การออกแบบเครื่องจำลองช่องสัญญาณ AWGN

AWGN CHANNAL EMULATOR

โดย

นายณัฐพร ภาสภิรมย์

นายผาณิต ละมุล

นายศรายุทธ์ ศุภโชคภากร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

การออกแบบเครื่องจำลองช่องสัญญาณ AWGN

AWGN CHANNAL EMULATOR

โดย

นายณัฐพร ภาสภิรมย์ 43015013

นายผาณิต ละมุล 43015023

นายศรายุทธ์ ศุภโชคภากร 43015036

อาจารย์ที่ปรึกษา

อาจารย์ อัครพล ตรีรัตน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การออกแบบเครื่องจำลองช่องสัญญาณแบบ AWGN**

AWGN Channel Emulator

ผู้จัดทำ

1. นาย ณัฐพร ภาสภิรมย์ 43015013

2. นาย ผาณิต ละมุล 43015023

3. นาย ศรายุทธ์ ศุภโชคภากร 43015036

อ.ดร. อัครพล ตรีรัตน์

..... อาจารย์ที่ปรึกษา

(อาจารย์ อัครพล ตรีรัตน์)

การออกแบบเครื่องจำลองช่องสัญญาณ AWGN

AWGN CHANNAL EMULATOR

โดย นายณัฐพร ภาสภิรมย์ 43015013
นายผาณิต ละมุล 43015023
นายศรายุทธ์ ศุภโชคภากร 43015036

อาจารย์ที่ปรึกษา อาจารย์ อัครพล ตรีรัตน์

บทคัดย่อ

โครงงานนี้เป็นการศึกษาและออกแบบ เครื่องสร้างสัญญาณรบกวนแบบ AWGN (AWGN-Channel Emulator) เพื่อใช้ในการจำลองช่องสัญญาณ (Channel) ในระบบ Telecommunication ด้วยอุปกรณ์ FPGA (Field Programmable Gate Array) พร้อมกับใช้โปรแกรม MAX+PLUS II ช่วยในการพัฒนาระบบ โดยใช้ภาษา VHDL [VHSIC (Very High Speed Integrated Circuit) Hardware Description- Language] ในการอธิบายพฤติกรรมการทำงานของระบบ

ABSTRACT

This project is to study the design of AWGN emulator by using a channel emulation in telecommunication system with FPGA device (Field Programmable Gate Array). In this project, we use MAX PLUS II program for development and use VHDL [VHSIC (Very High-Speed Integrated Circuit) Hardware Description Language] to describe the behavior of system

สารบัญ

สารบัญรูปภาพ

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาของหัวข้อปริญญานิพนธ์

การสื่อสารในระบบโทรคมนาคมปัจจุบันและในอดีตมักจะนำเรื่องสัญญาณรบกวนมาเป็นตัวพิจารณาคุณภาพของระบบต่างๆที่ใช้ออกแบบมาเพื่อใช้สำหรับการสื่อสารข้อมูล ดังที่ทราบดีว่าสัญญาณรบกวนเป็นสัญญาณที่เกิดขึ้นแบบสุ่ม ทำให้ไม่สามารถควบคุมสัญญาณได้โดยตรง แต่ในทางทฤษฎีเราสามารถพิสูจน์และหาค่ารูปแบบการกระจายตัวของสัญญาณแบบสุ่มที่มีค่าไม่จำกัดได้ ดังเช่นการกระจายตัวแบบปกติหรือเกาส์เซียน และการกระจายตัวแบบนี้จะถือว่าเป็นการกระจายตัวที่เกิดขึ้นกับทุกสิ่งในโลกที่มีค่าไม่จำกัดและที่สำคัญที่สุดสัญญาณรบกวนก็มีการกระจายแบบเกาส์เซียน เช่นกัน ดังนั้นปริญญานิพนธ์ฉบับนี้จึงได้นำสัญญาณรบกวนแบบเกาส์เซียน มาพิจารณาโดยการสร้างและออกแบบเครื่องจำลองช่องสัญญาณแบบ AWGN เพื่อใช้เป็นอุปกรณ์ช่วยในการพิจารณาประสิทธิภาพของอุปกรณ์สื่อสารข้อมูลและเทคนิคในการสื่อสารข้อมูลแบบต่างๆเพื่อจะนำมาสู่การพัฒนาและวิจัยเพื่อให้ได้ซึ่งประสิทธิภาพสูงสุดของการสื่อสารข้อมูล

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1.2.1 เพื่อศึกษาทฤษฎีการแจกแจงแบบเกาส์เซียน

1.2.2 เพื่อศึกษาและพิสูจน์ทฤษฎีของ Box-Muller

1.2.3 เพื่อศึกษาและพิสูจน์ทฤษฎีและออกแบบ LFSR (Linear feed back shift register)

1.2.4 เพื่อศึกษาและทำความเข้าใจทฤษฎีเซนทรัลลิมิต(Central limit)

1.2.5 เพื่อศึกษาภาษา VHDL เพื่อในการออกแบบและสร้างเครื่องจำลองช่องสัญญาณแบบ AWGN (Additive White Gaussian noise) และสามารถนำไปพัฒนาใช้กับงานวิจัยอื่นๆได้

1.2.6 เพื่อศึกษาโครงสร้างและการทำงานของอุปกรณ์ FPGAs(Field Programmable Gate Arrays)

1.3 ขอบเขตของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้เป็นการออกแบบเครื่องจำลองช่องสัญญาณแบบ AWGN ซึ่งออกแบบบนพื้นฐานของอุปกรณ์ FPGAs (Field Programmable Gate Arrays) ซึ่งใช้แนวคิดจากทางทฤษฎีของ Box-Mullerและทฤษฎีเซนทรัลลิมิตออกแบบและพัฒนาโดยใช้ภาษา VHDL ในการโปรแกรมลงบนชิป FPGAs เพื่อจำลองให้เป็นช่องสัญญาณในระบบการสื่อสาร

1.4 เนื้อหาของปริญญานิพนธ์

บทที่ 2 กล่าวถึงทฤษฎีและแนวคิดต่างๆที่ใช้ในการออกแบบเครื่องจำลองช่องสัญญาณแบบ AWGN

บทที่ 3 กล่าวถึงเทคนิคในการสร้างและข้อจำกัดต่างๆเพื่อประยุกต์นำมาใช้งานในทางปฏิบัติ

บทที่ 4 การทดลองและผลการทดลองแสดงในรูปของการจำลองด้วยโปรแกรมคอมพิวเตอร์

บทที่ 5 วิจารณ์ปัญหาที่เกิดขึ้น แนวทางในการพัฒนาและสรุปผล

ภาคผนวก ตาราง Cumulative Normal Distribution

ภาคผนวกโปรแกรม

หนังสืออ้างอิง

บทที่2
ทฤษฎีและหลักการ

2.1 สัญญาณรบกวน

2.1.1 คุณสมบัติทั่วไปของสัญญาณรบกวน

ในธรรมชาติจะมีปริมาณทางกายภาพหลายอย่างที่มีการเปลี่ยนแปลงอยู่ตลอดเวลา โดยหากฎเกณฑ์ที่แน่นอนไม่ได้ ในเรื่องของสัญญาณไฟฟ้าเช่นกัน สัญญาณบางชนิดเกิดเปลี่ยนแปลงอยู่ตลอดเวลาตามธรรมชาติและอาจจะมารบกวนสัญญาณข่าวสารที่ต้องการให้เกิดความผิดเพี้ยน หรือไม่ชัดเจนไปได้ เราเรียกสัญญาณเช่นนี้ว่า สัญญาณรบกวน (noise) ถ้าจะกล่าวโดยทั่วไป สัญญาณรบกวน ก็คือสัญญาณที่เราไม่พึงปรารถนา ไม่ว่าจะเกิดขึ้นตามธรรมชาติ หรืออาจจะเกิดขึ้นจากการกระทำของมนุษย์ ถ้าหากว่าเป็นสัญญาณที่เราไม่พึงต้องการซึ่งไม่เกี่ยวข้องกับสัญญาณที่เราต้องการแล้ว เราจะจัดว่ามันเป็นสัญญาณรบกวนทั้งสิ้น ตัวอย่างของสัญญาณรบกวนอันอาจเกิดจากการกระทำของมนุษย์ (man made noise) ได้แก่ สัญญาณจากการจุดระเบิดของหัวเทียนเครื่องยนต์ สัญญาณอันเกิดจากเครื่องใช้ไฟฟ้า เช่น สว่าน หรือเลื่อยไฟฟ้า ดังนี้เป็นต้น สัญญาณรบกวนตามธรรมชาติ (natural noise) ได้แก่ สัญญาณอันเนื่องจากอุณหภูมิ (thermal noise) และสัญญาณรบกวนจากระบบสุริยะ (solar noise) เป็นต้น

เนื่องจากสัญญาณรบกวนนั้นมีคุณสมบัติโดยทั่วไปที่คาดเดาอะไรกับมันโดยแน่นอนไม่ได้ (ranndom)กล่าวคือ คุณสมบัติของสัญญาณรบกวนโดยทั่วไปมักจะเป็นสัญญาณสุ่ม ดังนั้นการที่จะบอกถึงคุณสมบัติอะไรเกี่ยวกับสัญญาณรบกวนนั้น จึงบอกได้แต่คุณสมบัติที่เป็นค่าเชิงสถิติเท่านั้น จากนั้นจึงอาศัยคุณสมบัติเหล่านั้นมาเป็นเครื่องเชื่อมโยงเกี่ยวเนื่อง เพื่ออธิบายเกี่ยวกับคุณสมบัติทางกายภาพของสัญญาณรบกวนนั้นคุณสมบัติทางสถิติที่เราควรรู้ไว้ เมื่อเริ่มศึกษาถึงคุณสมบัติของสัญญาณรบกวนนั้น ได้แก่ ค่าเฉลี่ยในลักษณะต่างๆเช่น ค่าเฉลี่ย (average value) หรือ ค่ามัชฌิม (mean value) ค่ากำลังสองเฉลี่ย (mean square value) และ ค่าความแปรปรวน (variance) เป็นต้น

ถ้าสมมุติให้ $n(t)$ เป็นฟังก์ชันของเวลาที่ใช้แทนสัญญาณรบกวน ค่าเฉลี่ยของสัญญาณรบกวนในเชิงสถิติ ตามที่ได้อ้างถึงมาแล้วนั้น มีการนิยามดังต่อไปนี้

2.1.1.1 ค่าสถิติของสัญญาณรบกวน

ค่าเฉลี่ยของสัญญาณรบกวน $n(t)$ เขียนแทนด้วยสัญญาลักษณ์ $\overline{n(t)}$ มีนิยามดังนี้ คือ

$$\overline{n(t)} \equiv \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} n(t)\,dt \qquad (2.1)$$

ในที่นี้การใช้เครื่องหมายเส้นขีดเหนือค่าฟังก์ชันใด เป็นเครื่องหมายแสดงว่า นั่นคือค่าเฉลี่ยของฟังก์ชันนั้นต่อเวลา ค่าเฉลี่ย $\overline{n(t)}$ นี้บอกให้เรารู้ถึงคุณสมบัติทางกายภาพ ของสัญญาณรบกวน $n(t)$ นั้นว่ามีระดับไฟตรงที่เป็นค่าเฉลี่ย ของสัญญาณรบกวนนั้นอยู่เป็นค่าเท่าใด

ในการค่าค่าเฉลี่ยทางปฏิบัติ ค่าระยะเวลาT ใน (2.1)นั้นจะใช้เพียงค่า ระยะเวลาที่ยาวนานค่าหนึ่งเท่านั้น ซึ่งถ้าค่าระยะนี้ยาวนานพอสมควรแล้ว ค่า$n(t)$ที่หามาได้ก็จะใช้ประมาณบอกถึงค่าเฉลี่ยตาม(2.1) ได้ใกล้เคียงกับค่าทางทฤษฎี

2.1.1.2 ค่ากำลังสองเฉลี่ยของสัญญาณรบกวน

ค่ากำลังสองเฉลี่ยของสัญญาณรบกวน $\overline{n^2(t)}$ หาได้ดังนี้

$$n^2(t) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} |n(t)|^2 \, dt \qquad (2.2)$$

ค่าเฉลี่ยตาม(2.2)นี้ แสดงถึงปริมาณทางกายภาพ คือ ค่ากำลังเฉลี่ย (Average power) หรือค่ากำลังประสิทธิผล (effective power) ทั้งหมดของสัญญาณรบกวนนั้นว่ามีค่าเท่าไร

2.1.1.3 ค่าความแปรปรวนของสัญญาณรบกวน

ค่าความแปรปรวนของสัญญาณรบกวน σ_n^2 หาได้จากนิยามดังต่อไปนี้ คือ

$$\sigma_n^2 = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \left\{ n(t) - \overline{n(t)} \right\}^2 dt \qquad (2.3)$$

เนื่องจาก$n(t) - \overline{n(t)}$นั้นแสดงถึงส่วนของ$n(t)$ที่เบี่ยงเบนไปจากค่าเฉลี่ย $\overline{n(t)}$ดังนั้นค่า $\left\{ n(t) - \overline{n(t)} \right\}$ จึงมีความหมายแทนปริมาณส่วนที่เป็นสลับของสัญญาณรบกวน$n(t)$ที่ขี่ซ้อนอยู่บนค่าระดับไฟตรงเฉลี่ย$\overline{n(t)}$ด้วยเหตุนี้ ค่าความแปรปรวนของสัญญาณรบกวนσ_n^2 จึงแสดงให้เรารู้ถึงค่ากำลังเฉลี่ยของส่วนที่เป็นไฟสลับของสัญญาณรบกวน$n(t)$ และค่าความเบี่ยงมาตรฐานσ_n ของสัญญาณรบกวน$n(t)$จะบอกให้เรารู้ถึงค่าอาร์เอ็มเอส (RMS) ของสัญญาณรบกวน$n(t)$ นั้น

ค่าทางสถิติตาม (2.1) ถึง (2.3) นั้นมีความสัมพันธ์ดังต่อไปนี้

$$\overline{n^2(t)} = \sigma_n^2 + \left\{ \overline{n(t)} \right\}^2 \qquad (2.4)$$

ตามความสัมพันธ์ (2.4) นี้สามารถพิสูจน์ได้โดยอาศัยการบวก $\overline{n(t)}$เข้าไปกับ $n(t)$แล้วลบ $\overline{n(t)}$ออกพร้อมๆกันซึ่งจะไม่ทำให้ค่าของ $n(t)$เปลี่ยนไป แล้วอาศัย (2.2) และทำการกระจายพจน์ จะได้ผลลัพธ์ดังต่อไปนี้ คือ

$$\begin{aligned}\overline{n^2(t)} &= \overline{[\{n(t)-\overline{n(t)}\}+\overline{n(t)}]^2} \\ &= \overline{\{n(t)-\overline{n(t)}\}^2+2\overline{n(t)}\{n(t)-\overline{n(t)}\}+\{\overline{n(t)}\}^2} \\ &= \overline{\{n(t)-\overline{n(t)}\}^2}+2\overline{n(t)}\,\overline{\{n(t)-\overline{n(t)}\}}+\{\overline{n(t)}\}^2\end{aligned}$$

เนื่องจาก $\overline{\{n(t)-\overline{n(t)}\}}$ มีค่าเท่ากับศูนย์ เพราะฉะนั้นโดยอาศัยคำจำกัดความของ σ_n^2 ตาม (2.3) จะทำให้สามารถสรุปผลจากสมการบนได้ว่า คือ (2.4)

สมการ (2.4) มีความหมายในทางกายภาพว่า กำลังสองทั้งหมดของสัญญาณรบกวนมีค่าเท่ากับกำลังสองเฉลี่ยของส่วนที่เป็นไฟสลับของสัญญาณรบกวน รวมกับกำลังสองของส่วนที่เป็นไฟตรงของสัญญาณรบกวนนั้น

2.1.2 อัตราส่วนสัญญาณต่อสัญญาณรบกวน

เพราะเราไม่สามารถจะคาดการณ์ว่าสัญญาณรบกวนนั้นที่เวลาต่างๆจะมีค่าเป็นเท่าไร ดังนั้นเมื่อเราต้องการจะวิเคราะห์เกี่ยวกับสัญญาณรบกวน จึงสมควรที่จะพิจารณาใช้ค่ากำลังเฉลี่ยของสัญญาณรบกวนเหล่านั้นที่เป็นไปตาม (2.1) ถึง (2.3) มาเป็นเกณฑ์ในการช่วยวิเคราะห์สัญญาณรบกวนนั้น เรื่องสำคัญที่ควรพิจารณาเมื่อทำปฏิบัติการเกี่ยวสัญญาณที่มีกำลังอยู่ในระดับต่ำ ก็คือจะพบว่าสัญญาณรบกวนมักจะเข้ามามีอิทธิผลครอบงำสัญญาณนั้น การที่จะบอกว่าสัญญาณนั้น ถูกรบกวนโดยสัญญาณรบกวนมากน้อยเท่าใดนั้นอย่างหนึ่งก็คือ ใช้ ค่าอัตราส่วนสัญญาณต่อสัญญาณรบกวน (signal to noise ratio) ซึ่งนิยมเขียนแทนด้วยสัญลักษณ์ $\frac{S}{N}$ และนิยมเรียกว่าย่อว่าค่า เอสเอ็นอาร์ (SNR) เป็นตัวช่วยแสดงให้เรารู้ถึงค่าความแตกต่างระหว่างค่ากำลังของสัญญาณทั้งสองนั้น

ค่าเอสเอ็นอาร์นั้น ถูกนิยามว่า คือค่าอัตราส่วนของกำลังเฉลี่ยของสัญญาณ $\overline{s^2(t)}$ ต่อค่ากำลังเฉลี่ยของสัญญาณรบกวน $\overline{n^2(t)}$ ซึ่งเขียนเป็นสมการได้ดังนี้

$$\frac{S}{N} \equiv \frac{\overline{s^2(t)}}{\overline{n^2(t)}} \qquad (2.5)$$

ค่าเอสเอ็นอาร์นี้ ปรกติมักนิยมแสดงค่าในหน่วย เดซิเบล (Decibel) ซึ่งเขียนย่อว่า dB การแปลงค่าเอสเอ็นอาร์ ตาม (2.5) ให้มีหน่วยเดซิเบลทำได้ดังนี้

$$\left.\frac{S}{N}\right|_{dB} = 10\log\left(\frac{\overline{s^2(t)}}{\overline{n^2(t)}}\right) \qquad (2.6)$$

ค่าเอสเอ็นอาร์อานี้ เป็นค่าซึ่งแสดงถึงคุณภาพของสัญญาณที่กำลังพิจารณาว่ามีระดับกำลังสูงกว่าระดับกำลังของสัญญาณที่ปนอยู่ในสัญญาณนั้นมากน้อยเท่าไร

2.1.3 สัญญาณรบกวนขาว และสัญญาณรบกวนเนื่องจากอุณหภูมิ

สัญญาณรบกวนเกิดจากสาเหตุต่างๆกัน และมีรูปแบบพีเอสดีที่แตกต่างกัน สัญญาณรบกวนที่ควรสนใจมากที่สุด ถ้าไม่ได้มีการกล่าวบอกถึงค่าเฉลี่ยของมัน เราก็จะหมายถึงสัญญาณรบกวนที่มีค่าเฉลี่ยของมันเป็นศูนย์ เราก็อาจจะคิดเทียบได้ว่า สัญญาณรบกวนนั้น คือ สัญญาณรบกวนที่มีค่าเฉลี่ยเป็นศูนย์รวมอยู่กับสัญญาณไฟตรงที่มีค่าเท่ากับค่าเฉลี่ยของสัญญาณรบกวนนั้นได้

สัญญาณรบกวนที่มีการแจกแจงของค่ากำลังเท่ากันของทุกความถี่ บนแกนความถี่ข้างเดียวเท่ากับ η วัตต์ต่อเฮิรตซ์ จะมีค่าฟังก์ชันพีเอสดีในรูปแบบของสเปกตรัมสองข้าง คือ

$$S_n(\omega) = \frac{\eta}{2}, \quad \text{ที่ทุกความถี่} \tag{2.7}$$

เป็นที่ควรสังเกตว่า การกำหนดสัญญาณรบกวนขาวว่า คือสัญญาณรบกวนที่มีค่าพีเอสดีคงที่ตลอดทุกความถี่นั้น เป็นการกำหนดโดยความเป็นลักษณะตามอุดมคติ ทั้งนี้เพราะว่า การกำหนดเช่นนี้จะทำให้ ค่ากำลังเฉลี่ยของสัญญาณรบกวนชนิดนี้ มีค่ามากอนันต์ กล่าวคือ

$$\overline{n^2(t)} = \frac{1}{2\pi}\int_{\omega\to-\infty}^{\omega\to\infty}\left(\frac{\eta}{2}\right)d\omega \quad \to \infty \tag{2.8}$$

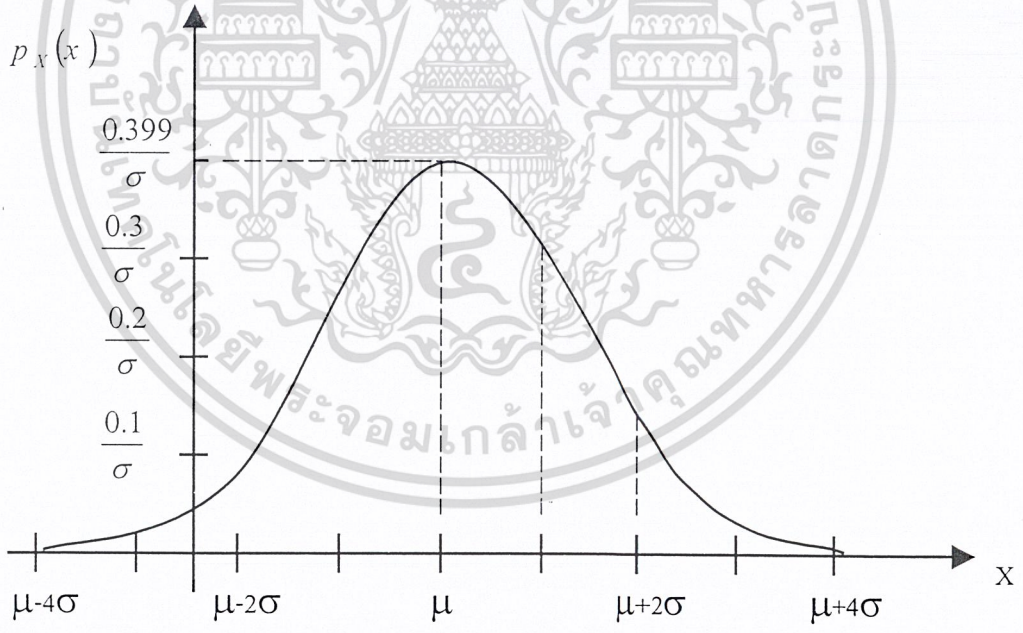
ซึ่งสัญญาณที่มีค่ากำลังเฉลี่ยมหาศาลเช่นนี้ ย่อมไม่มีในทางปฏิบัติ แต่อย่างไรก็ตาม การกำหนดนิยามเช่นนี้ นับได้ว่าเป็นรูปแบบที่ดีสำหรับเมื่อแบนด์วิดท์ของระบบที่กำลังใช้งานนั้น แคบกว่าแบนด์วิดท์ของสัญญาณรบกวนที่มีอยู่มาก ในกรณีเช่นนี้ระบบนั้นก็จะปฏิบัติการอยู่ในช่วงแบนด์วิดท์ที่จำกัดของสัญญาณเท่านั้น เพราะฉะนั้นโดยทางปฏิบัติแล้ว สัญญาณรบกวนขาวที่เราสนใจก็จะเป็นสัญญาณรบกวนที่อยู่ในลักษณะ สัญญาณรบกวนขาวที่มีความถี่จำกัด (band-limited white noise) เท่านั้น ซึ่งเมื่อเป็นเช่นนี้ก็จะเห็นว่าเพียงพอสำหรับการวิเคราะห์ระบบนั้น กล่าวอีกในหนึ่งได้ว่าในทางปฏิบัตินั้น ถึงแม้ระบบที่เรากำลังใช้อยู่จะถูกรบกวนด้วยสัญญาณรบกวนขาวตามอุดมคติ ส่วนประกอบของสัญญาณรบกวนขาวที่มีความถี่พ้นไกลไปจากแบนด์วิดท์ของระบบนั้น ก็จะไม่มีผลกระทบกับระบบที่เรากำลังให้ความสนใจอยู่เลย องค์ประกอบด้านความถี่ของสัญญาณรบกวนขาวที่มีอิทธิพลต่อระบบนั้น จะเป็นองค์ประกอบซึ่งมีความถี่ซึ่งอยู่ในย่านความถี่ที่จำกัดเท่านั้น ตัวอย่างของสัญญาณรบกวนขาวที่มีย่านความถี่จำกัด คือ สัญญาณรบกวนเนื่องจากอุณหภูมิ (thermal noise)

2.1.4 สัญญาณรบกวนแบบเกาส์เซียน

สำหรับสัญญาณรบกวนแบบเกาส์เซียนนั้น จะเป็นรูปแบบที่มีความสำคัญสำหรับการคำนวณเกี่ยวกับระบบสื่อสาร เนื่องมาจากสัญญาณรบกวนในรูปแบบของ Thermal noise ที่เกิดขึ้นในระบบสื่อสารนั้น จะมีลักษณะในลักษณะในการเกิดที่คล้ายกับรูปแบบของฟังก์ชั่นเกาส์เซียนโดยที่กระบวนการสุ่มแบบเกาส์เซียน นั้นเป็นกระบวนการสุ่มแบบต่อเนื่อง ซึ่งมีลักษณะของพีดีเอฟ(probability density function) ที่มีลักษณะดังสมการต่อไปนี้

$$P_x(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (2.9)$$

โดยในที่นี้ μ และ σ^2 คือค่าเฉลี่ยและค่าความแปรปรวนของพีดีเอฟ(ตัวแปรสุ่มใต้พีดีเอฟ)ตามลำดับ ค่า $\frac{1}{\sigma\sqrt{2\pi}}$ เป็นค่าน้ำหนักที่ใช้ถ่วงเพื่อทำให้พื้นที่ใต้เส้นโค้งของพีดีเอฟ มีค่าเท่ากับ 1 รูปกราฟของพีดีเอฟแบบเกาส์เซียนนี้มีดังแสดงในรูปที่ 2.1



รูปที่ 2.1 พีดีเอฟแบบเกาส์เซียน

2.1.5 สัญญาณรบกวนขาวแบบเกาส์เซียน (White Gaussian Noise)

ในทางวิศวกรรมไฟฟ้า กระบวนการสุ่มแบบเกาส์เซียนจะเป็นโมเดลของแรงดันไฟฟ้าของสัญญาณรบกวนในตัวต้านทาน และเป็นโมเดลของสัญญาณรบกวนในเครื่องรับของระบบการสื่อสาร สัญญาณรบกวนเป็นรูปคลื่นที่ไม่สามารถทำนายได้ โดยที่เราจะให้โมเดลของกระบวนการสุ่มแบบเกาส์เซียนเป็น $W(t)$ และปราศจากส่วนประกอบทางแรงดันไฟฟ้ากระแสตรง(DC component) ดังนั้น

$$E[W(t_1)] = \mu_W = 0 \tag{2.10}$$

โดยธรรมชาติกระบวนการของสัญญาณรบกวน เราสมมุติให้ผลสะสมของชั่วเวลาใดๆ $t_1,...,t_k, W(t_1),...,W(t_k)$ เป็นเซตของตัวแปรสุ่มอิสระ ในกรณีนี้ค่าของสัญญาณรบกวนที่เวลา t_i ไม่ได้บอกอะไรเกี่ยวกับค่าของสัญญาณรบกวน t_j เป็นผลให้ที่เวลา $\tau \neq 0$

$$R_W(\tau) = E[W(t)W(t+\tau)] = E[W(t)]E[W(t+\tau)] = 0 \tag{2.11}$$

ค่า $W(t)$จะสมบูรณ์ก็ต่อเมื่อเราหาค่า $R_W(0)$ แล้วพิจารณาค่าความหนาแน่นกำลังงาน (Power spectral density) $S_W(f)$ โดยมีนิยามเป็นดังนี้

$$S_W(f) = \int_{-\infty}^{\infty} R_W(\tau) e^{-j2\pi f\tau} d\tau \tag{2.12}$$

โดยที่ $R_W(\tau) = 0$ เมื่อ $\tau \neq 0$และ $S_W(f)$ จะคงที่ทุกความถี่ ค่าคงที่จะมีค่าเป็นศูนย์ยกเว้นที่ $R_W(\tau) = \delta(\tau)$

2.1.6 สัญญาณรบกวนทางเฟสและสัญญาณรบกวนสี (Phase Noise, Colored Noise)

นอกจากสัญญาณรบกวนในรูปแบบของสัณญาณรบกวนขาวแล้ว จะมีรูปแบบของการเกิดความผิดพลาดของสัญญาณในหลายรูปแบบด้วยกัน ได้แก่ Phase noise หรือ timing jitter และ colored noise

สำหรับการส่งสัญญาณในรูปแบบของข้อมูลดิจิตอลผ่านระบบสื่อสารนั้น นอกจากจะต้องมีการพิจารณาถึงสัญญาณรบกวนต่างๆที่เกิดขึ้นในระบบสื่อสารแล้ว จะต้องมีการพิจารณาถึงความผิดเพี้ยนของสัญญาณในรูปแบบของ Phase noise หรือ timing-jitter ด้วย ซึ่งจะเป็นความผิดเพี้ยนของสัญญาณที่เกิดมาจากคุณสมบัติของช่องสัญญาณที่ใช้สำหรับส่งข้อมูลนั้น มีผลตอบสนองทางความถี่ในแต่ละความถี่ที่ไม่เท่ากัน โดยจะส่งผลให้มีการลดทอนในแต่ละความถี่ที่ไม่เท่ากัน ดังนั้น ผลที่ตามมาคือจะทำให้ลักษณะของสัญญาณที่รับได้นั้น อาจมีรูปแบบที่ผิดเพี้ยนไป เช่นมีความคมของลูกคลื่นลดลง ซึ่งอุปกรณ์อย่างหนึ่งที่จะนำไปใช้ในการพิจารณาได้แก่ eye-diagram

