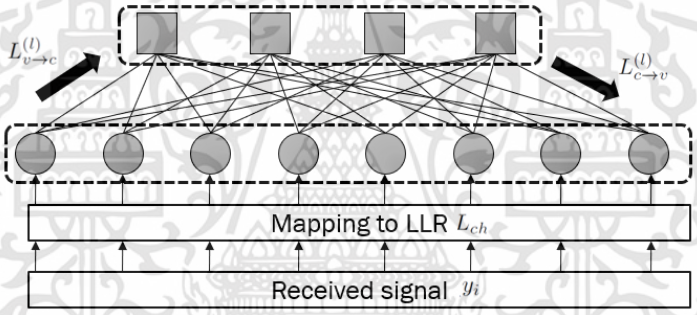
$$L_{ch}(y_i) = \log\left(\frac{p_{ch}(y_i \mid c_i = 0)}{p_{ch}(y_i \mid c_i = 1)}\right) \tag{2.22}$$

โดย L_{ch} คืออัตราค่าความน่าจะเป็นแบบลอการิทึมของช่องสัญญาณ

y_i คือสัญญาณที่เครื่องรับได้รับ

c_i คือคำรหัสจากเครื่องส่งสัญญาณ

$p_{ch}(y_i \mid c_i)$ คือฟังก์ชันการกระจายของสัญญาณที่รับได้เมื่อเครื่องส่งสัญญาณส่งคำรหัส



รูปที่ 2.3 การถอดรหัส LDPC ด้วยวิธีการ Belief Propagation

จากนั้นนำค่าอัตราค่าความน่าจะเป็นแบบลอการิทึมไปเก็บไว้เป็นข่าวสารแบบซอฟต์ (Soft Information) ที่ Check Node จากนั้นแปลงข่าวสารแบบซอฟต์ที่ Variable Node $L_{c\to v}(l)$ ดังสมการที่ 2.23 ในทำนองเดียวกันสามารถแปลงข่าวสารแบบซอฟต์ (Soft Information) จาก Variable Node ไปยัง Check Node $L_{v\to c}(l)$ ดังสมการที่ 2.24

ข่าวสารแบบซอฟต์จะถูกแลกเปลี่ยนระหว่างโหนดจนกว่าค่าวนซ้ำ Iteration (l) จะถึงค่าที่กำหนด เมื่อถึงแล้วจะทำการแปลงค่าข่าวสารครั้งสุดท้ายดังสมการที่ 2.25

$$L_{c\to v}(l) = 2\tanh^{-1}\left(\prod_{v' \in N(c)-\{v\}} \tanh\left(\frac{L_{v'\to c}(l)}{2}\right)\right) \tag{2.23}$$

$$L_{v\to c}(l) = L_{ch} + \sum_{c' \in M(v)-\{c\}} L_{v'\to c'}(l-1) \tag{2.24}$$

$$L_v(l_{\max}) = L_{ch} + \sum_{c \in M(v)} L_{c \to v}(l_{\max} - 1) \tag{2.25}$$

โดย $L_{c \to v}(l)$ คือ ข่าวสารแบบซอฟต์ (Soft Information) จาก Check Node ไปยัง Variable Node

$L_{v \to c}(l)$ คือ ข่าวสารแบบซอฟต์ (Soft Information) จาก Variable Node ไปยัง Check Node

L_{ch} คือ อัตราค่าความน่าจะเป็นแบบลอการิทึมของช่องสัญญาณ

N คือ จำนวน Check Node ทั้งหมดที่เชื่อมระหว่าง Check Node กับ Variable Node ยกเว้นโหนดที่ต้องการทราบค่า

M คือ จำนวน Variable Node ทั้งหมดที่เชื่อมระหว่าง Check Node กับ Variable Node ยกเว้นโหนดที่ต้องการทราบค่า

ขั้นตอนสุดท้ายคือการตัดสินใจบิตผลลัพธ์ของข้อมูลข่าวสาร สามารถคำนวณได้ดังสมการที่ 2.26

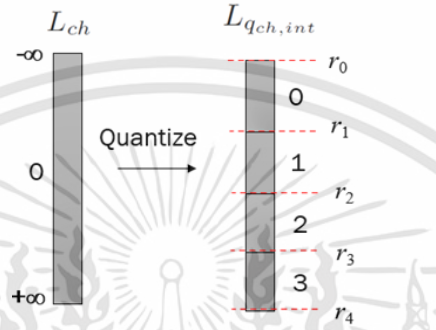
$$\hat{c}_v = \begin{cases} 1, & if \quad L_v(l_{\max}) \le 0 \\ 0, & else \end{cases} \tag{2.26}$$

โดย $\hat{c}_v$ คือ ข้อมูลข่าวสารหลังถอดรหัส

$L_v(l_{\max})$ คือ ข่าวสารแบบซอฟต์ค่าสุดท้ายที่ได้จาก Variable Node

2.1.5 การถอดรหัส LDPC ด้วย LUT (Lookup Table)

การถอดรหัสด้วย LUT เริ่มต้นจากการนำเอาต์พุตของการหาค่าอัตราค่าความน่าจะเป็นแบบลอการิทึมที่เป็นจำนวนจริงจากสมการที่ (2.21) มาทำการควอนไทซ์ (Quantize) ให้เป็นจำนวนเต็ม ซึ่งค่าที่นำมาควอนไทซ์จะเป็นดังรูปที่ 2.4



รูปที่ 2.4 ตัวอย่างการ Quantize 2 บิต

จากนั้นแทนค่าชุดข้อมูลที่ได้จากการควอนไทซ์ ในแต่ละหลักของเมทริกซ์ตรวจสอบพาริตีที่มีค่าบิตข้อมูลเป็น 1 เท่านั้นดังสมการที่ (2.27) และ (2.28)

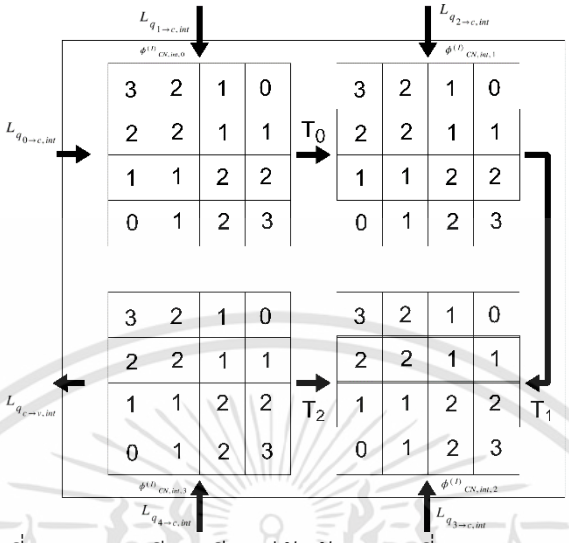
$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{2.27}$$

$$L_{q_{ch,\text{int}}} = \begin{bmatrix} 3 & 1 & 0 & 0 & 2 & 0 & 3 & 0 \\ 0 & 1 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 3 & 3 \\ 0 & 0 & 0 & 3 & 0 & 0 & 3 & 3 \end{bmatrix} \tag{2.28}$$

โดย H คือเมทริกซ์ตรวจสอบความถูกต้องพาริตี

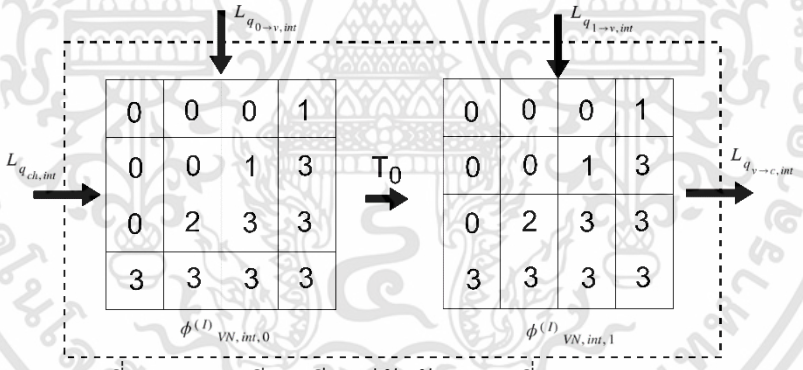
$L_{q_{ch,\text{int}}}$ คือเมทริกซ์ Log-Likelihood Ratio ของช่องสัญญาณ

หลังจากผ่านกระบวนการ Quantize จึงนำชุดอัตราค่าความน่าจะเป็นแบบลอการิทึมของช่องสัญญาณแต่ละตำแหน่งเข้าสู่กระบวนการเปรียบเทียบคู่อันดับโดยใช้ LUT ของ Check Node ดังรูปที่ 2.5



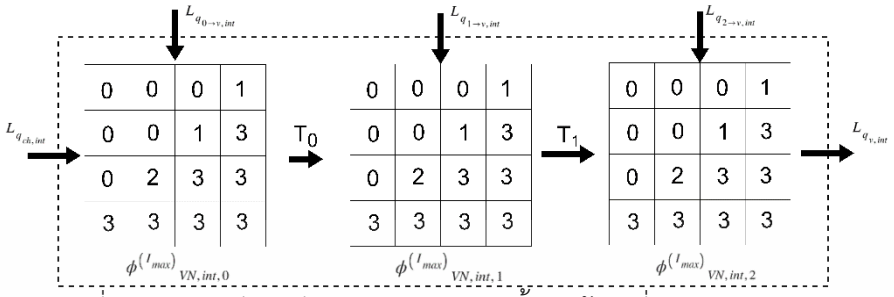
รูปที่ 2.5 การเปรียบเทียบคู่อันดับ LUT ที่ Check Node

นำผลลัพธ์ที่ได้ไปเข้ากระบวนการเปรียบเทียบคู่อันดับโดยใช้ LUT ของ Variable Node ดังรูปที่ 2.6



รูปที่ 2.6 การเปรียบเทียบคู่อันดับ LUT ที่ Variable Node

ผลลัพธ์ที่ได้จะถูกแลกเปลี่ยนระหว่าง Variable Node และ Check Node จนกว่าค่าวนซ้ำ iteration (I) จะถึงค่าที่กำหนด เมื่อถึงแล้วจะทำการแปลงค่าข่าวสารครั้งสุดท้ายด้วยการเปรียบเทียบคู่อันดับครั้งสุดท้ายที่ Variable Node ดังรูปที่ 2.7



รูปที่ 2.7 การเปรียบเทียบคู่อันดับ LUT ครั้งสุดท้าย ที่ Variable Node

จากนั้นนำข่าวสารปที่ได้จากกระบวนการเปรียบเทียบคู่อันดับโดยใช้ LUT ไปเข้ากระบวนการ Hard Decision ซึ่งเป็นกระบวนการพิจารณาข้อมูลโดยใช้เลือกดังสมการที่ (2.26)

2.2 การสร้างสัญญาณสุ่ม

2.2.1 การสร้างสัญญาณสุ่มด้วยวิธีการ Tausworthe Generators

การสร้างสัญญาณรบกวนเริ่มต้นจากการสุ่มข้อมูล โดยวิธีการสุ่มข้อมูลที่ใช้คือแบบ Tausworthe ซึ่งเป็นทฤษฎีที่มีความเหมาะสมต่อการสุ่มข้อมูลเป็นจำนวนหลายชุด แต่ละชุดข้อมูลถูกสร้างโดยการสร้างลำดับสุ่มของบิต และบิตข้อมูลตามจำนวนบิต K บิตตามต้องการ โดยสมการตั้งต้นของวิธีนี้เป็นดังสมการที่ (2.29)

$$b_n = c_{q-1}b_{n-1} \oplus c_{q-2}b_{n-2} \oplus c_{q-3}b_{n-3} \oplus \ldots \oplus c_0 b_{n-q} \qquad (2.29)$$

โดย b_n คือค่าคงที่ของ Tausworthe

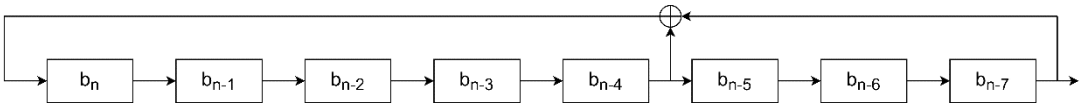
c_i คือตัวแปรเลขฐานสอง

b_i คือตัวแปรเลขฐานสอง

วิธีสุ่มข้อมูลแบบ Tausworthe จะใช้สมการสัญกรณ์พหุนามเพื่อระบุลำดับการสุ่มของบิตดังสมการที่ (2.30)

$$x^7 + x^3 + 1 \qquad (2.30)$$

การสุ่มข้อมูลแบบ Tausworthe จากสมการสัญกรณ์พหุนามดังสมการที่ (2.30) เป็นดังรูปที่ 2.7



รูปที่ 2.8 การสุ่มลำดับบิตข้อมูลแบบ Tausworthe

จากนั้นนำชุดข้อมูลที่สุ่มได้ไปเข้ากระบวนการ Box-Muller ซึ่งเป็นกระบวนการในการกระจายตัวของตัวแปรสุ่ม

2.2.2 การสร้างสัญญาณสุ่มด้วยวิธีการ Box-Muller

การสร้างสัญญาณรบกวนด้วยวิธีการ Box-Muller เป็นวิธีการที่ใช้กันอย่างแพร่หลายในการสุ่มตัวอย่างทางสถิติ และง่ายต่อการนำไปใช้งาน

ทฤษฎี Box-Muller คือ การเก็บจำนวนตัวอย่างแบบสุ่มเพื่อสร้างคู่อันดับที่เป็นอิสระต่อกัน การแปลง Box-Muller มีขอบเขตตั้งแต่ $[0,1]$ และมีรูปแบบดังสมการที่ (2.31) และ (2.32)

$$Z_0 = R\cos(\Theta) = \sqrt{-2U_1}\cos(2\pi U_2) \quad (2.31)$$

$$Z_1 = R\sin(\Theta) = \sqrt{-2U_1}\sin(2\pi U_2) \quad (2.32)$$

โดย Z_0 คือตัวแปรสุ่มอิสระ

Z_1 คือตัวแปรสุ่มอิสระ

U_1 คือตัวอย่างการสุ่มแบบอิสระ

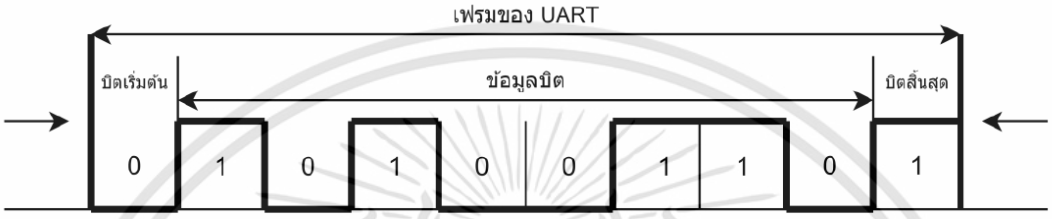
U_2 คือตัวอย่างการสุ่มแบบอิสระ

2.3 UART (Universal Asynchronous Receiver Transmitter)

UART คือโปรโตคอลสื่อสารข้อมูลอนุกรมแบบอะซิงโครนัส โดยผู้ใช้จำเป็นต้องกำหนดรูปแบบที่ใช้ในการรับส่งข้อมูลและกำหนดอัตราเร็วในการรับส่งข้อมูลให้แก่อุปกรณ์ที่ทำการติดต่อสื่อสารให้มีค่าตรงกันจึงสามารถทำการสื่อสารกันได้

2.3.1 เฟรมข้อมูลของ UART

เฟรมข้อมูลของ UART มีการกำหนดบิตเริ่มต้นเป็น 0 และบิตสิ้นสุดเป็น 1 เริ่มส่งข้อมูลจาก Least Signification Bit (LSB) ไป Most Signification Bit (MSB) มีข้อมูลข่าวสารเป็นข้อมูลขนาด 8 บิต อยู่ระหว่างบิตเริ่มต้นและบิตสิ้นสุด แสดงดังรูปที่ 2.9



รูปที่ 2.9 รูปแบบการรับและส่งข้อมูล UART

2.4 โปรแกรมที่ใช้

2.4.1 Xilinx Vivado

Xilinx Vivado เป็นโปรแกรมที่ถูกพัฒนาโดยบริษัท Xilinx ซึ่งเป็นโปรแกรมที่ใช้ในการออกแบบวงจรเพื่อใช้งานกับ FPGA รองรับการสร้างวงจรด้วยภาษา VHDL

2.4.1.1 ภาษา VHDL

VHDL (Very High Speed Integrated Circuit Hardware Description Language) เป็นภาษาที่บรรยายพฤติกรรมการทำงานของวงจรหรือระบบอิเล็กทรอนิกส์ ซึ่งพฤติกรรมดังกล่าวเป็นพฤติกรรมการทำงานของวงจรที่สอดคล้องกับการทำงานทางกายภาพ

VHDL ถูกคิดค้นและมีจุดเริ่มต้นมาจากกระทรวงกลาโหมสหรัฐอเมริกา (Department of Defense: DoD) ในปี ค.ศ.1981 ซึ่งภาษาระดับสูงเช่นเดียวกับภาษา C แต่มีความสามารถที่จะบรรยายพฤติกรรมการทำงานของวงจรดิจิทัล นอกจากนี้ยังเป็นภาษาที่นำไปเขียนเป็นรูปแบบดิจิทัลและรวมโครงสร้างของระบบดิจิทัลได้

2.4.2 Xilinx Vivado HLS

Xilinx Vivado HLS เป็นโปรแกรมที่ถูกพัฒนาโดยบริษัท Xilinx ซึ่งเป็นโปรแกรมที่ใช้ในการออกแบบวงจรเพื่อใช้งานกับ FPGA รองรับการสร้างวงจรด้วยภาษา C และ C++ โดยการแปลงโปรแกรมภาษาดังกล่าวให้กลายเป็นภาษา VHDL

2.4.2.1 ภาษา C++

C++ คือภาษา C Programming language รุ่นใหม่ ถูกพัฒนาโดย Dr.Bjarne Stroustrup ซึ่งเป็นนักวิจัยอยู่ที่ห้องปฏิบัติการ Bell Labs ประเทศสหรัฐอเมริกาในระหว่างปี พ.ศ. 2525-2528 ภาษา C++ เกิดจากแนวคิดในการเพิ่มประสิทธิภาพภาษา C โดยได้นำความสามารถของภาษา C มาพัฒนาให้เป็นภาษาที่มีความเป็นโปรแกรมเชิงวัตถุ (Object Oriented Programming) ทุกสิ่งที่ภาษา C ทำได้ ภาษา C++ ก็สามารถทำได้เช่นกัน แต่บางสิ่งที่ภาษา C++ ทำได้ ภาษา C อาจทำไม่ได้ ภาษา C++ ถูกออกแบบมาสำหรับการทำงานภายใต้ระบบปฏิบัติการ UNIX