2.2 ทฤษฎีความน่าจะเป็นและกระบวนการสุ่ม

2.2 .1 ความน่าจะเป็นหรือโอกาสการเกิด

ในการทดลองบางอย่างจะให้ผลลัพธ์ที่ไม่มีความแน่นอน คือ มีความเป็นไปได้หลายอย่าง เช่นในการทดลองลูกเต๋าครั้งหนึ่งผลลัพธ์ที่ได้ออกมาอาจจะเป็นได้ถึง 6 ค่า คือ ค่าจาก 1 ถึง 6 ถ้ากำหนดให้ A แทนค่าผลลัพธ์ที่อาจเกิดขึ้นในการทดลองเช่นนั้นค่าหนึ่ง เช่นในกรณีของการทอดลูกเต๋า A นี้อาจเป็นค่าเลขบนหน้าลูกเต๋า หน้าใดหน้าหนึ่งที่ปรากฏขึ้นมาภายหลังจากการทอดลูกเต๋าเป็นต้น ในการทอดลูกเต๋าทั้งหมด N ครั้งสมมุตว่าได้ผลออกมาเป็น A จำนวน N_A ครั้งอัตราส่วน $\frac{N_A}{N}$ นี้ ถูกนิยามว่า คือ ความถี่สัมพัทธ์ (relative frequency) ของเหตุการณ์นี้ ซึ่งค่าอัตราส่วนนี้คงจะใช้บอกอะไรไม่ได้มากนัก ถ้าหากว่าจำนวนการทดลอง N นั้นมาก ค่าความถี่สัมพัทธ์ของเหตุการณ์นั้นจะมีค่าลิมิต (limit) เข้าสู่ค่าหนึ่งๆ ค่าลิมิตนี้ถูกนิยามว่า คือ ค่าความน่าจะเป็น (probability) หรือโอกาสการเกิดของผลลัพธ์นั้น โอกาสการเกิดของผลลัพธ์ A เขียนเป็นนิยามในรูปแบบเชิงคณิตศาสตร์ได้ คือ

$$P(A) = \lim_{N\to\infty} \frac{N_A}{N} \tag{2.13}$$

ควรสังเกตจาก(2.12)ว่า ค่าของ$P(A)$นั้นมีได้สูงสุดเท่ากับ 1

$$0 \le P(A) \le 1 \tag{2.14}$$

การทดลองนี้ให้ผลลัพธ์ คือ A ออกมานั้น ในบางครั้งในวิชาสถิติอาจจะกล่าวว่าในการทดลองนั้นมี เหตุการณ์ (Event) A เกิดขึ้น ซึ่งก็ควรทำความเข้าใจว่า เหตุการณ์ต่างๆก็คือ ผลลัพธ์ต่างๆนั้นเอง

เหตุการณ์สองเหตุการณ์ถูกนิยามว่า ไม่มีส่วนร่วม(Disjoint) หรือไม่เกิดร่วมกันและกัน(mutually exclusive) ถ้าเหตุการณ์ทั้งสองนั้นไม่มีโอกาสเกิดขึ้นพร้อมๆกัน กล่าวคือ ในกรณีที่ถ้าเหตุการณ์ A เกิดขึ้นแล้วเหตุการณ์ B ก็จะไม่เกิดขึ้นหรือในทางกลับกันก็คือถ้าเหตุการณ์ B เกิดขึ้นแล้วเหตุการณ์ A ก็จะไม่เกิดขึ้น เราจะกล่าวเหตุการณ์ A และ B นั้นไม่มีส่วนร่วมกัน อธิบายได้ด้วยสัญลักษณ์ทางคณิตศาสตร์ดังนี้คือ

$$P(A \cap B) = P(AB) = 0 \tag{2.15}$$

ในกรณีของเหตุการณ์ที่ไม่มีส่วนร่วมกันเช่นนี้ เราจะพบว่าโอกาสการเกิดของเหตุการณ์ A หรือเหตุการณ์ B ซึ่งใช้สัญลักษณ์ $P(A \cup B)$มีค่าดังนี้ คือ

$$P(A \cup B) = \lim_{N \to \infty} \frac{N_A + N_B}{N} = P(A) + P(B) \quad (2.16)$$

โดยในที่นี้ N_A และ N_B คือจำนวนผลลัพธ์จากการทดลองที่เป็นเหตุการณ์ A และ B ตามลำดับ

2.2.2 โอกาสใต้เงื่อนไข และความเป็นอิสระทางสถิติ

เหตุการณ์ 2 เหตุการณ์ คือ A และ B ที่อาจเกิดขึ้นได้พร้อมกันในบางครั้งนั้น ถ้ามีบางส่วนของเหตุการณ์ A เกิดขึ้นพร้อมเหตุการณ์ B โอกาสของการเกิดเหตุการณ์ A และ B ขึ้นพร้อมกันนี้เราเรียกว่าโอกาสการเกิดร่วม (joint probability)ของ A และ B ในการทดลองN ครั้ง ถ้าให้โอกาสในการเกิด A และ B ร่วมกันคือ N_{AB} แล้ว โดยใช้หลักการของความถี่สัมพัทธ์จะได้

$$P(AB) = \lim_{N \to \infty} \frac{N_{AB}}{N} \quad (2.17)$$

ย่อมเป็นเรื่องปกติที่ $N_{AB} \leq N_A$ และ $N_{AB} \leq N_B$ ทั้งนี้เนื่องจากเหตุการณ์ A และเหตุการณ์ B ไม่จำเป็นที่จะต้องเกิดขึ้นพร้อมกันเสมอ

ในการทดลองบางอย่างการเกิดขึ้นของเหตุการณ์ B อาจเกี่ยวพันกับการเกิดเหตุการณ์ A โอกาสที่เหตุการณ์ B เกิดขึ้น เมื่อเหตุการณ์ A ได้เกิดขึ้นแล้ว มีชื่อเรียก โอกาสใต้เงื่อนไข (Conditional probability) ของเหตุการณ์ B เมื่อกำหนดเหตุการณ์ A มีสัญลักษณ์เป็น $P(B/A)$ ซึ่งถ้านิยามตามหลักของความถี่สัมพัทธ์จะได้

$$P(B/A) = \lim_{N \to \infty} \frac{N_{AB}}{N_A} = \lim_{N \to \infty} \frac{N_{AB}/N}{N_A/N}$$

$$= \frac{P(AB)}{P(A)} \text{ เมื่อ } P(A) \neq 0 \quad (2.18)$$

ในทำนองเดียวกันจะได้ว่า

$$P(A/B) = \frac{P(AB)}{P(B)} \text{ เมื่อ } P(B) \neq 0 \quad (2.19)$$

จาก (2.18) และ (2.19) จะได้

$$P(AB) = P(A/B)P(B) = P(B/A)P(A) \quad (2.20)$$

หรือ

$$P(B/A) = \frac{P(B)P(A/B)}{P(A)} \tag{2.21}$$

ความสัมพันธ์ตาม (2.21) รู้จักกันทั่วไปว่าเป็น กฎของเบส์ (Bayes' rule)

ถ้าโอกาสใต้เงื่อนไขของเหตุการณ์ B เมื่อกำหนดเหตุการณ์ A มีค่าเท่ากับโอกาสการเกิดของเหตุการณ์ B โดยปกติ กล่าวคือ

$$P(B/A) = P(B) \tag{2.22}$$

เราจะกล่าวว่าเหตุการณ์ B เป็นอิสระทางสถิติกับเหตุการณ์ A จงสังเกตว่าการเป็นอิสระของเหตุการณ์ทางสถิติไม่ได้หมายความว่าเหตุการณ์ A เกิดแล้ว เหตุการณ์ B จะไม่เกิดขึ้น แต่มันบอกว่าเมื่อเหตุการณ์ A เกิดขึ้นแล้ว โอกาสที่เหตุการณ์นั้นจะเป็นเหตุการณ์ B ด้วย จะมีค่าเท่ากับโอกาสของเหตุการณ์ B ที่จะเกิดขึ้นโดยลำพังเมื่อเทียบกับเหตุการณ์ในการทดลองทั้งหมด หรือถ้าใช้หลักของความถี่สัมพัทธ์ก็จะได้ว่าในกรณีที่ B เป็นอิสระทางสถิติกับเหตุการณ์ A จะได้

$$\lim_{N\to\infty} \frac{N_{AB}}{N} = \lim_{N\to\infty} \frac{N_B}{N} \tag{2.23}$$

จาก (2.18) และ (2.22) ทำให้เราได้เงื่อนไขของเหตุการณ์ A และเหตุการณ์ B ที่เป็นอิสระกันทางสถิติ คือ

$$P(AB) = P(A)P(B) \tag{2.24}$$

การสังเกตว่าคุณสมบัติความเป็นอิสระกันทางสถิติของเหตุการณ์ 2 เหตุการณ์นั้น แตกต่างกันจากคุณสมบัติความไม่มีส่วนร่วม หรือไม่เกิดร่วมกันของเหตุการณ์ 2 เหตุการณ์ ซึ่งแสดงว่าเหตุการณ์ A เกิดขึ้นแล้วเหตุการณ์ B จะไม่เกิดขึ้นอย่างแน่นอน คือ $P(AB) = 0$

2.2.3 ตัวแปรสุ่ม

ผลลัพธ์ของการทดลอง (Random experiment) อาจเป็นค่าตัวเลข เช่นผลลัพธ์ที่เกิดจากทอดลูกเต๋า หรืออาจจะเป็นผลลัพธ์ที่ไม่เป็นค่าตัวเลข เช่นการติดหรือดับของดวงไฟ แต่อย่างไรก็ดีในการพิจารณาเชิงคณิตศาสตร์นั้น ต้องการที่จะพิจารณาผลลัพธ์ที่เป็นค่าเชิงตัวเลข เพราะมันอาจจะอำนวยความสะดวกในการประเมินผล ด้วยเหตุนี้เองจึงได้เกิดแนวความคิดที่จะกำหนดค่าตัวเลขจริงให้กับผลลัพธ์จากการทดลองทางสถิติที่ไม่ได้เป็นค่าตัวเลขโดยตรง เพื่อแปลงค่าผลลัพธ์นั้นให้กลายเป็นผลลัพธ์เชิงตัวเลขนั้น

โดยอาศัยกฎเกณฑ์บางอย่างที่เหมาะสม เช่นถ้าผลลัพธ์จากการทดลองมีจำนวนจำกัด เราอาจใช้สัญลักษณ์แทนผลการทดลองนั้นเป็น λ_i โดยที่ i นั้นคือลำดับของผลการทดลองนั้นได้ และใช้ความสัมพันธ์ในลักษณะเชิงฟังก์ชัน เช่น $X(\lambda)$ เพื่อกำหนดค่าของตัวเลขจริงให้กับผลลัพธ์ λ แต่ละตัวนั้น ค่า $X(\lambda)$ ที่ได้ขึ้นมานี้มีชื่อเรียกว่าตัวแปรสุ่ม (random variable) โดยทั่วไปตัวแปรสุ่มนิยมที่จะเขียนเป็นอักษรตัวใหญ่ ตัวแปรสุ่ม X ที่มีค่าเป็น $x_i = X(\lambda_i)$เมื่อ i คือดัชนีซึ่งบอกให้รู้ถึงลำดับของผลลัพธ์ในการทดลองนั้น ในส่วนบทบาทของตัวแปรสุ่ม และโอกาสการเกิดหรือความน่าจะเป็นนั้นต่างกันโดยสิ้นเชิง โอกาสการเกิดของตัวแปรสุ่ม X ที่มีค่า x_i นั้น นิยมเขียน สัญลักษณ์แทนด้วย $P_X(x_i)$

ตัวแปรสุ่มจะเป็นชนิดดิสครีต เมื่อ x_i มีค่าแยกกันอย่างชัดเจน และเงื่อนไขสำหรับตัวแปรชนิดดีสครีตที่เกี่ยวกับโอกาสการเกิดของมันก็คือ

$$\sum_i P_X(x_i) = 1 \qquad (2.25)$$

ตัวแปรสุ่มชนิดดีสครีตจะมีจำนวนที่จำกัด ในกรณีที่ตัวแปรสุ่มมีจำนวนไม่จำกัด และค่าของมันต่อเนื่องอยู่ในช่วงของตัวแปรที่แน่นอน ตัวแปรสุ่มชนิดนี้ จะถูกนิยามว่าเป็นตัวแปรสุ่มชนิดต่อเนื่อง (Continuous random variable)

แนวความคิดในเรื่องโอกาสการเกิด เมื่อนำมารวมกับแนวความคิดในเรื่องตัวแปรสุ่มจะทำให้เกิดประโยชน์อย่างยิ่ง เมื่อเราเกี่ยวข้องกับตัวแปรสุ่ม X และY การกำหนดค่าโอกาสเกิดร่วมกัน (Joint probability) ของมัน คือ $P_{XY}(x_i, y_j)$ นับว่าเป็นสิ่งที่ควรสนใจ $P_{XY}(x_i, y_j)$ หมายถึงโอกาสที่ $X = x_i$ และ$Y = y_j$ นั้นมีสัดส่วนหรือความน่าจะเป็นเท่าไร ความสัมพันธ์ที่ควรสนใจก็คือ

$$\sum_i \sum_j P_{XY}(x_i, y_j) = 1 \qquad (2.26)$$

โอกาสใต้เงื่อนไขของ $X = x_i$ เมื่อกำหนด$Y = y_j$ จะใช้สัญลักษณ์ คือ $P_{X/Y}(x_i/y_j)$จะมีเงื่อนไขสอดคล้องกับหลักความน่าจะเป็น คือ

$$\sum_i P_{X/Y}(x_i/y_j) = \sum_j P_{Y/X}(y_j/x_i) = 1 \qquad (2.27)$$

โดยใช้คุณสมบัติสอดคล้องกับ (2.20) เราจะได้

$$\begin{aligned} P_{XY}(x_i, y_j) &= P_{X/Y}(x_i/y_j) P_Y(y_j) \\ &= P_{Y/X}(y_j/x_i) P_X(x_i) \end{aligned} \qquad (2.28)$$

ซึ่งจะเป็นผลที่นำไปสู่ กฎของเบส์สำหรับตัวแปรสุ่ม คือ

$$P_{Y/X}(y_j/x_i) = \frac{P_Y(y_j)P_{X/Y}(x_i/y_j)}{P_X(x_i)} \tag{2.29}$$

จาก (2.28) จะได้

$$\begin{aligned} P_{XY}(x_i,y_j) &= P_{X/Y}(x_i/y_j)P_Y(y_j) \\ &= P_Y(y_j)\sum_i P_{X/Y}(x_i/y_j) \\ &= P_Y(y_j) \end{aligned} \tag{2.30}$$

ในทำนองเดียวกันจะได้

$$\sum_j P_{XY}(x_i,y_j) = P_X(x_i) \tag{2.31}$$

โอกาสการเกิด $P_X(x_i)$และ $P_Y(y_j)$มีชื่อเรียกว่า โอกาสช่วงขอบ หรือ ความน่าจะเป็นช่วงขอบ (Marginal probability) สมการ (2.30) และ (2.31) แสดงความสัมพันธ์ของการหาค่าโอกาสช่วงขอบของตัวแปรสุ่มจากค่าโอกาสการเกิดร่วมกันของตัวแปรสุ่มนั้น

2.2.4 ฟังก์ชันการแจกแจงสะสม และฟังก์ชันความหนาแน่นของความน่าจะเป็น

โอกาสการเกิดของค่าตัวแปรสุ่ม X เมื่อ X น้อยกว่าหรือเท่ากับ x ถูกนิยามว่าคือ ฟังก์ชันการแจกแจงสะสม (Cumulative distribution function) ของ X เรียกเป็นคำย่อว่า ซีดีเอฟ (CDF) และใช้สัญลักษณ์คือ $F_X(x)$ตามคำนิยามนี้เมื่อเขียนในเชิงคณิตศาสตร์จะได้ว่า

$$F_X(x) = P(X \le x) \tag{2.32}$$

เพราะว่า ซีดีเอฟ $F_X(x)$ นั้นมีรากฐานอยู่บนเนื้อหาของความน่าจะเป็น ดังได้อธิบายมาแล้วดังนั้นจึงมีคุณสมบัติดังต่อไปนี้คือ

$$0 \le F_X(x) \le 1 \tag{2.33}$$

$$F_X(x_1) \le F_X(x_2) \qquad \text{เมื่อ } x_1 \le x_2 \tag{2.34}$$

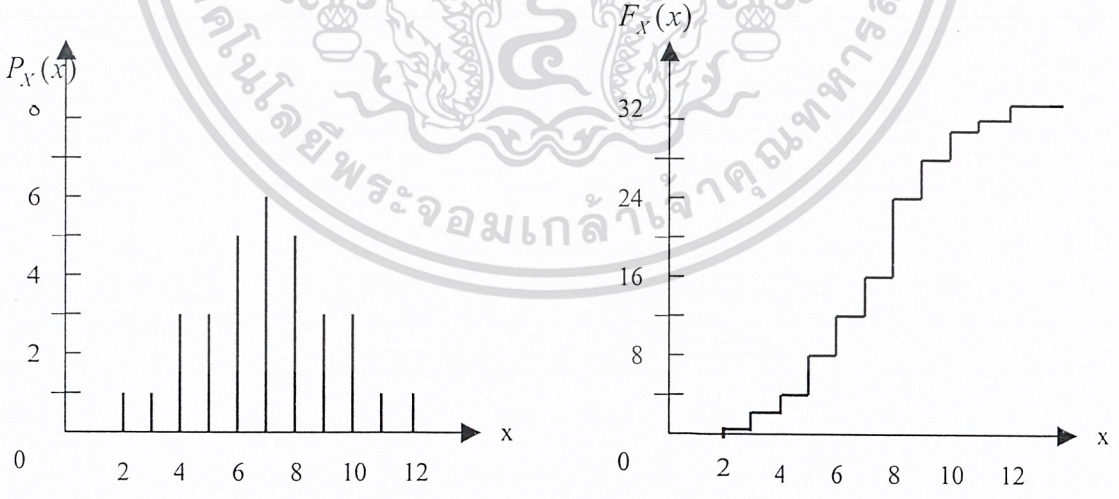
$$F_X(-\infty) = 0 \tag{2.35}$$

$$F_X(\infty) = 1 \tag{2.36}$$

สำหรับตัวแปรสุ่มชนิดดิสครีต $X = X(\lambda_i)$ซึ่งมีความน่าจะเป็น คือ P_i นั้น ค่าซีดีเอฟสามารถแสดงได้ดังต่อไปนี้ คือ

$$F_X(x) = \sum_i P_i\, u(x - x_i) \qquad (2.37)$$

ดังนั้น ซีดีเอฟของตัวแปรสุ่มชนิดดิสครีตจะเป็นอนุกรมของฟังก์ชันขั้นหรือฟังก์ชันขั้นบันได ซึ่งขอบของขั้นบันไดจะเกิดขึ้นที่จุด $X = x_i$ และความสูงของขั้นบันไดขึ้นกับค่า $P_i = P(X = x_i)$และในระหว่างจุด $X = x_i$ และ $X = x_{i+1}$ ค่า $F_X(x)$จะมีค่าคงที่คล้ายกับตัวแผ่นบันได ตัวอย่างลักษณะความสัมพันธ์ระหว่าง $P_X(x)$และ $F_X(x)$นั้นมีดังแสดงในรูปที่ 2.4ก และ 2.4ข ตามลำดับ ในกรณีของตัวแปรสุ่มชนิดค่าต่อเนื่องในช่วงตัวแปรที่กำหนดให้นั้น จะมีจำนวนของตัวแปรรวมกันอยู่มหาศาล ค่าโอกาสการเกิดหรือความน่าจะเป็นของตัวแปรสุ่ม ที่จุดใดจุดหนึ่ง เช่นที่ $X = x$ จะไม่สามารถกำหนดขึ้นมาได้ เนื่องจากจุดไม่มีขนาด กล่าวคือขนาดเป็นศูนย์ เมื่อจุดไม่มีขนาด เราก็ไม่สามารถที่จะหาความน่าจะเป็นของมันได้ ค่าความน่าจะเป็นที่จุดใดจุดหนึ่งนั้นหาไม่ได้ เพราะมีค่าลิมิตไปสู่ศูนย์ แม้ว่าจะพิจารณาว่าการนำจุดมาต่อๆกัน จะทำให้เกิดเป็นเส้นขึ้นก็ตาม แต่จุดไม่มีขนาด ดังนั้นโอกาสของการเกิดจุดในเส้นๆนั้น จึงมีค่าน้อยมากจนกำหนดไม่ได้ หรือถือว่าเป็นศูนย์นั้นเอง ด้วยเหตุผลดังกล่าวนี้ สำหรับค่าตัวแปรสุ่มชนิดที่มีค่าต่อเนื่อง การกล่าวถึงความน่าจะเป็นของ $X = x$ จึงมีความหมายที่ไม่สมบูรณ์ ยกเว้นแต่จะกล่าวใหม่ให้ถูกต้องว่าความน่าจะเป็นที่บริเวณ $X = x$ หรือ $x < X \le x + \Delta x$ นั้นมีค่าเป็นอย่างไร ดังนั้นจะเห็นได้ว่าในทัศนะเช่นนี้ แนวความคิดเรื่องซีดีเอฟน่าจะอำนวยความสะดวกและเหมาะสมดี เพราะสำหรับในกรณีนี้



(ก) ความน่าจะเป็นแบบดิสครีต (ข) ซีดีเอฟของความน่าจะเป็นตามรูป(ก)

รูปที่ 2.2 กราฟแสดงความน่าจะเป็น $P_X(x)$และแสดงค่าซีดีเอฟ $F_X(x)$

ความน่าจะเป็นสำหรับ $x < X \leq x + \Delta x$ ก็คือ $F_X(x+\Delta x) - F_X(x)$ นั่นเองทั้งนี้เพราะเรารู้ว่า

$$\begin{aligned} F_X(x+\Delta x) &= P(X \leq x+\Delta x) \\ &= P(X \leq x) + P(x < X \leq x + \Delta x) \\ &= F_X(x) + P(x < X \leq x+\Delta x) \end{aligned} \tag{2.38}$$

ด้วยค่าซีดีเอฟมีความเหมาะสมที่จะใช้งาน ดังที่เห็นได้จากดังที่อธิบายแล้วนี้ ในกรณีที่เราคิดว่า $\Delta x \rightarrow 0$ โดยใช้อนุกรมของเทย์เลอร์ (Taylor's series) จะได้

$$F_X(x+\Delta x) = F_X(x) + \frac{dF_X(x)\Delta x}{dx} \tag{2.39}$$

จาก (2.38) และ (2.39) นี้เองทำให้เราได้

$$\lim_{\Delta x \to 0} \frac{d\,F_X(x)\Delta x}{dt} = P(x < X \leq x+\Delta x) \tag{2.40}$$

ค่าอนุพันธ์ของ $F_X(x)$ เมื่อเทียบกับ x นี้ จะนิยามแทนด้วย $p_X(x)$ กล่าวคือ

$$\frac{d\,F_X(x)}{dx} = p_X(x) \tag{2.41}$$

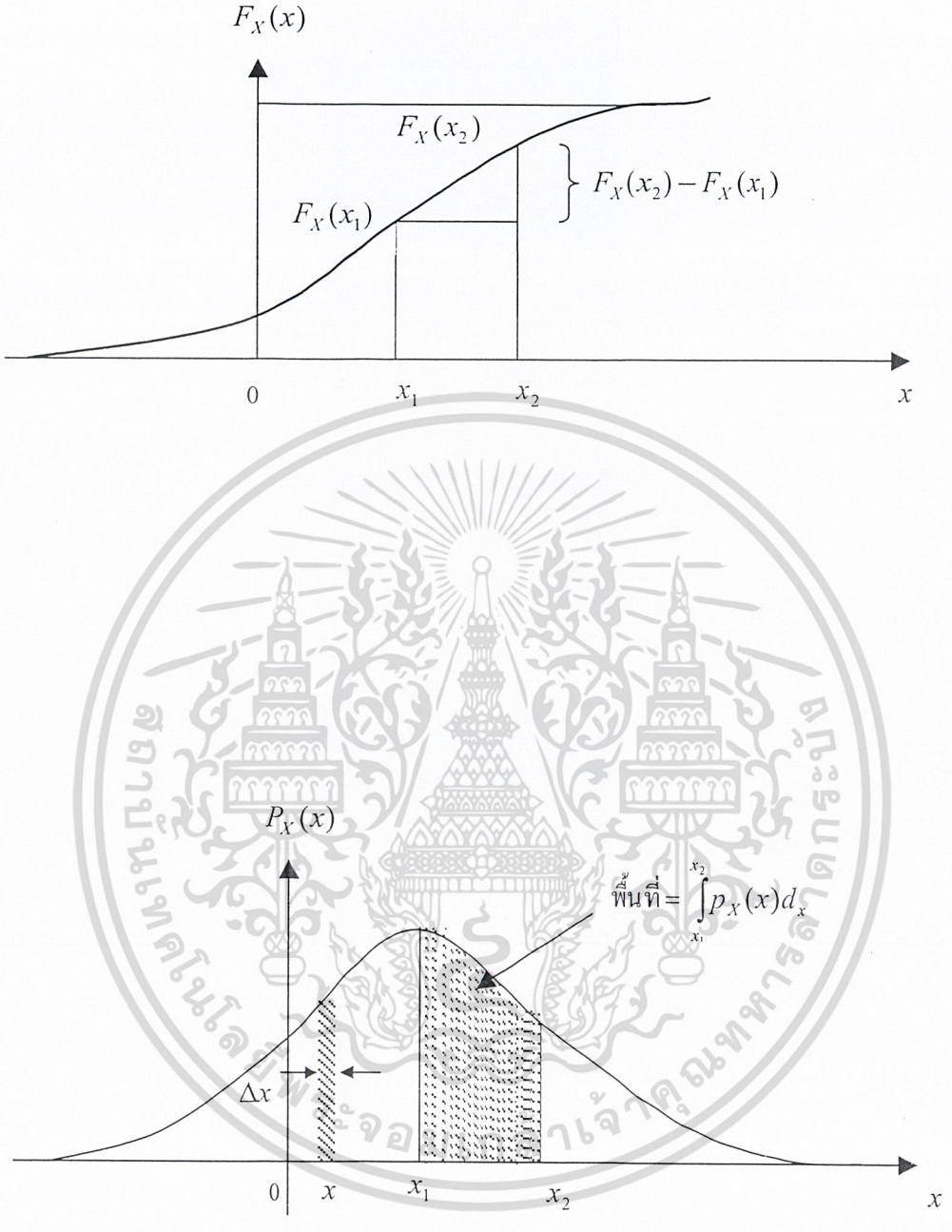
ฟังก์ชัน $p_X(x)$ นี้มีชื่อเรียกว่า ฟังก์ชันความหนาแน่นของความน่าจะเป็น (Probability density function) ของตัวแปรสุ่ม X ซึ่งต่อจากนี้จะใช้คำย่อแทนว่า พีดีเอฟ (PDF) ให้พิจารณาว่า ความน่าจะเป็นของตัวแปรสุ่ม X ในช่วง $(x\,,\,x+\Delta x\,]$ นี้มีค่าเท่าไร $p_X(x)\Delta x$ นั่นคือ พื้นที่ใต้พีดีเอฟ $p_X(x)$ ตลอดช่วง $(x\,,\,x+\Delta x\,]$ จาก (2.44) เราจะได้ว่า

$$F_X(x) = \int_{-\infty}^{x} p_X(x)dx \tag{2.42}$$

และจาก (2.40) เมื่อ $x = x_1$ และ $x + \Delta x = x_2$ เราจะได้

$$\begin{aligned} P(x_1 < X \leq x_2) &= F_X(x_2) - F_X(x_1) \\ &= \int_{-\infty}^{x_2} p_X(x)\,dx - \int_{-\infty}^{x_1} p_X(x)\,dx \\ &= \int_{x_1}^{x_2} p_X(x)\,dx \end{aligned} \tag{2.43}$$

ดังนั้นโอกาสการพบค่า X ในช่วง (x_1, x_2) จะมีค่าเท่ากับพื้นที่ใต้พีดีเอฟในช่วง (x_1, x_2) ดังแสดงในรูปที่ 2.3



รูปที่ 2.3 ค่าฟังก์ชันแจกแจงสะสม และฟังก์ชันความหนาแน่นของความน่าจะเป็น

เพราะ $F_X(\infty) = 1$ ดังนั้นจะได้

$$\int_{-\infty}^{\infty} p_X(x)\, dx \quad = \quad 1 \tag{2.44}$$

2.2.5 ค่าเฉลี่ยของตัวแปรสุ่ม

ค่าเฉลี่ยของปริมาณที่เกิดเป็นตัวแปรสุ่ม จะเป็นพารามิเตอร์ที่สำคัญประจำตัวของปริมาณนั้น ถ้าในการทดลอง N ครั้ง มีผลการทดลองออกมา n แบบ คือ $x_1, x_2, ..., x_n$ โดยผลลัพธ์แต่ละแบบกล่าวคือ x_i มีจำนวนการเกิดเป็น N_i แล้ว ค่าเฉลี่ยของผลลัพธ์จากการทดลองนี้คือ $\overline{X}$ จะหาได้จาก

$$\overline{X} = \frac{N_1 x_1 + N_2 x_2 + ... + N_n x_n}{N} \tag{2.45}$$

โดยในที่นี้ $N = \sum_i N_i$ นี้อาจเขียนแยกได้ในรูปความสัมพันธ์ต่อไปนี้คือ

$$\overline{X} = \frac{N_1}{N} x_1 + \frac{N_2}{N} x_2 + ... + \frac{N_n}{N} x_n \tag{2.46}$$

จะเห็นว่าเมื่อ $N \to \infty$ ค่าอัตราส่วน $\frac{N_i}{N} \to P_X(x_i)$ ดังนั้นจะได้

$$\overline{X} = \sum_i x_i \, P_X(x_i) \tag{2.47}$$

ค่าเฉลี่ยนี้บางครั้งก็เรียกว่า ค่ามัชฌิม(mean) หรือ ค่าคาดหมาย (expected value) ของตัวแปรสุ่ม X นั้นซึ่งมีสัญลักษณ์คือ $\mathrm{E}[x]$ จึงสรุปได้ว่า

$$\overline{X} = \mathrm{E}[x] = \sum_i x_i \, P_X(x_i) \tag{2.48}$$

และในกรณีที่ X เป็นตัวแปรสุ่มชนิดมีค่าต่อเนื่องจะได้

$$\overline{X} = \mathrm{E}[x] = \int_{-\infty}^{\infty} x \, P_X(x) dx \tag{2.49}$$

2.2.6 โมเมนต์ของตัวแปรสุ่ม

โมเมนต์ที่ n (n^{th} Moment) ของตัวแปรสุ่ม X ถูกนิยามว่า คือ ค่าเฉลี่ยของ X^n

$$\overline{X^n} = \int_{-\infty}^{\infty} x^n \, p_X(x) dx \tag{2.50}$$

และ เซนทรัลโมเมนต์ที่ n (n^{th} Central moment) ของตัวแปรสุ่ม X ถูกนิยามว่า คือ ค่าความคาดหมายของ $\left(X-\overline{X}\right)^n$ กล่าวคือ

$$\overline{\left(X-\overline{X}\right)^n} = \int_{-\infty}^{\infty}\left(X-\overline{X}\right)^n p_X(x)dx \qquad (2.51)$$

ค่าเซนทรัลโมเมนต์ที่ 2 มีความสำคัญเป็นพิเศษในทางสถิติมีชื่อพิเศษเฉพาะว่า ความแปรปรวน (variance) ของตัวแปรสุ่ม X นิยมใช้สัญลักษณ์แทนด้วย σ_X^2 และค่ารากที่ 2 ของความแปรปรวนคือ σ_X มีชื่อเรียกเฉพาะว่า ความเบี่ยงเบนมาตรฐาน (standard deviation) ของตัวแปรสุ่ม X เพราะฉะนั้นตามคำจำกัดความจะได้

$$\sigma_X^2 = \overline{\left(X-\overline{X}\right)^2} = \overline{X^2}-\overline{X}^2 \qquad (2.52)$$

2.2.7 การแจกแจงความน่าจะเป็นร่วมกันของฟังก์ชันของตัวแปรสุ่ม

ให้ X_1 และ X_2 เป็นตัวแปรสุ่มชนิดต่อเนื่อง โดยมีฟังก์ชันความหนาแน่นของความน่าจะเป็นร่วมกันเป็น p_{X_1,X_2} ในบางครั้งมีความจำเป็นที่จะต้องได้ค่าแจกแจงร่วมกันของตัวแปรสุ่มในรูปของ Y_1 และ Y_2 ซึ่งเกิดจากฟังก์ชันของ X_1 และ X_2 กำหนดให้ $Y_1=g_1(X_1,X_2)$ และ $Y_2=g_2(X_1,X_2)$ สำหรับฟังก์ชัน g_1 และ g_2 สมมุติให้ต้องเป็นไปตามเงื่อนไขดังต่อไปนี้

1. สมการ $y_1=g_1(x_1,x_2)$ และ $y_2=g_2(x_1,x_2)$ สามารถมีวิธีเฉพาะที่ทำให้ x_1 และ x_2 อยู่ในรูปของ y_1 และ y_2 ซึ่งเป็นวิธีที่พูดได้ว่า $x_1=h_1(y_1,y_2)$, $x_2=h_2(y_1,y_2)$

2. ฟังก์ชัน g_1 และ g_2 เป็นฟังก์ชันต่อเนื่องที่สามารถหาอนุพันธ์ย่อย (partial derivatives) ได้ทุกๆ จุดของ (x_1,x_2) และสามารถหาดีเทอร์มิแนนท์ (determinant) ขนาด 2x2 ได้

$$J(x_1,x_2) = \begin{vmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} \end{vmatrix} = \frac{\partial g_1}{\partial x_1}\cdot\frac{\partial g_2}{\partial x_2} - \frac{\partial g_1}{\partial x_2}\cdot\frac{\partial g_2}{\partial x_1} \neq 0 \qquad (2.53)$$

ภายใต้เงื่อนไข 2 ข้อนี้ สามารถที่จะแสดงตัวแปรสุ่ม Y_1 และ Y_2 ด้วยฟังก์ชันความหนาแน่นร่วมกัน (joint density function) โดย

$$p_{Y_1Y_2}(y_1,y_2) = p_{X_1X_2}(x_1,x_2)\left|J(x_1,x_2)\right|^{-1} \qquad (2.54)$$

เมื่อ $x_1=h_1(y_1,y_2)$, $x_2=h_2(y_1,y_2)$

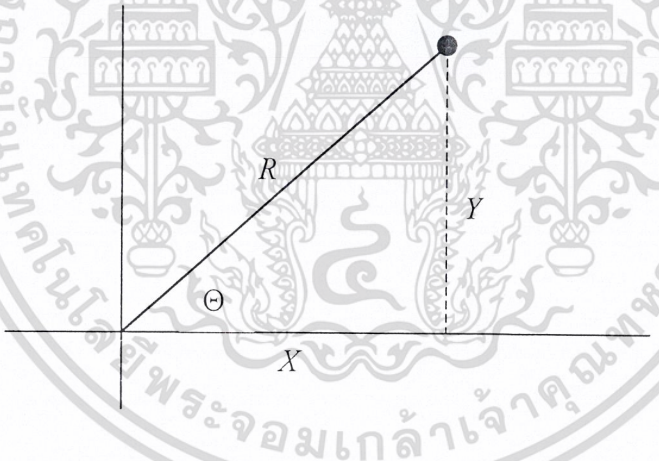
การจะพิสูจน์สมการ (2.54) นั้นสามารถทำได้แต่จะยาวและยุ่งยาก ดังนั้นจึงขอให้ดูสมการในบรรทัดต่อไป คือ

$$P\{Y_1 \le y_2, Y_2 \le y_2\} = \iint_{(x_1,x_2):} p_{X_1X_2}(x_1,x_2)dx_1dx_2 \qquad (2.55)$$

$$g_1(x_1,x_2) \le y_1$$
$$g(x_1,x_2) \le y_2$$

ฟังก์ชันความหนาแน่นร่วมกันสามารถที่จะหามาได้จากการหาอนุพันธ์ของสมการ (2.55) โดยสนใจค่า y_1 และ y_2 และผลลัพธ์ที่เกิดจากการหาอนุพันธ์จะเท่ากับ สมการ (2.54)

ตัวอย่างถ้า (X,Y) เป็นจุดที่สุ่มในระนาบ และสมมุติให้เป็นระนาบของพิกัดฉาก X และ Y เป็นอิสระต่อกัน และ เป็นตัวแปรสุ่มแบบปกติ แต่สิ่งที่เราสนใจในการแจกแจงร่วมกันคือ R, Θ ในพิกัดเชิงขั้ว ซึ่งแสดงจุดนี้ ในรูปที่ 2.4



รูปที่ 2.4 ● = จุดสุ่ม
$(X,Y) = R, \Theta$

ให้ $r = g_1(x,y) = \sqrt{x^2+y^2}$ และ $\theta = g_2(x,y) = \tan^{-1}\left(\frac{y}{x}\right)$ เราจะเห็นได้ว่า

$$\frac{\partial g_1}{\partial x} = \frac{x}{\sqrt{x^2+y^2}} \qquad \frac{\partial g_1}{\partial y} = \frac{-y}{\sqrt{x^2+y^2}}$$

$$\frac{\partial g_2}{\partial x} = \frac{1}{1+(y/x)^2}\left(\frac{-y}{x^2}\right) = \frac{-y}{x^2+y^2}$$

$$\frac{\partial g_2}{\partial y} = \frac{1}{1+(y/x)^2}\left(\frac{1}{x}\right) = \frac{x}{x^2+y^2}$$

ดังนั้น

$$J(x,y) = \frac{x^2}{\left(x^2+y^2\right)^{\frac{3}{2}}} + \frac{y^2}{\left(x^2+y^2\right)^{\frac{3}{2}}} = \frac{1}{\sqrt{x^2+y^2}} = \frac{1}{r}$$

ถ้ามีฟังก์ชันความหนาแน่นร่วมกันของ X และY เป็น

$$p(x,y) = \frac{1}{2\pi} e^{\frac{-\left(x^2+y^2\right)}{2}} \tag{2.56}$$

ดังนั้นเราจะเห็นได้ว่า ฟังก์ชันความหนาแน่นร่วมกันของ $R = \sqrt{x^2+y^2}$, $\Theta = \tan^{-1}\left(\frac{y}{x}\right)$ สามารถเขียนได้เป็น

$$p(r,\theta) = \frac{1}{2\pi} r e^{-\frac{r^2}{2}} \qquad 0<\theta<2\pi \quad , \quad 0<r<\infty \tag{2.57}$$

แฟคเตอร์ความหนาแน่นร่วมกันนี้ได้ชื่อว่าเป็นความหนาแน่นช่วงขอบ (Marginal densities) สำหรับR และΘ ซึ่งเราได้ R และΘ ที่เป็นตัวแปรสุ่มที่อิสระต่อกัน โดย Θ เป็นการแจกแจงแบบยูนิฟอร์มบนช่วง$(0,2\pi)$ และR เป็นการแจกแจงแบบเรลี่ย์(Raleigh)โดยมีความหนาแน่นเป็น

$$p(r) = r\, e^{-\frac{r^2}{2}} \qquad 0<r<\infty \tag{2.58}$$

รูปแบบของผลลัพธ์ที่ได้ในสมการที่ (2.57) สามารถที่สร้างตัวแปรสุ่มแบบปกติ โดยการทำทรานฟอร์มบนตัวแปรสุ่มแบบยูนิฟอร์ม ให้ U_1 และU_2 เป็นตัวแปรสุ่มอิสระของแต่ละการแจกแจงแบบยูนิฟอร์มบนช่วง $(0,1)$ เราจะทำการทรานฟอร์ม U_1, U_2 ไปเป็นตัวแปรสุ่มอิสระอีก2 ตัว คือ X_1 และ X_2 โดยจะแสดงในรูปของพิกัดเชิงขั้ว (R,Θ) ของ เวกเตอร์สุ่ม (X_1, X_2) วิธีการนี้บางทีเรียกว่า วิธี Box-Muller

2.2.7.1คุณสมบัติของการทำอินเวอร์สทราสฟอร์ม

ให้ U เป็นตัวแปรสุ่มแบบยูนิฟอร์ม ที่มีค่าอยู่ในช่วง $(0,1)$ สำหรับทุกๆฟังก์ชั่นการแจกแจงชนิดต่อเนื่อง F ถ้ากำหนดให้ Y เป็นตัวแปรสุ่มโดย

$$Y = F^{-1}(U) \tag{2.59}$$

ดังนั้นตัวแปรสุ่ม Y มี F เป็นฟังก์ชั่นการแจกแจง $[F^{-1}(x)$ มีเท่ากับ y เพราะฉะนั้น $F(y) = x]$

พิสูจน์

$$F_y(a) = P\{Y \le a\}$$

$$= P\{F^{-1}(U) \le a\} \tag{2.60}$$

$F(x)$ เป็นโมโนโทนฟังก์ชัน(Monotone) จาก $F^{-1}(U) \le a$ เพราะฉะนั้นจะเป็นได้อย่างเดียว คือ $U \le F(a)$ ดังนั้นในสมการ (2.59) จะได้ว่า

$$F_y(a) = P\{U \le F(a)\}$$
$$= F(a) \tag{2.61}$$

ถ้าดูจากคุณสมบัติแล้วสามารถสร้างตัวแปรสุ่ม X ซึ่งมีฟังก์ชันการแจกแจงชนิดต่อเนื่องเป็น F โดยสร้างจากจำนวนสุ่ม U และกำหนดให้ $X = F^{-1}(U)$

เพราะฉะนั้นสมการ(2.61) เมื่อใช้วิธีการทำอินเวอร์สทราสฟอร์มจะได้ค่า $R^2 = -2\log_e U_1$ ส่วน $2\pi U_2$ นั้นเป็นตัวแปรสุ่มแบบยูนิฟอร์มในช่วง $(0,2\pi)$ เราจะใช้ในการสร้าง Θ ดังนั้นเราจะได้ว่า

$$R^2 = -2\log_e U_1 \tag{2.62}$$

$$\Theta = 2\pi U_2 \tag{2.63}$$

ดังนั้น R^2 คือการยกกำลังสองของระยะทางจากจุดศูนย์กลาง และ θ ก็เป็นมุมของของ (X_1, X_2) จาก $X_1 = R\cos\Theta$, $X_2 = R\sin\Theta$ เราจะได้ว่า

$$X_1 = \sqrt{-2\log_e U_1}\,\cos(2\pi U_2) \tag{2.64}$$

$$X_2 = \sqrt{-2\log_e U_1}\,\sin(2\pi U_2) \tag{2.65}$$

โดยตัวแปรสุ่ม X_1, X_2 เป็นอิสระต่อกัน

2.2.8 ทฤษฎีเซนทรัลลิมิต (The central limit theorem)

ทฤษฎีเซนทรัลลิมิตกล่าวถึง ผลของทฤษฎีความน่าจะเป็นว่า พีดีเอฟของผลลัพธ์ของตัวแปรที่เป็นอิสระหลาย ๆ ตัว ที่เกิดอย่างสุ่มจะมีแนวโน้มเข้าใกล้ พีดีเอฟของเกาส์เซียนมากขึ้นทุกที เมื่อจำนวนตัวแปรเพิ่มมากขึ้น

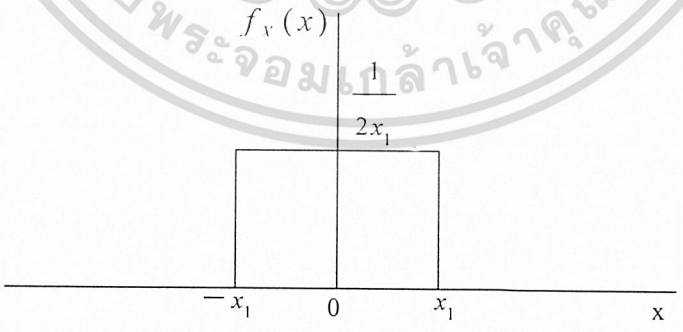
ทฤษฎีเซนทรัลลิมิต

ให้ X_1, X_2 .... เป็นลำดับของตัวแปรสุ่มที่มีการแจกแจงอย่างเดียวกัน และเป็นอิสระต่อกัน โดยมีค่าเฉลี่ยเป็น μ และค่าความแปรปรวนเป็น σ^2 แล้วการแจกแจงของ

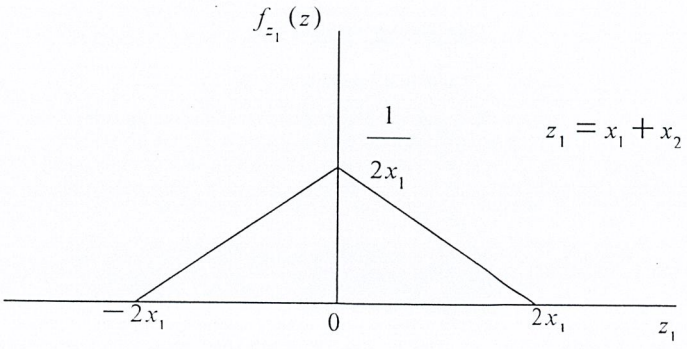
$$\frac{X_1 + \ldots + X_n - n\mu}{\sigma\sqrt{n}} \tag{2.66}$$

จะมีการแจกแจงปกติมาตรฐานที่มีค่าเฉลี่ย $\mu = 0$ ค่าความแปรปรวน $\sigma^2 = 1$ เมื่อ $n \to \infty$ และสำหรับค่า a อยู่ในช่วง $-\infty < a < \infty$ จะได้ว่า

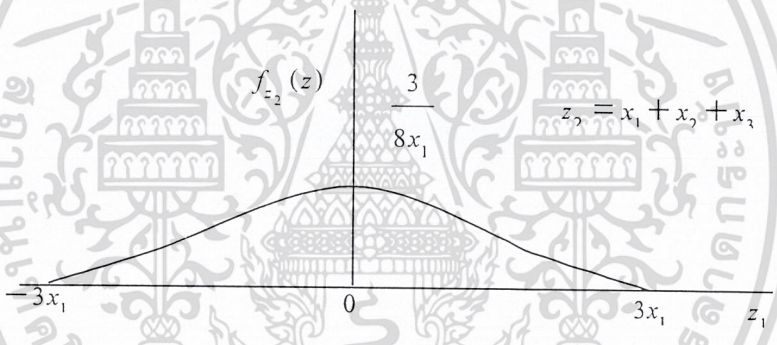
$$P\left\{\frac{X_1 + \ldots + X_n - n\mu}{\sigma\sqrt{n}} \le a\right\} \to \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{\frac{-X^2}{2}}\, dx \text{ เมื่อ } n \to \infty \tag{2.67}$$



(ก) ตัวแปรสุ่ม x ที่มีการกระจายแบบยูนิฟอร์ม



(ข) พีดีเอฟ ของตัวแปรสุ่ม $x_1 + x_2$



(ค) พีดีเอฟ ของตัวแปรสุ่ม $x_1 + x_2 + x_3$

รูปที่ 2.5 แสดงพีดีเอฟ ของผลลัพธ์ของหลายตัวแปรที่มีการกระจายแบบยูนิฟอร์มจะเข้าใกล้พีดีเอฟของเกาส์เซียนเมื่อจำนวนตัวแปรเพิ่มมากขึ้น

2.3 การสร้างสัญญาณไบนารี่แบบสุ่มโดยใช้วงจรนับ LFSR

Linear – feedback shift registers (LFSR) เป็นทางเลือกหนึ่งในการสร้างวงจรนับไบนารี่ซึ่งสามารถลดจำนวนวงจรลอจิกให้น้อยลงได้ดีกว่าวิธีอื่นๆ ผลของวงจรนับไบนารี่ของLFSRโดยปกติแล้วจะมีลำดับของการนับเป็น $2^m - 1$ สภาวะ และมีลักษณะเป็นไบนารี่แบบสุ่ม ทำให้ Register เหล่านี้มีชื่อเรียกอื่น ๆ เช่น pseudo – random sequence generator (PRSG)

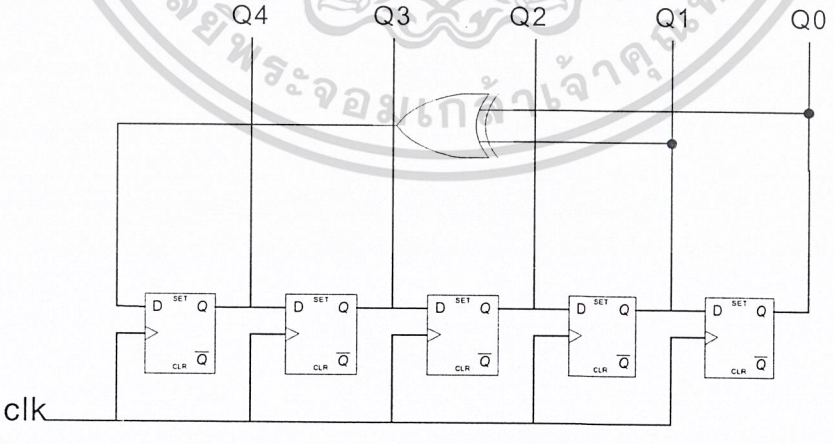
วัตถุประสงค์หลักๆ ในการใช้ LFSR ก็เพื่อให้การนับมีจำนวนใกล้เคียงกับสภาวะที่เป็นไปได้ $(2^m - 1)$ ให้มากที่สุด และมีการประยุกต์ใช้งานกันมากในเรื่องการเข้ารหัสแบบดิจิตอล (digital coding)

2.3.1 คุณสมบัติของ LFSR ในการออกแบบ

- LFSR ที่มีฟลิปฟลอบ (flip – flop) m ตัว สามารถที่จะสร้างจำนวนสภาวะ (state) เป็น $2^m - 1$ โดยที่สภาวะที่เป็นศูนย์จะไม่ยอมให้เกิดขึ้นเนื่องจาก LFSR จะไม่ทำงานได้(หรือ counter locks up)
- เลือกใช้โพลิโนเมียลที่มั่นใจแล้วว่า $2^m - 1$ สภาวะจะต้องไม่ซ้ำกัน โดยจำเป็นจะต้องรู้ต้นแบบโพลิโนเมียล
- ในการสร้างวงจรนับต้องคำนึงถึงโพลิโนเมียลที่เหมาะสม เช่น วงจรนับ 35 จะต้องใช้โพลิโนเมียลที่มี อันดับเท่ากับ 6 ผลลัพธ์ที่เป็นไปได้คือ $2^6 - 1 = 63$ ซึ่งสามารถหาได้จากตารางโพลิโนเมียล

2.3.2 วงจรนับของ LFSR

ตัวอย่างของวงจรนับ LFSR ที่ต้องการ m = 4จะใช้ค่าจากตารางที่ 2.1 โดยเลือกเอาโพลิโนเมียลอันดับ 5 คือ $x^5 + x + 1$ เพราะว่ายอมให้เกิดได้ 31 สภาวะ โดยที่สามารถสร้างวงจรโพลิโนเมียลลอจิกเป็น " many to one " หรือ " one to many " ในรูปที่ 2.8 จะแสดงโครงสร้าง LFSR ที่เป็นแบบ one to many



รูปที่ 2.6 แสดงโครงสร้าง LFSR แบบ one to many

m	โพลิโนเมียล	m	โพลิโนเมียล
1	$x+1$	21	$x^{21}+x^{2}+1$
2	$x^{2}+x+1$	22	$x^{22}+x+1$
3	$x^{3}+x+1$	23	$x^{23}+x^{5}+1$
4	$x^{4}+x+1$	24	$x^{24}+x^{4}+x^{3}+1$
5	$x^{5}+x+1$	25	$x^{25}+x^{3}+1$
6	$x^{6}+x+1$	26	$x^{26}+x^{8}+x^{7}+x+1$
7	$x^{7}+x+1$	27	$x^{27}+x^{8}+x^{7}+x+1$
8	$x^{8}+x^{6}+x^{5}+x+1$	28	$x^{28}+x^{3}+1$
9	$x^{9}+x^{4}+1$	29	$x^{29}+x^{2}+1$
10	$x^{10}+x^{3}+1$	30	$x^{30}+x^{16}+x^{15}+x+1$
11	$x^{11}+x^{2}+1$	31	$x^{31}+x^{3}+1$
12	$x^{12}+x^{7}+x^{4}+x^{3}+1$	32	$x^{32}+x^{28}+x^{27}+x+1$
13	$x^{13}+x^{4}+x^{3}+x+1$	33	$x^{33}+x^{13}+1$
14	$x^{14}+x^{12}+x^{11}+x+1$	34	$x^{34}+x^{15}+x^{14}+x+1$
15	$x^{15}+x+1$	35	$x^{35}+x^{2}+1$
16	$x^{16}+x^{5}+x^{3}+x^{2}+1$	36	$x^{36}+x^{11}+1$

ตารางที่ 2.1 แสดงลักษณะของโพลิโนเมียลที่ m ค่าต่างๆ

2.4 VHDL

2.4.1 ประวัติความเป็นมาของภาษา VHDL

วิวัฒนาการของภาษา VHDL นั้นเริ่มต้นประมาณปี ค.ศ. 1981 โดยที่กระทรวงกลาโหมสหรัฐอเมริกาหรือ Department of Defense (DOD) มองเห็นว่าอุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร เป็นอุปกรณ์ที่ได้รับการพัฒนามาเมื่อประมาณ 20 ปีก่อนเพราะเทคโนโลยีในขณะนั้นทำให้การพัฒนาอุปกรณ์อิเล็กทรอนิกส์เป็นไปอย่าล่าช้า ซึ่งเป็นสภาพที่ไม่อาจยอมรับได้ในปัจจุบัน เพราะเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์ ได้รับการพัฒนาไปอย่างรวดเร็ว ดังที่จะเห็นได้ว่ามีวงจรอิเล็กทรอนิกส์หลายวงจร ที่แต่เดิมถูกสร้างขึ้นมาจากชิ้สส่วนอุปกรณ์อิเล็กทรอนิกส์จำนวนหลายชิ้น ถูกนำประกอบกันอยู่บนแผงวงจรไฟฟ้า ที่มีขนาดใหญ่แต่ในปัจุบันสามรถที่จะใช้เทคโนโลยีการออกแบบและผลิตวงจรรวมขนาดใหญ่มาก (Very Large Scale Integration หรือ VLSI) รวมอุปกรณ์ต่างๆเหล่านั้นให้อยู่บนชิ้นอุปกรณ์สารกึ่งตัวนำ ที่มีขนาดประมาณ 1-2 ตร.ซม. ได้ซึ่งเป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้น (ความเร็วในการทำงานของวงจร) ตลอดจนความน่าเชื่อถือ และความคงทนต่อสภาพแวดล้อมสูง ทาง DOD จึงตั้งโครงการขึ้นมาเพื่อศึกษา วิธีการที่จะช่วยพัฒนาวงจรอิเล็กทรอนิกส์โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามารถนำผลิตได้เร็วขึ้น และโครงการดังกล่าวมีชื่อว่า Very High Speed Integrated Circuits หรือ VHSIC ในระยะแรกโครงการเป็นความลับด้านความมั่นคงของประเทศ ทาง DOD ได้ออกความต้องการมาตรฐานของภาษาที่ใช้สำหรับบรรยายพฤติกรรมของวงจรหรือ Hardware ของระบบสำหรับโครงการ VHSIC ซึ่งมีสาระสำคัญพอสรุปได้ดังนี้

- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถจะเข้าใจได้ทั้งคนและเครื่องโดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้ (Project Documentation)
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองกรทำงานของวงจร (Simulation Language)

Top-Down Design

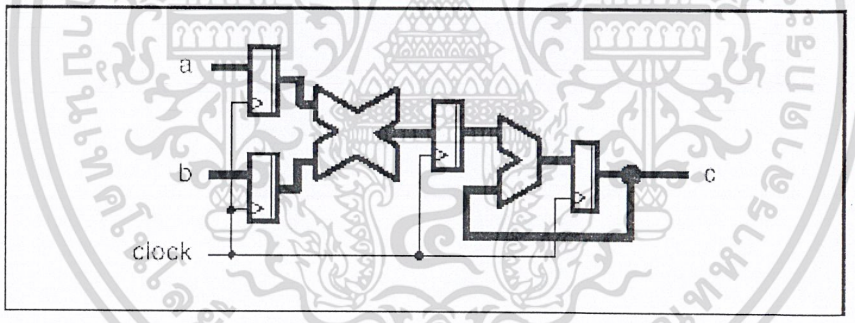
ภาษา VHDL ใช้กำหนดพฤติกรรมฟังก์ชันการทำงานของ Hardware ในระบบระบบดิจิตอลนั้นคือ ความอ่อนตัวของภาษาที่สามารถจำลองการทำงานจากหลักการของรูปแบบ (Simulate Conceptual designs) แต่ในขณะเดียวกันก็สามารถจำลองการทำงานของ Hardware ที่ให้รายละเอียดเกี่ยวกับเวลาอย่างถูกต้อง (timing based simulation) และจากโครงสร้างของภาษายังสามารถจำลองการทำงานในรูปของลำดับชั้น (hierarchy of simulation levels) ความสามารถดังกล่าวนี้จึงช่วยให้วิศวกรออกแบบ สามารถที่จะเขียนรูปแบบบรรยายจากระดับบนสุดของวงจรที่อยู่ในรูปสังเขป(high level of abstraction)
ลงสู่รายละเอียดในระดับล่างของวงจรได้เช่น Gate level เป็นต้น ในช่วงเวลานั้นเองวงการอุสาหกรรมไมโครอิเล็กทรอนิกส์ ตลอดจนสถาบันวิจัยและศึกษา กำลังพัฒนาภาษาที่จะใช้สำหรับการสังเคราะห์วงจรแบบอัตโนมัติ เพื่อลดเวลาในการพัฒนาวงจรลง ภาษา VHDL จึงถูกนำเข้าพิจารณาในโครงการนี้ด้วย โดยเพิ่มขีด เพื่อขีดความสารถของภาษาขึ้นอีกประการหนึ่ง นอกเหนือจากสิ่งที่ทาง DOD กำหนดในครั้งแรกคือเป็นภาษาที่ใช้สำหรับสังเคราะห์วงจร (Synthesis language)

ภาษา VHDL เป็นภาษาที่สนับสนุนการเขียนรูปแบบในทุกๆ ลักษณะและวิธีการ ดังเช่นตัวอย่างที่ แสดงในรูปที่ 2.7 คือรูปแบบ (model) ของลำดับขั้นตอนของการคูณและหาผลรวม (multiply accumulate algorithm) ของตัวแปรสองตัว a และ b ส่วนผลลัพธ์คือ c ในลักษณะของเลขฐานสอง (binary number) แบบนี้แสดงในระดับที่สังเขปที่สุดของแนวความคิดที่จะแก้ปัญหา เพื่อหาผลลัพธ์ โดยไม่ได้คำนึงถึงโครงสร้างของวงจรอย่างที่เคยชิน เช่นอุปกรณ์วงจรรวม Arithmetic and Logic Unit (ALU)

```
FOR i IN 1 TO 1024 LOOP
        result := result + a(i) * b(i):
END LOOP;
c <= result;
```

รูปที่ 2.7 แสดงVHDL Statement สำหรับ Multiply Accumulate Algorithm

ในขณะที่อีกด้านหนึ่งของมุมมองในปัญหาเดียวกันนี้ ภาษา VHDL ก็สามรถใช้บรรยายลำดับขั้นตอนของการคูณและหาผลรวม (multiply accumulate algorithm) โดยแสดงได้ในรายละเอียดของการสร้างวงจรดังกล่าวจริงๆ ตามที่เห็นได้จากรูปที่2.8