2.4.3 Visual Studio Code

2.4.3.1 ภาษา Python

Python คือภาษาการเขียนโปรแกรมระดับสูง ที่นำข้อดีของภาษาต่าง ๆ มารวมไว้ด้วยกัน ถูกออกแบบมาให้เรียนรู้ได้ง่าย และมีไวยากรณ์ที่ช่วยให้เขียนโปรแกรมสั้นกว่าภาษาอื่น ๆ มีการจัดการหน่วยความจำอัตโนมัติ สนับสนุนกระบวนทัศน์การเขียนโปรแกรม (Programming paradigms) ประกอบด้วย การเขียนโปรแกรมเชิงวัตถุ (OOP : Object Oriented Programming) การเขียนโปรแกรมเชิงคำสั่ง (Imperative Programming) และการเขียนโปรแกรมเชิงฟังก์ชัน (Functional)

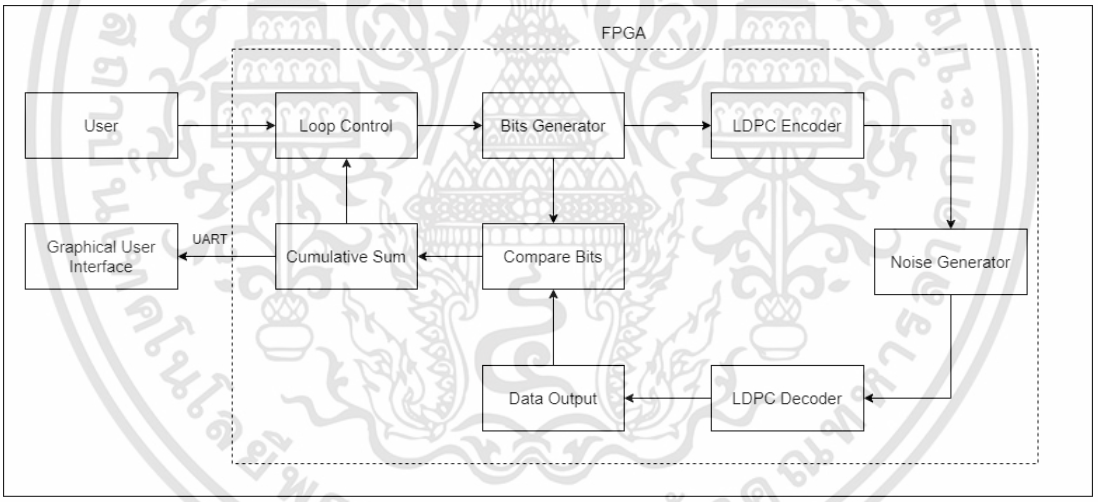
2.4.4 Quartus Prime

เป็นโปรแกรมวิเคราะห์และสังเคราะห์วงจรที่ด้วยภาษา Hardware Description Language (HDL) ซึ่งช่วยให้นักพัฒนาฮาร์ดแวร์สามารถทำการวิเคราะห์แผนภาพ RTL และจำลองการทำงานของวงจรที่ออกแบบต่อการรับอินพุตที่แตกต่างกัน ภาษาที่รองรับประกอบไปด้วย VHDL และ Verilog

บทที่ 3

การออกแบบและการจัดทำปริญญานิพนธ์

การออกแบบและสร้างวงจรรับส่งข้อมูลตามมาตรฐาน CCSDS บน FPGA ด้วยภาษา C++ และ VHDL ซึ่งกระบวนการทำงานของระบบเริ่มต้นจากการควบคุมการทำงานจากผู้ใช้ เพื่อสั่งการวงจรควบคุมการวนซ้ำให้ไปควบคุมวงจรสร้างสัญญาณสุ่มให้สุ่มข่าวสารเพื่อทำการเข้ารหัสที่วงจรเข้ารหัส LDPC จากนั้นนำคำรหัสที่ได้ไปผ่านวงจรสร้างสัญญาณรบกวน และนำมาถอดรหัสที่วงจรถอดรหัส LDPC นำข่าวสารที่ได้ไปเปรียบเทียบกับข่าวสารที่ได้จากวงจรสร้างสัญญาณสุ่มเพื่อให้ได้จำนวนบิตผิดพลาด เก็บค่าจำนวนบิตผิดพลาดที่ได้ไว้ที่วงจรผลรวมสะสมเพื่อนำมาบวกค่าจำนวนบิตผิดพลาดของการวนซ้ำถัดไป ทำเช่นเดิมจนครบจำนวนการวนซ้ำ และส่งค่าผลรวมบิตผิดพลาดสะสมไปแสดงผลผ่าน Terminal Program แสดงดังรูปที่ 3.1



รูปที่ 3.1 ภาพรวมการออกแบบวงจรตามตรฐาน CCSDS

3.1 การออกแบบชนิดข้อมูลแบบ Floating Point บนบอร์ด FPGA

เนื่องจากจำนวนจริง (Real Number) ที่เป็นจำนวนอตรรกยะ เช่น ค่าพาย (π) เป็นต้น ซึ่งเป็นไปไม่ได้ที่คอมพิวเตอร์จะสามารถเก็บจำนวนจริงบางจำนวนได้ในหน่วยความจำที่จำกัด แต่คอมพิวเตอร์สามารถเก็บจำนวนจุดลอยตัว (Floating Point) ซึ่งเป็นการประมาณค่าจำนวนจริง

โดยใช้ค่าแมนทิสซา (Mantisa) และค่าชี้กำลัง (Exponent) ที่มีจำนวนบิตจำกัด เช่น 1.2345 แทนได้ด้วย 12345×10^{-4} ค่าแมนทิสซาคือ 12345 และค่าชี้กำลังคือ -4 ตามมาตรฐาน IEEE 754 การเก็บจำนวนจุดลอยตัวเป็นแบบความแม่นยำเดี่ยว (Single precision) ขนาด 32 บิต แบ่งเป็น 3 ส่วน ดังนี้ [4]

1) บิตที่สำคัญที่สุด (MSB) ขนาด 1 บิตเรียกว่าบิตเครื่องหมาย (Sign bit)

2) 8 บิตถัดมาแทนค่าชี้กำลัง

3) 23 บิตสุดท้ายแทนค่าแมนทิสซา

การแปลงเลขจำนวนจุดลอยตัวดังกล่าวให้กลายเป็นเลขฐานสิบมีขั้นตอนดังนี้

1) สร้างจำนวนจุดตรึงขึ้นมาก่อน โดยนำแมนทิสซาไปต่อท้ายเลข เช่น 1 แมนทิสซาคือ 01101 เป็น 1.01101

2) นำจำนวนแบบจุดตรึงที่ได้ไปคูณกับ $2^{(\exponent-127)}$ สำหรับเลขฐานสอง การคูณสองคือการเลื่อนบิตทั้งหมดไปทางซ้าย 1 หลัก การหารสองคือการเลื่อนบิตทั้งหมดไปทางขวา 1 หลัก

3) บิตแสดงเครื่องหมายเป็น 0 จะแทนค่าเป็นจำนวนบวก กรณีบิตแสดงเครื่องหมายเป็น 1 จะแทนค่าเป็นจำนวนลบ

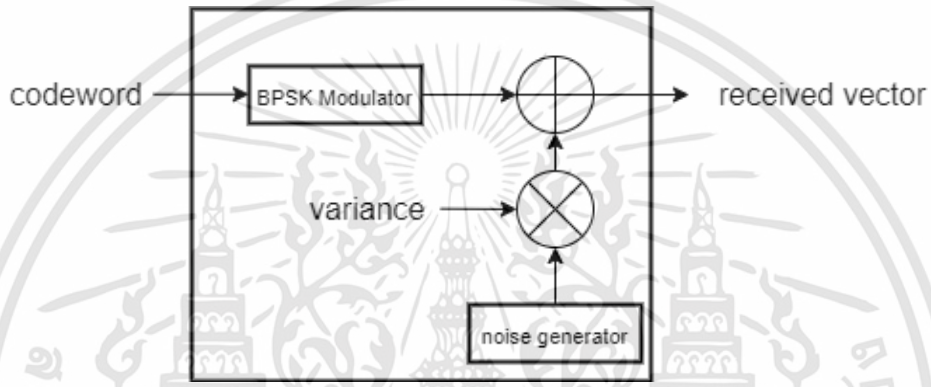
ข้อยกเว้นของค่าชี้กำลังและค่าแมนทิสซาเป็นดังตารางที่ 3.1

ตารางที่ 3.1 ข้อยกเว้นของค่าชี้กำลังและค่าแมนทิสซา

Exponent	Mantisa	ความหมาย
0	0	ค่าศูนย์
$2^8-1(11111111)$	0	Infinity
$2^8-1(11111111)$	Non zero	NaN (not a number)
0	Non zero	Denormalized number
1 ถึง 2^8-2	Any	Normalized number

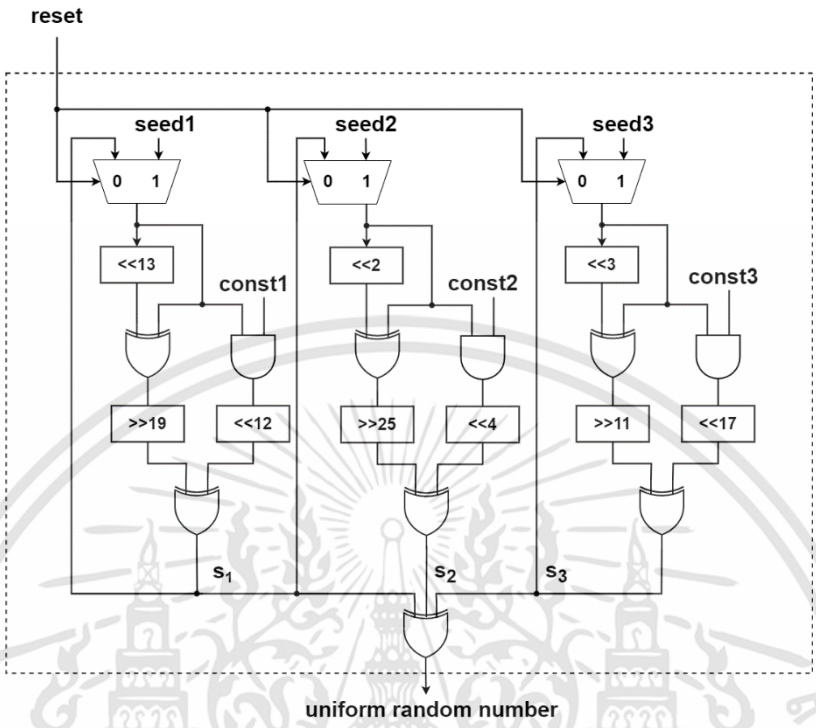
3.2 วงจรสร้างสัญญาณสุ่มบนบอร์ด FPGA

วงจรสร้างสัญญาณสุ่มมีหน้าที่ในการจำลองสัญญาณสุ่มเพื่อทำให้คำรหัสที่ส่งมาจากวงจรเข้ารหัสมีความคลาดเคลื่อนเสมือนการส่งสัญญาณจริง ๆ จากภาคส่งไปยังภาครับโดยการสร้างสัญญาณสุ่มมาคูณกับค่าความแปรปรวน และบวกกับคำรหัสที่ได้รับจากวงจรเข้ารหัส ประมวลผลออกมาเป็นสัญญาณเวกเตอร์เพื่อส่งให้วงจรถอดรหัสดังรูป 3.2



รูปที่ 3.2 ภาพรวมของวงจรสร้างสัญญาณสุ่ม

สร้างวงจรสร้างสัญญาณด้วยวงจรการสุ่มข้อมูล 3 วงจร กำหนดค่าเริ่มต้น 3 ชุด ชุดละ 32 บิต เพื่อมาทำการเข้ากระบวนเลื่อนบิตดังรูปที่ 3.3



รูปที่ 3.3 โครงสร้างของวงจรสุ่มข้อมูล

นำข้อมูลที่ผ่านกระบวนการเลื่อนบิตข้อมูลทั้ง 3 ชุดมาทำการบวกแบบไม่คิดตัวทด (*XOR*) จะได้ชุดข้อมูลสุ่มมา 1 ชุด ชุดละ 32 บิต และวงจรสุ่มข้อมูลจะทำงานซ้ำไปเรื่อย ๆ ตามจำนวนที่ต้องการดังสมการที่ (3.1)

$$ramdom_n = s_1 \text{xor} s_2 \text{xor} s_3 \tag{3.1}$$

โดย $random_n$ คือชุดข้อมูลสุ่ม

s_1 คือข้อมูลชุดที่ 1 จากกระบวนการเลื่อนบิตข้อมูล

s_2 คือข้อมูลชุดที่ 2 จากกระบวนการเลื่อนบิตข้อมูล

s_3 คือข้อมูลชุดที่ 3 จากกระบวนการเลื่อนบิตข้อมูล

ทำการแปลงชุดข้อมูลสุ่มให้มีค่าระหว่าง 0 กับ 1 โดยการนำชุดข้อมูลสุ่มหารด้วยค่าสูงสุดที่เป็นไปได้ของชุดบิตดังสมการที่ (3.2)

$$rand = \frac{random_n + 1}{randMax + 1} \tag{3.2}$$

โดย $rand$ คือชุดข้อมูลสุ่มที่มีค่าระหว่าง 0 กับ 1

$random_n$ คือชุดข้อมูลสุ่ม

$randMax$ คือค่าสูงสุดที่เป็นไปได้ของชุดบิต 32

นำชุดข้อมูลสุ่มที่มีค่าระหว่าง 0 กับ 1 เข้ากระบวนการแปลงเป็นสัญญาณสุ่มดังสมการที่ (3.3)

$$noise = \sqrt{-2 \times \log(rand_n)} \times \cos(2\pi \times rand_{n+1}) \quad (3.3)$$

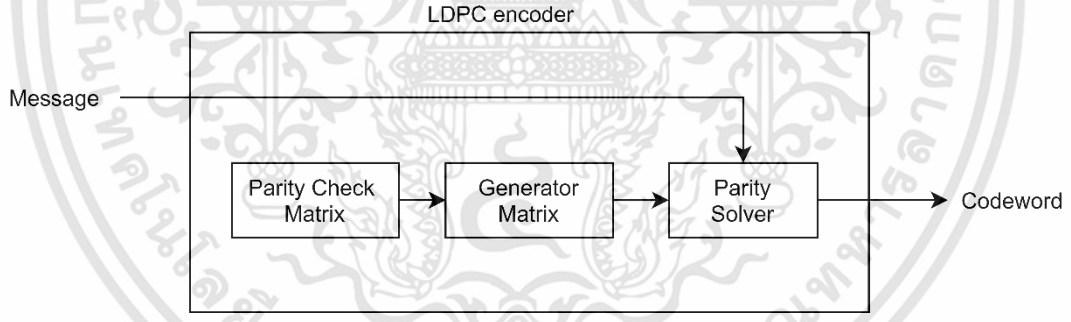
โดย $noise$ คือสัญญาณสุ่ม

$rand_n$ คือชุดข้อมูลสุ่มที่มีค่าระหว่าง 0 กับ 1

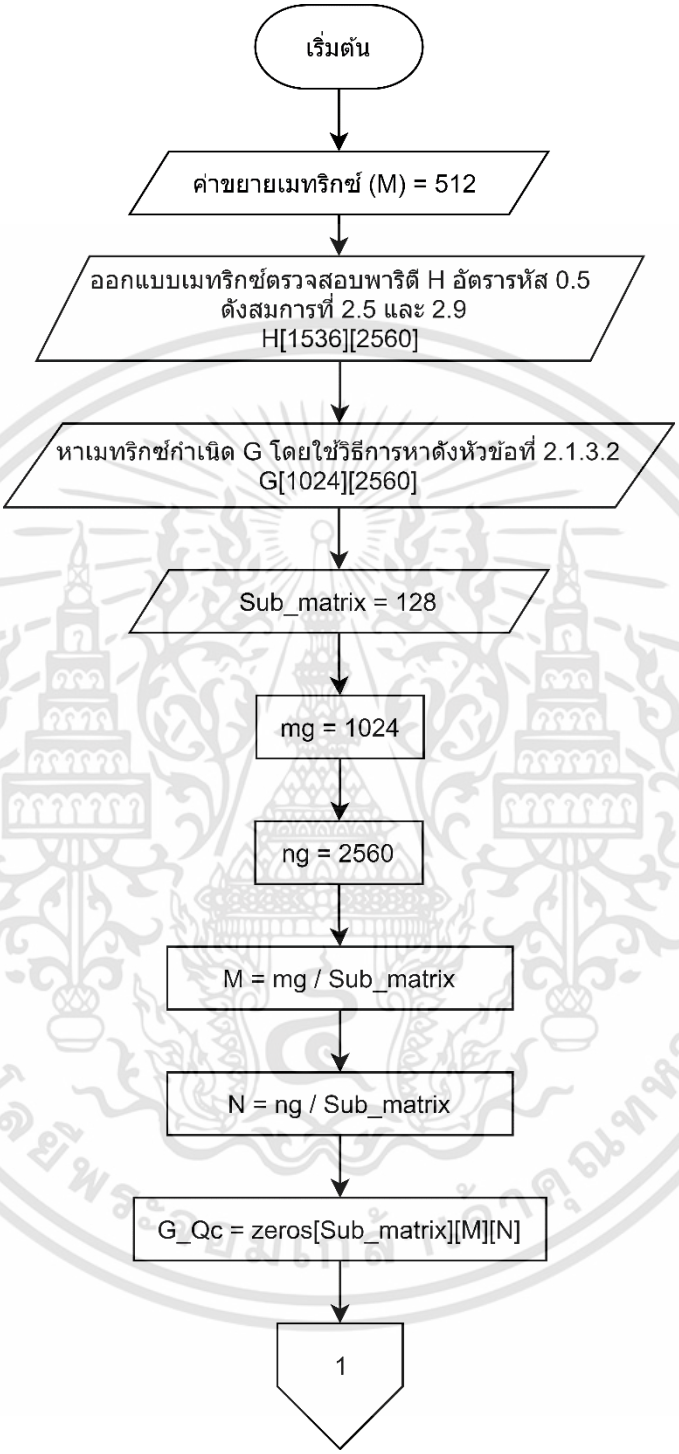
$rand_{n+1}$ คือชุดข้อมูลสุ่มที่มีค่าระหว่าง 0 กับ 1 ที่ทำการสุ่มครั้งถัดไป