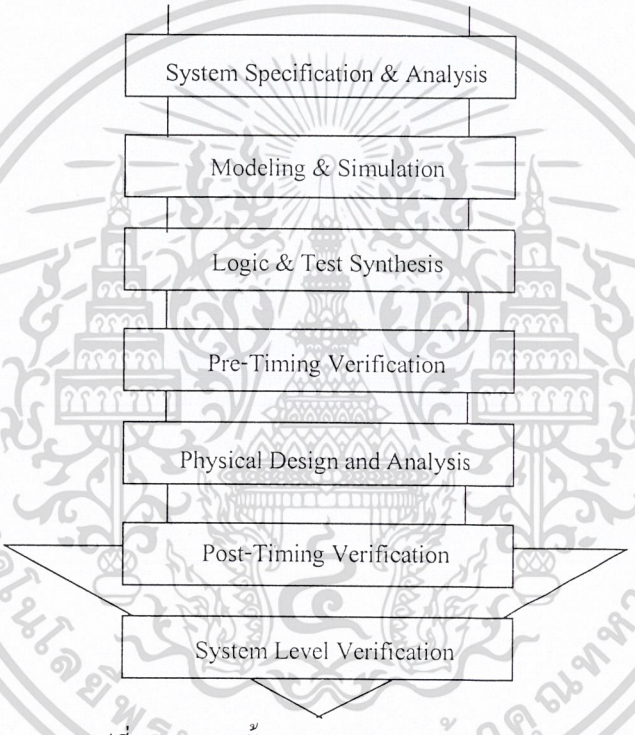


รูปที่ 2.8 แสดงโครงสร้างของ Multiply-Accumulate Unit

ฉนั้นจากความสามารถที่จะเขียนรูปแบบ (Modeling) ได้ในลักษณะต่างๆ นี้เองจึงเปิดโอกาสให้วิศวกรผู้ออกแบบได้พัฒนาและจำลองการทำงานของรูปแบบได้เร็ว ตั้งแต่ในระยะเริ่มต้นของแนวความคิดเกี่ยวกับฟังก์ชั่นการทำงานของวงจรอย่างสังเขป โดยที่ยังไม่ต้องไปคำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนั้น VHDL ยังเป็นภาษาที่สนับสนุนลักษณะต่างๆของระบบดิจิตอล (digital system) ที่มีความซับซ้อนได้ทั้งหมด

การเริ่มต้นด้วยวิธีการเขียนรูปแบบจากแนวความคิดอย่างสังเขป พร้อมทั้งการจำลองการทำงานรูปแบบที่เขียนขึ้น เพื่อตรวจสอบความถูกต้อง ประกอบกับการกลั่นกรองเพิ่มเติมรายละเอียดลงสู่ระบบดิจิตอลที่สมบูรณ์ในรูปของวงจรไฟฟ้าที่ละขั้น นั้นเป็นขบวนการของ Top-Down Design การที่เริ่มต้น

ด้วยการเขียนรูปแบบ (modeling) ในระดับบน (top-level) ของแนวความคิดอย่างสังเขปนั้น วิศวกรออกแบบสามารถที่จะพัฒนาสภาพแวดล้อมต่างๆ เพื่อการตรวจสอบการทำงานของวงจร (test environment) ได้ตั้งแต่ในระยะแรกๆ ของการออกแบบ เพื่อใช้สำหรับตรวจสอบความถูกต้องของแนวความคิดกับสิ่งที่ต้องการจริงหรือ specification ของงานดังนั้นจึงเป็นได้ยากที่ในระดับล่างลงมา โดยที่ได้เพิ่มรายละเอียดของรูปแบบให้มากขึ้นตามลำดับจะเกิดข้อผิดพลาด หรือเบี่ยงเบนไปจากจุดประสงค์เดิม เพราะในแต่ละขั้นตอนย่อยจะมีการสร้างสภาพแวดล้อมเพื่อการตรวจสอบขึ้นใหม่ โดยอ้างอิงจากระดับที่อยู่สูงกว่าขึ้นไปเสมอ จากรูปที่ 2.9 แสดงให้เห็นขั้นตอนของการออกแบบในลักษณะของ Top-Down Design ทั้งนี้ในทางปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย ก็เนื่องมาจากตอนของการผลิต (implantation) สามารถกระทำได้ในหลายๆ เทคโนโลยี เช่น Programmable Logic Devices อันได้แก่ PLA , FPGA หรือ CPLD เป็น นอกจากนั้นยังมี Semi-Custom IC (Gate Array , Standard Cell) และ Full Custom IC



รูปที่ 2.9 แสดงขั้นตอนของ Top-Down Design

- Physical Design and Analysis คือ ขั้นตอนของการผลิตเป็นวงจร (technology and device mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้า (Printed Circuit Board: PCB) ที่ประกอบด้วยอุปกรณ์หลายๆชิ้น หรืออยู่ในรูปของวงจรรวมเฉพาะงาน (ASIC)

- Post-Timing Verification หลังจากที่ได้วงจริงมาแล้ว ยังต้องมีความจำเป็นที่ต้องตรวจสอบการทำงานที่คำนึงเวลาด้วย เพื่อความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอลเพราะในขั้นตอนนี้วงจรที่ออกแบบจะประกอบด้วย Input และ output pad ซึ่งเป็นจุดต่อสำหรับรับและส่งสัญญาณกับภายนอก

- System Level Verification หลังจากที่นำวงจรที่ออกแบบรวมเข้าอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอลแล้วนั้น จะต้องทดสอบการทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่นๆ อีกครั้งเป็นการควบคุมคุณภาพของผลิตภัณฑ์

จากความอ่อนตัวของภาษา และความสามารถที่จะเขียนรูปแบบได้หลายลักษณะนี้เอง VHDL จึงเป็นเครื่องมือที่ใช้สำหรับออกแบบตั้งแต่ขั้นต้นบนสุด คือแนวความคิดที่จะแก้ปัญหาลงไปที่ละขั้นตอนของการผลิตวงจรจริง (form idea to implementation) ข้อดีที่เห็นได้ชัดของการนำ VHDL มาใช้ในการออกแบบลักษณะ top-down นี้คือ วิศวกรออกแบบสามารถที่จะสร้างรูปแบบ (model) และจำลองการทำงานเพื่อตรวจสอบความถูกต้องกับข้อกำหนด (specification) ตั้งแต่เริ่มแรกที่มีแนวความคิดอย่างสังเขป จากการจำลองการทำงานในระยะต้นๆของการออกแบบนั้น หลักการต่างๆ ที่ถูกกำหนดขึ้นใช้ในการแก้ปัญหา (สร้างวงจรให้เป็นตามความต้องการของข้อกำหนด) จะถูกตรวจสอบด้วยทุกครั้ง ก่อนที่จะมีการลงทุนในขั้นตอนสุดท้ายของการออกแบบ หรือการสร้างวงจรนั้นเอง นั้นหมายความว่าข้อผิดพลาดที่อาจจะเกิดขึ้นได้จากหลักการที่กำหนดขึ้น จะถูกตรวจพบและจัดการแก้ไขให้ถูกต้องได้ ก่อนที่จะทำงานในขั้นตอนต่อๆ ไปของขบวนการออกแบบ

ดังนั้นคุณสมบัติหลักของภาษา VHDL คือความสามรถที่จะใช้บรรยาย hardware ได้ในทุกๆ ระดับของภาพรวมทั้งระบบ ฉนั้นวิศวกรออกแบบจึงสามารถใช้เครื่องมือ (ภาษา) เพียงอันเดียวในการบรรยายทั้งระบบ ซึ่งก็เช่นเดียวกันกับเครื่องมือจำลองการทำงาน (simulator)

Terminology and Conventions

การเขียนรูปแบบของระบบดิจิตอลด้วยภาษา VHDL นั้น จะมีศัพท์เทคนิคเฉพาะ ฉนั้น ในบทนี้จะเป็นการบรรยาย และอธิบายศัพท์บางคำที่ต้องใช้งานบ่อยๆ

ลักษณะของรูปแบบ(Model Styles): ลักษณะของการเขียนรูปแบบ (model) ด้วยภาษา VHDL สามารถแบ่งออกได้เป็น

- **Behavioral Model:** หรือเรียกอีกอย่างหนึ่งว่า **Algorithmic description** เป็นรูปแบบที่บรรยายพฤติกรรมของระบบดิจิตอล ในส่วนที่บรรยายมีโครงสร้างคล้ายกับภาษาชั้นสูง (high level language) ทั่วๆไป เช่น Pascal หรือ C เป็นต้น ในการจำลองการทำงาน (simulation) คำสั่งแต่ละคำสั่ง (statement) จะถูกประเมินผลเป็นไปตามลำดับ (sequential) จากบนลงล่าง ยกเว้นในกรณีของคำสั่ง Loop หรือการเรียกใช้โปรแกรมย่อย รูปแบบลักษณะนี้จะไม่ให้รายละเอียดที่เกี่ยวกับการผลิต หรือโครงสร้างของ hardware แต่ในทางตรงข้ามที่จะให้รายละเอียดที่เกี่ยวกับความสัมพันธ์ระหว่าง input กับ output ที่ดี

- **Dataflow Model:** เรียกอีกอย่างหนึ่งว่า " Register Transfer Level "(RTL) เป็น รูปแบบที่ถูกเขียนขึ้น เพื่อจุดประสงค์ที่จะใช้เครื่องมือสำหรับสังเคราะห์วงจรอัตโนมัติ รูปแบบลักษณะนี้ส่วนใหญ่จะเป็น procedural constructs และ functional operators

- **Structural Model** เป็นรูปแบบที่แสดงการเชื่อมต่อกันระหว่างอุปกรณ์ต่างๆ ที่ประกอบกันขึ้นเป็นวงจรหรือระบบดิจิตอล และสามารถเรียกอีกอย่างหนึ่งว่า "nettles representation" เป็นการเขียนที่แสดงให้เห็นโครงสร้างของ Hardware

- **Mixed-Level Model** จากคุณสมบัติที่อ่อนตัวของภาษา VHDL จึงสามารถที่จะเขียนรูปแบบโดยใช้ลักษณะต่างๆ ที่กล่าวมาข้างต้น บรรยายบวงจรหรือระบบดิจิตอลเดียวกัน ฉนั้นรูปแบบเช่นนี้จึงมีการเขียนแบบผสม

Concurrency: ในภาษา VHDL นั้น ชุดคำสั่ง (statements) แต่ละชุดจะทำงานในเวลาเดียวกันและอิสระต่อกัน ลักษณะเช่นนี้เป็นคุณสมบัติที่เป็นความจริงทางฟิสิกค์ของวงจรอิเล็กทรอนิกส์ ชุดคำสั่งนี้เรียกว่า " **concurrent statement** " และจะทำงานก็ต่อเมื่อมีการเปลี่ยนแปลงค่าของสัญญาณ

Sequential: นอกจากความสารถที่ชุดคำสั่งจะทำงานแบบ concurrent แล้วบางครั้งการเขียนรูปแบบในลักษณะที่บรรยายพฤติกรรมของวงจร มีความจำเป็นที่จะต้องให้ชุดคำสั่งทำงานเป็นลำดับขั้นเรียงกันจากบน ลงสู่ล่าง อย่างเช่นการเขียนแบบ Behavioral Model เป็นต้น ชุดคำสั่งที่เป็น Sequential นี้จะใช้ในโปรแกรมย่อย (Subprogram) และ process statement ซึ่งจะกล่าวถึงต่อไปในภายหลัง

Driver: สัญญาณต่างๆ (Signal) ใน VHDL นั้นจะถูกควบคุมด้วยตัวขับหรือ "driver" สัญญาณเหล่านี้จะรับค่าใหม่ (ระดับของสัญญาณ) ได้ด้วยตัวขับนี้เอง

Transaction: การเกิด transaction กับ signal นั้นจะเกิดขึ้นเมื่อมีการกำหนดค่าๆ หนึ่งให้กับ signal นั้น ค่าใหม่ที่ signal ได้รับอาจจะมีผลหรือไม่มีผลทำให้เกิดการเปลี่ยนแปลงของระดับสัญญาณ (event) เช่นการเปลี่ยนจากค่า Logic '0' เป็นค่า Logic "1" เป็นต้น

Event: คือการเปลี่ยนระดับค่าของ Signal จากระดับหนึ่งไปสู่ระดับอื่น อย่างเช่นในระบบดิจิตอลการเปลี่ยนจาก Logic '0' เป็นค่า Logic "1" หรือในทางตรงกันข้ามถือว่า signal นั้นเกิด "event" ได้ นั้นจะต้องเกิด transaction ด้วย แต่ในทางตรงกันข้ามการเกิด transaction ไม่จำเป็นต้องเกิด event ทุกครั้ง

Sensitivity List: คือรายชื่อของ signal ต่างๆ ที่มีผลให้เกิดการทำงานของ concurrent statement เมื่อเกิด event ขึ้นกับ signal ตัวใดตัวหนึ่งหรือหลายตัวพร้อมกันในรายชื่อนั้น

Objects: ในภาษา VHDL นั้นคำว่า objects ใช้เขียนเพื่อบงบอกถึงองค์ประกอบส่วนหนึ่งของรูปแบบ ซึ่งเปรียบได้เหมือนกับภาชนะที่มีไว้บรรจุค่าต่างๆ สามารถแบ่งออกได้เป็นสามชั้น (class) ด้วยกัน คือ

- **Constant:** ได้ Object ประเภทหนึ่งที่เมื่อกำหนดค่าเริ่มต้นให้แล้ว จะคงค่านั้นไว้ตลอด ไม่สามารถดัดแปลง หรือแก้ไขได้ สามารถประกาศใช้ได้ในส่วนที่เป็นส่วนประกาศต่างๆ ของรูปแบบ (model)

- **Signal:** หมายถึง Object ประเภทหนึ่งที่สามารถกำหนดค่าที่สัมพันธ์กับเวลาให้ได้นั้น หมายความว่า signal สามารถรับค่าได้เพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง signal จะรับค่าๆ หนึ่งได้จากตัวขับสัญญาณหรือ driver ซึ่งตัวขับนี้อาจจะเก็บค่าในอนาคตสำหรับ signal ไว้ด้วย signal สามารถประกาศใช้ได้ในส่วนที่เป็นเนื้อที่ของ concurrent body เท่านั้น ดังนั้น signal จึงสามารถถูกนำไปใช้ได้ตลอดโครงสร้างของรูปแบบ(model) หรือที่เรียกว่า global object

- **Variable:** หรือตัวแปรได้แก่ object ที่สามารถกำหนดค่าใดๆ ให้ได้และสามารถที่จะเปลี่ยนแปลงค่าได้ตลอดการจำลองการทำงาน แต่จะเก็บค่าเพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง เนื่องจาก

Variable สามารถประกาศใช้ได้ในส่วนที่เป็น sequential body เท่านั้นอันได้แก่ส่วนประกาศของ Process, Function หรือ Procedure ดังนั้น Variable จึงสามารถนำไปใช้ได้เฉพาะในขอบเขตที่ถูกประกาศใช้เท่านั้น.

การประกาศใช้ Object (object declaration): การที่จะใช้ Object ชั้นต่างๆ ตามที่กล่าวมาแล้วในการเขียนรูปแบบด้วยภาษา VHDL นั้นจะต้องมีการประกาศใช้ก่อน การประกาศที่ Object สามารถใช้ชุดคำสั่งตามโครงสร้างดังนี้

Object_class identifier : **TYPE** [:= initial_value];

ซึ่ง Object_calss ได้แก่ Constant, signal หรือ Variable การตั้งชื่อ (identifier) เป็นไปตามกฎของภาษา VHDL ซึ่งจะกล่าวต่อไป Type คือการกำหนดประเภทของ object ที่ประกาศนั้นๆ นอกจากนั้นยังสามารถกำหนดค่าเริ่มต้นของ object ได้ (initial_value) ซึ่งส่วนนี้เป็นเพียง option และการประกาศจะต้องอยู่ในพื้นที่ที่กำหนดให้ของแต่ละส่วนของรูปแบบ (declarative area)

การตั้งชื่อ Object: การตั้งชื่อจะต้องไปตามกฎต่อไปนี้

1. ชื่อ (Identifier)ประกอบด้วยตัวหนังสือ (พยัญชนะและตัวเลข) ในภาษาอังกฤษได้แก่
   - พยัญชนะ A-Z, a-z
   - ตัวเลข 0,1,2,3,4,5,6,7,8,9
   - เครื่องหมายขีดเส้นใต้ (Underscore) “_”
2. ชื่อจะต้องขึ้นต้นด้วยพยัญชนะเสมอ
3. ชื่อสามารถประกอบด้วยพยัญชนะตัวเลขและ เครื่องหมายขีดเส้นใต้จำนวนไม่จำกัด
4. การใช้เครื่องหมายขีดเส้นใต้(_) ทุกครั้ง จะต้องนำหน้าด้วยพยัญชนะหรือตัวเลขและตามหลังด้วยพยัญชนะหรือตัวเลขเช่นกัน
5. พยัญชนะตัวใหญ่หรือตัวเล็กไม่มีความแตกต่างกัน (Case insensitive)
6. ห้ามใช้คำสงวน (Reserved word) ของภาษา VHDL

ค่าเริ่มต้น (Initial value): การประกาศใช้ object ทุกครั้งควรจะต้องกำหนดค่าเริ่มต้นให้ด้วย เพราะค่าที่กำหนดนี้จะถูกนำไปใช้ เมื่อเริ่มต้นจำลองการทำงาน (simulation) ของรูปแบบ

SIGNAL example_signal : BIT := ‘1’;

ในกรณีที่ไม่มีการกำหนดค่าเริ่มต้น ระบบจำลองการทำงานจะนำค่าที่น้อยที่สุด (อยู่ทางซ้ายมือสุดของกลุ่มค่า) ของแต่ละประเภท (Type) มาเป็นค่าเริ่มต้นแทน

ประเภทข้อมูล (Data type): ได้แก่ type ของ object ที่จะเป็นตัวกำหนดว่าค่าใดบ้างในกลุ่มของค่า (set of type) ของแต่ละ type สามารถที่จะกำหนดให้กับ object ได้นอกจากนั้น type ยังเป็นตัวกำหนดการทำงานในลักษณะต่างๆ (operation) ของ object นั้นๆ type แบ่งออกเป็น 4 ประเภทคือ

1. Scalar ได้แก่ตัวเลข (numeric) ซึ่งในภาษา VHDL มีตัวเลขจำนวนเต็มบวก (Integer) เช่น 1.0,30., 1E2 เป็นต้น

2. Enumeration กลุ่มของค่าประเภทนี้ได้แก่ ตัวหนังสือ หรือชื่อต่างๆ

3. Physical ได้แก่หน่วยทางฟิสิกส์ในระบบ SI

4. File เป็น type ภายนอกสามารถมีค่าได้หลายๆอย่าง

ประเภทย่อยของข้อมูล(Subtype): สำหรับ Type ที่กำหนดไว้แล้ว (Predefined type และ user-defined type) สามารถที่จะแบ่งออกเป็นกลุ่มย่อยลงไปได้อีก โดยที่องค์ประกอบของ Type ใหม่นี้จะเป็นส่วนหนึ่งของ type เดิม หรือที่เรียกว่า **Subtype**

```
TYPE qit IS ('0','1','Z','X');
SUBTYPE tit IS qit RANGE '0' TO 'Z';
```

ประเภทของ Object ที่กำหนดไว้แล้ว (Predefined type): ได้แก่ type ที่กำหนดไว้ใน package ชื่อ STANDARD และกำหนดโดย IEEE ว่าจะต้องมีในระบบที่ใช้พัฒนา VHDL ฉนั้นจึงไม่จำเป็นต้องประกาศใช้ในทุกๆรูปแบบที่เขียนขึ้น type ประเภทนี้ได้แก่

1. Boolean คือ กลุ่มของค่า False และTrue
2. Bit คือ กลุ่มของค่า "0" และ "1"
3. Integer คือ กลุ่มของค่า –214748347 ถึง 214748347
4. Real คือ กลุ่มของค่า –1.0E38 ถึง 1.05E38
5. Character คือกลุ่มของคำ พยัญชนะ 'A'-'Z','a'-'z' อักษรหรือเครื่องหมายพิเศษและตัวอักษรควบคุม
6. Time ได้แก่หน่วยเวลาที่มีพื้นฐานเวลาเป็นวินาที (second ย่อด้วย s หรือ S)
7. Severity Level คือกลุ่มของค่า Note, Warning, Error, Failure

Operator: เนื่องจาก Type ของ object จะเป็นตัวบ่งบอกถึง operator ที่ object นั้นๆสามารถกระทำได้ ตลอดจน Type ของผลลัพธ์ที่ได้จากการทำงาน ในภาษา VHDL แบ่ง operator ออกได้เป็นประเภทต่างๆตามที่แสดงในรูปที่ 2.12

Delay: หมายถึงช่วงระยะเวลาที่เริ่มต้นของสาเหตุ จนกระทั่งเป็นผลออกมาให้เห็นของปรากฏการณ์หนึ่งๆ ในความเป็นจริงทางธรรมชาติอุปกรณ์ hardware ทุกอย่าง อาทิ เช่น gate ต่างๆ ในระบบดิจิตอลจะมี delay แฝงอยู่เสมอ (ที่เรียกว่า propagation delay time) การที่ภาษา VHDL เป็นภาษาที่ใช้บรรยาย hardware จึงสามารถบรรยายพฤติกรรมของ delay ได้ ซึ่งจำแนก delay ออกเป็น

1. Delay Selection คือการกำหนด delay ที่มีผลต่อ Signal ในรูปแบบลักษณะใด **Inertial delay** ได้แก่ปฏิกิริยาต่อต้านการเปลี่ยนแปลง ระบบที่ประกอบด้วย inertial delay จะแสดงผลของการหน่วยเวลาต่อเมื่อ สัญญาณที่มาบังคับให้เกิดการเปลี่ยนแลปลง (จาก driver) มีช่วงระยะเวลานานกว่า inertial delay ของระบบนั้นๆ

2.Transport Delay จากความหมายของคำว่า transport คือการขนส่งจากจุดหนึ่งไปยังอีกจุดหนึ่ง ฉะนั้น transport Delay จึงแสดงผลของการหน่วงเวลาทุกครั้งเสมอไม่ว่าการเปลี่ยนแปลงของตัวขับ (driver) นั้นจะมีช่วงระยะเวลานานเท่าไร

3. Internal Delay ได้แก่ delay ภายใยตัวของระบบ VHDL ทั้งนี้เนื่องมาจากภาษา VHDL เป็นภาษาที่ชุดคำสั่งทั้งหลายทำงานแบบ concurrent ต่อกัน ซึ่งแตกต่างไปจากภาษาโปรแกรมชั้นสูงทั่วๆ ไป แต่การทำงานของระบบจำลองการทำงานด้วยเครื่องคอมพิวเตอร์ (simulator) เครื่องไม่สามารถที่จะทำงานสองคำสั่ง (หรือมากว่า) ได้กลไกลที่จะทำให้เป็นไปตามหลักการของ concurrent statement ได้นั้นเรียกว่า delta delay (δ) ที่หมายถึง internal delay ของระบบพัฒนา VHDL

LRM: คือคู่มืออ้างอิง (Language Reference Manual) ของมาตรฐาน IEEE 1076

Predefined Operators	
Logical Operators:	NOT, AND, OR, NAND, NOR, XOR
Operand TYPE:	BIT, BOOLEAN
Result TYPE:	BIT, BOOLEAN
Relational Operations:	=, /=, <, <=, >, >=
Operand TYPE:	TYPE ใดๆก็ได้
Result TYPE:	BOOLEAN
Arithmetic Operator:	+, -, *, /, MOD, REM, ABS
Operand TYPE:	INTEGER, REAL, Physical
Result TYPE:	INTEGER, REAL, Physical
Concatenation Operation:	&
Operand TYPE:	Array ของ TYPE ทุกประเภท
Result TYPE:	Array ของ TYPE ทุกประเภท

รูปที่ 2.10 แสดงOperator ที่กำหนดไว้ใน ภาษา VHDL

การแสดงกฎเกณฑ์การเขียนคำสั่งในภาษา VHDL จะอยู่ในรูปของ Batches Nauru Format (BNF) ซึ่งแน่นอนที่ว่าไม่สามารถค้นคว้าเพิ่มเติมได้จากหนังสือคู่มือ IEEE Standard 1076 Language Reference

Manual (LRM) การแสดงวิธีการเขียนแบบ BNF นั้นจะมีการดัดแปลงไปบ้างเล็กน้อยเพื่อความสะดวกในการศึกษาและมีสิ่งที่ต้องอธิบาย ณ ที่นี้ดังนี้

เครื่องหมายวงเล็บเหลี่ยม [Square brackets]

- สิ่งทีอยู่ในเครื่องหมายนี้เป็น Option สามารถที่จะเขียนหรือไม่เขียนได้

เครื่องหมายวงเล็บปีกกา

- สิ่งที่อยู่ในเครื่องหมายแสดงว่า ในภาษา VHDL อนุญาติให้มีได้ 0,1 หรือหลายๆครั้ง

ตัวพิมพ์ใหญ่ 'Capitalize'

- ในที่นี้ใช้ในความหมายที่แสดงว่า คำที่เขียนด้วยตัวพิมพ์ใหญ่ เป็นคำของ VHDL และคำที่อยู่ใน Standard Package ที่สามารถนำมาใช้ได้ทันทีโดยไม่ต้องมีการประกาศใหม่ ปกติภาษา VHDL เป็นภาษาที่มี case insensitive เช่นการเขียนในลักษณะนี้จะมีความหมายเดียวกันหมด

entity = Entity = EnTiTy = ENTITY

นอกจากนี้ยังมีจุดประสงค์ เพื่อที่จะแยกความหมายทั่วไปของภาษาประจำวัน กับภาษา VHDL เช่น ฟังก์ชัน (Function) ในภาษาทั่วไปหมายถึงหน้าที่ในการทำงาน เพื่อให้ได้ผลอย่างใดอย่างหนึ่งออกมา แต่ในขณะที่ Function เป็นโปรแกรมย่อยประเภทหนึ่งของภาษา VHDL

ตัวพิมพ์หนาทึบ **'BOLD'**

มีจุดประสงค์เพื่อต้องการเน้น ไม่ได้เป็นส่วนของกฎเกณฑ์ของภาษาแต่อย่างใดมาดูตัวอย่างของการเขียนในลักษณะ BNF จาก IF-THEN-ELSE statement

```
IF condition THEN
  {Sequential-statement(s)}
[{ELSEIF condition THEN}
   {Sequential-statement(s)}]
[ELSE
   {Sequential-statement(s)}]
   EBD IF;
```

ตามกฎเกณฑ์การเขียนที่สามารถที่จะเขียน ELSIF ได้หลายครั้ง หรือไม่มีก็ได้ คำว่า ELSIF เป็น option จะเขียนหรือไม่ก็ได้ แต่ถ้าเขียนจะมีเพียงครั้งเดียว ในขอบเขตของ sequential statement ต่างๆลงไปได้หลายๆคำสั่งหรือไม่ก็ได้

นอกจากนั้นยังมีคำศัพท์ที่มีความหมายเฉพาะสำหรับภาษา VHDL ซึ่งจะใช้ทับศัพท์โดยสามารถเข้าใจได้ทันทีโดยไม่ต้องแปลงกลับมาเป็น ภาษาอังกฤษอีก สุดท้ายจะขอยกตัวอย่างบางคำเช่น

Function หรือ ฟังก์ชัน หมายถึงหน้าที่ที่กระทำการใดการหนึ่งแต่ Function หมายถึงโปรแกรมย่อย (subprogram) ชนิดหนึ่งในภาษา VHDL

Process หมายถึงขบวนการใดๆ ในการทำงาน แต่ Process หมายถึง concurrent statement ในภาษา VHDL

Signal หมายถึงสัญญาณต่างๆ แต่ Signal หมายถึงการประกาศใช้ object ที่มีชั้นเป็นประเภท Signal

2.4.2 ส่วนต่างๆของแบบ (Design Unit)

ภาษา VHDL นั้นประกอบด้วยส่วนต่างๆ ที่สำคัญและเป็นพื้นฐานของการเขียนโปรแกรมระบบดิจิตอลที่สำคัญ 4 หน่วยคือ

1.Entity Design Unit

2. Architecture Design Unit

3. Package Design Unit

4. Configuration Design Unit

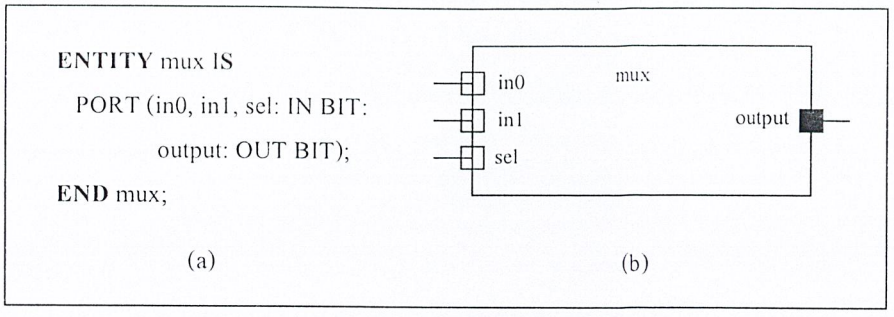
2.4.2.1 Entity Design Unit

หน่วยของแบบ (Design Unit) ส่วนที่ใช้สำหรับติดต่อระหว่างโลกภายนอกกับรูปแบบ(model) ที่จะเขียนขึ้น ส่วนนี้เรียกว่า " entity Design Unit" ในส่วนนี้ใช้กำหนดจุดต่อ(connection point) ของรูปแบบ กำหนดทิศทางการไหลของสัญญาณตามจุดต่างๆ (mode) และประเภทของค่า (type of value) ที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆ (PORT) ของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้น

```
ENTITY component name IS
    input and output ports
    physical and other parameters
END [component_name];
```

รูปที่ 2.11 แสดงโครงสร้างอย่างง่ายๆของ Entity Design Unit

ส่วนนี้จะขึ้นต้นด้วยคำว่า ENTITY และ IS ระหว่างคำทั้งสองเป็นส่วนสำหรับชื่อของรูปแบบที่ต้องการจะเขียน (component_name) สำหรับการตั้งชื่อนั้นจะต้องเป็นไปตามกฎเกณฑ์ของภาษาหลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทาง เข้า –ออก ของข้อมูล (input-output) รวมทั้งพารามิเตอร์อื่นๆ ส่วนนี้เรียกว่าส่วนหัว (entity header) และที่สำคัญคือ entity design unit จะต้องปิดท้ายด้วยคำว่า END และเครื่องหมายอัฒภาค หรือ semicolon (;)

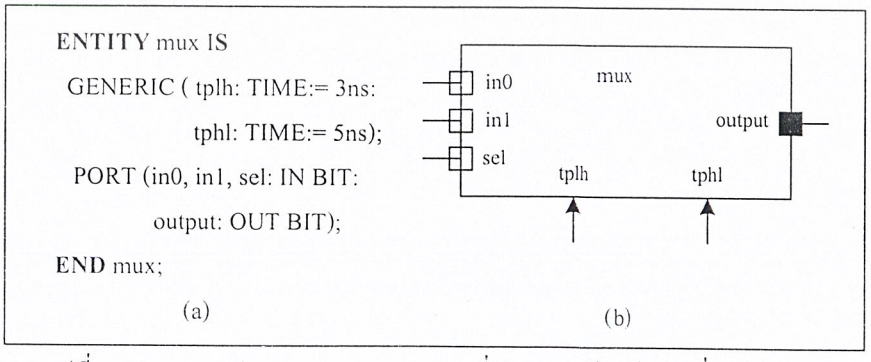


รูปที่ 2.12 แสดงรูปแบบของ 2:1 multiplexer, (a) VHDL entity design, (b) มุมมองของ Interface

ในรูปที่ 2.12 เป็น Entity design unit ที่บรรยายอุปกรณ์ที่มีชื่อว่า mux ในส่วนหัวของ entity (header) มีการกำหนดจุดต่อ 4 จุด ภายใต้ชุดคำสั่ง PORT โดยที่ 3 จุดแรกเป็นจุดให้ข้อมูลไหลผ่านเข้า (input) ได้แก่ in0, in1 และ sel ซึ่งกำหนดด้วยทิศทางการติดต่อกับภายนอก (mode) เป็นการไหลเข้า (IN) ที่แสดงด้วยรูปสี่เหลี่ยมโปร่งในรูปที่ 2.10 ส่วนจุด output เป็นจุดให้ข้อมูลไหลออก (output) ซึ่งกำหนดด้วยทิศทางการติดต่อกับภายนอกเป็นการไหลออก (out) ที่แสดงด้วยรูปสี่เหลี่ยมทึบรูปที่ 2.12 ส่วนประเภท (type) ของข้อมูลที่จะไหล เข้า-ออก นั้น เป็นประเภท bit ที่สามารถมีค่าได้เพียงค่าคือ '0' และ '1' เท่านั้น

นอกจากนั้นผู้ออกแบบยังสามารถกำหนดค่าพารามิเตอร์ทางฟิสิกส์ที่เป็นข้อมูลเพิ่มเติมอื่นๆลงในส่วนหัวของ Entity ได้อีก อาทิเช่น ข้อมูลเกี่ยวกับความเร็วในการทำงานของอุปกรณ์ อันได้แก่ propagation delay time พารามิเตอร์เหล่านี้ เรียกว่า generic ที่กำหนดด้วยคำสั่ง generic จากตัวอย่างในรูปที่ 2.11 สามารถที่สามารถที่จะเพิ่มข้อมูลเกี่ยวกับความเร็วในการทำงานได้ตามที่แสดงในรูปที่ 2.13

เช่นเดียวกับตัวแรกหน่วยของแบบนี้ช่องทางติดต่อกับภายนอก 4 จุด แต่ได้เพิ่มข้อมูลของ Propagation delay time สำหรับนำไปใช้ในการบรรยายพฤติกรรมของอุปกรณ์ ตัวพารามิเตอร์หรือที่เรียกว่า generic นี้มีชื่อว่า tplh และ tphl มีประเภทของข้อมูลเป็นเวลา (Time) และจะเป็นค่าตายตัว (Default value) สำหรับรูปแบบนี้เสมอ ซึ่งในที่นี้จะมีค่า 3 nanosecond และ 5 nanosecond ตามลำดับ ค่าตายตัวนี้สามารถที่จะเปลี่ยนแปลงให้มีค่าอื่นได้แล้วแต่กรณี ทั้งนี้ขึ้นอยู่กับอุปกรณ์ที่จะทำการเขียนรูปแบบ



รูปที่ 2.13 แสดงรูปแบบ 2:1 multiplexer ที่ประกอบด้วยข้อมูลเกี่ยวกับเวลา (a) VHDL entity design, (b) มุมมองของ interface

ในบางกรณีสามารถที่จะใช้ภาษา VHDL สร้างรูปแบบที่ปราศจากช่องทางไหล เข้า-ออก ของข้อมูล (input-output) ได้ ซึ่งส่วนใหญ่จะพบในการสร้างรูปแบบ สำหรับตรวจการทำงานของอีกรูปแบบหนึ่ง (VHDL test bench)

```
ENTITY test_bench IS
END test_bench
```

รูปที่ 2.14 แสดงEntity design unit ที่ไม่มีการกำหนดช่องติดต่อกับภายนอก

2.4.2.2 Architecture Design Unit

คือส่วนที่ใช้เขียนบรรยายกำหนดพฤติกรรมของรูปแบบ ในมุมมองของการจำลองการทำงาน (Simulation) พฤติกรรมต่างๆที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่าน เข้า-ออก ตรงช่องทางตลอดจนพารามิเตอร์ต่างๆ (port and generics) ที่กำหนดใน entity design unit

```
ARCHITECTURE idntifier OF component_name IS
    [declaration]
BEGIN
    specification of the functionality of the
    component in terms of its input lines and as
    influenced by physical and other parameters
END [identifier];
```

รูปที่ 2.15 แสดงให้เห็นโครงสร้างอย่างง่ายๆของ architecture design unit

ส่วนของ Architecture design unit นั้นเริ่มต้นด้วยคำว่า Architecture และตามด้วยชื่อ (identifier) สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่งที่แสดงให้เห็นว่า architecture นั้นใช้บรรยาย entity design unit ใด (OF *<entity design unit>* IS) ส่วนที่อยู่ระหว่าง Architecture และ Begin เป็นส่วนหนึ่งประกาศกำหนด (Architecture declarative area) ที่เป็นเพียงส่วนเผื่อก (option) ในบริเวณนี้สามารถใช้เขียนประกาศกำหนดค่าต่างๆที่จะนำไปใช้ภายใน Architecture นั้นได้ อาทิเช่นประเภท (type) ต่างๆ (ตัวอย่างเช่น Bit, Bit_vector), สัญญาณ (Signal) ล, ตัวคงที่ (Constant), โปรแกรมย่อย (ได้แก่ Function และ Procedure) และอุปกรณ์ (Component) ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า และไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง Port) นั้นจะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำว่า Begin กับ End ของ

Architecture Design Unit และนอกจากนั้นชุดคำสั่งทุกคำสั่งที่อยู่ภายใต้บริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (concurrent statement) เท่านั้น Architecture Design Unit
จะต้องปิดท้ายด้วยคำสั่ง End และชื่อของ architecture (identifier) นั้นๆที่เป็นส่วนหนึ่งเผื่อเลือก

โดยทั่วไปการเขียนรูปแบบระบบดิจิตอลด้วยภาษา VHDL สามารถเขียนได้ในลักษณะต่างๆตามที่กล่าวมาแล้วดังนี้

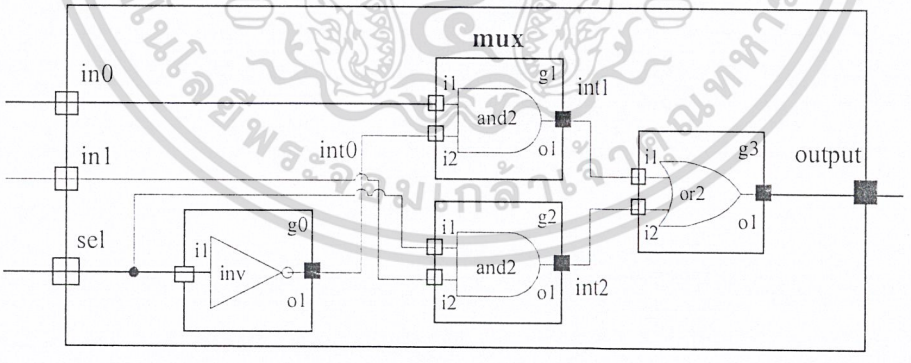
- Dataflow description
- Behavioral description
- Structure description
- Mixed model description

```
ARCHITECTURE data_flow OF mux IS
BEGIN
    output <= ((NOT sel) AND in0) OR (sel AND in1);
END data_flow;
```

รูปที่ 2.16 แสดงArchitecture design unit ของ 2:1 mux ตาม Boolean expression

$$output = (\overline{sel} \cdot in0) + (sel \cdot in1)$$

รูปที่ 2.17(ก) ส่วนที่บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า (in0,in1) กับข้อมูลที่ไหลออก (output) ประกอบด้วยชุดคำสั่งแบบแข่งขนานเพียงชุดเดียวซึ่งเป็นการบรรยายพฤติกรรมของ 2:1 mux การบรรยายลักษณะนี้เรียกว่า Dataflow description หรือ register transfer level (RTL)



รูปที่ 2.17 (ก) แสดงArchitecture description ของ 2:1 multiplexer (structural description)

รูปที่ 2.17 เป็น Architecture ของการบรรยาย 2:1 mux ในลักษณะของ structural description โดยใช้ inverter (inv ที่อุปกรณ์ g0) , And-gate 2 input จำนวน 2 gate (and2 ที่อุปกรณ์ g1 และ g2) ,OR-gate 2 input จำนวน 2 gate (or2 ที่อุปกรณ์ g3) มาสร้างตาม Boolean expression ของรูปที่ 2.17

```
ARCHITECTURE struct OF mux IS
   COMPONENT inv
   PORT( i1:IN BIT; o1: OUT BIT );
   END COMPONENT;
   COMPONENT and2
   PORT(i1, i2: IN BIT; o1: OUT BIT );
   END COMPONENT;
   COMPONENT or2
   PORT(i1, i2: IN BIT; o1: OUT BIT );
END COMPONENT;
   SIGNAL into, int1, int2: BIT;
BEGIN
   g0: inv
   PORT MAP( i1 => sel, o1 => int0);
   g1: and2
   PORT MAP( i1 => in0, i2 => int0, o1 => int1);
   g2: and2
   PORT MAP( i1 => sel, i2 => in1, o1 => int2);
   g3: or2
   PORT MAP( i1 => int1, i2 => int2, o1 => output);
END struct;
```

รูปที่ 2.17 (ข) : Architecture description ของ 2: 1 mux (structural description)

รูปที่ 2.17 (ก) เป็นโครงสร้าง (Structure) ของการต่อวงจรในโดยที่มีสัญญาณ in0,in1 และ in2 เป็นสัญญาณภายใน รูปที่ 2.17(ข) เป็น architecture ของ VHDL model ที่เขียนกำหนดการเชื่อมต่อภายในด้วย VHDL netlist

```
ARCHITECTURE behav OF mux IS
BEGIN
    PROCESS (in0, in1, sel)
    BEGIN
        IF (sel = '0') THEN
            output <= in0;
        ELSE
            output <= in1;
        END IF;
    END PROCESS;
END behav;
```

รูปที่ 2.18 แสดงArchitecture description ของ 2:1 mux (behavioral description)

การบรรยาย 2:1 mux ในลักษณะของ behavioral description ได้แสดงให้เห็นอีกครั้งในรูปที่ 2.18 ซึ่งจะเห็นได้ว่าส่วนที่เป็น Architecture design unit ทั้งหมด ต่างบรรยายพฤติกรรมเดียวกัน และจะให้ผลลัพธ์จากการจำลองการทำงานที่เหมือนกัน

2.4.2.3 Package Design Unit

ข้อมูลต่างๆตลอดจนโปรแกรมย่อย (Subprogram) ที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบดิจิตอล สามถรเก็บไว้ในส่วนที่เรียกว่า Package ได้ และข้อมุลเหล่านี้สามารถนำไปใช้ได้โดย entity design unit, architecture design unit หรือจาก package design unit อื่นๆ ด้วยชุดคำสั่ง USE statement นอกจากนั้นสิ่งที่นิยมทำกันมากคือรูปแบบ (model) มาตรฐานต่างๆ อาทิเช่น standard components (model ของ IC ตระกูล 74xx) จะถูกเก็บไว้ใน package ที่ทุกคนสามารถเข้าถึงและนำไปใช้ได้ สิ่งที่สามารถประกาศหรือบรรจุได้ใน package ได้แก่

- Subprogram
- Type
- Constants
- Signals
- Aliases
- Attributes
- Component
- Disconnection Specification

โดยปกติแล้ว Package จะแบ่งเป็นสองส่วนคือ

1. Package declaration
2. Package body

เนื่องจาก Package ถูกสร้างเป็นส่วนแยกต่างหากออกจากรูปแบบ (Model) ที่กำลังเขียนอยู่ ฉนั้นการที่นำ Package ไปใช้นั้นจะต้องมีเชื่อมโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษา VHDL สามารถกระทำได้ด้วยชุดคำสั่ง use statement

Package declaration

ส่วนที่มีความสำคัญที่สุด Package (มองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ Package declaration เพราะจะเป็นส่วนที่กำหนดชื่อ (identifier) ของสิ่งที่ประกาศอยู่ภายใน Package สำหรับนำไปใช้ภายนอกตัวของ Package เอง ถ้าสิ่งใดๆ ถูกประกาศในส่วนของ Package body แต่ไม่ถูกประกาศใน Package declaration จะไม่สามารถนำค่า และพฤติกรรมไปใช้จากส่วนนอกได้ ซึ่งสามารถเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของ entity declaration คือ interface ที่มีหน้าที่ติดต่อกับโลกนอก ฉนั้นโดยทั่วไปแล้ว Package สามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วน body และยังสามารถถูกนำไปใช้จากรูปแบบ (model) ภายนอกได้เช่นใช้สำหรับประกาศ type หรือ signal (global) เช่นเดียวกันกับ Package body ที่ไม่

จำเป็นต้องมี Package declaration แต่ Package นั้นจะไม่สามารถถูกนำไปใช้จากรูปแบบ (model) อื่นได้ การเขียน Package declaration มีกำเกณฑ์ตามที่แสดงในรูปที่ 2.19

```
PACKAGE package_name IS
    package_decleerative_part
END package_name;
```

รูปที่ 2.19 แสดงโครงสร้างของ Package declaration

คำว่า Package และ End package กำหนดขอบเขตของ Package declaration ระหว่างนั้นจะเป็นส่วนที่ใช้ประกาศต่างๆสิ่งที่สามารถประกาศในส่วนนี้ได้แก่

- ส่วนประกาศกำหนดโปรแกรมย่อย (Subprogram declarations)
- Type declaration
- Subtype declaration
- Object declaration (signal, constants)
- Alias declaration
- Attribute specification
- Component specifications
- Disconnection specifications

รูปที่ 2.20 เป็นตัวอย่างของการเขียน Package declaration ที่มีการประกาศ Type, Constant, Component และ Signal

```
PACKAGE example IS
    TYPE cd IS ('C', 'D');
    CONSTANT pi: REAL:= 3.14159;
    COMPONENT ttl_74163 IS
        PORT( a,b: IN BIT;
              c: OUT BIT );
    END COMPONENT;
    SIGNAL global_clock: BIT;
END example;
```

รูปที่ 2.20แสดงตัวอย่างของ Package declaration

Package body

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆ ในรูปของคำสั่งลำดับ (Sequential statement) ที่ใช้บรรยายฟังก์ชันการทำงานโปรแกรมย่อย (subprogram) ทั้งหลายที่ชื่อของโปรแกรมย่อยนั้นๆที่ถูกประกาศในส่วน

ของ Package declaration แล้ว จะถูกเก็บไว้ใน Package body ทั้งนี้รวมทั้ง deferred constant (อันได้แก่ตัวคงที่ถูกประกาศชื่อก่อนในส่วนของ declaration แต่ถูกกำหนดค่าในส่วนของ body ของ Package) ฉนั้นส่วน Package body จึงไม่จำเป็นต้องมี ถ้าในส่วน Package declaration ไม่มีการประกาศชื่อ (identifier) ที่มีโปรแกรมย่อย (subprogram) หรือ deferred constant การเขียน Package body นั้นเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 2.21

```
PACKAGE BODY package_name IS
        declarative part
END pack_name;
```

รูปที่ 2.21 แสดงโครงสร้างของ Package body

ชื่อของ Package_name ที่ใช้ใน Package body จะต้องเป็นชื่อเดียวกับชื่อที่กำหนดไว้ใน Package declaration ในรูปที่ 2.22 แสดงตัวอย่างของการเขียน Package (declaration และ body ที่สัมพันธ์กัน) โดยนำการกำหนดโปรแกรมย่อย (subprogram) ประเภท function

```
-- package declaration
PACKAGE pack_funct IS
    FUNCTION mean (a, b, c: REAL) RETURN REAL;
END pack_funct;
--package body
PACKAGE BODY pack_funct IS
    FUNCTION mean (a, b, c: REAL) RETURN REAL IS
    BEGIN
        RETURN (a+b+c)/3.0;
    END mean;
END pack_funct;
```

รูปที่ 2.22 แสดงตัวอย่างการเขียน Package

ในส่วน ของ Package declaration จะบรรจุส่วนที่เรียกว่า function declaration ในที่นี้เป็นการประกาศชื่อ ของโปรแกรมย่อย (Function) mean และส่วนที่บรรยายการทำงานของโปรแกรมย่อย mean (เรียกว่า function body) จะถูกเก็บไว้ในส่วน Package body แต่สิ่งที่สำคัญอย่างหนึ่งของการเขียน Package คือ ก่อนที่จะนำ Package ไปใช้ (โดยการอ้างอิงจากภายนอก) Package นั้นจะต้องถูกวิเคราะห์เสียก่อนว่าถูกต้อง หรือพูดง่ายๆได้ว่า จะต้องผ่านการ compile ก่อนนั้นเอง

Configuration Design Unit

ดังที่ทราบกันแล้วว่ารูปแบบ หนึ่งของระบบดิจิตอลไม่ว่าจะเป็นอะไร จะมี Entity design unit ได้เพียงหน่วยเดียวเท่านั้น แต่ในขณะที่ Entity design unit หนึ่งหน่วยนี้อาจจะมี architecture ที่เป็นหน่วยรองได้หลายหน่วย

ดังนั้นในการจำลองการทำงานของรูปแบบ (Model) นั้น simulator จะนำ architecture อันไหนนั้นจะต้องบอกให้ simulator ทราบและในรูปแบบ VHDL นั้นการบอกหรือการกำหนดคือการใช้ Configuration ประกอบ entity กับ architecture design unit ที่ต้องการเข้าด้วยกัน รูปที่2.23 แสดงกฎเกณฑ์การเขียน Configuration

```
CONFIGURATION identifier OF entity_name IS
        configuration_declarative_part
END;
```

รูปที่ 2.23 แสดงโครงสร้างของ Configuration

ในรูปที่ 2.24 เป็นรูปแบบ (model) ง่ายๆของ And-gate2 input ที่มีส่วนรองอันได้แก่ architecture สองแบบคือ dataflow description และ behavioral description

```
ENTITY and2 IS
    GENERIC (ttl_delay: TIME:= 3ns);
    PORT( in1, in2: IN BIT;
          output: OUT BIT );
END and2;

ARCHITECTURE dataflow OF and2 IS
BEGIN
    output <= in1 AND in2 AFTER ttl_delay;
END dataflow;

ARCHITECTURE behave OF and2 IS
BEGIN
    PROCESS(in1, in2)
    BEGIN
        IF( in1 = '1' AND in2 = '1') THEN
            output <= '1' AFTER ttl_delay;
        ELSE
        output <= '0';
    END PROCESS;
END behave;
```

รูปที่ 2.24 แสดง VHDL model ของ And-gate2 input

ก่อนที่จะนำรูปแบบ(Model) ไปจำลองการทำงานจะต้องมีการประกอบ Architecture ที่ต้องการเข้ากับ Entity design unit เสียก่อน (configuration) ซึ่งระบบ VHDL ส่วนใหญ่ถ้าไม่กำหนดการประกอบ Architecture เครื่อง simulation จะนำ Architecture หน่วยสุดท้ายที่ผ่านการวิเคราะห์ไปใช้งานจำลองการทำงาน ในรูปแบบที่ 2.25 แสดงการประกอบ Architecture ชื่อ dataflow เข้ากับ Entity design unit

```
CONFIGURATION dataflow_and OF and2 IS
    FOR dataflow
    END FOR;
END dataflow_and;
```

รูปที่ 2.25 แสดง Configuration ของรูปแบบ (Model) and2

ในตัวอย่างแสดงการประกอบโครงร่าง (Configuration) ชื่อ dataflow_and ให้เป็นตัวกำหนดการเชื่อม entity ชื่อ and2 เข้ากับ architecture ชื่อ dataflow ซึ่งจะเห็นได้ว่าเป็นการประกอบโครงร่างอย่างง่ายๆ ในรูปที่ 2.26 จะเป็นโครงสร้างที่ซับซ้อนมากขึ้น

ตัวอย่างในรูปที่ 2.26 (ก) configuration ชื่อ decode_llcon เชื่อมต่อ entity ชื่อ decode กับ architecture ชื่อ structure และภายใน architecture ชื่อ structure นี้ประกอบด้วยอุปกรณ์ย่อยๆซึ่งแต่ละตัวมีลักษณะการประกอบกับส่วนอื่นด้สยโครงสร้าง For...End และ For...End ที่กำหนด configuration จำเพาะที่อยู่ภายใต้อีกครั้ง (เรียกว่า configuration specification)และ configuration ภายในนี้จะต้องผ่านการวิเคราะห์สำหรับ configuration ปัจจุบัน(วงนอก) ผลลัพธ์ที่ได้จากการ วิเคราะห์จะต้องถูกต้องเก็บไว้ภายใต้ working directory (symbolic name Work) การกำหนดอุปกรณ์จำเพาะนั้น สามารถที่จะ กระทำได้โดยการบอกชื่อ label เช่นในกรณีของ i1 สำหรับ inv หรือใช้ all หรือ others ดังเช่นในกรณีของอุปกรณ์ and3 เป็นต้น ในที่นี้หมายความว่าอุปกรณ์ inv จะใช้ configuration ชื่อ invcon จาก library Work (หรือ working directory ที่มีชื่อตายตัวว่า Work) ที่กำลังทำงานอยู่ จะเห็นได้ว่าเป็นการสร้าง configuration ซ้อน configuration การที่จะใช้งาน configuration บนสุด (decode_llcon) ได้ configuration ล่างสุด (invcon และ and3con) จะต้องผ่านการวิเคราะห์ว่าถูกต้องมาแล้ว การสร้างโครงสร้างเช่นนี้เรียกว่า lower-level configuration

รูปที่ 2.26 (ข) ใช้ Entity และ architecture design unit เดียวกับรูป (ก) สำหรับ inv (กำหนดในส่วน architecture declaration) ถูกกำหนดจำเพราะด้วย label i1 กับอุปกรณ์ที่มี entity ชื่อ inv (สามารถมีชื่อเหมือนหรือต่างกันได้) และใช้ architecture ชื่อ behave ที่ยังคงเหลือในโครงสร้าง (ถ้ามีการใช้อุปกรณ์ inv มากว่าหนึ่งตัว) จะถูกประกอบเข้ากับอุปกรณ์ที่มี entity ชื่อ inv เช่นกันแต่จะใช้ architecture ชื่อ dataflow

แทน ทั้งนี้ entity design unit ชื่อ inv และ architecture design unit ชื่อ behave และ dataflow จะต้องถูกวิเคราะห์ก่อน ผลลัพธ์ที่ได้จากการวิเคราะห์จะต้องถูกเก็บไว้ภายใต้ working directory การประกอบลักษณะนี้เรียกว่า entity architecture pair configuration

```
ARCHITECTURE examp_config OF decode IS
    COMPONENT inv PORT( in1: IN BIT; o1: OUT BIT);
    END COMPONENT;
    COMPONENT and3 PORT( in1, in2, in3: BIT; o1: OUT BIT);
    END COMPONENT;
    ⋮
END examp_config;
```

```
CONFIGURATION decode_llcon OF decode IS
    FOR structural
        FOR i1: inv USE CONFIGURATION WORK.invcon;
        END FOR;
        FOR i2: inv USE CONFIGURATION WORK.invcon;
        END FOR;
        FOR ALL: and3 USE CONFIGURATION WORK.and3con;
        END FOR;
    END FOR;
END decode_llcon;
```

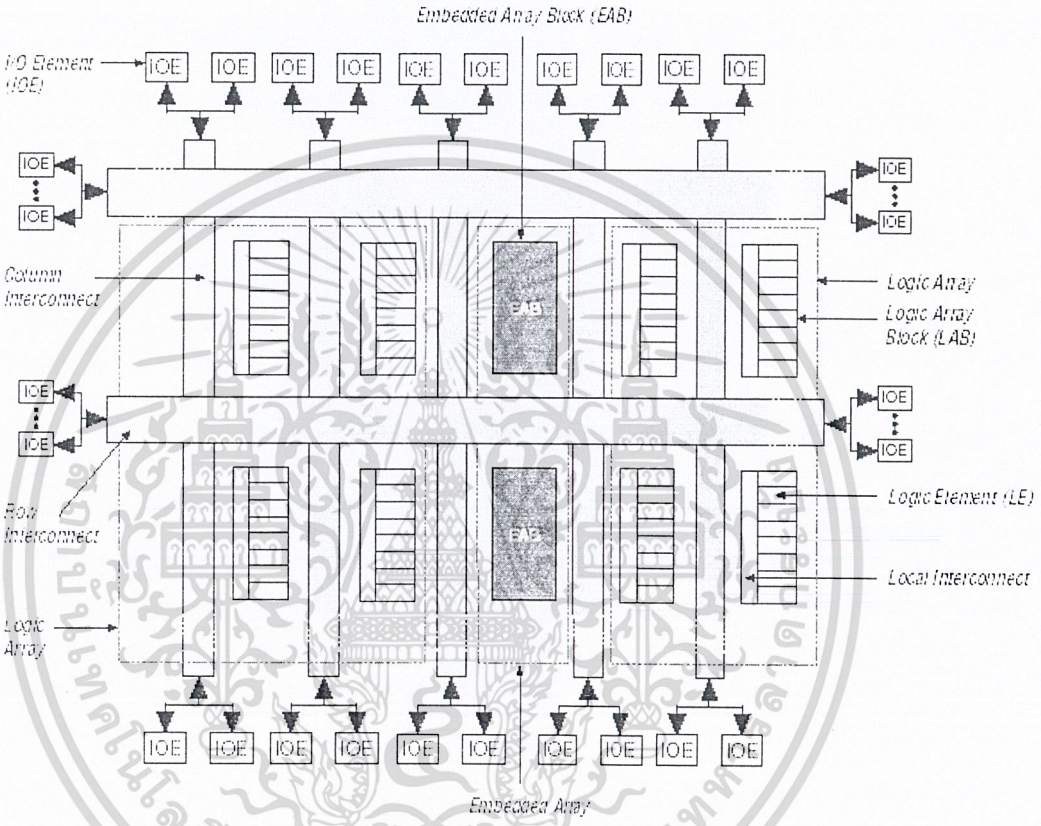
รูปที่ 2.26 (ก) แสดงตัวอย่างของ Configuration

```
CONFIGURATION decode_eacon OF decode IS
    FOR structural
        FOR i1: inv USE ENTITY WORK.inv (behave);
        END FOR;
        FOR OTHER: inv USE ENTITY WORK.inv (dataflow);
        END FOR;
        FOR a1: and3 USE ENTITY WORK.and3 (behave);
        END FOR;
        FOR OTHER: and3 USE ENTITY WORK.and3 (dataflow);
        END FOR;
    END FOR;
END decode_llcon;
```

รูปที่ 2.26 (ข) แสดงตัวอย่างของ Configuration

2.5 Field Programmable Gate Arrays (FPGAs)

FPGAs ของบริษัท Altera ตระกูล FLEX 10 K เป็นอุปกรณ์ที่มีความหนาแน่นเกทประมาณตั้งแต่ 10,000 – 250,000 เกท โดยการจัดโครงสร้าง (Configuration) จะใช้วิธีโหลดโครงสร้างเข้าไปในSRAM ภายใน ซึ่งหมายความว่าถ้าไม่ได้มีการจ่ายไฟเลี้ยงให้ โครงสร้างที่จัดเอาไว้ก็จะหายไป FPGAs ประเภทนี้จะสามารถโปรแกรมซ้ำได้ไม่จำกัดจำนวนครั้ง และการทำงานตามลอจิกฟังก์ชั่นจะใช้วิธีการเปิดตารางดู (Look Up table : LUT) โดยโครงสร้างของ FPGAs ตระกูล FLEX 10 K แสดงดังรูปที่ 2.27