3.3 วงจรเข้ารหัส LDPC บนบอร์ด FPGA

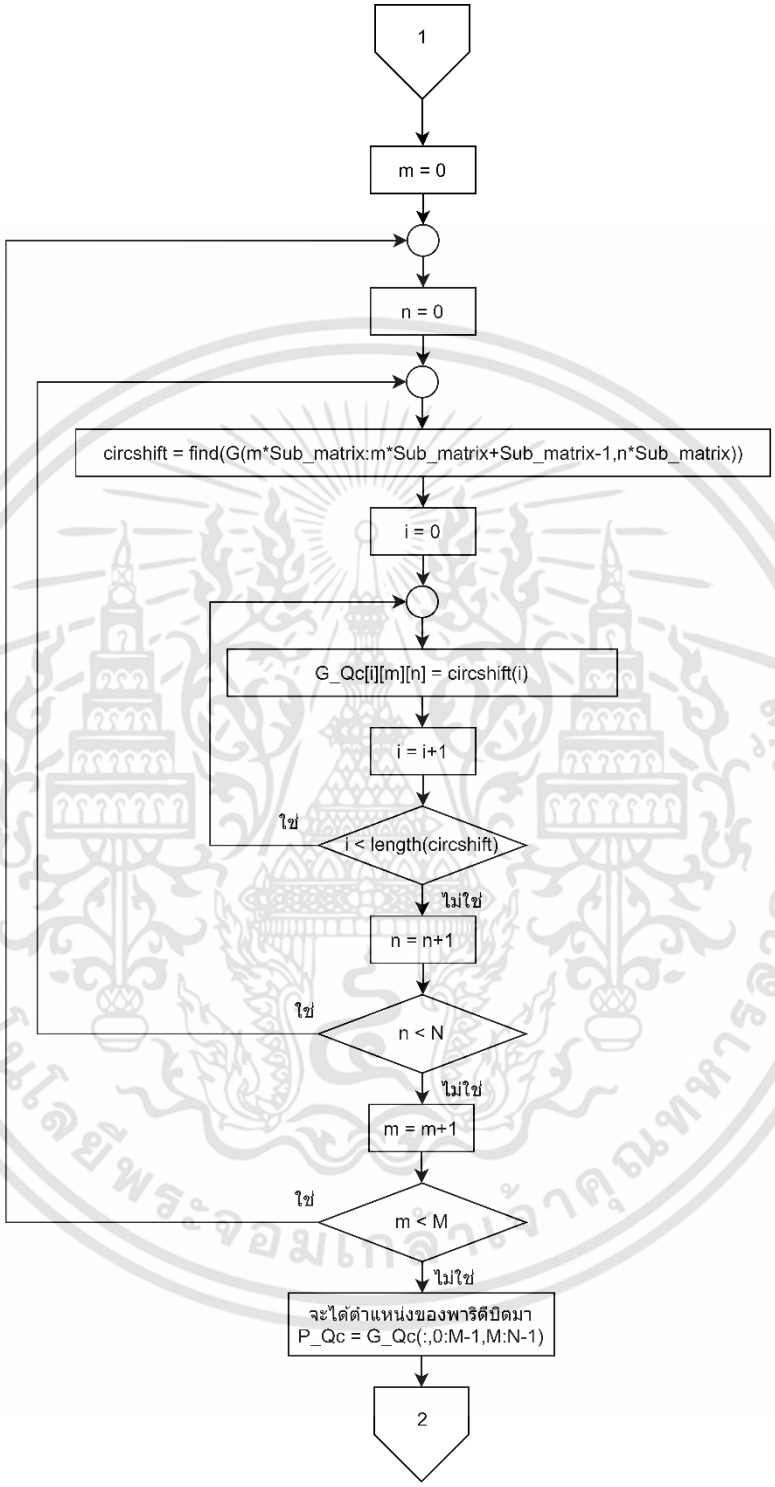
การทำงานของวงจรเข้ารหัส LDPC คือการนำข้อมูลอินพุตจาก Bits Generator 1024 บิต มาผ่านวงจรเข้ารหัส LDPC เพื่อที่จะได้คำรหัสออกมาดังรูปที่ 3.4 และมีผังงานกระบวนการทำงานดังรูปที่ 3.5



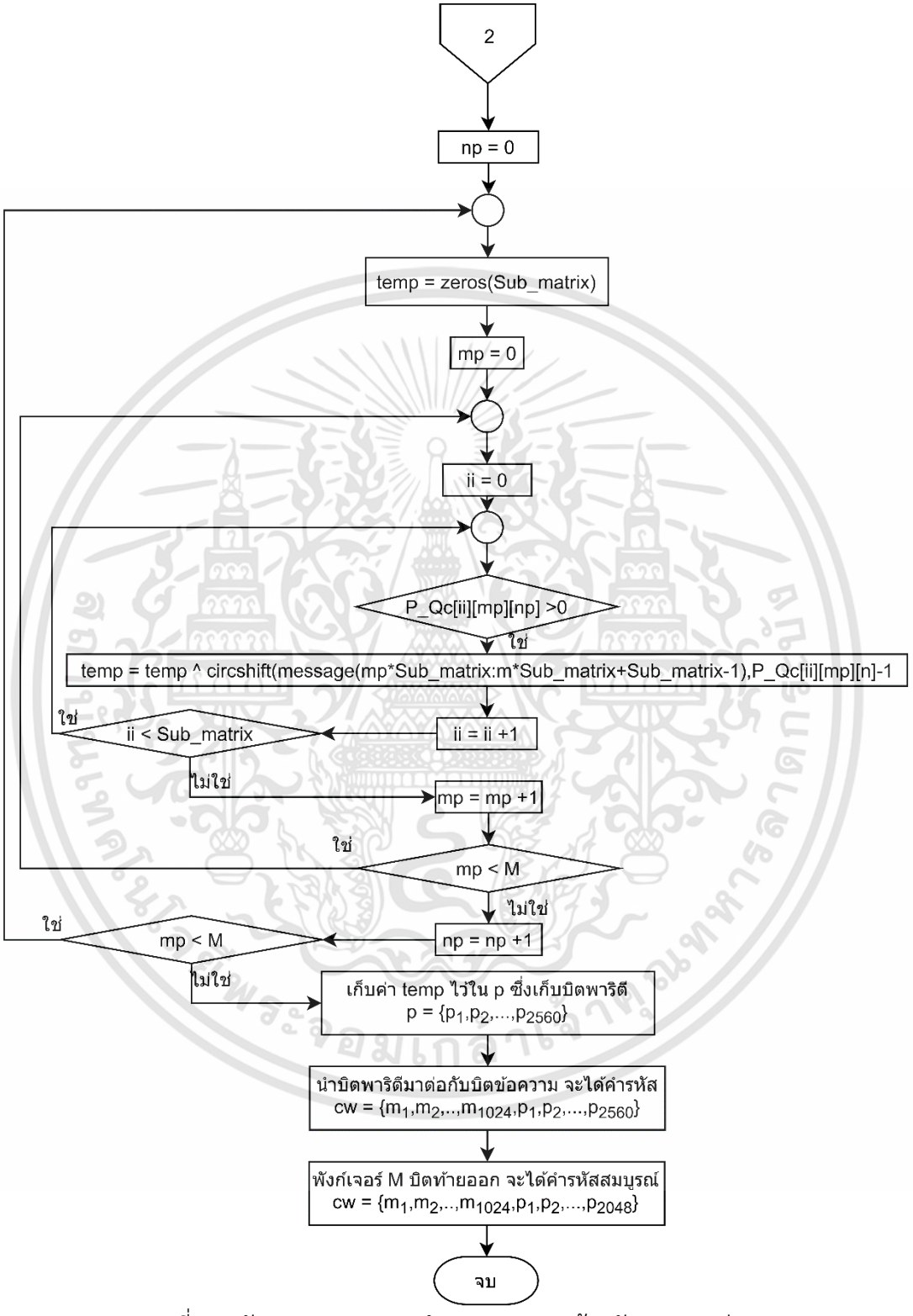
รูปที่ 3.4 ภาพรวมการทำงานของวงจรเข้ารหัส LDPC



รูปที่ 3.5 ผังงานกระบวนการทำงานของวงจรเข้ารหัส LDPC



รูปที่ 3.6 ผังงานกระบวนการทำงานของวงจรเข้ารหัส LDPC (ต่อ)



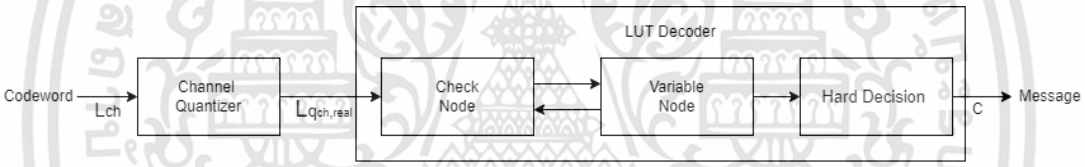
รูปที่ 3.7 ผังงานกระบวนการทำงานของวงจรเข้ารหัส LDPC (ต่อ)

จากรูปที่ 3.5 ถึง 3.7 คือผังกระบวนการจัดรูปเมทริกซ์ตรวจสอบพาริตีตามมาตรฐานของ CCSDS โดยใช้อัตรารหัส 0.5 และใช้ค่าขยายเมทริกซ์เท่ากับ 512 จัดรูปเมทริกซ์ตรวจสอบพาริตีโดยใช้วิธีเกาส์ ให้อยู่ในรูปแบบดังสมการที่ (2.4)

จากนั้นได้ทำการหาตำแหน่งของบิตพาริตี เพื่อนำมาหาบิตพาริตีว่าแต่ละบิตมีค่า 0 หรือ 1 โดยจะมีทั้งหมด 1536 บิต และได้นำบิตพาริตีมาต่อกับบิตข้อความแล้วทำพังก์เจอร์โดยตัดบิตสุดท้ายออกทั้งหมด 512 บิต จึงได้คำรหัสที่สมบูรณ์

3.4 วงจรถอดรหัส LDPC บนบอร์ด FPGA

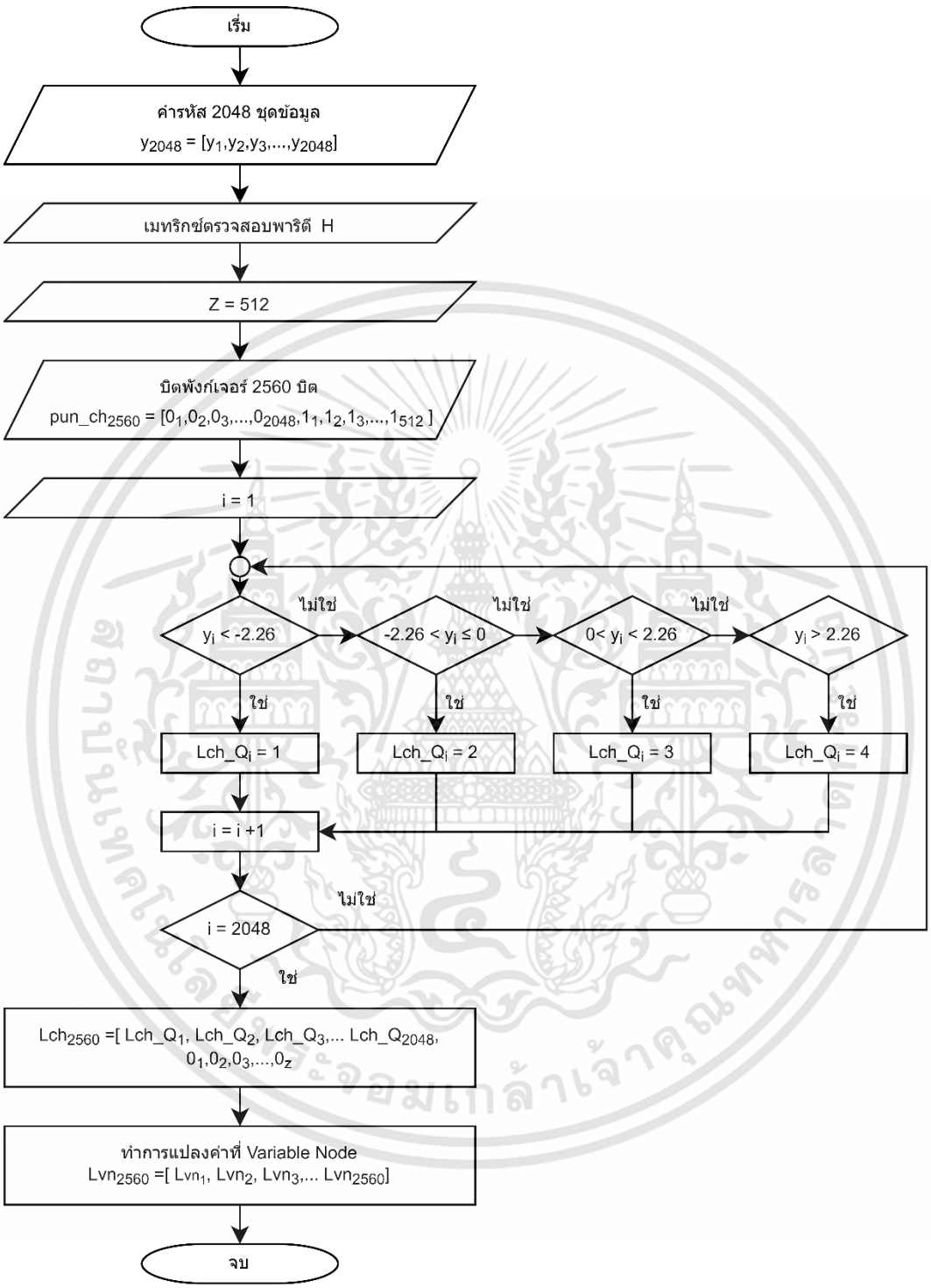
การสร้างวงจรถอดรหัส LDPC สำหรับมาตรฐาน CCSDS ชนิด AR4JA ด้วยการใช้ Lookup Table (LUT) เพื่อใช้ในการหาผลลัพธ์คำรหัสจาก Check Node และ Variable Node ที่กำหนดโดยเมทริกซ์ตรวจสอบพาริตี วงจรถอดรหัส LDPC มีโครงสร้างดังรูปที่ 3.8



รูปที่ 3.8 โครงสร้างของวงจรถอดรหัส LDPC ชนิด AR4JA โดยใช้ LUT

3.4.1 วงจรควอนไทซ์

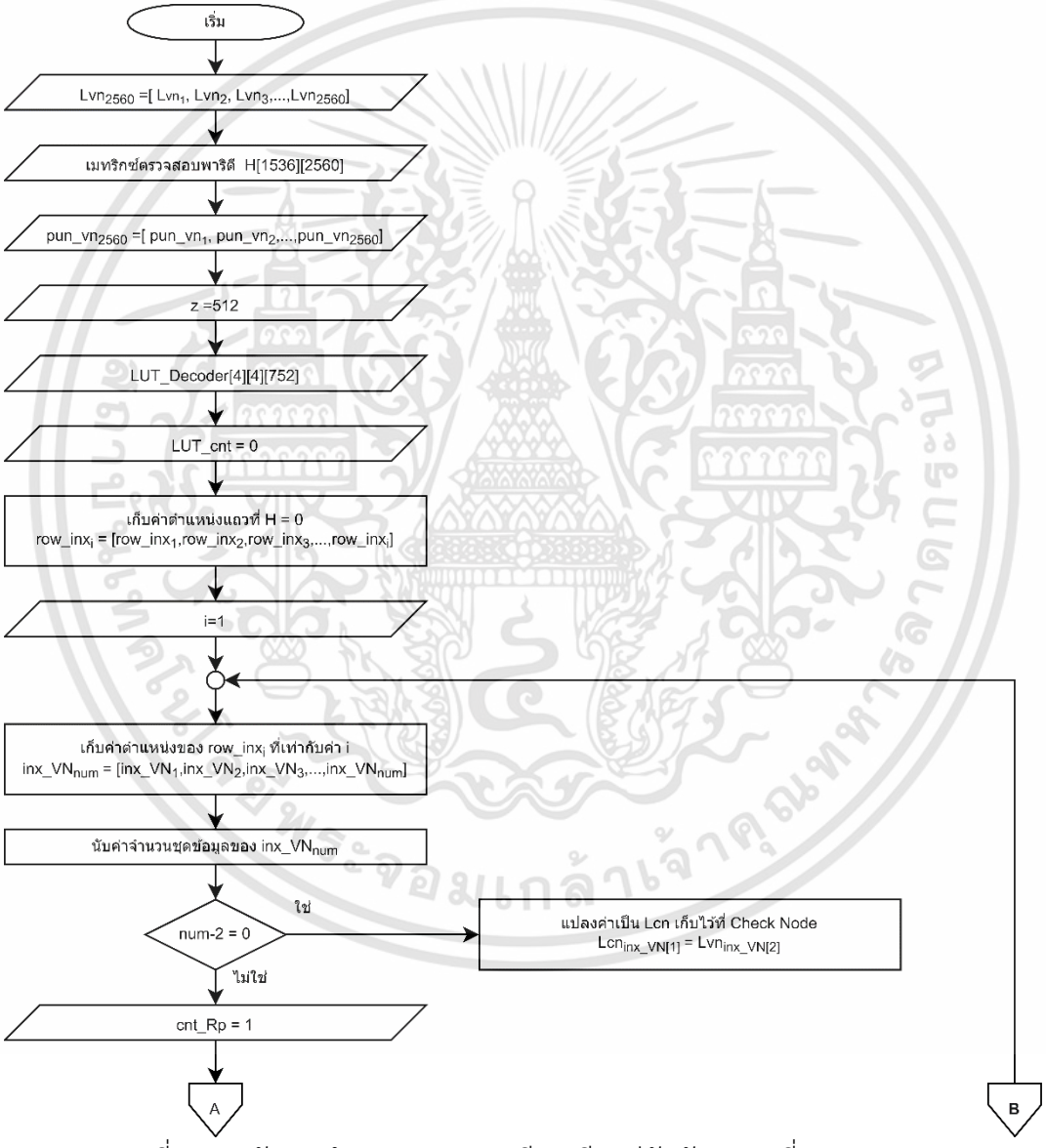
วงจรควอนไทซ์มีหน้าที่แปลงข้อมูลที่เป็นจำนวนจริงให้กลายเป็นจำนวนเต็ม เริ่มต้นทำงานจากการรับชุดข้อมูลคำรหัสทั้งหมด 2048 ชุด ที่ได้รับจากวงจรสร้างสัญญาณรบกวนมาทำการควอนไทซ์เพื่อให้ได้เป็นจำนวนเต็ม จากนั้นนำข้อมูลที่ได้มาเติมบิตพังก์เจอร์ 512 บิต จนครบ 2560 บิต และทำการแปลงค่าแล้วเก็บไว้ที่ Variable Node ดังกระบวนการในรูปที่ 3.9