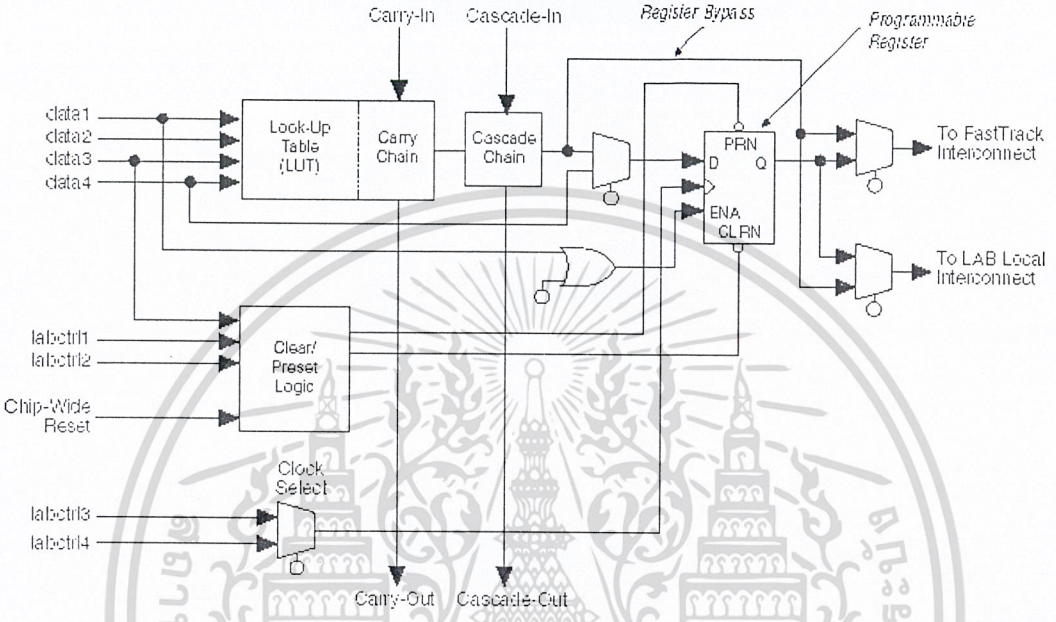


รูปที่ 2.27 แสดงโครงสร้างของ FPGAs ตระกูล FLEX 10K

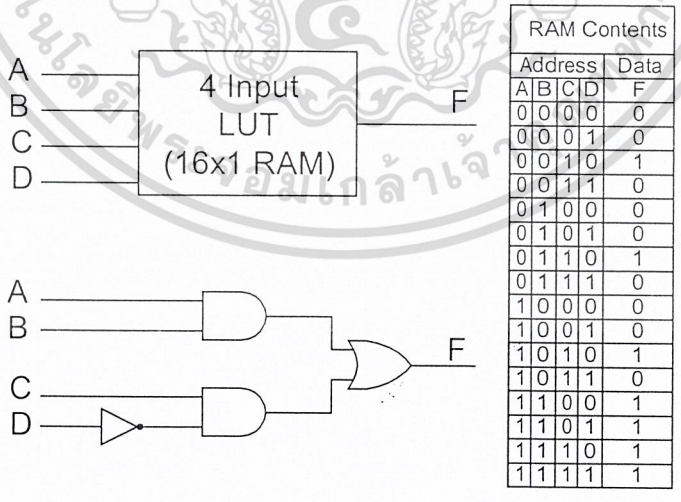
ในโครงสร้างของ FPGAs ตระกูล FLEX 10 K สามารถที่จะแบ่งเป็นส่วนต่างๆได้ดังนี้

2.5.1 Logic Element (LE)

ในรูปที่ 2.28 แสดงโครงสร้างภายในของ LE โดยการกระทำทางบูลีนของลอจิกเกทจะสร้างด้วยวิธีการ LUT โดย LUT คือ 1x16 SRAM ซึ่ง LUT เพียงตัวเดียวสามารถนำมาทำโครงข่ายของลอจิกเกทที่มี 4 อินพุท และ 1 เอาท์พุท โดยโครงข่ายของลอจิกเกทจะถูกแปลงไปเป็นตารางค่าความจริง (Truth Table) ดังแสดงในรูปที่ 2.28



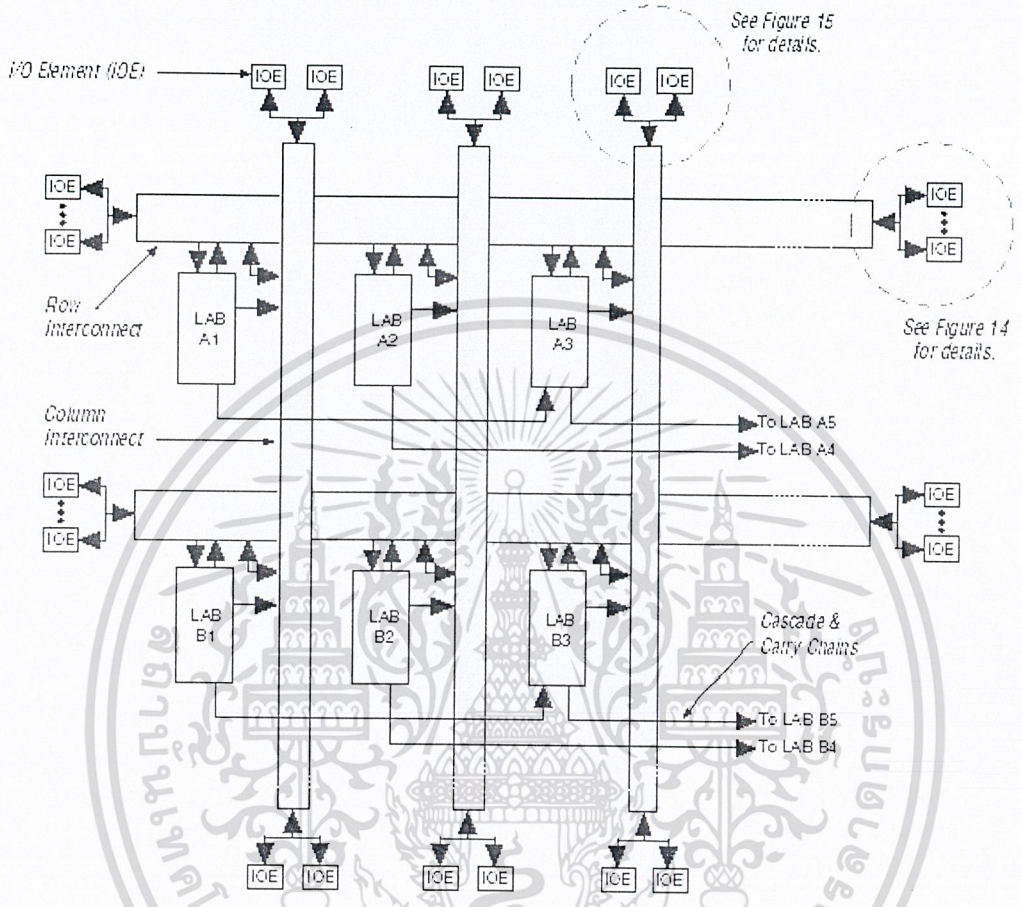
รูปที่ 2.28 แสดงโครงสร้างภายในของ LE



RAM Contents				
Address				Data
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

รูปที่ 2.29 การใช้งาน LUT เป็นโครงข่ายของลอจิก

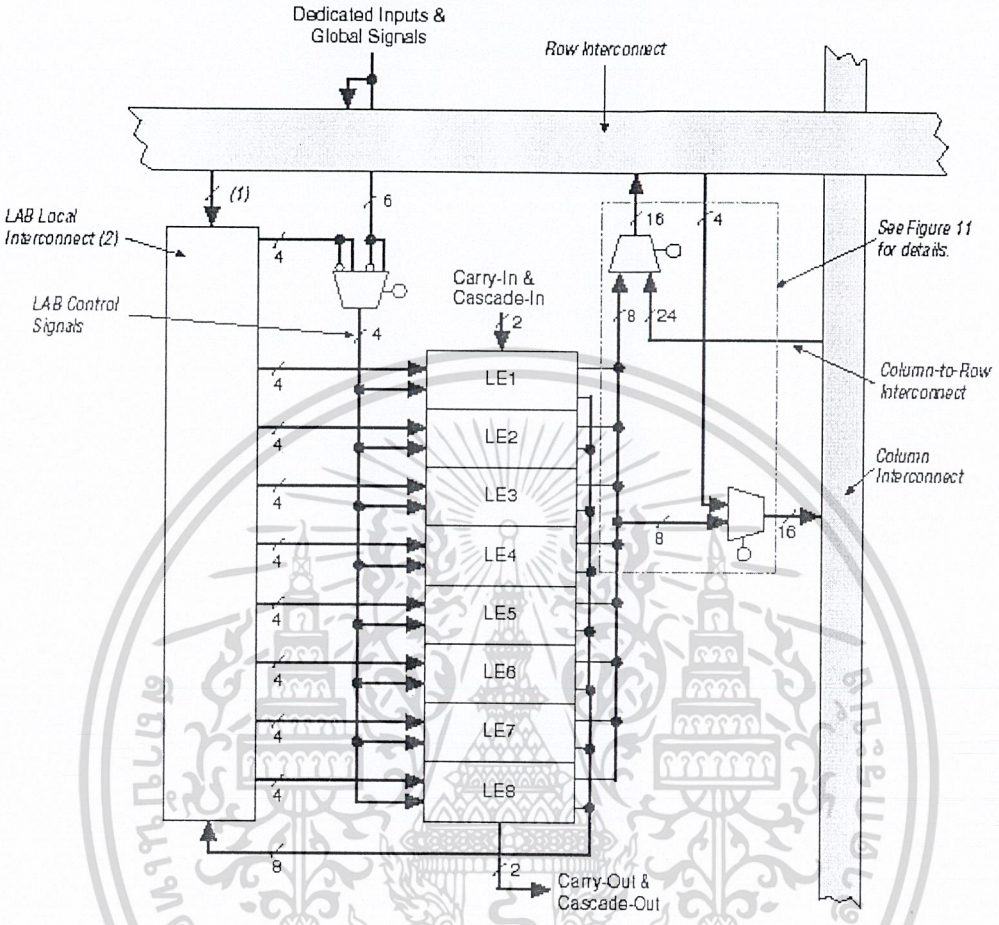
ถ้าโครงข่ายของลอจิกเกทมีความซับซ้อนขึ้นจะต้องใช้ LUT ของแต่ละ LE เป็นจำนวนหลายตัว โดยเอาท์พุทของ LUT จะส่งต่อไปยังฟลิปฟลอปและต่อไปยังโครงข่ายการเชื่อมต่อ (Interconnection Network) ดังแสดงในรูปที่ 2.30



รูปที่ 2.30 แสดงโครงข่ายของการเชื่อมต่อ

2.5.2 Logic Array Block (LAB)

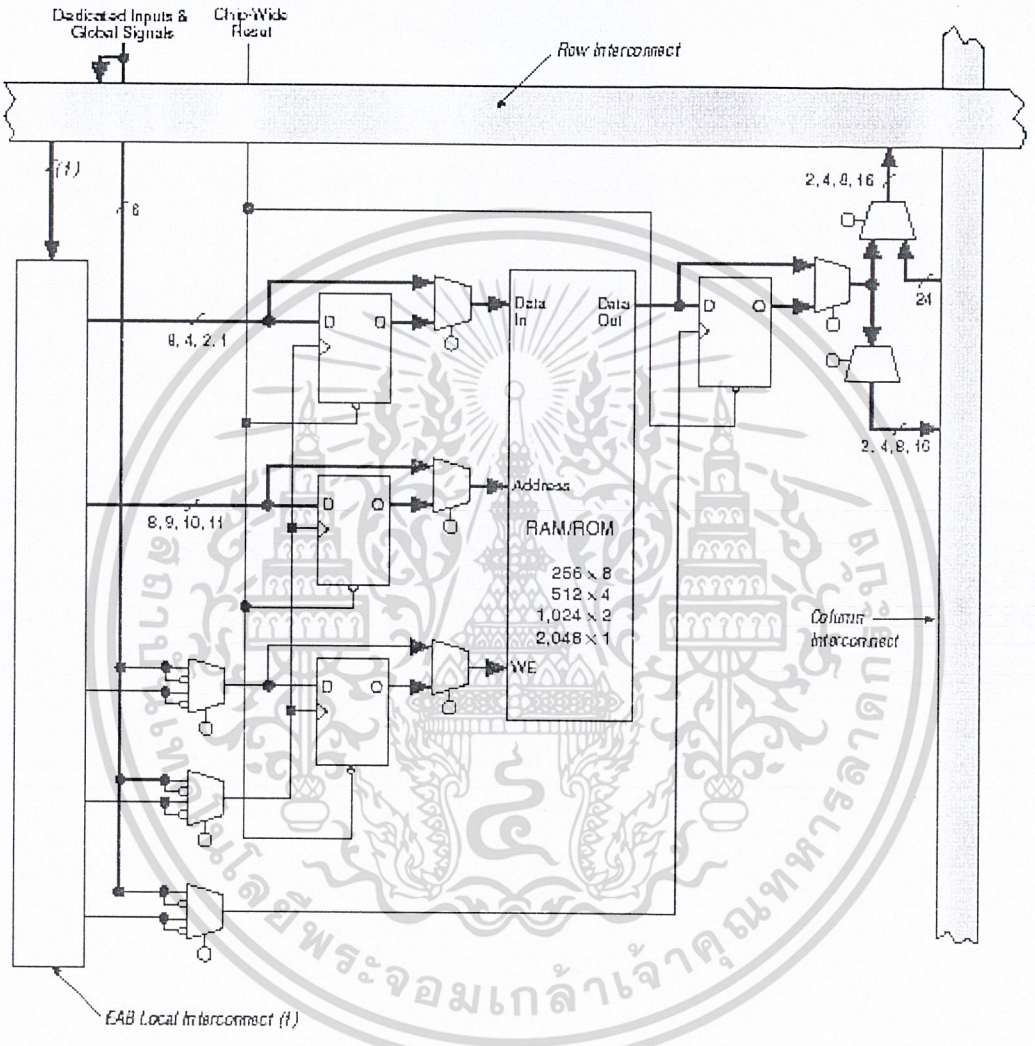
LAB 1 ตัว จะประกอบไปด้วย 8 LE ดังแสดงในรูปที่ 2.31



รูปที่ 2.31 แสดงโครงสร้างภายในของ LAB

2.5.3 Embedded Array Block (EAB)

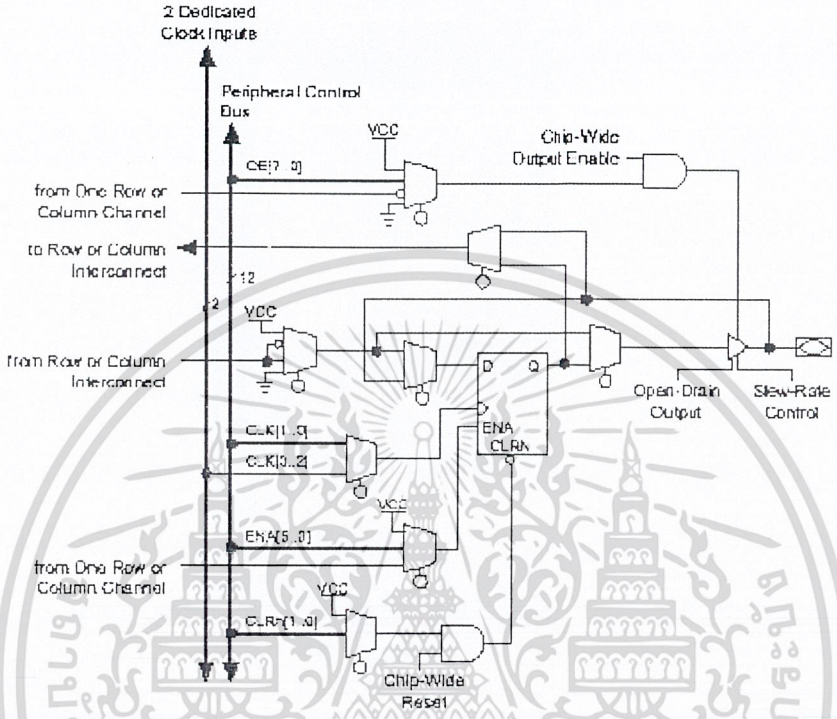
สถาปัตยกรรมโดยทั่วไปของ FLEX 10 K จะมีลักษณะของ LAB ที่มีการจัดเรียงแบบเมตริกซ์ และ EAB ซึ่งมีการเชื่อมต่อผ่านทางแถวและคอลัมน์ โดยในแต่ละแถวจะมี 1 EAB ซึ่ง 1 EAB จะมีขนาด 2048 บิท และสามารถกำหนดความกว้าง (Width) ความลึก (Depth) ของ EAB ได้โดยไม่ส่งผลต่อความเร็ว



รูปที่ 2.32 แสดงโครงสร้างภายใน EAB

2.5.4 Input Out Element (IOE)

IOE จะถูกต่ออยู่กับขา I/O โดยจะประกอบด้วยส่วนของวงจรที่เป็น Tri State และส่วนที่เป็นฟลิปฟลอป ซึ่งเป็น option ดังแสดงในรูปที่ 2.33



รูปที่ 2.33 แสดงโครงสร้างภายในของ IOE

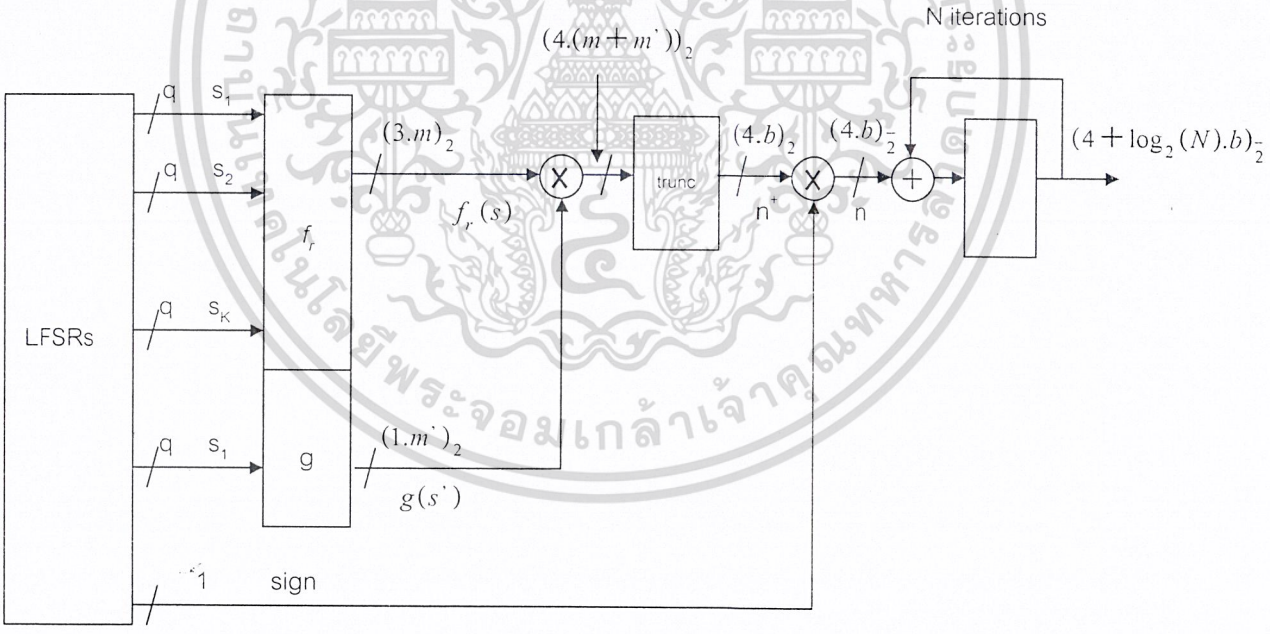
บทที่ 3

การคำนวณและการสร้าง

จากทฤษฎีสามารถออกแบบเครื่องจำลองช่องสัญญาณ AWGN โดยแบ่งออกเป็น 2 ส่วนใหญ่ๆ ได้ดังนี้

1. การออกแบบในส่วนของ Box-Muller ซึ่งเป็นการประมาณค่าการแจกแจงแบบเกาส์เซียนอย่างคร่าวๆ จะประกอบไปด้วย วงจร LFSR ที่มีโพลิโนเมียล m = 5 , m = 12 และ m = 14 หน่วยความจำของฟังก์ชั่นเอฟ(ROM_f(x)) หน่วยความจำของฟังก์ชั่นจี(ROM_g(x)) วงจรการคูณขนาด(10x7)บิต และวงจรทูคอมพลีเมนท์

2. การออกแบบในส่วนของเซนทรัลลิมิต จะประกอบไปด้วย วงจรแอคคิวมูเลเตอร์ ซึ่งส่วนต่างๆสามารถเขียนเป็น บล็อกไดอะแกรมได้ดังนี้



รูปที่ 3.1 แสดงบล็อกไดอะแกรมโดยรวมของ AWGN

3.1 การออกแบบในส่วนของ Box-Muller

3.1.1 การควอนไทซ์ค่าที่ได้จากวิธี Box-Muller

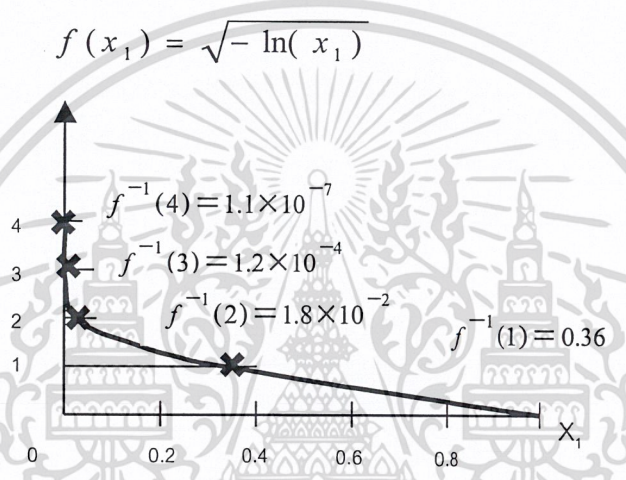
ค่าที่ได้จากวิธีการของ Box-Muller คือ

$$f(x_1)=\sqrt{-\ln(x_1)} \tag{3.1}$$

$$g(x_2)=\sqrt{2}\cos(2\pi x_2) \tag{3.2}$$

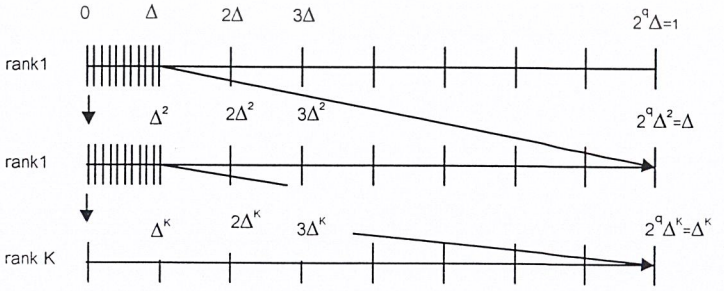
จะเริ่มจากการควอนไทซ์จากฟังก์ชัน $f(x_1)$ และตามด้วยการควอนไทซ์ฟังก์ชัน $g(x_2)$

1) การควอนไทซ์แบบไม่เชิงเส้นของฟังก์ชัน $f(x_1)$



รูปที่ 3.2 แสดงกราฟของฟังก์ชันของ $f(x_1)$

จากรูปที่ 3.2 ในช่วงที่ x_1 มีค่าเข้าใกล้ศูนย์ $f(x_1)$ จะมีความชันสูงและมีค่าที่โตประมาณ 4 เพราะฉะนั้นในช่วงนี้จำเป็นต้องมีช่วงของการควอนไทซ์(quantized step)ที่เล็กมาก เพื่อให้ได้ความถูกต้องสูง ในช่วง $[0,1]$ ของ x_1 จะทำการควอนไทซ์ไม่เป็นเชิงเส้นโดยพิจารณาในรูปที่ 3.3



รูปที่ 3.3 แสดงการควอนไทซ์ไม่เป็นเชิงเส้นในช่วง $[0,1]$

ค่า K คือจำนวนระดับของ rank และ q คือ จำนวนบิตที่เราต้องการ (q=4 บิต) ในแต่ rank จะแบ่งออกเป็น 2^q ส่วน ซึ่งมีช่วงของการควอนไทซ์ เป็น $\Delta^r = 2^{-rq}$ โดยค่า r เป็นลำดับของหน่วยความจำ (ROM_f(x)) เพราะฉะนั้นใน rank1จะแบ่งออกเป็น 16 ส่วน มีช่วงการควอนไทซ์เป็น $\frac{1}{16},\frac{2}{16},.....,1$ ในส่วนของ rank2 ก็แบ่งออกเป็น 16 ส่วน แต่ช่วงของการควอนไทซ์เป็น $\Delta^2 = 2^{-2(4)}$ มีค่าเป็น $\frac{1}{258},\frac{2}{256},.....,\frac{16}{256}$ ลักษณะการควอนไทซ์จะเป็นเช่นนี้จนกระทั่ง K = 5 การเก็บค่าจากการควอนไทซ์ฟังก์ชัน $f(x_1)$ของแต่ละ rank ค่าที่เก็บในแต่ละแอดเดรส s จะใช้สมการต่อไปนี้ในการพิจารณา

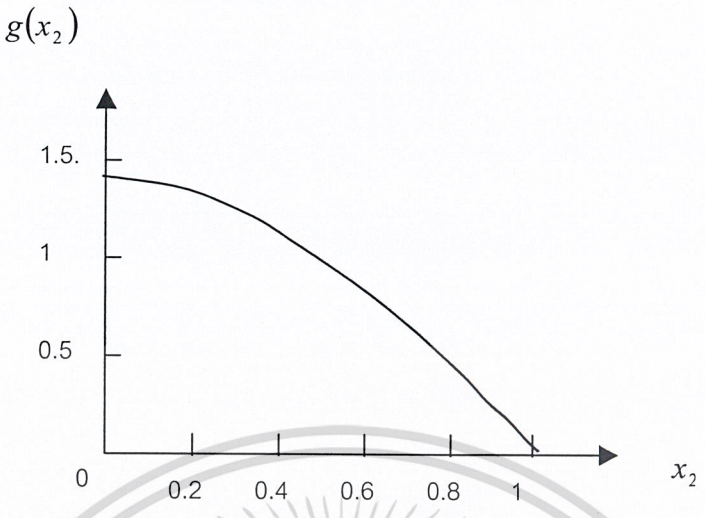
$$f_r(S) = \left\lfloor 2^m \sqrt{-\ln((S+\delta)\Delta^r)} \right\rfloor \left(X 2^{-m}\right) \qquad (3.3)$$

โดยที่ค่า r จะเปลี่ยนแปลงในช่วง 1 ถึง K ส่วนค่าที่ได้จากการควอนไทซ์ใน rank1,2,..,5 จะถูกเก็บไว้ใน ROM1,2,....,5 ตามลำดับ s จะเปลี่ยนแปลงในช่วง 1 ถึง 15 และ m เป็นจำนวนบิตที่อยู่หลังจุดทศนิยม ของ $f_r(s)$ { ตัวอย่างเช่น $f_r(s)$เป็นการเข้ารหัสของ $3+m$ บิต ก็หมายความว่า 3 บิตเป็นส่วนของจำนวนเต็ม และ m บิตเป็นส่วนของทศนิยม } $\lfloor x \rfloor$ เป็นเครื่องหมายที่ใช้บอกปัดเศษลงเป็นจำนวนเต็มเท่ากับ x ส่วน δ เป็นจำนวนจริงที่อยู่ในช่วง 0 ถึง 1 ซึ่งสัมพันธ์กับตำแหน่งของแซมเปิล ในแต่ละช่วงของการควอนไทซ์ ค่าต่างๆที่นำไปเก็บไว้ใน ROM_f(x) ของ สมการที่(3.3)แสดงในตารางที่3.1

ROM f Address	1		2		3		4		5	
	ฐาน10	ฐาน2	ฐาน10	ฐาน2	ฐาน10	ฐาน2	ฐาน10	ฐาน2	ฐาน10	ฐาน2
0	0	0000000000	0	0000000000	0	0000000000	0	0000000000	0	0000000000
1	1.5458	0011000101	2.2720	0100100010	2.8168	0101101000	3.2722	0110100010	3.6715	0111010101
2	1.3673	0010101111	2.1546	0100010011	2.7230	0101011100	3.1918	0110011000	3.6000	0111001100
3	1.2366	0010011110	2.0741	0100001001	2.6598	0101010100	3.1380	0110010001	3.5524	0111000110
4	1.1295	0010010000	2.0121	0100000001	2.6117	0101001110	3.0974	0110001100	3.5166	0111000010
5	1.0363	0010000100	1.9612	0011111011	2.5757	0101001001	3.0646	0110001000	3.4877	0110111110
6	0.9518	0001111001	1.9179	0011110101	2.5399	0101000101	3.0370	0110000100	3.4636	0110111011
7	0.8730	0001101111	1.8801	0011110000	2.5114	0101000001	3.0133	0110000001	3.4427	0110111000
8	0.7978	0001100110	1.8463	0011101100	2.4863	0100111110	2.9924	0101111111	3.4244	0110110110
9	0.7244	0001011100	1.8159	0011101000	2.4637	0100111011	2.9736	010111100	3.4081	0110110100
10	0.6514	0001010011	1.7880	0011100100	2.4433	0100111000	2.9567	0101111010	3.3933	0110110010
11	0.5772	0001001001	1.7623	0011100001	2.4245	0100110110	2.9412	0101111000	3.3799	0110110000
12	0.4995	0000111111	1.7384	0011011110	2.4072	0100110100	2.9270	0101110110	3.3675	0110101111
13	0.4151	0000110101	1.7161	0011011011	2.3911	0100110010	2.9138	0101110100	3.3560	0110101101
14	0.3174	0000101000	1.6951	0011011000	2.3761	0100110000	2.9015	0101110011	3.3453	0110101100
15	0.1841	0000010111	1.6753	0011010110	2.3620	0100101110	2.8899	0101110001	3.3353	0110101010

ตารางที่ 3.1 แสดงค่าที่ใช้เก็บในหน่วยความจำ(ROM_f(x)) ของฟังชัน $f(x_1)$

2) การควอนไทซ์แบบเชิงเส้นของฟังก์ชัน $g(x_2)$ โดย $g(x_2)=\sqrt{2}\cos\left(\frac{2\pi x_2}{4}\right)$



รูปที่ 3.4 แสดงกราฟของฟังก์ชัน $g(x_2)$

การควอนไทซ์ฟังก์ชัน $g(x_2)$ จะอาศัยความสมมาตรของฟังก์ชัน Cosine ซึ่งจะใช้แค่หนึ่งในสี่ส่วน โดยกำหนดให้ s', q' บิต เป็นตัวแปรสุ่ม และ $\Delta' = 2^{-q'}$ เป็นช่วงของการควอนไทซ์ในส่วนของฟังก์ชัน $g(x_2)$ จะใช้ $q'=8$ บิตทำให้ $g(x_2)$ ถูกแบ่งเป็น 256 ส่วน [เครื่องหมาย (sign) ของ cosine จะเก็บไปพิจารณาทีหลัง] ค่าของ $g(x_2)$ ที่ได้จากการควอนไทซ์จะได้จากสมการ

$$g(s')=\left\lfloor 2^{m'}\sqrt{2}\cos\left(\frac{\pi\Delta'(s'+\delta')}{2}\right)\right\rfloor\left(2^{-m'}\right) \tag{3.4}$$