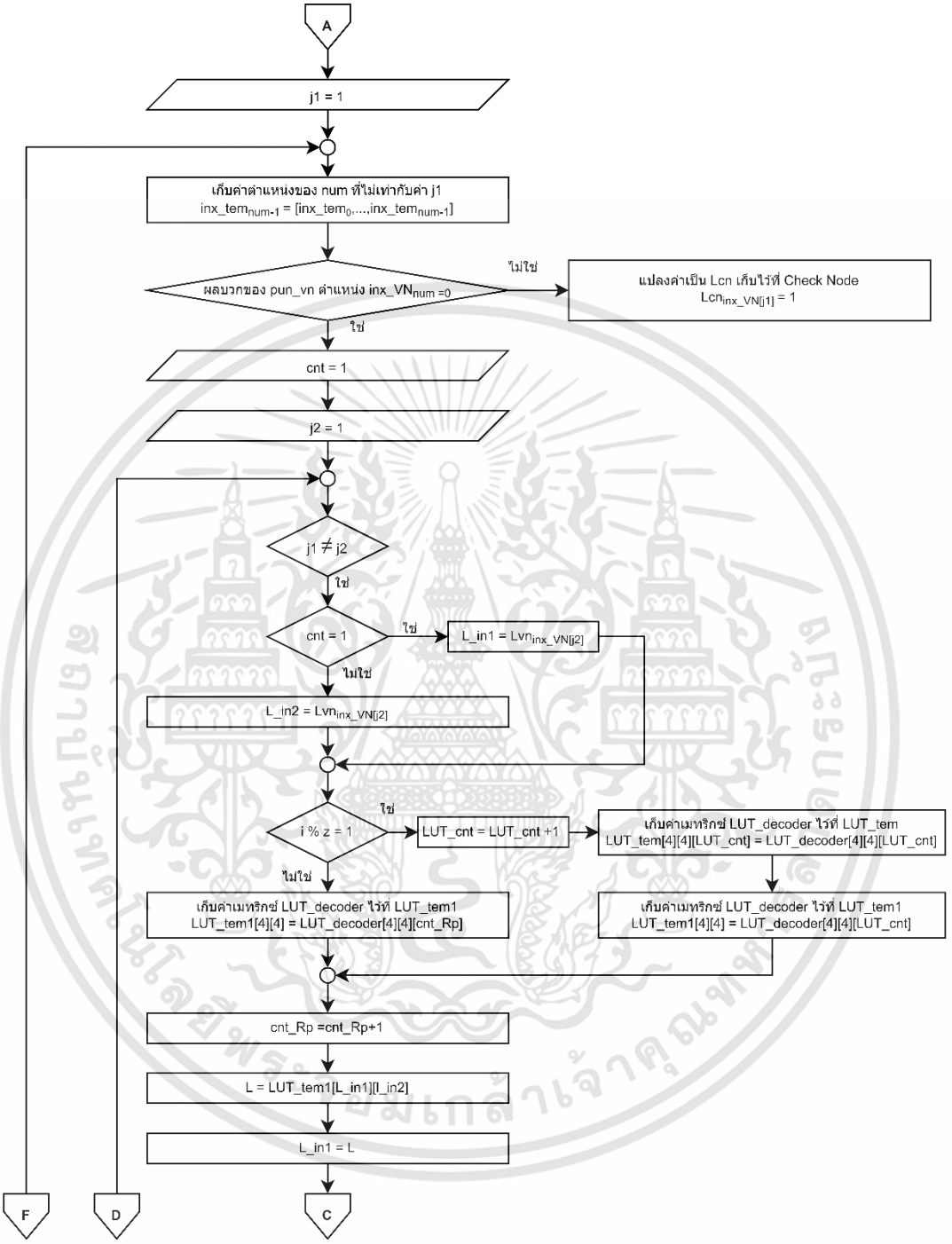
รูปที่ 3.9 ผังการทำงานของวงจรควอนไทซ์

3.4.2 วงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node

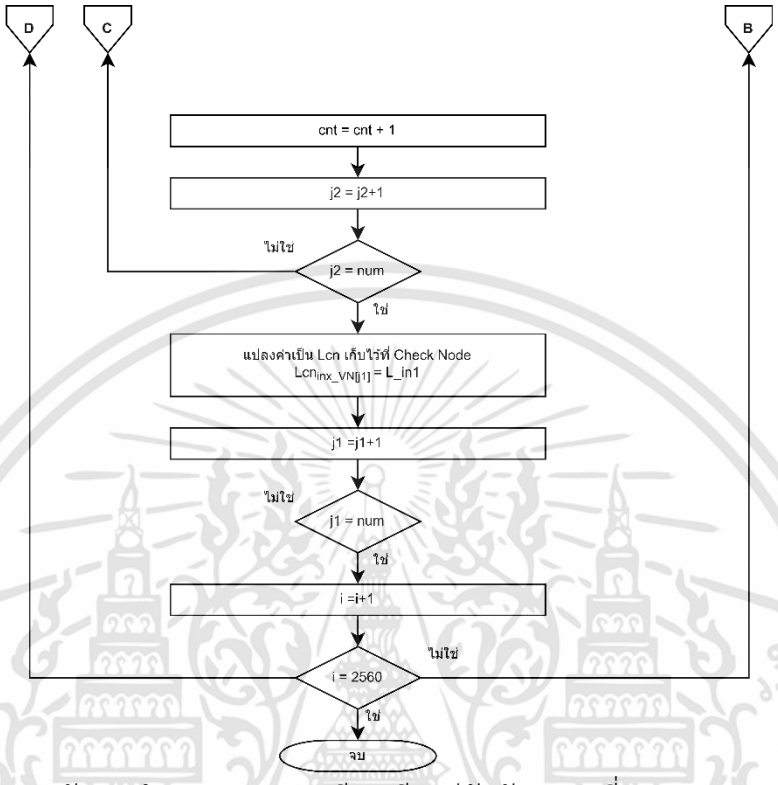
วงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node จะนำ Lvn ที่ได้จากการวงจรควอนไทซ์ที่เก็บค่าไว้ที่ Variable Node มาทำการถอดรหัสโดยการเปรียบเทียบคู่อันดับทีละ Check Node เพื่อให้ได้ผลลัพธ์เป็นค่า Lcn แล้วไปเก็บไว้ที่ Check Node ดังกระบวนการในรูปที่ 3.10 ถึง 3.12



รูปที่ 3.10 ผังการทำงานของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node



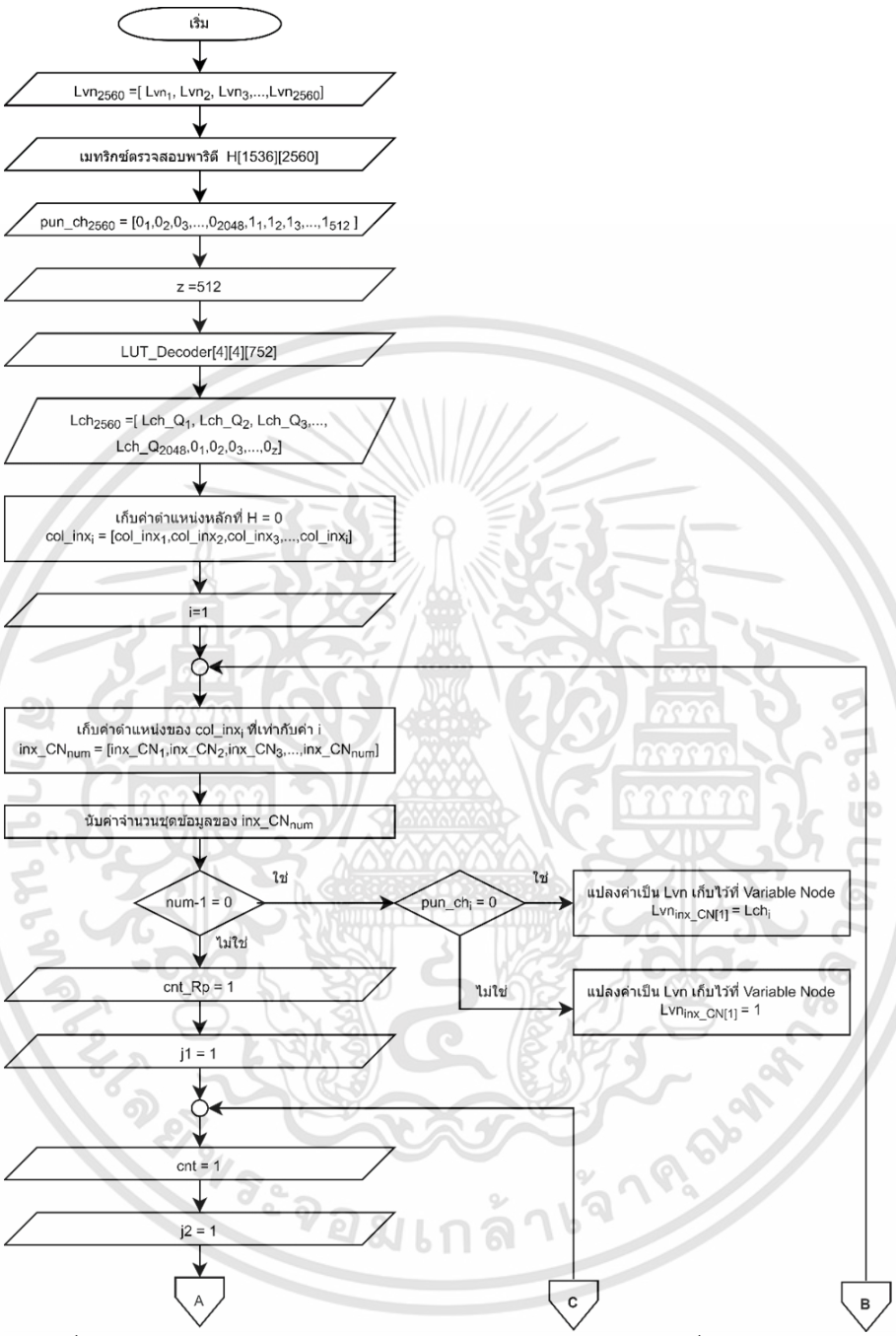
รูปที่ 3.11 ผังการทำงานของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node (ต่อ)



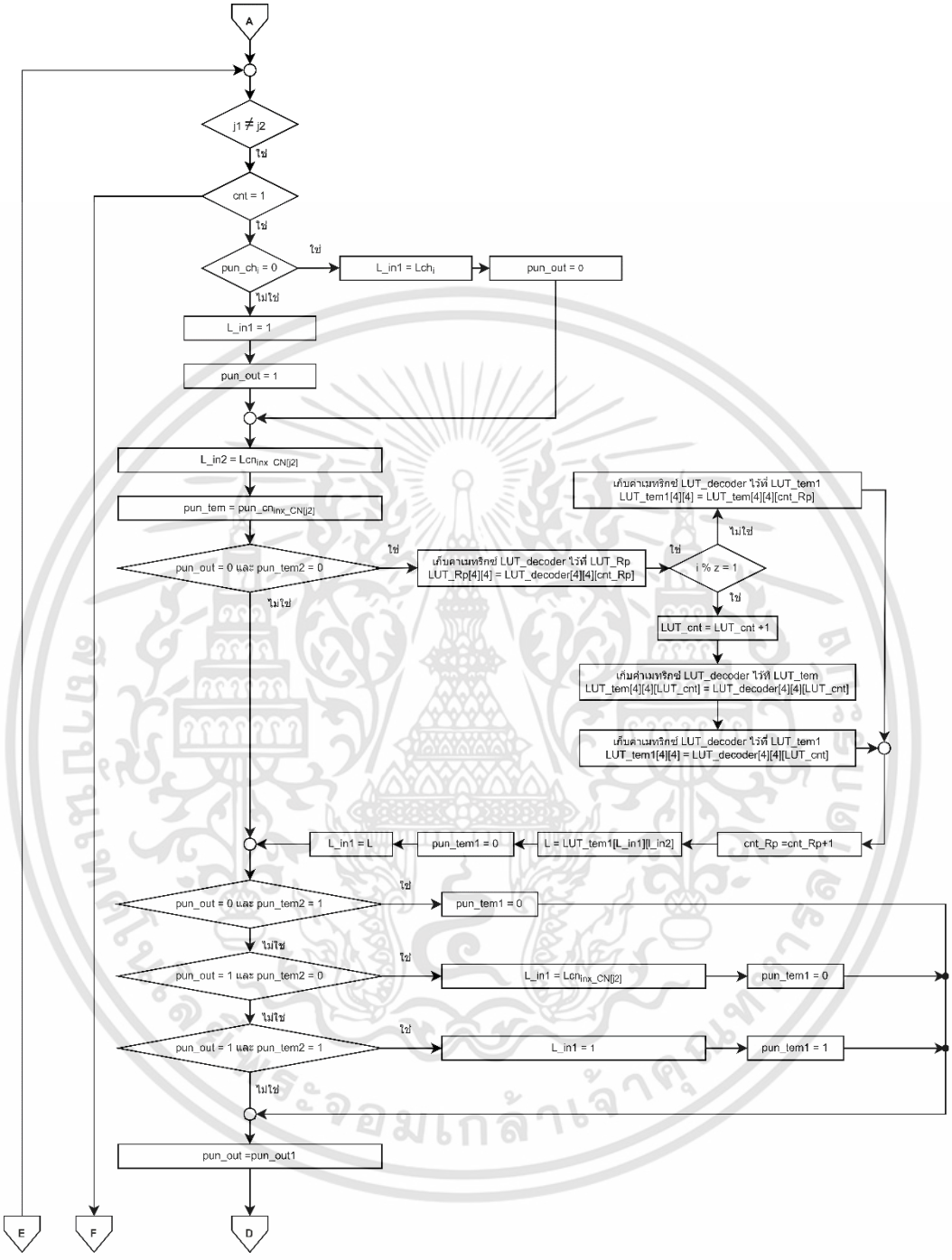
รูปที่ 3.12 ผังการทำงานของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node (ต่อ)

3.4.3 วงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node

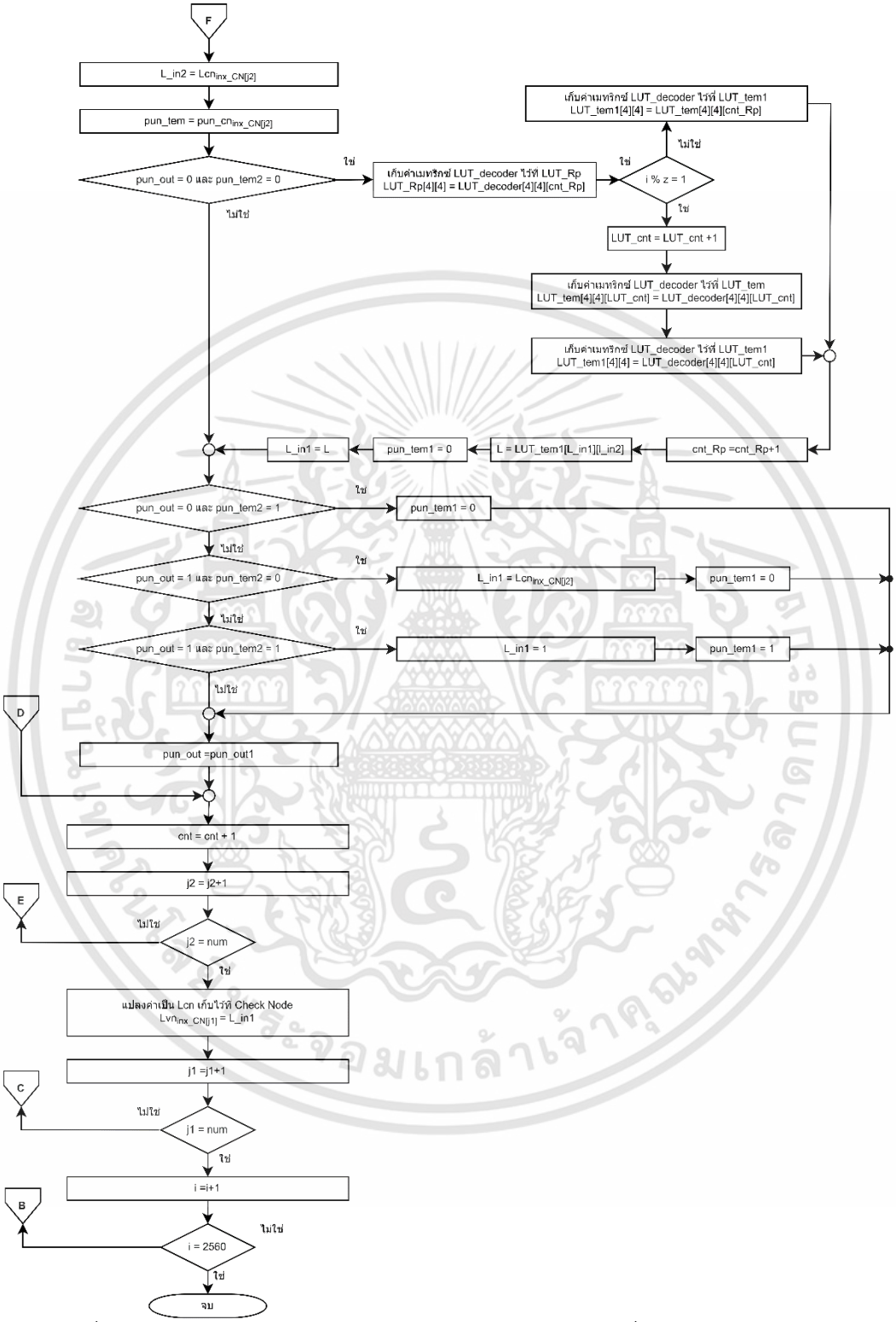
วงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node เริ่มต้นทำงานจากการรับค่า Lcn จากวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node จากนั้นทำการถอดรหัสให้อยู่ในรูปของ Lvn เก็บค่าไว้ที่ Variable Node โดยต้องเปรียบเทียบคู่อันดับของ Variable Node แรกก่อนเสมอเพื่อเป็นค่าตั้งต้นในการเปรียบเทียบคู่อันดับที่ Variable Node ถัดไปดังกระบวนการในรูปที่ 3.13 ถึง 3.15



รูปที่ 3.13 ผังการทำงานของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node



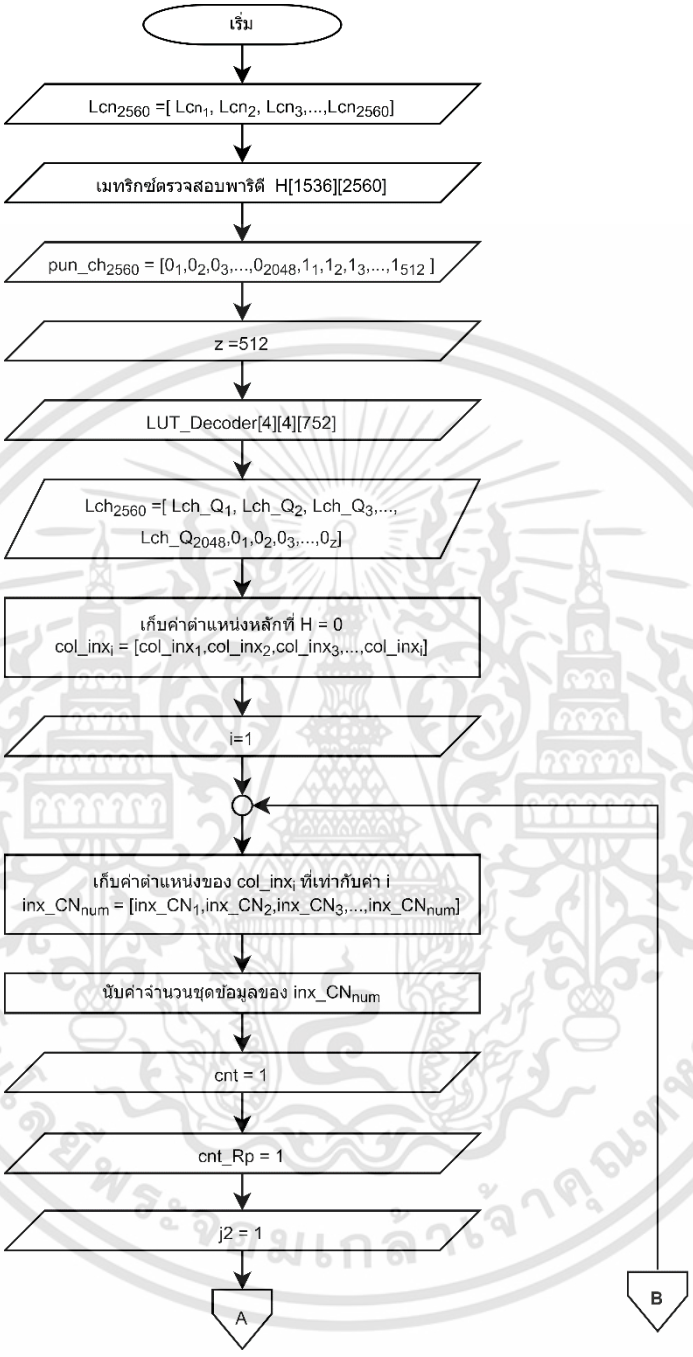
รูปที่ 3.14 ผังการทำงานของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node (ต่อ)



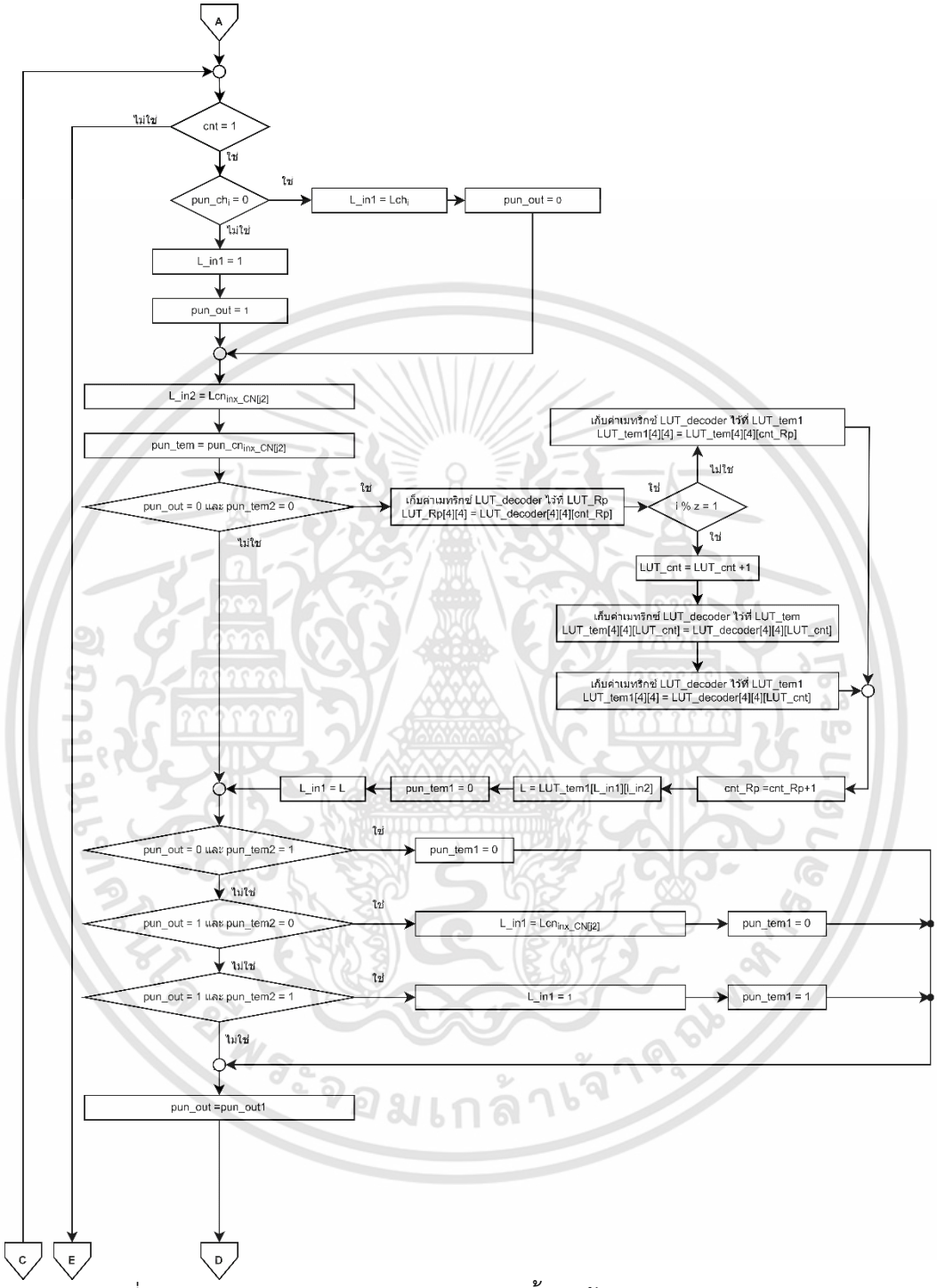
รูปที่ 3.15 ผังการทำงานของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node (ต่อ)

3.4.4 วงจรถอดรหัสครั้งสุดท้าย ที่ Variable Node

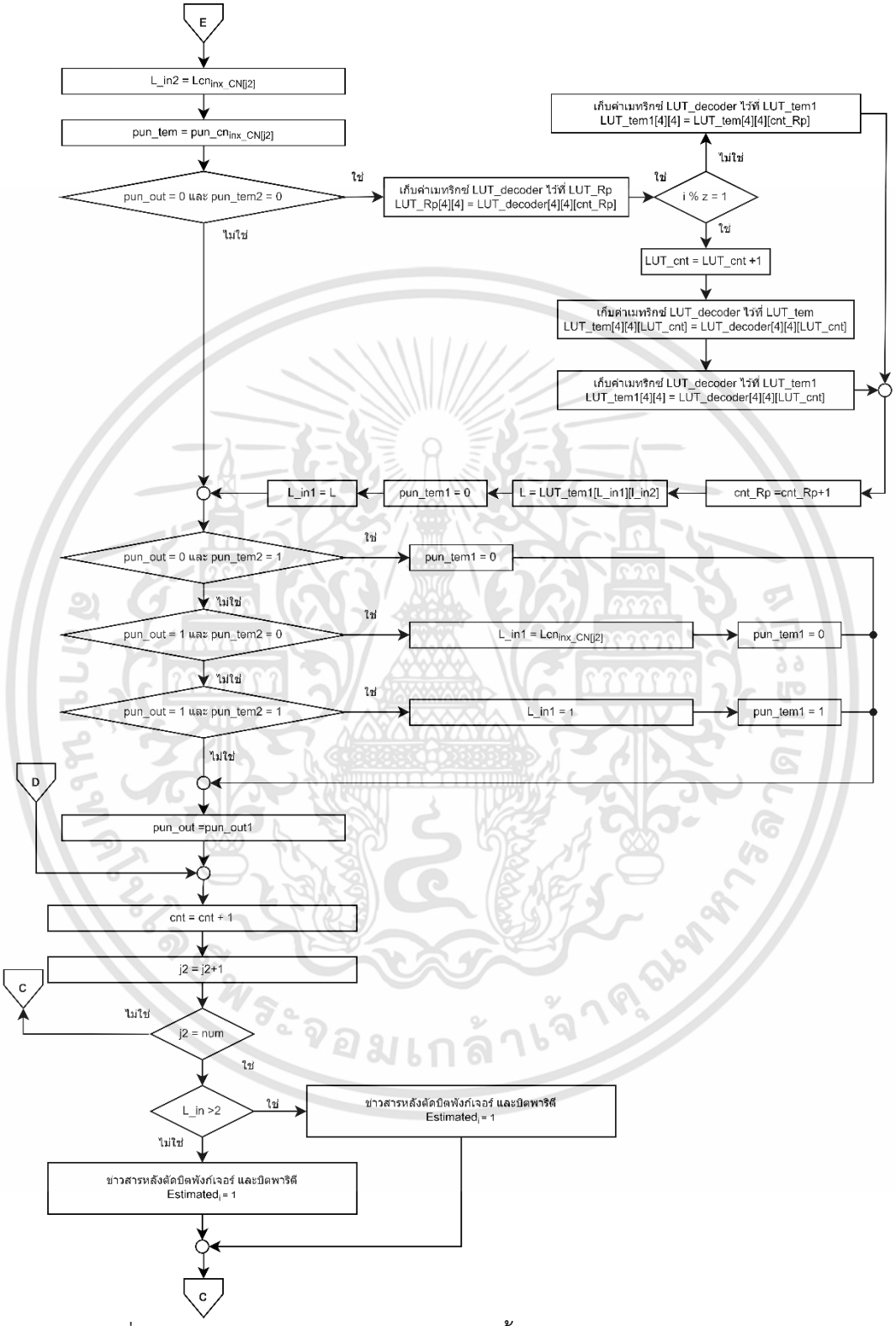
ผลลัพธ์ Lvn และ Lcn ที่ได้จะถูกแลกเปลี่ยนระหว่าง Variable Node และ Check Node จนกว่าค่าวนซ้ำ Iteration (I) จะถึงค่าที่กำหนด เมื่อถึงแล้วจะทำการแปลงค่าข่าวสารครั้งสุดท้ายด้วยการเปรียบเทียบคู่อันดับครั้งสุดท้ายที่ Variable Node จากนั้นนำข่าวสารที่ได้จากกระบวนการเปรียบเทียบคู่อันดับ LUT ไปเข้ากระบวนการ Hard Decision ซึ่งเป็นกระบวนการพิจารณาข้อมูลโดยใช้เลือกใช้เกณฑ์คงที่มาทำการวิเคราะห์ว่าบิตที่กำลังทำการถอดรหัสควรเป็น 1 หรือ 0 หาก L_in มากกว่าค่า ‘2’ จะถอดรหัสเป็น 1 และหากน้อยกว่าหรือเท่ากับ 2 จะถอดรหัสเป็น 0 ข่าวสารที่ได้จะเป็นชุดข้อมูลไบนารี่ 2560 บิต ทำการตัดบิตพังก์เจอร์ 512 บิตท้าย และตัดบิตพาริตี 1024 บิตท้าย จะได้ข้อมูลที่ถอดรหัสออกได้ทั้งหมด 1024 บิตดังกระบวนการในรูปที่ 3.16 ถึง 3.19



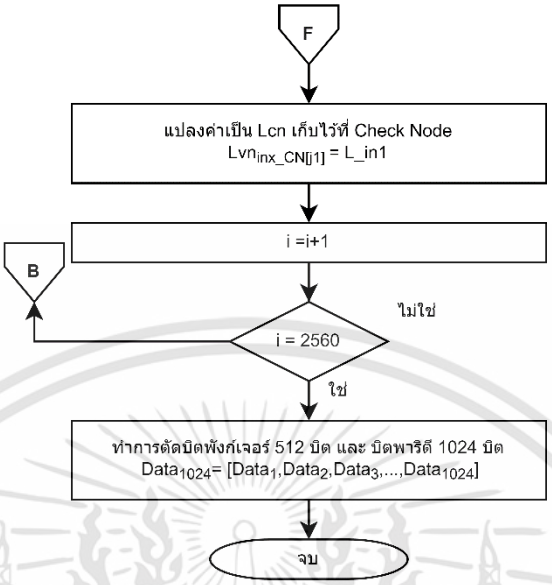
รูปที่ 3.16 ผังการทำงานของวงจรถอดรหัสครั้งสุดท้าย Variable Node



รูปที่ 3.17 ผังการทำงานของวงจรถอดรหัสครั้งสุดท้าย Variable Node (ต่อ)



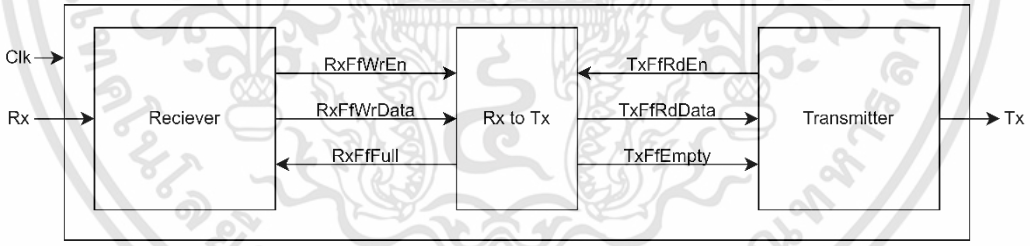
รูปที่ 3.18 ผังการทำงานของวงจรถอดรหัสครั้งสุดท้าย Variable Node (ต่อ)



รูปที่ 3.19 ผังการทำงานของวงจรถอดรหัสครั้งสุดท้าย Variable Node (ต่อ)

3.5 การออกแบบวงจรผ่าน UART

วงจรของ UART จะทำการรับและส่งข้อมูลด้วยเฟรมข้อมูลดังหัวข้อที่ 2.3.1 ซึ่งมีโครงสร้างของวงจรดังรูปที่ 3.20



รูปที่ 3.20 โครงสร้างวงจร UART

สัญญาณตามโครงสร้างวงจร UART มีหน้าที่ดังต่อไปนี้

1) สัญญาณ Clk เป็นสัญญาณนาฬิกาเพื่อกำกับจังหวะการทำงานของวงจรทั้งหมด

2) สัญญาณ RxFfFull เป็นสัญญาณแสดงสถานะปัจจุบันของ Rx to Tx ว่าพร้อมใช้งานหรือไม่

3) สัญญาณ RxFfWrData เป็นสัญญาณข่าวสาร 8 บิตจากวงจร Receiver ไปยัง Rx to Tx

4) สัญญาณ RxFfWrEn เป็นสัญญาณเพื่อกระตุ้นเพื่อให้ Rx to Tx รับสัญญาณ RxFfWrData

5) สัญญาณ TxFfRdData เป็นสัญญาณข่าวสาร 8 บิตจาก Reciever ไปยังวงจร Transmitter

6) สัญญาณ TxFfRdEn เป็นสัญญาณเพื่อกระตุ้นเพื่อให้ Rx to Tx ส่งสัญญาณ TxFfRdData

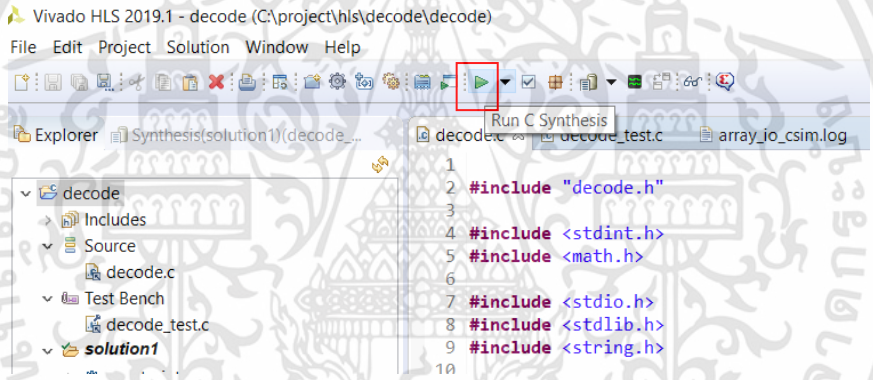
7) สัญญาณ TxEmpty เป็นสัญญาณเพื่อแจ้งวงจร Transmitter ว่า Reciever ไม่มีข่าวสาร

8) สัญญาณ Rx เป็นสัญญาณที่รับข้อมูลพารามิเตอร์จากส่วนต่อประสานกราฟิกกับผู้ใช้ไปยัง FPGA

9) สัญญาณ Tx เป็นสัญญาณที่ส่งอัตราความผิดพลาดบิตจาก FPGA ไปยังส่วนต่อประสานกราฟิกกับผู้ใช้

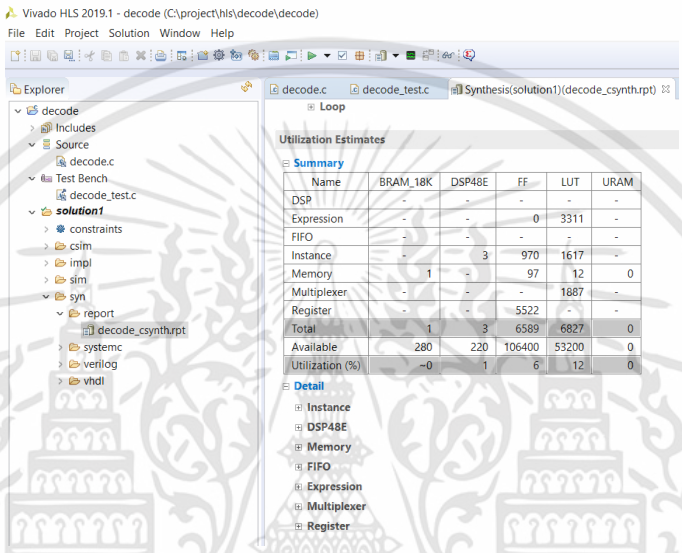
3.6 การสังเคราะห์วงจรที่ทำการออกแบบ

เมื่อทำการออกแบบวงจรตามหัวข้อ 3.1 ถึง 3.4 เสร็จเรียบร้อย จึงทำการสังเคราะห์วงจรด้วยโปรแกรม Vivado HLS เพื่อเปลี่ยนจากซอฟต์แวร์ภาษา C++ ที่เขียนให้กลายเป็นฮาร์ดแวร์ที่ทำงานได้จริง โดยทำการสังเคราะห์วงจรด้วยการกดปุ่ม Run C Synthesis ดังรูปที่ 3.21



รูปที่ 3.21 ปุ่ม Run C Synthesis บนโปรแกรม Vivado HLS

เมื่อสังเคราะห์วงจร จะได้รับไฟล์ตารางสรุปการใช้ทรัพยากรของวงจร และสัญญาณที่ทำงานในวงจรจริงดังรูปที่ 3.22



รูปที่ 3.22 ผลสรุปการใช้ทรัพยากรของวงจรและสัญญาณที่ทำงานในวงจร

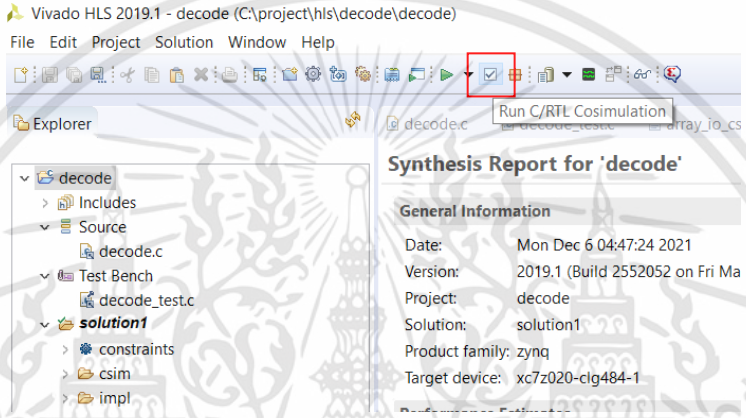
3.7 การจัดเก็บผลการทดลอง

ในการจัดเก็บผลการทดลองของวงจรเข้ารหัส วงจรสร้างสัญญาณสุ่ม และวงจรถอดรหัส จะจัดเก็บผลการทดลอง 4 ชนิด ได้แก่ กราฟจำลองการทำงานของสัญญารภายในวงจร ผลลัพธ์ Register Transfer Level (RTL) ของวงจร ผลลัพธ์ที่แสดงผลบนส่วนประสานงานผู้ใช้ GUI และกราฟอัตราบิตผิดพลาดของบิตข้อมูลก่อนเข้าวงจรเข้ารหัสกับชุดบิตข้อมูลหลังออกจากวงจรถอดรหัส