หรือ

$$g(s')=\left\lfloor 2^{m'}\sqrt{2}\cos\left(\frac{\pi(s'+\delta')}{512}\right)\right\rfloor\left(2^{-m'}\right) \tag{3.5}$$

โดยที่ s' และ m' มีความหมายเหมือนกับ s และ m ของ $f_r(S)$ ซึ่งค่าที่จะเก็บไว้ใน ROM_g(x) ของสมการที่(3.5) แสดงไว้ในตารางที่3.2

S'	ฐาน10	ฐาน2	S'	ฐาน10	ฐาน2	S'	ฐาน10	ฐาน2	S'	ฐาน10	ฐาน2	S'	ฐาน10	ฐาน2
0	1.4142	1011010	52	1.3414	1010110	104	1.1332	1001000	156	0.8106	0110100	208	0.4061	0011010
1	1.4142	1011010	53	1.3387	1010101	105	1.1280	1001000	157	0.8035	0110011	209	0.3978	0011001
2	1.4140	1011010	54	1.3359	1010101	106	1.1228	1001000	158	0.7963	0110011	210	0.3895	0011001
3	1.4139	1011010	55	1.3330	1010101	107	1.1175	1000111	159	0.7891	0110010	211	0.3811	0011000
4	1.4139	1011010	56	1.3301	1010101	108	1.1121	1000111	160	0.7819	0110010	212	0.3728	0011000
5	1.4137	1011010	57	1.3271	1010101	109	1.1068	1000111	161	0.7747	0110001	213	0.3644	0010111
6	1.4131	1011010	58	1.3241	1010100	110	1.1013	1000110	162	0.7674	0110001	214	0.3560	0010111
7	1.4127	1011010	59	1.3210	1010100	111	1.0959	1000110	163	0.7601	0110000	215	0.3476	0010110
8	1.4123	1011010	60	1.3179	1010100	112	1.0904	1000110	164	0.7528	0110000	216	0.3392	0010101
9	1.4118	1011010	61	1.3147	1010100	113	1.0848	1000101	165	0.7454	0101111	217	0.3307	0010101
10	1.4113	1011010	62	1.3115	1010100	114	1.0792	1000101	166	0.7380	0101111	218	0.3223	0010100
11	1.4107	1011010	63	1.3082	1010011	115	1.0736	1000100	167	0.7306	0101111	219	0.3138	0010100
12	1.4101	1011010	64	1.3049	1010011	116	1.0679	1000100	168	0.7232	0101110	220	0.3054	0010011
13	1.4094	1011010	65	1.3015	1010011	117	1.0622	1000100	169	0.7157	0101110	221	0.2969	0010011
14	1.4086	1011010	66	1.2981	1010011	118	1.0565	1000011	170	0.7082	0101101	222	0.2884	0010010
15	1.4078	1011010	67	1.2946	1010011	119	1.0507	1000011	171	0.7007	0101101	223	0.2799	0010010
16	1.4070	1011010	68	1.2911	1010010	120	1.0449	1000011	172	0.6931	0101100	224	0.2714	0010001
17	1.4061	1011010	69	1.2875	1010010	121	1.0390	1000010	173	0.6855	0101100	225	0.2629	0010001
18	1.4051	1011010	70	1.2839	1010010	122	1.0331	1000010	174	0.6779	0101011	226	0.2543	0010000
19	1.4041	1011010	71	1.2802	1010010	123	1.0271	1000001	175	0.6703	0101011	227	0.2458	0001111
20	1.4030	1011010	72	1.2765	1010001	124	1.0211	1000001	176	0.6627	0101010	228	0.2372	0001111
21	1.4019	1011001	73	1.2728	1010001	125	1.0151	1000001	177	0.6550	0101010	229	0.2287	0001110
22	1.4008	1011001	74	1.2690	1010001	126	1.0091	1000000	178	0.6473	0101001	230	0.2201	0001110
23	1.3995	1011001	75	1.2651	1010001	127	1.0030	1000000	179	0.6395	0101001	231	0.2115	0001101
24	1.3983	1011001	76	1.2612	1010000	128	0.9968	1000000	180	0.6318	0101000	232	0.2030	0001101
25	1.3969	1011001	77	1.2573	1010000	129	0.9906	0111111	181	0.6240	0101000	233	0.1944	0001100
26	1.3956	1011001	78	1.2533	1010000	130	0.9844	0111111	182	0.6162	0100111	234	0.1858	0001100
27	1.3941	1011001	79	1.2492	1010000	131	0.9782	0111110	183	0.6084	0100111	235	0.1772	0001011
28	1.3926	1011001	80	1.2451	1001111	132	0.9719	0111110	184	0.6005	0100110	236	0.1685	0001011
29	1.3911	1011001	81	1.2410	1001111	133	0.9656	0111110	185	0.5927	0100110	237	0.1599	0001010
30	1.3895	1011001	82	1.2368	1001111	134	0.9592	0111101	186	0.5848	0100101	238	0.1513	0001001
31	1.3879	1011001	83	1.2326	1001111	135	0.9528	0111101	187	0.5769	0100101	239	0.1427	0001001
32	1.3862	1011000	84	1.2283	1001110	136	0.9464	0111100	188	0.5689	0100100	240	0.1340	0001000
33	1.3844	1011000	85	1.2240	1001110	137	0.9399	0111100	189	0.5610	0100100	241	0.1254	0001000
34	1.3826	1011000	86	1.2196	1001110	138	0.9334	0111011	190	0.5530	0100011	242	0.1167	0000111
35	1.3808	1011000	87	1.2152	1001110	139	0.9269	0111011	191	0.5450	0100011	243	0.1081	0000111
36	1.3789	1011000	88	1.2107	1001101	140	0.9203	0111011	192	0.5370	0100010	244	0.0994	0000110
37	1.3769	1011000	89	1.2062	1001101	141	0.9137	0111010	193	0.5289	0100010	245	0.0908	0000110
38	1.3749	1011000	90	1.2017	1001101	142	0.9071	0111010	194	0.5209	0100001	246	0.0821	0000101
39	1.3729	1011000	91	1.1971	1001100	143	0.9004	0111001	195	0.5128	0100001	247	0.0734	0000100
40	1.3708	1010111	92	1.1924	1001100	144	0.8937	0111001	196	0.5047	0100000	249	0.0648	0000100
41	1.3686	1010111	93	1.1877	1001100	145	0.8869	0111001	197	0.4966	0100000	250	0.0561	0000011
42	1.3664	1010111	94	1.1830	1001011	146	0.8802	0111000	198	0.4885	0011111	251	0.0474	0000011
43	1.3641	1010111	95	1.1782	1001011	147	0.8734	0111000	199	0.4803	0011110	252	0.0388	0000010
44	1.3618	1010111	96	1.1734	1001011	148	0.8665	0110111	200	0.4721	0011110	253	0.0214	0000001
45	1.3594	1010111	97	1.1685	1001011	149	0.8596	0110111	201	0.4639	0011101	254	0.0127	0000001
46	1.3570	1010111	98	1.1636	1001010	150	0.8527	0110110	202	0.4557	0011101	255	0.0041	0000000
47	1.3546	1010110	99	1.1587	1001010	151	0.8458	0110110	203	0.4475	0011100			
48	1.3520	1010110	100	1.1537	1001010	152	0.8388	0110101	204	0.4393	0011100			
49	1.3495	1010110	101	1.1486	1001001	153	0.8318	0110101	205	0.4310	0011011			
50	1.3468	1010110	102	1.1435	1001001	154	0.8248	0110101	206	0.4227	0011011			
51	1.3442	1010110	103	1.1384	1001001	155	0.8177	0110100	207	0.4145	0011010			

ตารางที่ 3.2 แสดงค่าที่ใช้เก็บในหน่วยความจำ(ROM_ g(x)) ของฟังชัน $g(x_2)$

3.1.2 การสร้าง HBM (Half Box-Muller)

การคูณ $f_r(s)$ และ $g(s')$ เรียกว่า การควอนไทซ์ตัวแปรสุ่ม HBM โดยจะใช้สมการ

$$n^+ = \left\lfloor \frac{f_r(s)xg(s')}{2^{m+m'-b}} \right\rfloor \left(x2^{-b}\right) \tag{3.6}$$

ซึ่งมีค่าความน่าจะเป็น $P\left(f_r(s), g(s')\right)$ เป็น

$$P\left(f_r(s), g(s')\right) = 2^{-(rq+q')} \tag{3.7}$$

กำหนดให้ s_n เป็นซับเซต (Subset) ของ $\{0,...,2^q-1\}\times\{1,..,K\}\times\{0,..,2^{-q'}-1\}$ ของทั้งสามตัวแปร (s,r,s') และที่ n^+ ในสมการที่ 3.7 จะมีความหนาแน่นเป็น $P(HBM=n^+)$ ที่ได้จากสมการ

$$P(HBM=n^+) = \sum_{(s,r,s')\in s_m} P\left(f_r(s), g(s')\right) \tag{3.8}$$

สมการที่ (3.6),(3.7) และ (3.8) ใช้ในการหาพีดีเอฟของ HBM แต่ส่วนที่ต้องการคือตัวแปรสุ่ม Box-Muller (BM)ซึ่งเกิดจากตัวแปรสุ่ม HBM ที่ใช้ตัวแปรสุ่มไบนารี่เป็นตัวแสดงเครื่องหมาย (sign)

$$n = (1-2\cdot sign)n^+ \tag{3.9}$$

ผลที่เกิดจากการคูณกันของ $f_r(s)$ และ $g(s')$ คือ n^+ ซึ่งจะมีจำนวนบิตมากขึ้น{ROM_f(x) มีขนาด 10 บิตในแต่ละแอดเดรส และ ROM_g(x) มีขนาด 7 บิตในแต่ละแอดเดรส ผลของการคูณจะทำให้ n^+ มีขนาด17บิต}ดังนั้นจำเป็นจะต้องมีกระบวนการในการตัดบิต(truncation)ออกเพื่อง่ายต่อการคำนวณในขั้นตอนต่อไป การตัดบิตนั้นจะตัดให้เหลือหลังจุดทศนิยมเพียง 6 บิต แล้วนำค่าที่ได้หลังจากการตัดบิตเข้าไปทำการกำหนดเครื่องหมายเพื่อให้เอาท์พุตนั้นเป็น BMที่สมบูรณ์ ซึ่งเครื่องหมายนั้นเราจะกำหนดโดยใช้ตัว LFSR เป็นตัวสร้างสัญญาณบิต 1 หรือ บิต 0 ขึ้นมาแบบสุ่มตามเงื่อนไขของ LFSR ถ้า LFSR สร้างบิต 1 ขึ้นมาผลลัพธ์ของ n จะมีค่าเท่ากับ n^+ ทำการ $2'$s complement แต่เมื่อ LFSR สร้างบิต 0 ผลลัพธ์ของ n จะมีค่าเท่ากับ n^+

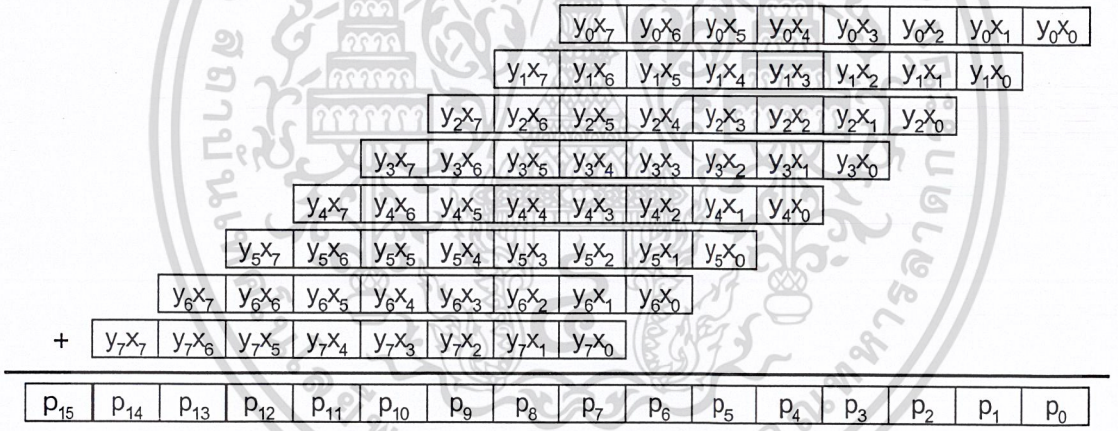
ในหัวข้อที่ 3.1.2 นี้จะประกอบไปด้วย 2 วงจร คือ การคูณเลขฐานสองและการทำทูคอมพลีเมนท์

3.1.2.1 การคูณเลขฐานสอง(Binary Multiplication)

ในการคูณเลขฐานสองจะใช้วิธี การคูณ การเลื่อน และการบวก โดยจะทำการคำนวณทีละบิต ดังนั้นผลที่เป็นไปได้ในการคูณแต่ละครั้งคือ 0 และ 1 ตัวอย่างเช่น

11	1011	multiplicand
× 13	× 1101	multiplier
33	1011	
11	0000	shifted multiplicand
143	1011	
	1011	
	10001111	product

จะสังเกตเห็นได้ว่า ถ้าอินพุตของ X และ Y มีขนาด n บิตจะทำให้ผลการคูณกันของ P = XY มีขนาด 2n บิต ในส่วนของโครงสร้างโดยทั่วไปของการคูณของ 8x8 บิต แสดงดังรูปที่ 3.5

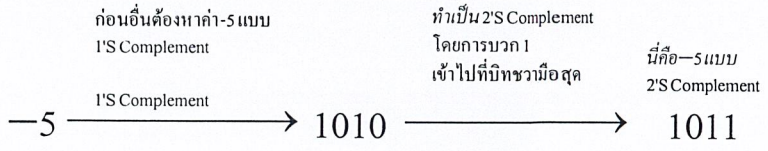


รูปที่ 3.5 แสดงโครงสร้างของการคูณขนาด 8x8 บิต

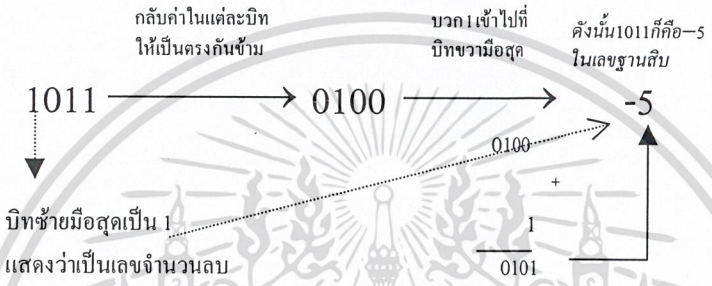
สำหรับการออกแบบวงจรการคูณ ไม่จำเป็นจะต้องเขียนโปรแกรมตามโครงสร้างที่ประกอบไปด้วยวงจรฟูลแอดเดอร์จำนวนมาก เพราะสามารถที่จะเรียกใช้ฟังก์ชั่นการคูณของโปรแกรม MAX PLUS II ได้ โดยจะอยู่ในส่วนของ LPM ฟังก์ชั่น ซึ่งจะมีส่วนของ LPM multiplier อยู่ซึ่งสามารถที่จะกำหนดการคูณขนาดต่างๆ ตามที่ต้องการได้ หรือสามารถที่จะเขียนโปรแกรมให้อยู่ของการคูณ P<= X*Y โดยการเรียกใช้ Package ของ IEEE. STD_LOGIC_ARITH และ IEEE .STD_LOGIC_UNSIGNED

3.1.2.2 การทำทูคอมพลีเมนท์ (2'S Complement)

เลขฐานสองแบบ 2'S Complement ก็คือเลขแบบ 1'S Complement ที่บวก 1 เข้าไปที่บิตขวามือสุด (LSB) โดยดูจากตัวอย่างการหาค่าเลขฐานสองแบบ 2'S Complement ของ -5



การทำเลขฐานสองแบบ 2'S Complement เพื่อหาค่าในฐานสิบสามารถทำได้โดย



เนื่องจากเลขฐานสองนั้นมีหลายรูปแบบเช่น

1. เลขฐานสองแบบไม่มีเครื่องหมาย (Unsigned Number)
2. เลขฐานสองแบบมีเครื่องหมายแบบ Signed Magnitude
3. เลขฐานสองแบบมีเครื่องหมายแบบ 1'S Complement
4. เลขฐานสองแบบมีเครื่องหมายแบบ 2'S Complement

และลองพิจารณาค่าของเลขฐานสองขนาด 3 หลักตามตารางนี้

	Unsigned Number	Signed Magnitude	1"S Complement	2"S Complement
011	3	3	3	3
010	2	2	2	2
001	1	1	1	1
000	0	0	0	0
111	7	-3	-0	-1
110	6	-2	-1	-2
101	5	-1	-2	-3
100	4	-0	-3	-4

ตารางที่ 3.3 แสดงเลขฐานสองแบบต่างๆ

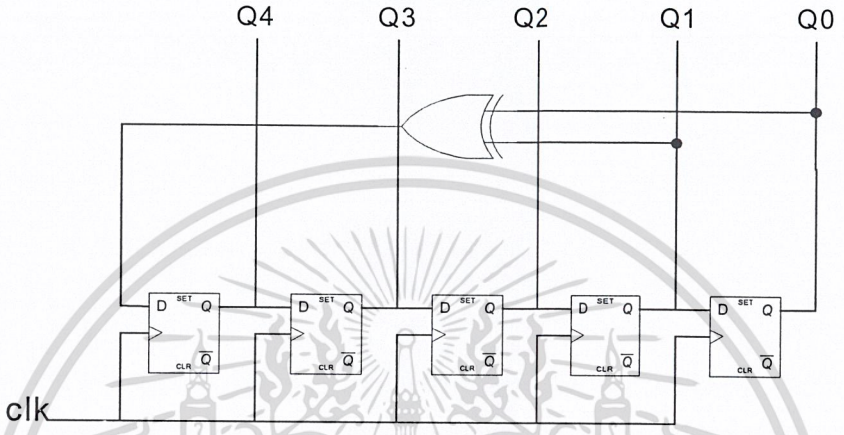
3.1.3 การสร้าง LFSR

LFSR จะเป็นตัวกำเนิดสัญญาณไบนารีที่มีการแจกแจงแบบยูนิฟอร์มซึ่งในโครงงานนี้จะใช้ตัว LFSR ทำหน้าที่ 3 อย่าง คือ

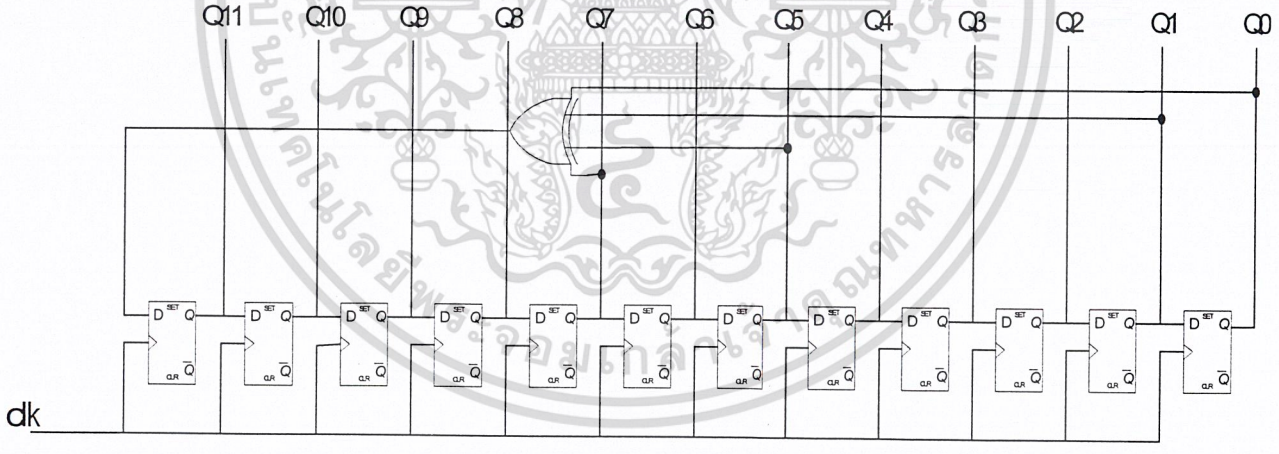
1)เป็นตัวชี้ตำแหน่งแอดเดรสของ ROM_f(x) โดยใช้ LFSR ที่มีโพลิโนเมียล m=5

2)เป็นตัวชี้ตำแหน่งแอดเดรสของ ROM_g(x) โดยใช้ LFSR ที่มีโพลิโนเมียล m=12

3)เป็นตัวสร้างบิตเครื่องหมาย(Sign) โดยใช้บิตใดบิตหนึ่งจาก LFSR ที่ออกแบบไว้



รูปที่ 3.6 แสดงLFSR ที่มีโพลิโนเมียล m=5 ($x^5 + x + 1$)



รูปที่ 3.7 แสดงLFSR ที่มีโพลิโนเมียล m=12($x^{12} + x^7 + x^5 + x + 1$)

ในส่วนของ LFSR ที่ออกแบบไว้สามารถที่จะเปลี่ยนแปลงให้ m มีขนาดมากขึ้นเพื่อเพิ่มจำนวนการสร้างตัวแปรสุ่ม

3.2 การออกแบบในส่วนของเซนทรัลลิมิต

การกระจายของ BM1 ของวิธี Box-Muller จะใช้พารามิเตอร์ในตารางที่ 3.3

Parameter	b	q	K	m	δ	q'	m'	δ'
BM_1	6	4	5	7	0.5	8	6	0.5

ตารางที่3.4 แสดงคุณลักษณะของ Box-Muller

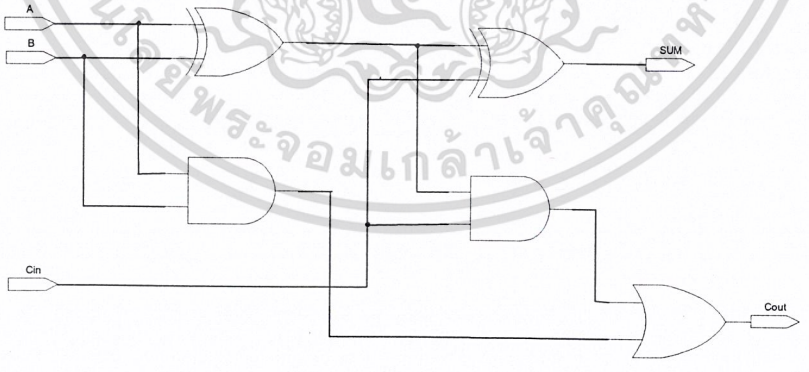
ค่าความละเอียดของการเปลี่ยนแปลงของการกระจาย BM1 ขึ้นอยู่กับจำนวน N ในการทำแอคคิวมูเลชั่น ซึ่งสามารถกระจายBM2ได้จากการทำออโต้คอนโวลูชั่น(auto-convolution) ของBM1 ($BM_2 = BM_1 \otimes BM_1$) เช่นเดียวกันการกระจายของ BM4 ได้จากการทำออโต้คอนโวลูชั่นของ $BM_2 (BM_4 = BM_2 \otimes BM_2)$

ผลที่ได้จาก Box-Muller เมื่อทำแอคคิวมูเลชั่น ค่าที่ได้หลังจากการทำแอคคิวมูเลชั่นจะมีค่าเฉลี่ยและค่าเบี่ยงเบนมาตรฐานเป็น

$$\mu\left(BM_{N,b}\right) = -N \cdot 2^{-b-1} \tag{3.10}$$

$$\sigma\left(BM_{N,b}\right) = \sqrt{N} \tag{3.11}$$

3.2.1 การสร้างวงจรฟูลแอดเดอร์(Full Adder)



รูปที่ 3.8 แสดงวงจรฟูลแอดเดอร์

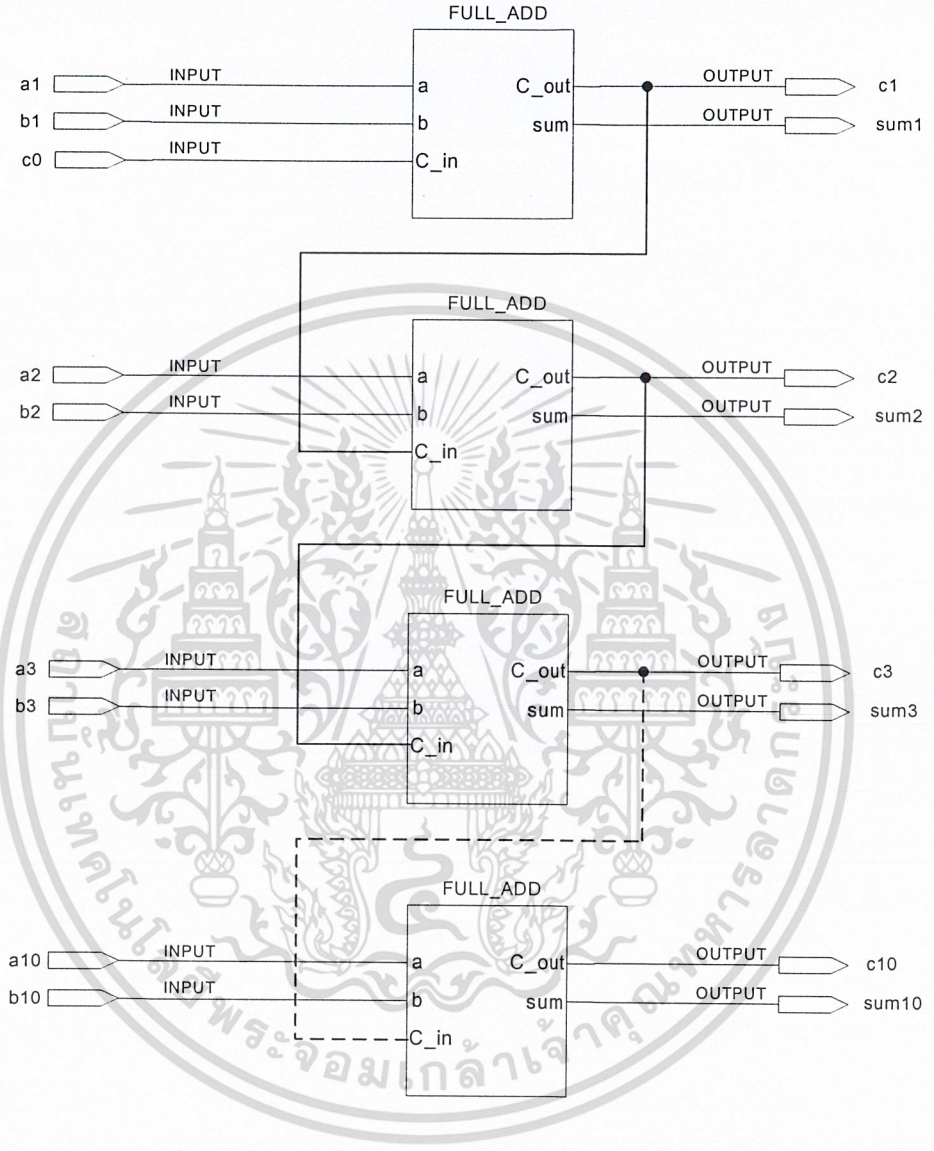
จากรูปที่ 3.7 สามารถเขียนสมการบูลีนได้ว่า

$sum = (A \oplus B) \oplus Cin$, $Cout = (A \oplus B)Cin + AB$

วงจรฟูลแอดเดอร์เป็นวงจรพื้นฐานที่ใช้ในการสร้างวงจรแอดเดอร์แบบขนานที่จะกล่าวต่อไป

3.2.2 การสร้างวงจรแอดเดอร์(Adder)

ในการสร้างวงจรแอดเดอร์จะนำเอาวงจรฟูลแอดเดอร์มาต่อกัน 10 ชุด เพื่อให้ได้การบวกแบบขนานทีละ10 บิต



รูปที่ 3.9 แสดงวงจรแอดเดอร์

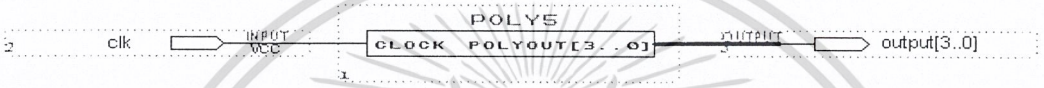
บทที่ 4

การทดลองและผลการทดลอง

การออกแบบส่วนต่างๆ ของเครื่องจำลองช่องสัญญาณแบบ AWGN (Additive White Gaussian noise) ทำการเขียนโปรแกรมโดยให้แต่ละส่วนทำงานตามที่ได้ออกแบบโดยใช้ภาษา VHDL ทำการ Compileแล้วทำการจำลองการทำงานของโปรแกรมแต่ละส่วนที่ได้เขียนขึ้นให้ได้ผลตามที่ออกแบบไว้