3.7.1 กราฟจำลองการทำงานของสัญญาณภายในวงจร

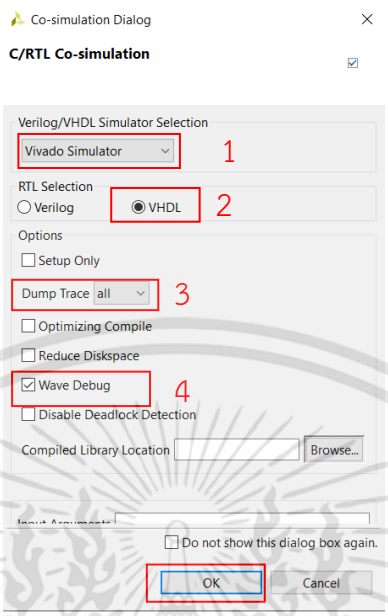
กราฟจำลองการทำงานของสัญญาณบ่งบอกถึงการทำงานของแต่ละสัญญาณอย่างละเอียดในแต่ละช่วงเวลา

หลังผ่านกระบวนการสังเคราะห์วงจรเรียบร้อยแล้วจึงกดปุ่ม Run C/RTL Cosimulation ดังรูปที่ 3.23 เพื่อทำการจำลอง



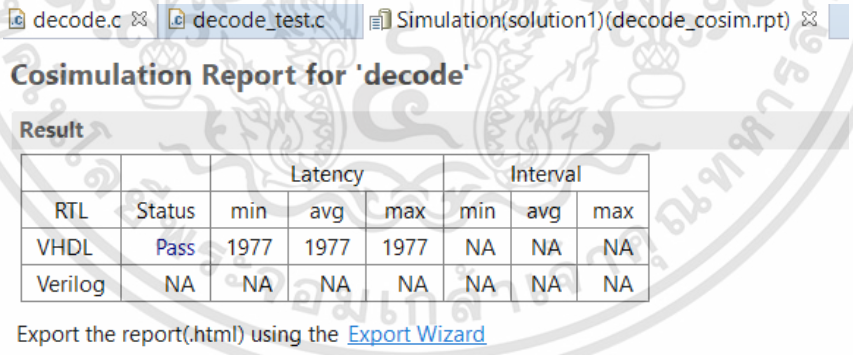
รูปที่ 3.23 ปุ่ม Run C/RTL Cosimulation บนโปรแกรม Vivado HLS

เลือกหัวข้อ “Vivado Simulator” จากนั้นเลือก RTL Selection เป็นภาษา “VHDL” ตามด้วยเปลี่ยน Dump Trace เป็น “all” และเลือกที่ตัวเลือก “Wave Debug” จากนั้นกด OK เพื่อให้โปรแกรมทำงานดังรูปที่ 3.24



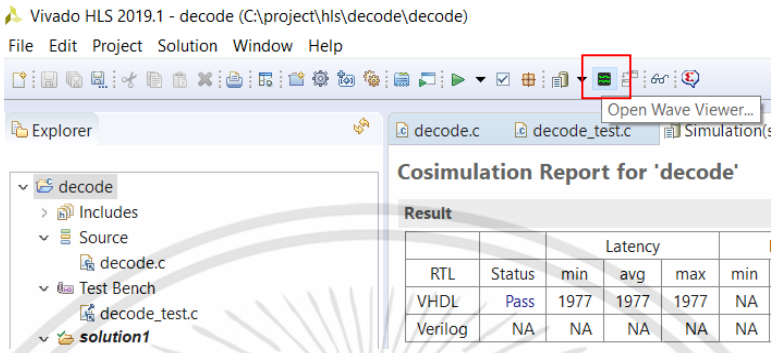
รูปที่ 3.24 หน้าต่างการตั้งค่าของการสร้างกราฟจำลองการทำงานของวงจร

เมื่อโปรแกรมทำงานเสร็จสิ้น จะได้ตารางสรุปเวลาการประมวลผลของวงจร ดังรูปที่ 3.25



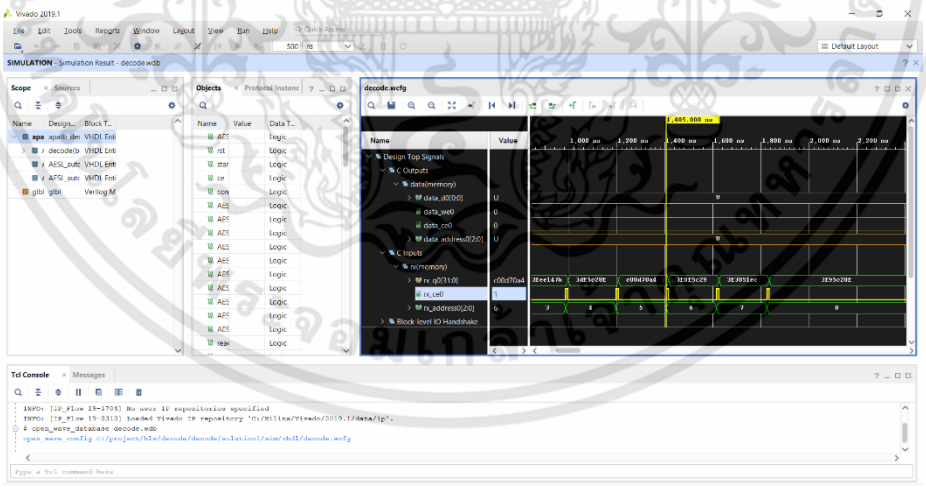
รูปที่ 3.25 ตารางสรุปเวลาการประมวลผลของวงจร

กดปุ่ม Open Wave Viewer เพื่อเปิดกราฟจำลองการทำงานของวงจรดังรูปที่ 3.26



รูปที่ 3.26 ปุ่ม Open Wave Viewer สำหรับเปิดกราฟจำลองการทำงานของวงจร

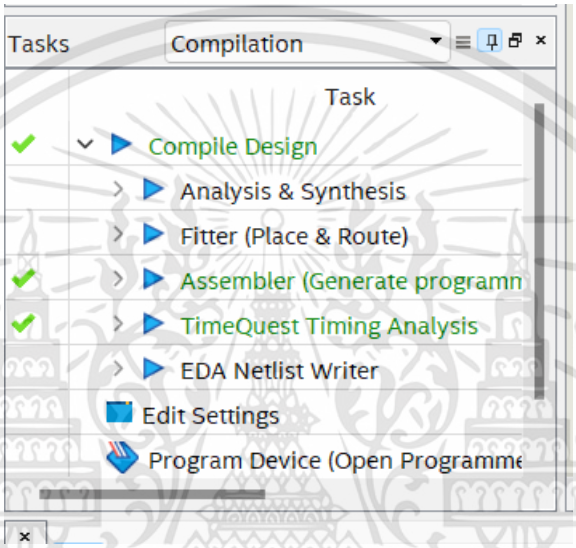
หลังจากนั้นจะได้กราฟจำลองการทำงานของวงจรที่ถูกเปิดขึ้นด้วยโปรแกรม Vivado ดังรูปที่ 3.27 กราฟจำลองการทำงานของวงจรสามารถบอกการทำงานจริงของสัญญาณภายในวงจร และทำให้สามารถตรวจสอบได้ว่าวงจรที่ออกแบบมาสามารถทำงานได้จริง ขั้นตอนสุดท้ายคือการเลือกดูสัญญาณเอาต์พุตที่ต้องการ และเปรียบเทียบค่ากับโปรแกรม MATLAB ว่าถูกต้องหรือไม่



รูปที่ 3.27 กราฟจำลองการทำงานของวงจร

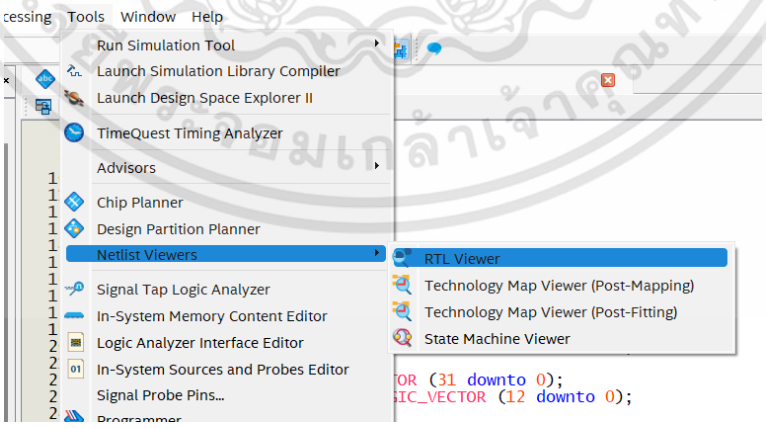
3.7.2 ผลลัพธ์ RTL ของวงจร

ผลลัพธ์ RTL ใช้สำหรับดูลักษณะ*ของวงจร*ที่สังเคราะห์ขึ้น เริ่มต้นจากการนำไฟล์ VHDL ที่ได้จากการสังเคราะห์วงจรจากหัวข้อที่ 3 มาสร้างโปรเจคใหม่ในโปรแกรม Quartus Prime จากนั้นกด Compile Design เพื่อตรวจความถูกต้องของโค้ด และประเมินผลการใช้งานของบอร์ดทดลอง ดังรูปที่ 3.28



รูปที่ 3.28 ปุ่มสำหรับ Compile Design วงจร

เลือกหัวข้อ Tools ตามด้วยหัวข้อ Netlist Viewers และ RTL Viewer เพื่อดูลักษณะ*ของวงจร*ที่สังเคราะห์ขึ้น ดังรูปที่ 3.29

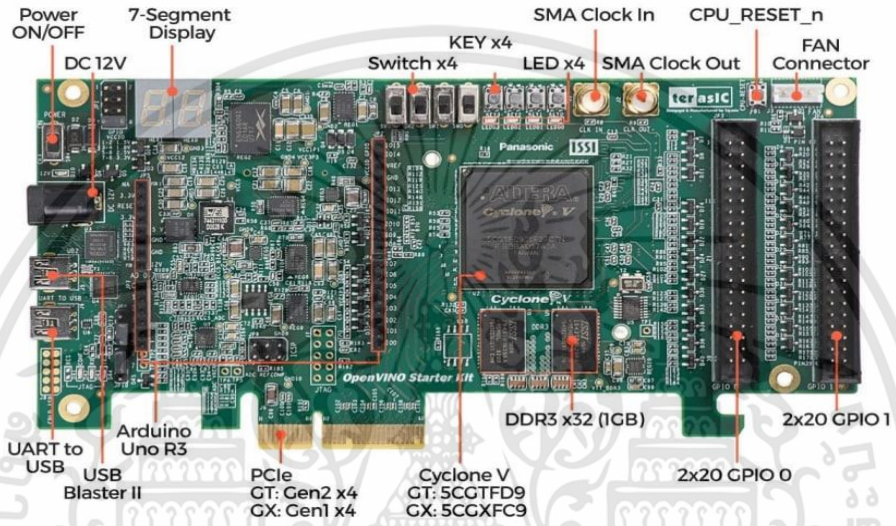


รูปที่ 3.29 การเลือกหัวข้อ RTL Viewer เพื่อดูผลลัพธ์ RTL

3.8 เครื่องมือที่ใช้ในการทดลอง

3.8.1 OpenVino Starter Kit GT Edition

OpenVino Starter Kit GT Edition มีลักษณะดังรูปที่ 3.30 และมีคุณสมบัติและส่วนประกอบดังตารางที่ 3.2



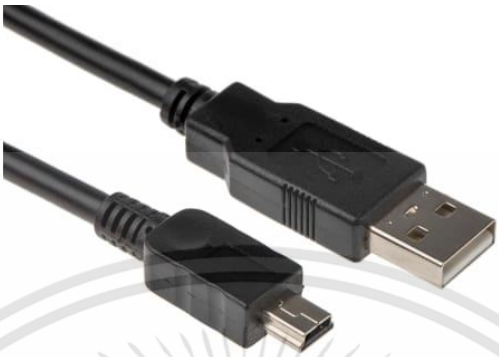
รูปที่ 3.30 OpenVino Starter Kit GT Edition

ตารางที่ 3.2 คุณสมบัติและส่วนประกอบของบอร์ด OpenVino Starter Kit GT Edition

อุปกรณ์ภายในบอร์ด	*คุณสมบัติ*
FPGA Device	Intel FPGA Cyclone® V GT 5CGTFD9D5F27C7N device
Memory	1GB DDR3 SDRAM (32-bit data bus) 64MB SDRAM (16-bit data bus)

อุปกรณ์ภายในบอร์ด	คุณสมบัติ
Configuration and Debug	Quad Serial Configuration device – EPCQ256 Onboard USB-Blaster II (Mini-B USB connector)
Communication	UART to USB (USB Mini-B connector) PCIe Gen2x4
Connectors	Two 40 Pin GPIO header One Arduino Uno Revision 3 header SMA IN/OUT 3.3V Single-end input and output
Switches/ Buttons/ Indicators	5 user Keys 4 user switches 4 LEDG Two 7-segment displays
Power	12V DC Input PCIe 12V Input
Cooling System	12V Fan with 5000 Rotational Speed

3.8.2 สาย USB to Mini USB



รูปที่ 3.31 สาย USB to Mini USB

3.8.3 Laptop

ใช้สำหรับเปิดใช้งานโปรแกรม Vivado โปรแกรม Vivado HLS และ Visual Studio Code และ Altera Quartus II Quartus Prime มี สเปกและคุณลักษณะดังรูปที่ 3.32 ถึง 3.34

รูปที่ 3.32 Laptop MSI Modern 15

MSI Modern 15 มีสเปกดังนี้

1) CPU : 11 th Gen Intel(R) Core(TM) i7-1165G7 @2.80GHz

2) GPU : NVIDIA GeForce MX450 2GB GDDR5

3) RAM : 8.0 GB

รูปที่ 3.33 ASUS TUF Gaming FX504GE

ASUS TUF Gaming FX504GE มีสเปกดังนี้

1) CPU : Intel(R) Core(TM) i7-8750H CPU @2.20GHz

2) GPU : NVIDIA GeForce GTX 1050Ti 4GB GDDR5

3) RAM : 8.0 GB

รูปที่ 3.34 Pavilion Gaming Laptop 15-ec1072AX

Pavilion Gaming Laptop 15-ec1072AX มีสเปกดังนี้

1) CPU : AMD Ryzen 5 4800H with Radeon Graphics @3.0GHz

2) GPU : NVIDIA GeForce GTX 1650 4GB GDDR6

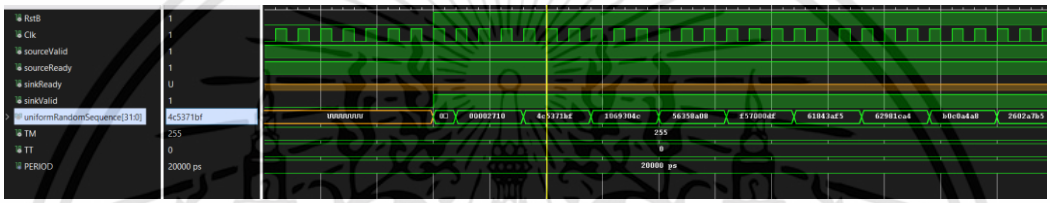
3) RAM : 8.0 GB

บทที่ 4

ผลการทดลอง

4.1 วงจรสร้างสัญญาณสุ่ม

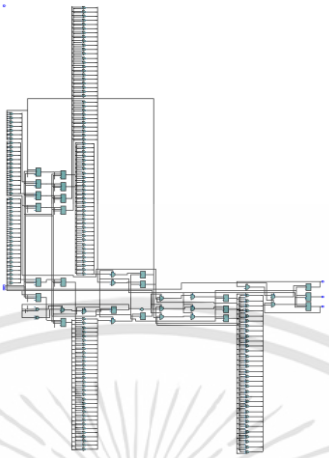
สังเคราะห์วงจรสร้างสัญญาณสุ่มด้วยโปรแกรม Vivado และได้ทำการจำลองการทำงานของวงจรด้วยการทดลองรับสัญญาณอินพุตเพื่อให้เริ่มต้นกระบวนการสุ่มบิตตามหลักการ Tausworthe จะได้เอาต์พุตออกมาเป็นชุดบิตสุ่มชุดละ 32 บิต แสดงดังรูปที่ 4.1



รูปที่ 4.1 กราฟการทำงานของวงจรสุ่ม

จากรูปที่ 4.1 จะพบว่าค่าผลลัพธ์การรับค่าอินพุตของวงจรสร้างสัญญาณสุ่ม ประกอบไปด้วย 1 สัญญาณ คือสัญญาณ sourceValid คือสัญญาณที่สั่งการให้วงจรสร้างสัญญาณสุ่มเริ่มการทำงาน วงจรจะเริ่มการทำงานก็ต่อเมื่อ sourceValid เท่ากับ ‘1’ และค่าผลลัพธ์ของเอาต์พุตประกอบไปด้วย 2 สัญญาณ ได้แก่ สัญญาณ UniformRamdomSequence คือสัญญาณที่เก็บค่าชุดบิตสุ่มจำนวน 32 บิต ที่ได้จากกระบวนการสร้างสัญญาณสุ่ม และสัญญาณ sinkValid คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ UniformRamdomSequence ซึ่งจะเป็น ‘1’ เมื่อเอาต์พุตพร้อมใช้งาน

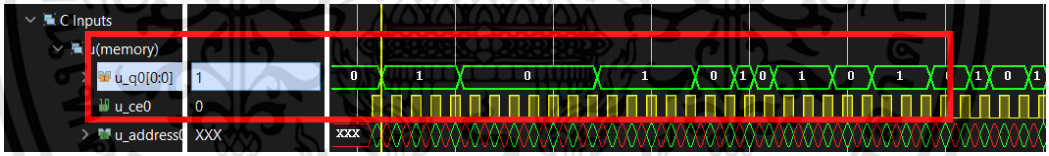
ทำการสังเคราะห์วงจรโดยใช้โปรแกรม Quartus Prime ซึ่งได้ผลลัพธ์ RTL ดังรูปที่ 4.2



รูปที่ 4.2 ผลลัพธ์ RTL ของวงจรสร้างสัญญาณสุ่ม

4.2 วงจรเข้ารหัส

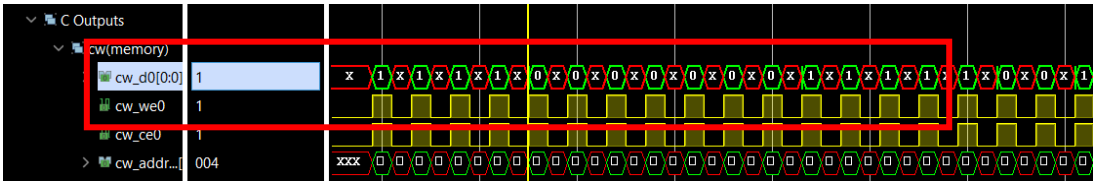
สังเคราะห์วงจรเข้ารหัสในโปรแกรม Vivado HLS และได้ทำการจำลองการทำงานของวงจรด้วยการทดลองรับค่าอินพุตจำนวนจริงจากวงจรสร้างสัญญาณสุ่มจำนวน 1024 บิต แสดงดังรูปที่ 4.3



รูปที่ 4.3 อินพุตบิตของวงจรเข้ารหัส

จากรูปที่ 4.3 จะพบว่าค่าผลลัพธ์การรับค่าอินพุตของวงจรเข้ารหัส ประกอบไปด้วย 3 สัญญาณ ได้แก่ สัญญาณ u_q0 คือสัญญาณที่เก็บค่า u ที่ได้จากวงจรสร้างสัญญาณสุ่ม สัญญาณ u_ce0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ u_ce0 ซึ่งจะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน และสัญญาณ u_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ u แต่ละข้อมูล

เอาต์พุตหลังจากการประมวลผลผ่านวงจรเข้ารหัสได้ผลลัพธ์ออกมาเป็นค่า cw จำนวน 2048 บิต แสดงดังรูปที่ 4.4



รูปที่ 4.4 เอาต์พุตของวงจรเข้ารหัส

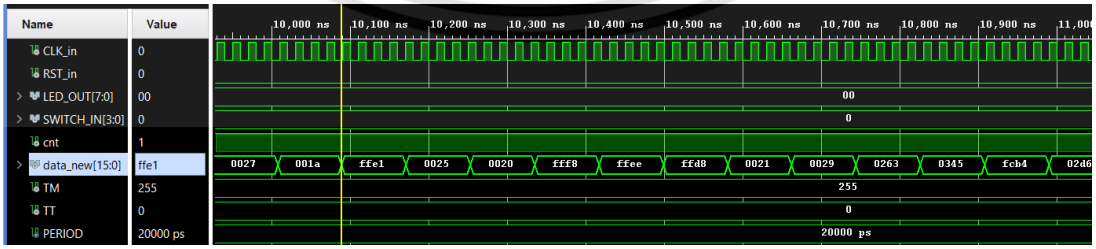
จากรูปที่ 4.4 จะพบว่าค่าผลลัพธ์ของเอาต์พุตวงจรเข้ารหัส ประกอบไปด้วย 4 สัญญาณ ได้แก่ สัญญาณ cw_d0 คือสัญญาณที่เก็บค่า cw ที่ได้จากกระบวนการเข้ารหัส สัญญาณ cw_we0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ cw_we0 ซึ่งจะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน สัญญาณ cw_ce0 คือสัญญาณที่บอกว่าวงจรกำลังทำงานอยู่จะเป็น ‘0’ ในช่วงที่ไม่มีค่าเอาต์พุต และสัญญาณ cw_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ cw แต่ละข้อมูล

ทำการสังเคราะห์วงจรโดยใช้โปรแกรม Quartus Prime ซึ่งได้ผลลัพธ์ RTL ดังรูปที่ 4.5

รูปที่ 4.5 ผลลัพธ์ RTL ของวงจรเข้ารหัส

4.3 วงจรสร้างสัญญาณรบกวน

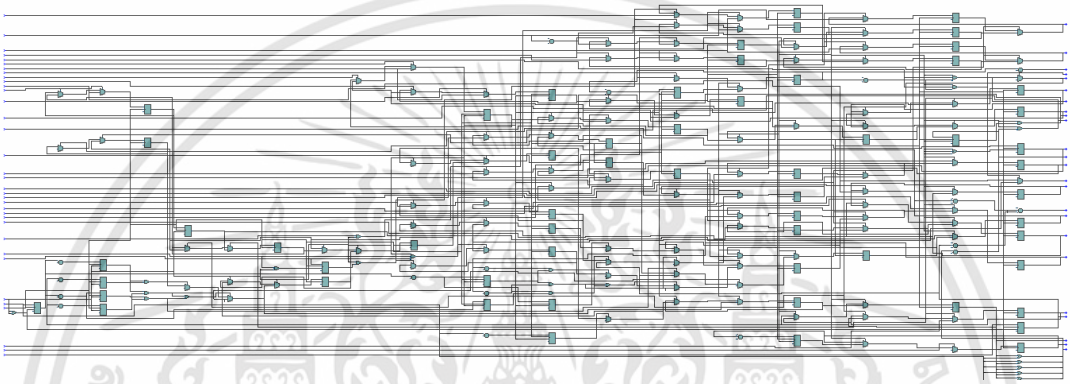
สังเคราะห์วงจรสร้างสัญญาณรบกวนด้วยโปรแกรม Vivado และได้ทำการจำลองการทำงานของวงจร โดยมีกระบวนการสุ่มบิตตามหลักการ Tausworthe และหลักการ Box-Muller จะได้เอาต์พุตออกมาเป็นชุดบิตสุ่มชุดละ 16 บิต แสดงดังรูปที่ 4.6



รูปที่ 4.6 กราฟการทำงานของวงจรสร้างสัญญาณรบกวน

จากรูปที่ 4.6 จะพบว่าไม่มีสัญญาณอินพุตของวงจรสร้างสัญญาณรบกวน และค่าผลลัพธ์ของเอาต์พุตมี 1 สัญญาคือ สัญญาณ Data_new คือสัญญาณที่เก็บค่าชุดบิตสุ่มชนิดจุดตรึง จำนวน 16 บิต ที่ได้จากกระบวนการสร้างสัญญาณรบกวน

ทำการสังเคราะห์วงจรโดยใช้โปรแกรม Quartus Prime ซึ่งได้ผลลัพธ์ RTL ดังรูปที่ 4.7

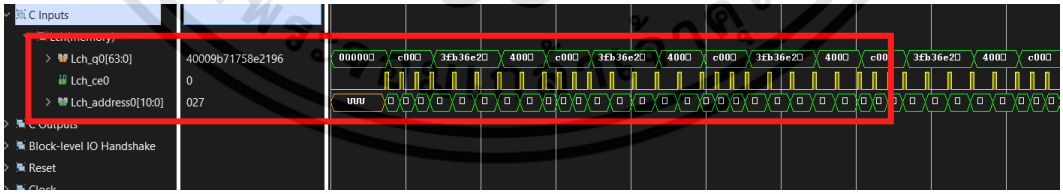


รูปที่ 4.7 ผลลัพธ์ RTL ของวงจรสร้างสัญญาณรบกวน

4.4 วงจรถอดรหัส

4.4.1 วงจรควอนไทซ์

สังเคราะห์วงจรควอนไทซ์ในโปรแกรม Vivado HLS และได้ทำการจำลองการทำงานของวงจรด้วยการทดลองรับค่าอินพุตจำนวนจริงหลังจากกระบวนการบวกสัญญาณข่าวสารกับสัญญาณรบกวนทั้งหมด 2048 ชุดข้อมูล แสดงดังรูปที่ 4.8

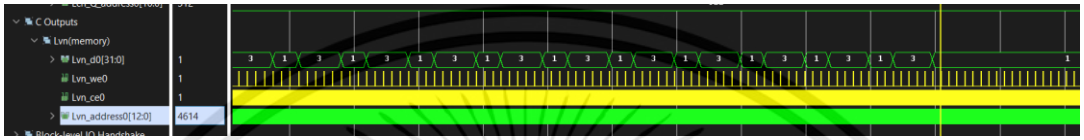


รูปที่ 4.8 อินพุตบิตของวงจรควอนไทซ์

จากรูปที่ 4.8 จะพบว่าค่าผลลัพธ์การรับค่าอินพุตของวงจรควอนไทซ์ ประกอบไปด้วย 3 สัญญาณ ได้แก่ สัญญาณ Lch_q0 คือสัญญาณที่เก็บค่า Lch ที่ได้จากกระบวนการบวกสัญญาณ

ข่าวสารกับสัญญาณรบกวน สัญญาณ Lch_ce0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ Lch_q0 จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน และสัญญาณ Lch_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ Lch แต่ละข้อมูล

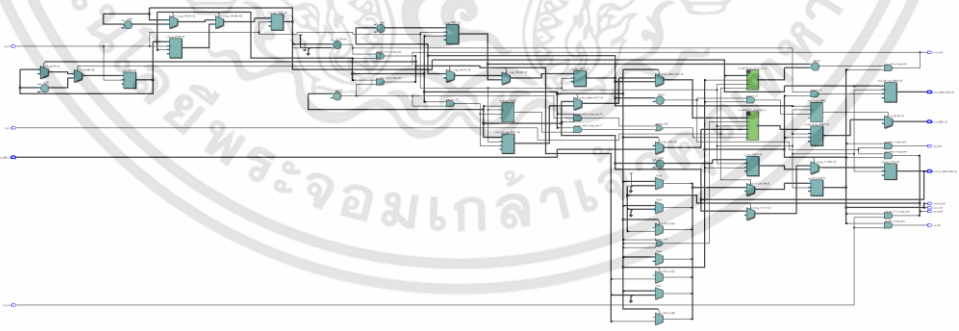
เอาต์พุตหลังจากการประมวลผลผ่านวงจรควอนไทซ์ได้ผลลัพธ์ออกมาเป็นค่า Lvn ซึ่งจะนำไปเก็บที่ Variable Node จำนวน 7680 ชุด แสดงดังรูปที่ 4.9



รูปที่ 4.9 เอาต์พุตของวงจรควอนไทซ์

จากรูปที่ 4.9 จะพบว่าค่าผลลัพธ์ของเอาต์พุตวงจรควอนไทซ์ ประกอบไปด้วย 4 สัญญาณ ได้แก่ สัญญาณ Lvn_d0 คือสัญญาณที่เก็บค่า Lvn ที่ได้จากกระบวนการแปลงค่าอัตราค่าความน่าจะเป็นแบบลอการิทึม สัญญาณ Lvn_we0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ Lvn_d0 จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน สัญญาณ Lvn_ce0 คือสัญญาณที่บอกว่าวงจรกำลังทำงานอยู่จะเป็น ‘0’ ก็ต่อเมื่อวงจรทำงานถึงค่าสุดท้ายของเอาต์พุต และสัญญาณ Lvn_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ Lvn แต่ละข้อมูล

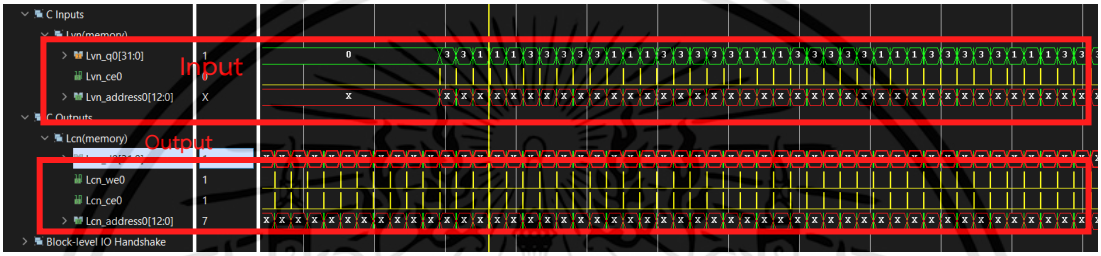
ทำการสังเคราะห์วงจรโดยใช้โปรแกรม Quartus Prime ซึ่งได้ผลลัพธ์ RTL ดังรูปที่ 4.10



รูปที่ 4.10 ผลลัพธ์ RTL ของวงจรควอนไทซ์

4.4.2 วงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node

สังเคราะห์วงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node ในโปรแกรม Vivado HLS และได้ทำการจำลองการทำงานของวงจรด้วยการทดลองรับค่าอินพุต Lvn ที่ได้รับจากวงจรควอนไทซ์ 7680 ค่า ทำการถอดรหัสออกมาเป็นเอาต์พุต Lcn ซึ่งจะนำไปเก็บที่ Check Node จำนวน 7680 ชุดข้อมูล แสดงดังรูปที่ 4.11

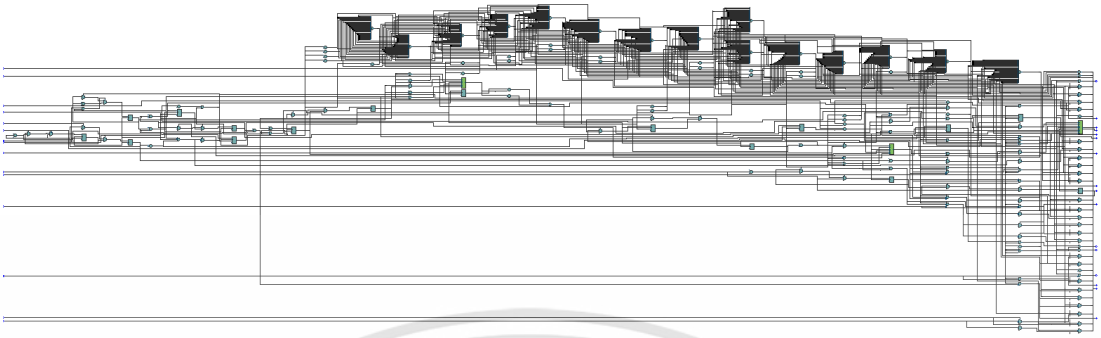


รูปที่ 4.11 อินพุตและเอาต์พุตของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node

จากรูปที่ 4.11 จะพบว่าค่าผลลัพธ์การรับค่าอินพุตของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node ประกอบไปด้วย 3 สัญญาณ ได้แก่ สัญญาณ Lvn_q0 คือสัญญาณที่เก็บค่า Lvn ที่ได้จากวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node สัญญาณ Lvn_ce0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ Lvn_q0 จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน และสัญญาณ Lvn_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ Lvn แต่ละข้อมูล

เอาต์พุตวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node ประกอบไปด้วย 4 สัญญาณ ได้แก่ สัญญาณ Lcn_d0 คือสัญญาณที่เก็บค่า Lcn ที่ได้จากกระบวนการถอดรหัสที่ Check Node สัญญาณ Lcn_we0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ Lcn_q0 จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน สัญญาณ Lcn_ce0 คือสัญญาณที่บอกว่าวงจรกำลังทำงานอยู่จะเป็น ‘0’ ก็ต่อเมื่อถึงค่าสุดท้ายของเอาต์พุต และสัญญาณ Lcn_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ Lcn แต่ละข้อมูล

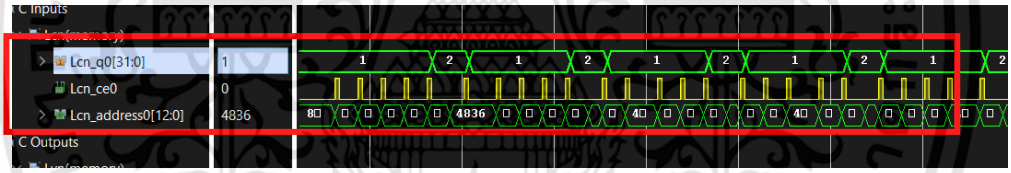
ทำการสังเคราะห์วงจรโดยใช้โปรแกรม Quartus Prime ซึ่งได้ผลลัพธ์ RTL ดังรูปที่ 4.12



รูปที่ 4.12 ผลลัพธ์ RTL ของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node

4.4.3 วงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node

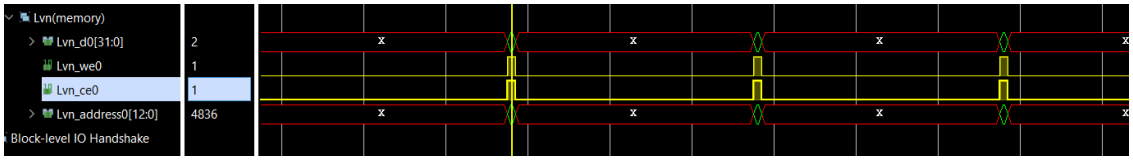
สังเคราะห์วงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node ในโปรแกรม Vivado HLS และได้ทำการจำลองการทำงานของวงจรด้วยการทดลองรับค่าอินพุต Lvn ที่ได้รับจากวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node 7680 ค่าแสดงดังรูปที่ 4.13



รูปที่ 4.13 อินพุตบิตของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node

จากรูปที่ 4.13 จะพบว่าค่าผลลัพธ์การรับค่าอินพุตของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node ประกอบไปด้วย 3 สัญญาณ ได้แก่ สัญญาณ Lcn_q0 คือสัญญาณที่เก็บค่า Lcn ที่ได้จากวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node สัญญาณ Lcn_ce0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ Lcn_q0 จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน และสัญญาณ Lcn_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ Lcn แต่ละข้อมูล

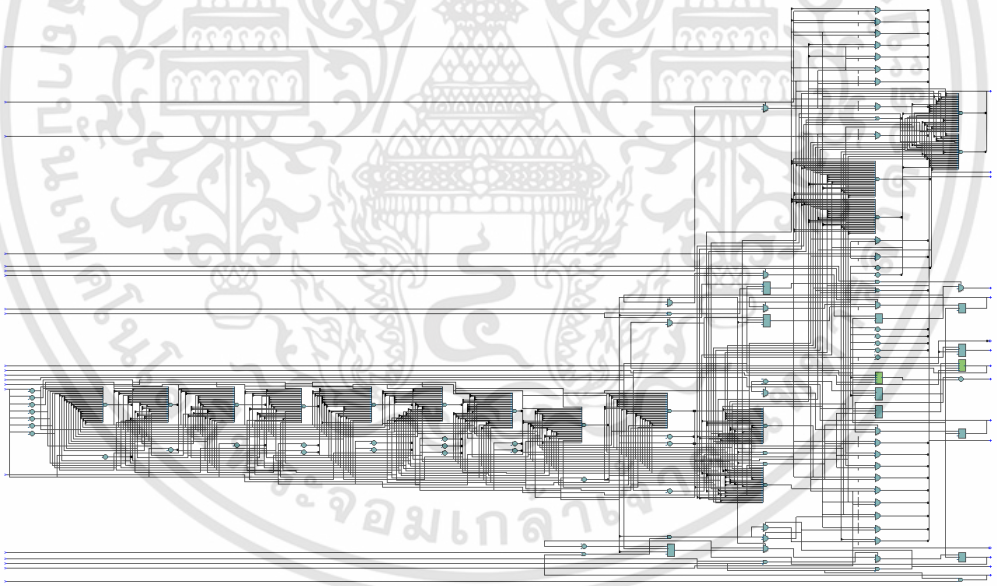
เอาต์พุตหลังจากการประมวลผลผ่านวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node ได้ผลลัพธ์ออกมาเป็นค่า Lvn ซึ่งจะนำไปเก็บที่ Variable Node จำนวน 7680 ชุด แสดงดังรูปที่ 4.14



รูปที่ 4.14 เอาต์พุตของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node

จากรูปที่ 4.8 จะพบว่าค่าผลลัพธ์ของเอาต์พุตวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node ประกอบไปด้วย 4 สัญญาณ ได้แก่ สัญญาณ Lvn_d0 คือสัญญาณที่เก็บค่า Lvn ที่ได้จากกระบวนการถอดรหัสที่ Variable Node Lvn_we0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ Lvn_d0 จะเป็น ‘1’ เมื่อค่าเอาต์พุตพร้อมใช้งาน สัญญาณ Lvn_ce0 คือสัญญาณที่บอกว่าวงจรกำลังทำงานอยู่จะเป็น ‘0’ ก็ต่อเมื่อถึงค่าสุดท้ายของเอาต์พุต และสัญญาณ Lvn_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ Lvn แต่ละข้อมูล

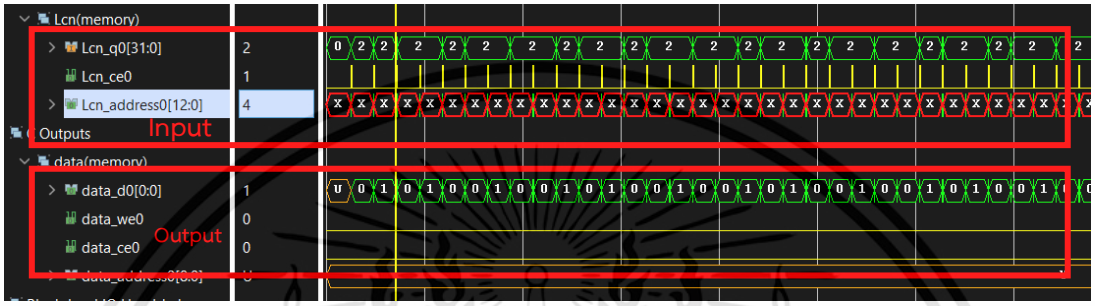
ทำการสังเคราะห์วงจรโดยใช้โปรแกรม Quartus Prime ซึ่งได้ผลลัพธ์ RTL ดังรูปที่ 4.15



รูปที่ 4.15 ผลลัพธ์ RTL ของวงจรเปรียบเทียบคู่อันดับ LUT ที่ Variable Node

4.4.4 วงจรถอดรหัสครั้งสุดท้าย

สังเคราะห์วงจรถอดรหัสครั้งสุดท้าย ในโปรแกรม Vivado HLS และได้ทำการจำลองการทำงานของวงจรด้วยการทดลองรับค่าอินพุต Lcn ที่ได้รับจากวงจรแปลงวงจรถอดรหัสที่ Variable Node 7680 ค่า และได้ถอดรหัสออกมาเป็นเอาต์พุตจำนวน 1024 บิตแสดงดังรูปที่ 4.16

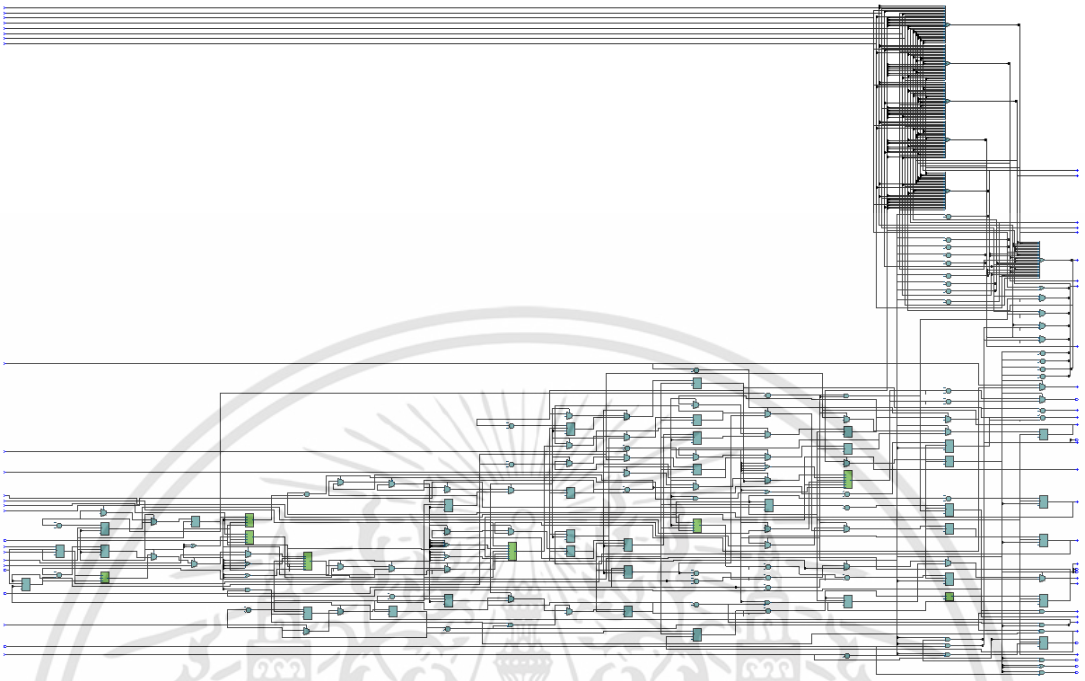


รูปที่ 4.16 อินพุตและเอาต์พุตของวงจรถอดรหัสครั้งสุดท้าย

จากรูปที่ 4.16 จะพบว่าค่าผลลัพธ์การรับค่าอินพุตของวงจรถอดรหัสครั้งสุดท้ายประกอบไปด้วย 3 สัญญาณ ได้แก่ สัญญาณ Lcn_q0 คือสัญญาณที่เก็บค่า Lcn ที่ได้จากวงจรเปรียบเทียบคู่อันดับ LUT ที่ Check Node สัญญาณ Lcn_ce0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ Lcn_q0 จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน และสัญญาณ Lcn_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของ Lcn แต่ละข้อมูล

เอาต์พุตวงจรวงจรถอดรหัสครั้งสุดท้ายประกอบไปด้วย 4 สัญญาณ ได้แก่ สัญญาณ data_d0 คือสัญญาณที่เก็บค่าข่าวสารที่ได้จากกระบวนการถอดรหัส สัญญาณ data _we0 คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ data _q0 จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน สัญญาณ data _ce0 คือสัญญาณที่บอกว่าวงจรกำลังทำงานอยู่จะเป็น ‘0’ ก็ต่อเมื่อถึงค่าสุดท้ายของเอาต์พุต และสัญญาณ data_address คือสัญญาณชี้ตำแหน่งที่อยู่ของสัญญาณของข่าวสารแต่ละข้อมูล

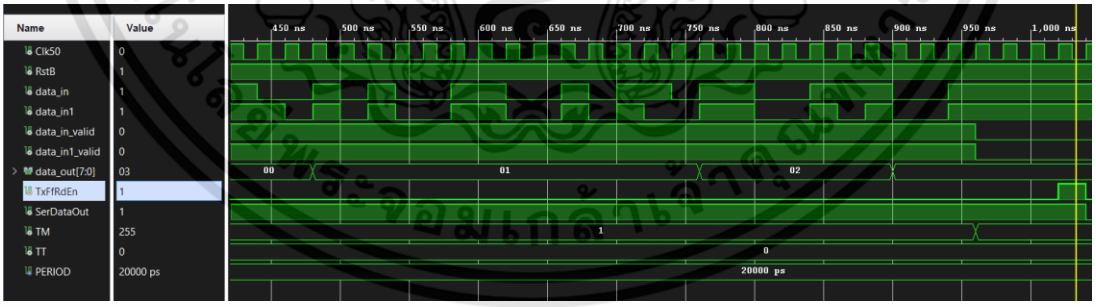
ทำการสังเคราะห์วงจรโดยใช้โปรแกรม Quartus Prime ซึ่งได้ผลลัพธ์ RTL ดังรูปที่ 4.17



รูปที่ 4.17 ผลลัพธ์ RTL ของวงจรถอดรหัสครั้งสุดท้าย

4.5 วงจรเปรียบเทียบคำรหัส

สังเคราะห์วงจรเปรียบเทียบทำรหัสด้วยโปรแกรม Vivado และได้ทำการจำลองการทำงานของวงจรด้วยการทดลองรับสัญญาณอินพุตจากวงจรสร้างสัญญาณสุ่มจำนวน 1024 บิต และวงจรถอดรหัสจำนวน 1024 บิต แสดงดังรูปที่ 4.18

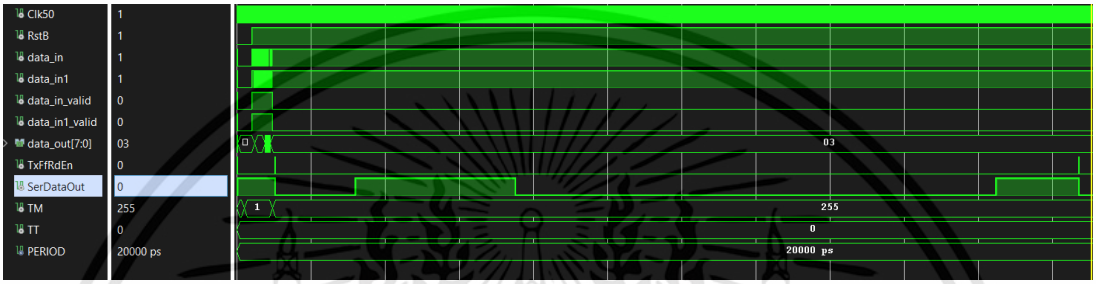


รูปที่ 4.18 อินพุตของวงจรเปรียบเทียบคำรหัส

จากรูปที่ 4.18 จะพบว่าค่าผลลัพธ์การรับค่าอินพุตของวงจรเปรียบเทียบคำรหัสประกอบไปด้วย 4 สัญญาณ ได้แก่ สัญญาณ data_in คือสัญญาณที่เก็บค่าข่าวสารที่ถูกสร้างโดยวงจรสร้างสัญญาณสุ่ม สัญญาณ data_in_valid คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ data_in จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน สัญญาณ data_in1 คือสัญญาณที่เก็บค่าข่าวสารที่

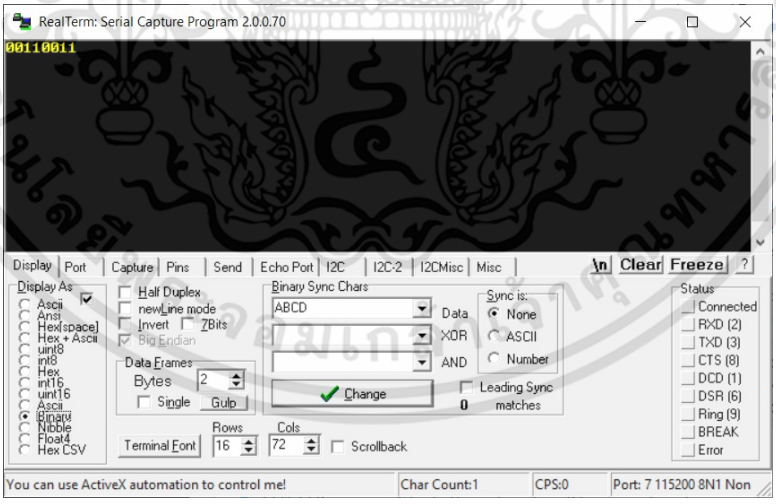
ได้จากวงจรสร้างถอดรหัส และสัญญาณ data_in1_valid คือสัญญาณที่ตรวจสอบความถูกต้องของสัญญาณ data_in1 จะเป็น ‘1’ เมื่อค่าอินพุตพร้อมใช้งาน

นำ 2 สัญญาณอินพุตมาเปรียบเทียบเพื่อนับบิตผิดพลาดที่เกิดจากความไม่ตรงกันของสัญญาณ จะได้ค่าออกมาเป็นจำนวนบิตผิดพลาดชนิดไบนารี่ 8 บิต จากนั้นส่งค่าจำนวนบิตผิดพลาดผ่าน TxSerial ได้เอาต์พุตออกมาทีละ 1 บิต ดังรูป 4.19



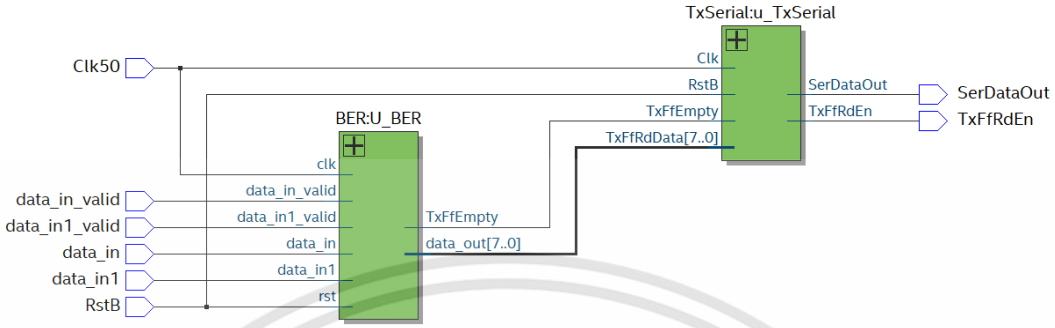
รูปที่ 4.19 เอาต์พุตของวงจรเปรียบเทียบคำรหัส

เอาต์พุตวงจรเปรียบเทียบคำรหัสมี 1 สัญญาณคือสัญญาณ SerDataOut เป็นสัญญาณที่เก็บค่าจำนวนบิตผิดพลาดของวงจรเปรียบเทียบคำรหัสทีละ 1 บิต ซึ่งมีค่าตรงกับค่าจำนวนบิตผิดพลาดส่งผ่าน TxSerial เพื่อแสดงผลบน Terminal Program ดังรูปที่ 4.20



รูปที่ 4.20 บิตผิดพลาดของวงจรเปรียบเทียบคำรหัสบน Terminal Program

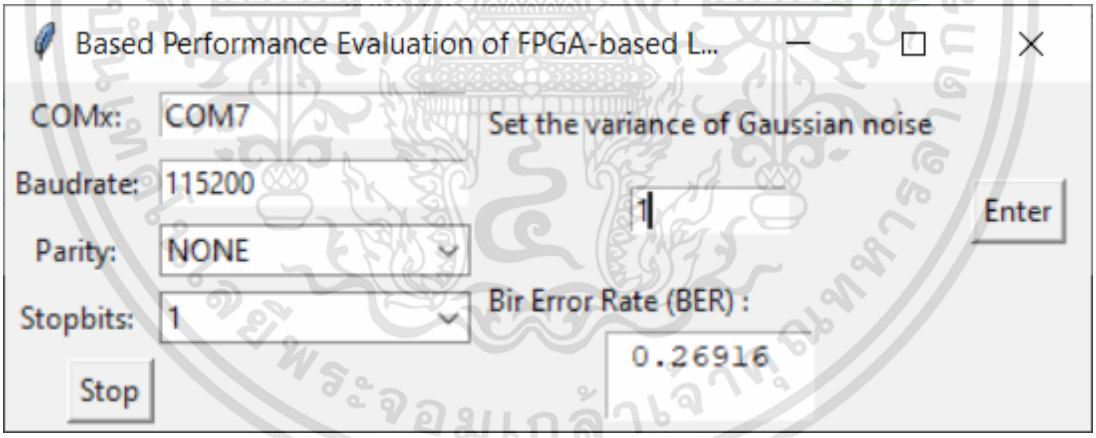
ทำการสังเคราะห์วงจรโดยใช้โปรแกรม Quartus Prime ซึ่งได้ผลลัพธ์ RTL ดังรูปที่ 4.21



รูปที่ 4.21 ผลลัพธ์ RTL ของวงจรเปรียบเทียบคำรหัส

4.6 ส่วนประสานงานผู้ใช้

ทำการส่งค่า Variance ไปยัง FPGA ผ่านส่วนประสานงานผู้ใช้ เพื่อแสดงผลอัตราบิตผิดพลาด (Bit Error Rate : BER) ดังรูปที่ 4.22



รูปที่ 4.22 การแสดงผลอัตราบิตผิดพลาดบนส่วนประงานผู้ใช้

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

ปริญญานิพนธ์นี้ได้เสนอการจำลองสมถรรถนะของรหัส LDPC ด้วย FPGA สำหรับมาตรฐาน CCSDS ซึ่งประกอบไปด้วยการออกแบบวงจรสร้างสัญญาณสุ่ม วงจรเข้ารหัส LDPC วงจรสร้างสัญญาณรบกวน วงจรถอดรหัส LDPC และวงจรเปรียบเทียบคำรหัส FPGA จากโปรแกรม Vivado HLS และ Quartus Prime เมื่อทำการส่งค่าจำนวนบิตผิดพลาดจาก FPGA ไปแสดงบนส่วนประสานงานผู้ใช้ (GUI) เพื่อเป็นการตรวจสอบความถูกต้องของข่าวสารหลังผ่านกระบวนการถอดรหัส ซึ่งพบว่าแบบจำลองสมถรรถนะของรหัส LDPC ที่มีบิตอินพุตเท่ากับ 1024 บิต ค่าอัตรารหัสเท่ากับ 0.5 และค่าความความแปรปรวนเท่ากับ 0.316 พบว่ามีอัตราความผิดพลาดบิตเท่ากับ 0.00108 ที่อัตราส่วนระหว่างสัญญาณข่าวสารต่อสัญญาณรบกวนเท่ากับ 5 เดซิเบล

5.2 ข้อเสนอแนะ

การจำลองสมรรถนะของรหัส LDPC ด้วย FPGA สำหรับมาตรฐาน CCSDS แสดงผลบนส่วนประสานงานผู้ใช้ GUI ควรมีผลลัพธ์การทดสอบที่หลากหลาย เพื่อให้ผู้ที่ศึกษาเกิดความเข้าใจง่ายในการจำลองสมรรถนะ

บรรณานุกรม

[1]. “Low Density Parity Check Codes For Use In Near-Earth And Deep Space Applications.” *Orange Book*. 2007 : 21-31.

[2] Louise, Ross, Martin. *The Zynq Book.* 1st Ed. Scotland, Uk. : Strathclyde Academic Media, 2014.

[3] Louise, Ross, Martin. *The Zynq Book Tutorials.* 2nd Ed. Scotland, Uk. : Strathclyde Academic Media, 2014.

[4] Eric W. Weisstein. “Box-Muller.” Https://En.Wikipedia.Org/Wiki/Eric_W._Weisstein.

[5] D. E. Knuth. *The Art Of Computer Programming*: *Seminumerical Algorithms.* 2nd Ed. Reading, Ma. : Addison-Wesley, 1981.

[6] Falook Gilco“Floating Point.” Http://Falookg.Blogspot.Com/2014/02/Floating-Point-Representation.Html

[7]J. P. R. Tootill, W. D. Robinson, And D. J. Eagle. *An Asymptotically Random Tausworthe Sequence*. J. Assoc. Comput, 1973.

[8] Xilinx. *Vivado Design Suite Tutorial High-Level Synthesis.* 1st Ed. California, Usa. : San Jose, 2014.

ภาคผนวก

โปรแกรมสำหรับการออกแบบการจำลองสมรรถนะของรหัสแอลดีพีซีด้วย FPGA สำหรับมาตรฐาน CCSDS

โปรแกรมสำหรับการออกแบบการจำลองสมรรถนะของรหัสแอลดีพีซีด้วย FPGA สำหรับมาตรฐาน CCSDS สามารถเข้าถึงได้จาก https://github.com/Siripansa/Performmance_Evaluation_of_FPGA-Based_LDPC_Codes_for_CCSDS