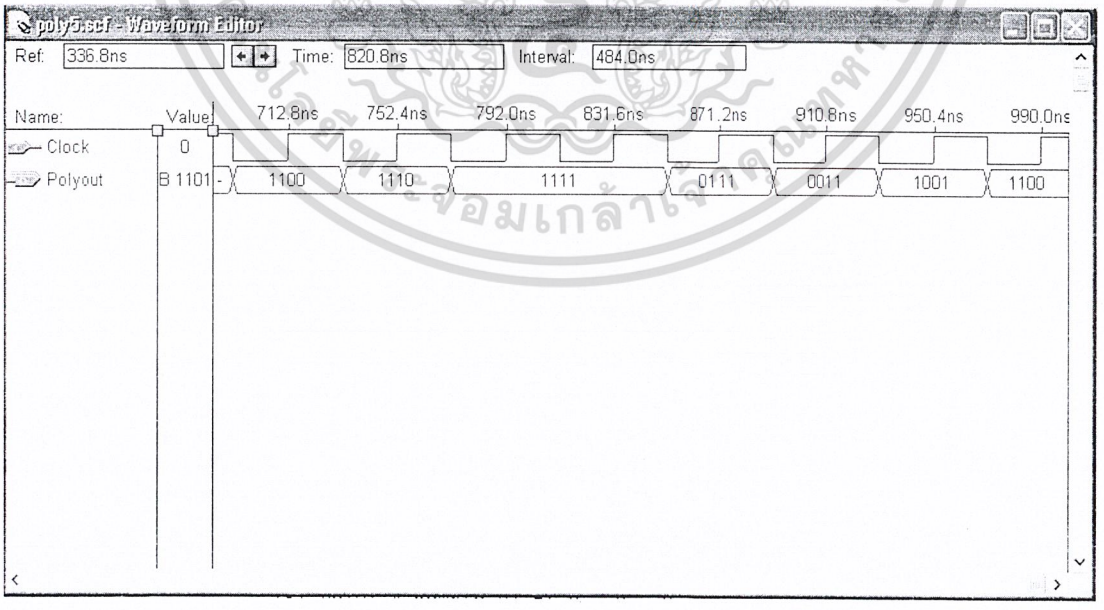
4.1 ส่วนการสร้างสัญญาณสุ่ม

4.1.1 วงจร LFSR ที่มีโพลิโนเมียล m = 5 สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์(Symbol) ดังนี้



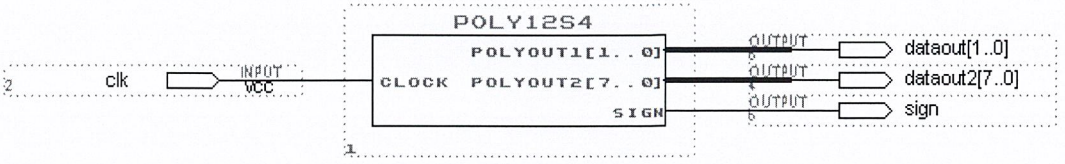
รูปที่ 4.1 แสดงสัญลักษณ์ของวงจร LFSR ที่มีโพลิโนเมียล m = 5
ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน(Simulation) ได้ดังนี้



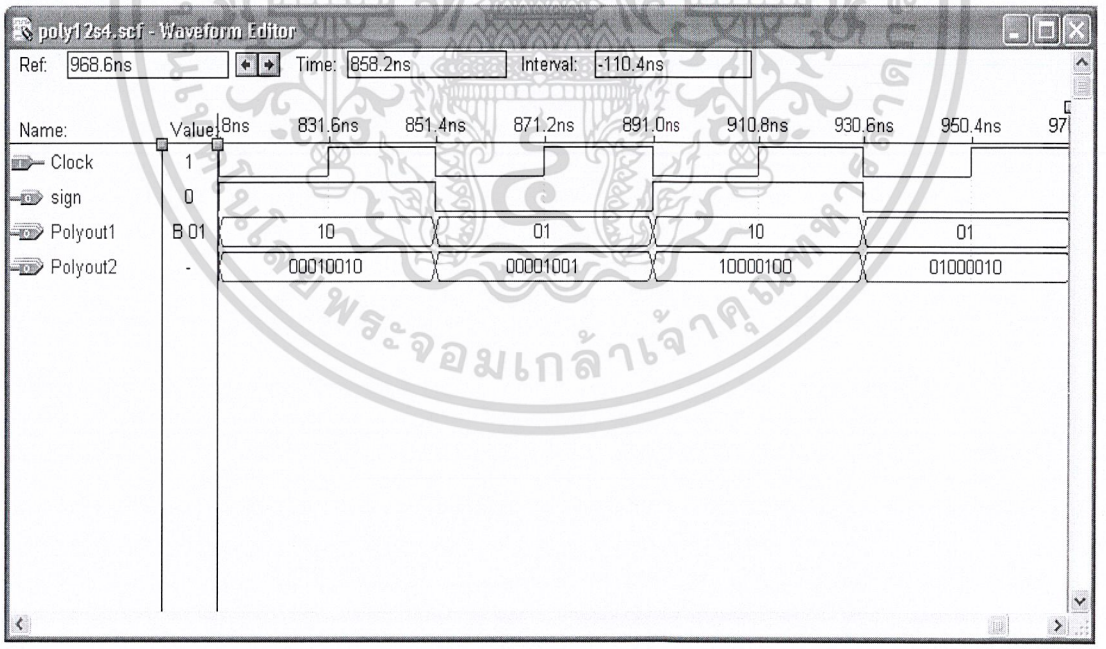
รูปที่ 4.2 แสดงการจำลองการทำงานของวงจร LFSR ที่มีโพลิโนเมียล m = 5

4.1.2 วงจร LFSR ที่มีโพลิโนเมียล m = 12 สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์ดังนี้



รูปที่ 4.3 แสดงสัญลักษณ์ของวงจร LFSR ที่มีโพลิโนเมียล m = 12 ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

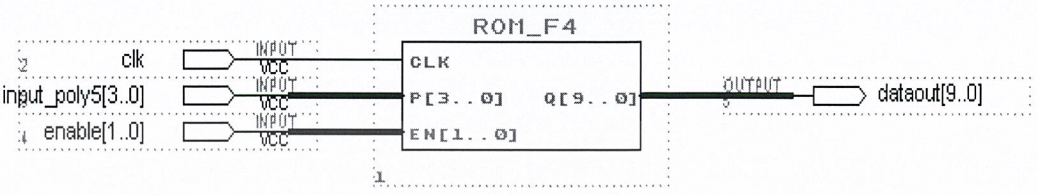
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังนี้



รูปที่ 4.4 แสดงการจำลองการทำงานของวงจร LFSR ที่มีโพลิโนเมียล m = 12

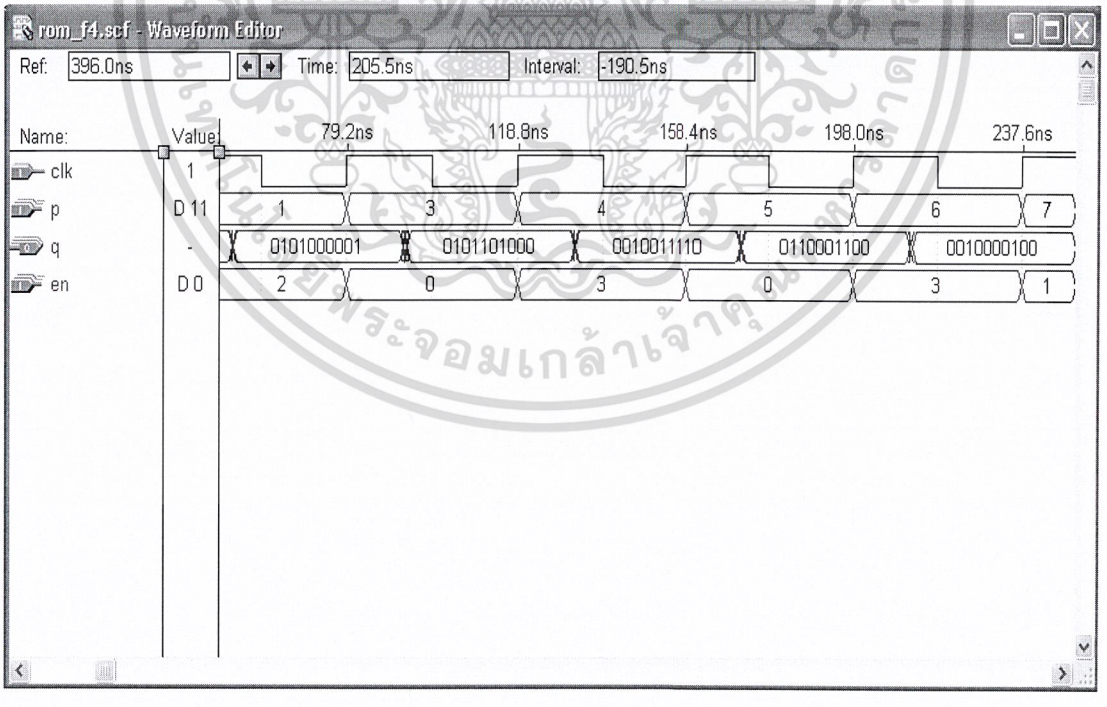
4.2 ส่วนการสร้างหน่วยความจำถาวร(ROM)

4.2.1 หน่วยความจำของฟังก์ชั่นเอฟ(ROM_f (x)) สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์ ดังนี้



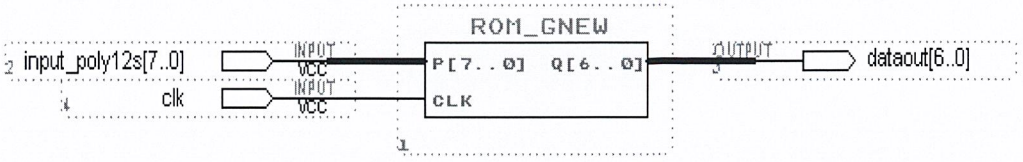
รูปที่ 4.5 แสดงสัญลักษณ์ของหน่วยความจำของฟังก์ชั่นเอฟ(ROM_f (x))
ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังนี้



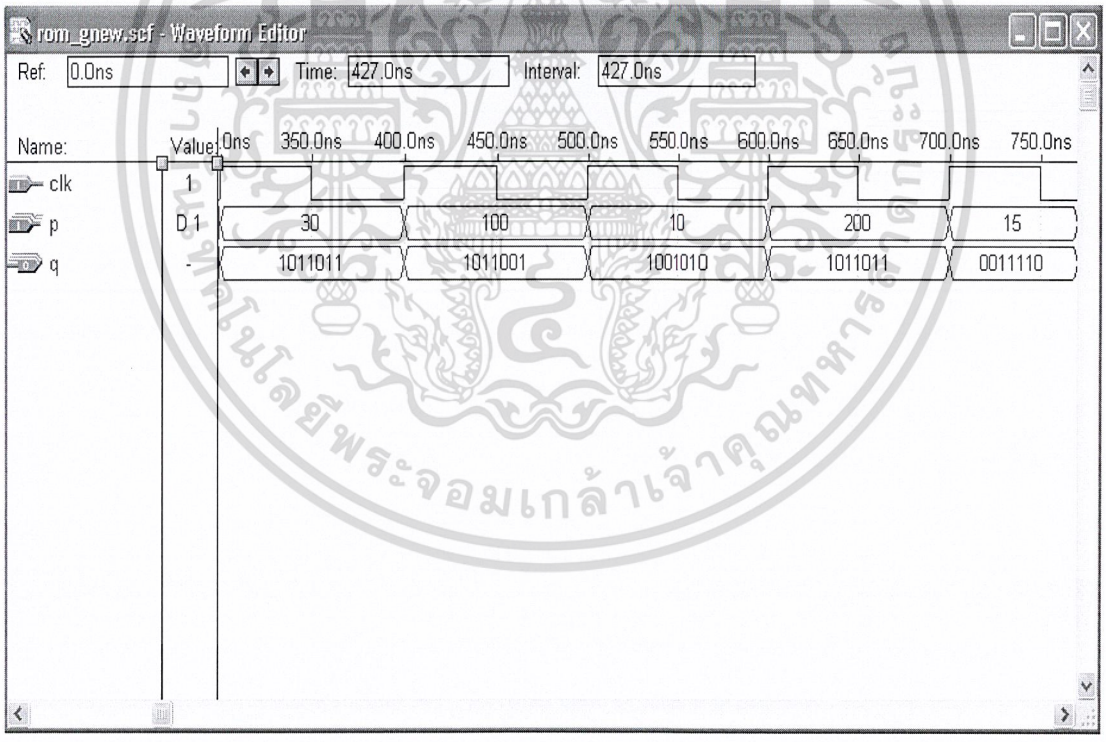
รูปที่ 4.6 แสดงการจำลองการทำงานของหน่วยความจำของฟังก์ชั่นเอฟ(ROM_f (x))

4.2.2 หน่วยความจำของฟังก์ชั่นจี (ROM_g(x)) สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์ดังนี้



รูปที่ 4.7 แสดงสัญลักษณ์ของหน่วยความจำของฟังก์ชั่นจี(ROM_g (x)) ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

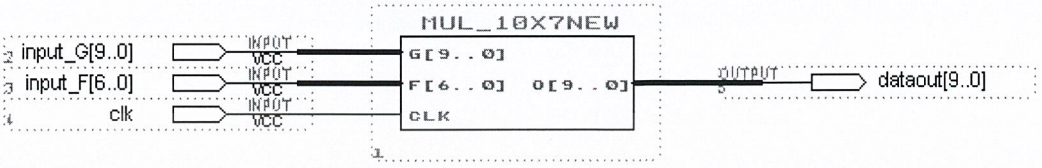
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังนี้



รูปที่ 4.8 แสดงการจำลองการทำงานของหน่วยความจำของฟังก์ชั่นจี(ROM_g (x))

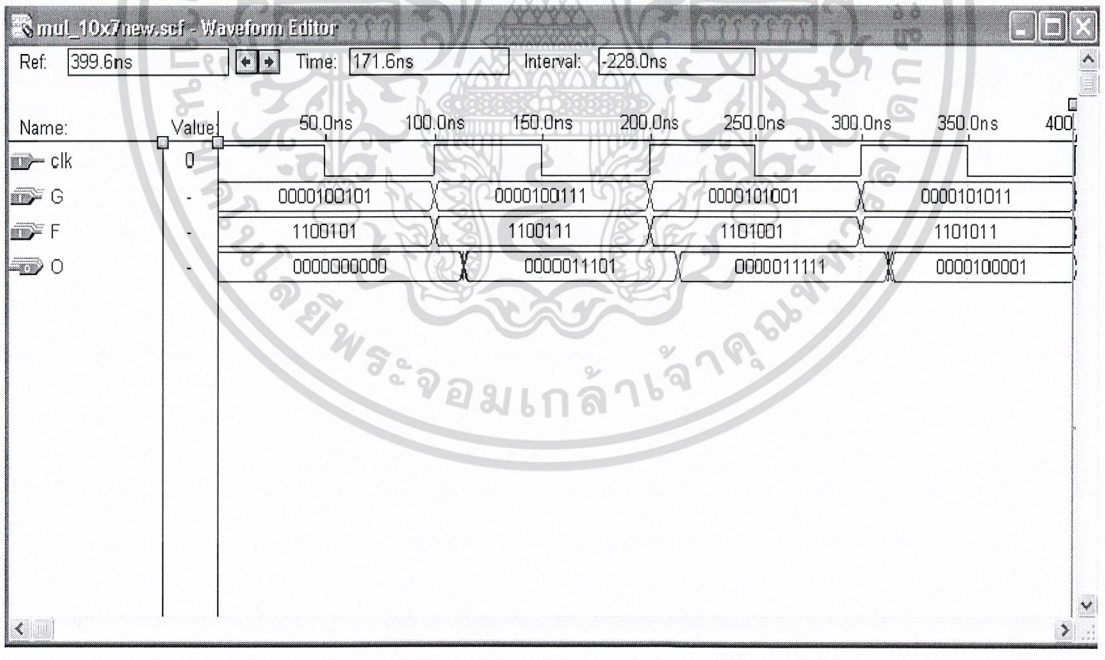
4.3 ส่วนการสร้างวงจรการคูณ (Multiplier)

วงจรการคูณขนาด(10x7) บิต สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์ดังนี้



รูปที่ 4.9 แสดงสัญลักษณ์ของวงจรการคูณขนาด(10x7)บิต
ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

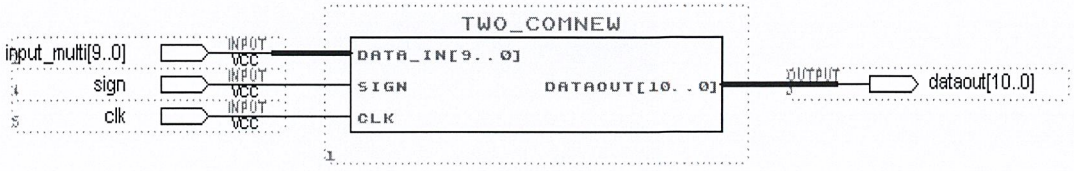
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังนี้



รูปที่ 4.10 แสดงการจำลองการทำงานของวงจรการคูณขนาด(10x7)บิต

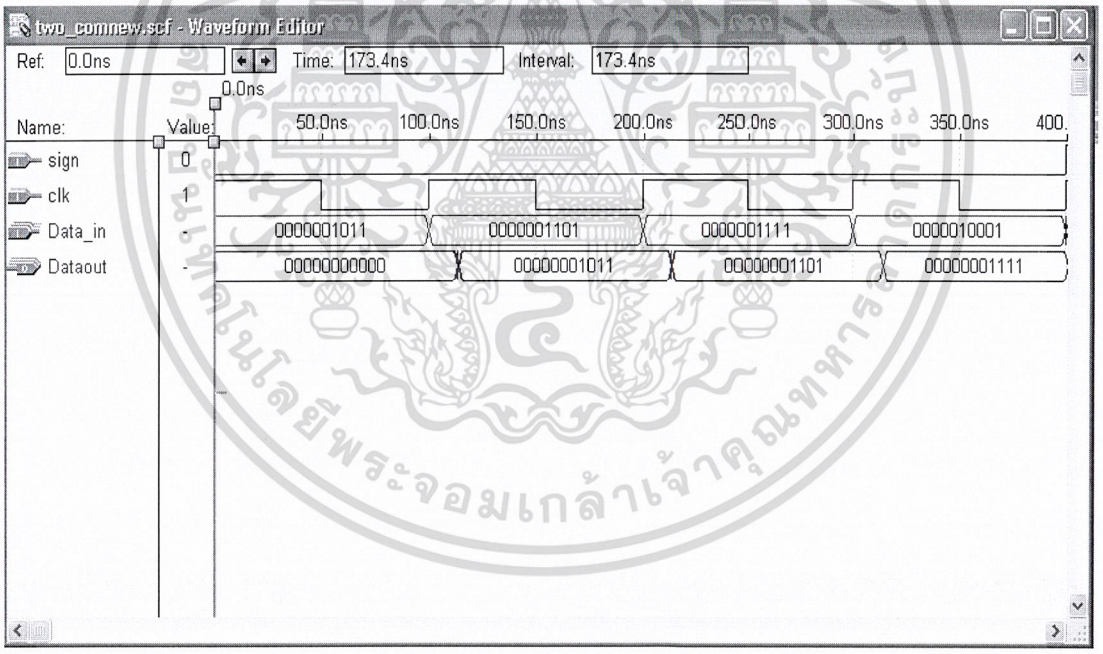
4.4 ส่วนการสร้างวงจรทูคอมพลีเมนท์ (2 ‘s complement)

วงจรทูคอมพลีเมนท์ สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์ดังนี้



รูปที่ 4.11 แสดงสัญลักษณ์ของวงจรทูคอมพลีเมนท์ ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

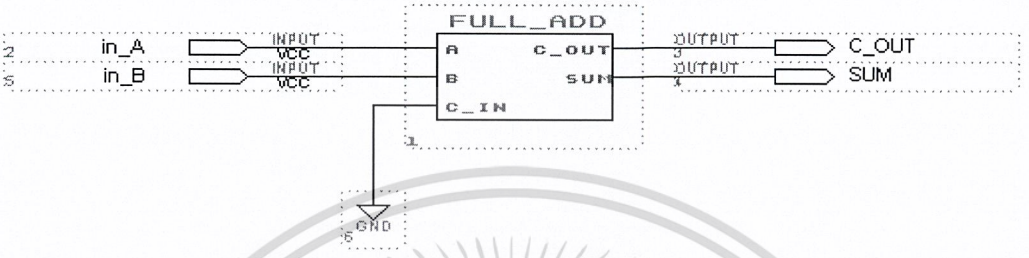
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังนี้



รูปที่ 4.12 แสดงการจำลองการทำงานของวงจรทูคอมพลีเมนท์

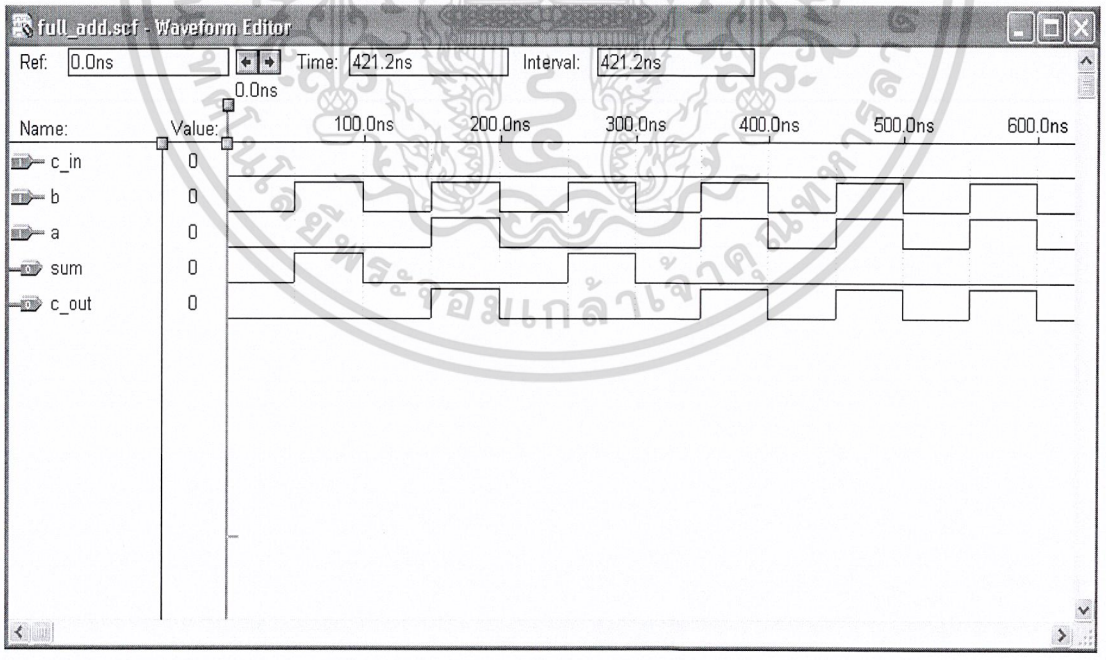
4.5 ส่วนการสร้างวงจรแอคคิวมูเลเตอร์(Accumulator)

4.5.1 วงจรฟูลแอดเดอร์(Full adder) สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์ดังนี้



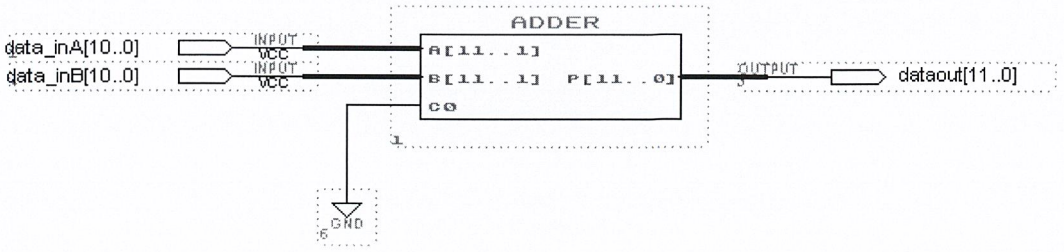
รูปที่ 4.13 แสดงสัญลักษณ์ของวงจรฟูลแอดเดอร์ ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังนี้



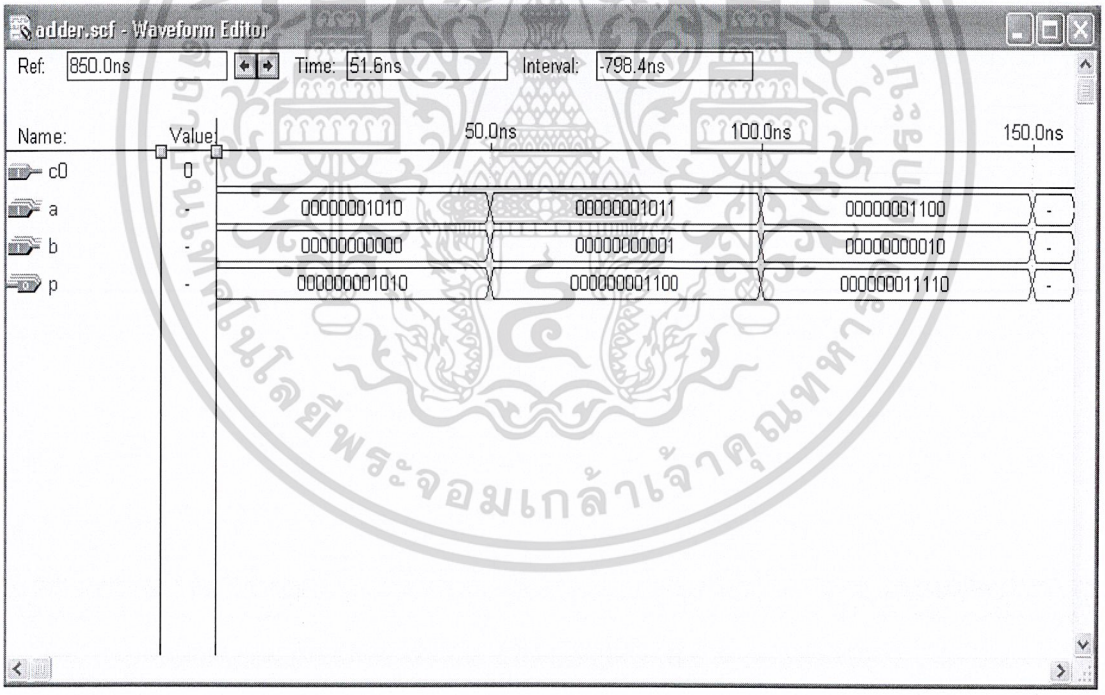
รูปที่ 4.14 แสดงการจำลองการทำงานของวงจรฟลูแอดเดอร์

4.5.2 วงจรการบวก(Adder) สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์ดังนี้



รูปที่ 4.15 แสดงสัญลักษณ์ของวงจรการบวก ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

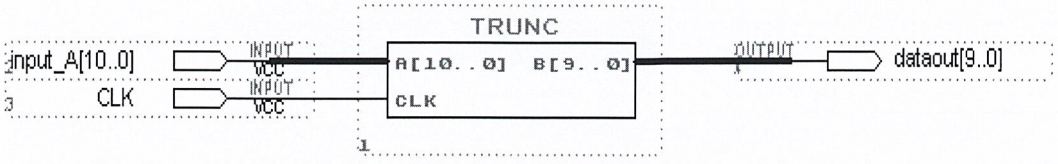
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังนี้



รูปที่ 4.16 แสดงการจำลองการทำงานของวงจรการบวก

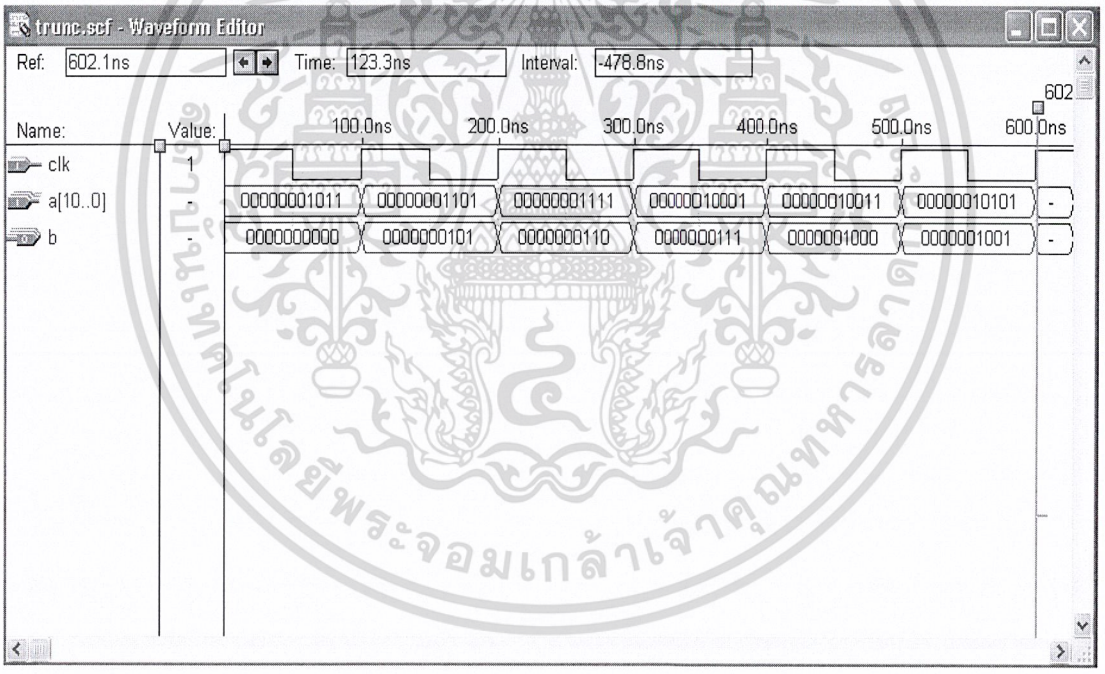
4.6 ส่วนการสร้างวงจรตัดบิต (Truncation)

วงจรตัดบิตที่ทำการสังเคราะห์จากการเขียนโปรแกรมด้วยภาษา VHDLสามารถแสดงสัญลักษณ์ได้ดังนี้



รูปที่ 4.17 แสดงสัญลักษณ์ของวงจรตัดบิต ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

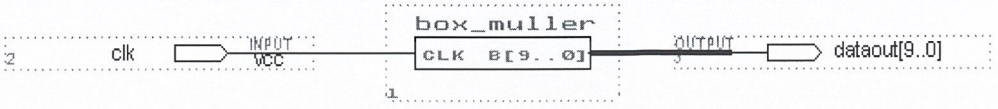
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังนี้



รูปที่ 4.18 แสดงการจำลองการทำงานของวงจรตัดบิต

4.7 ส่วนของการสร้าง Box-Muller

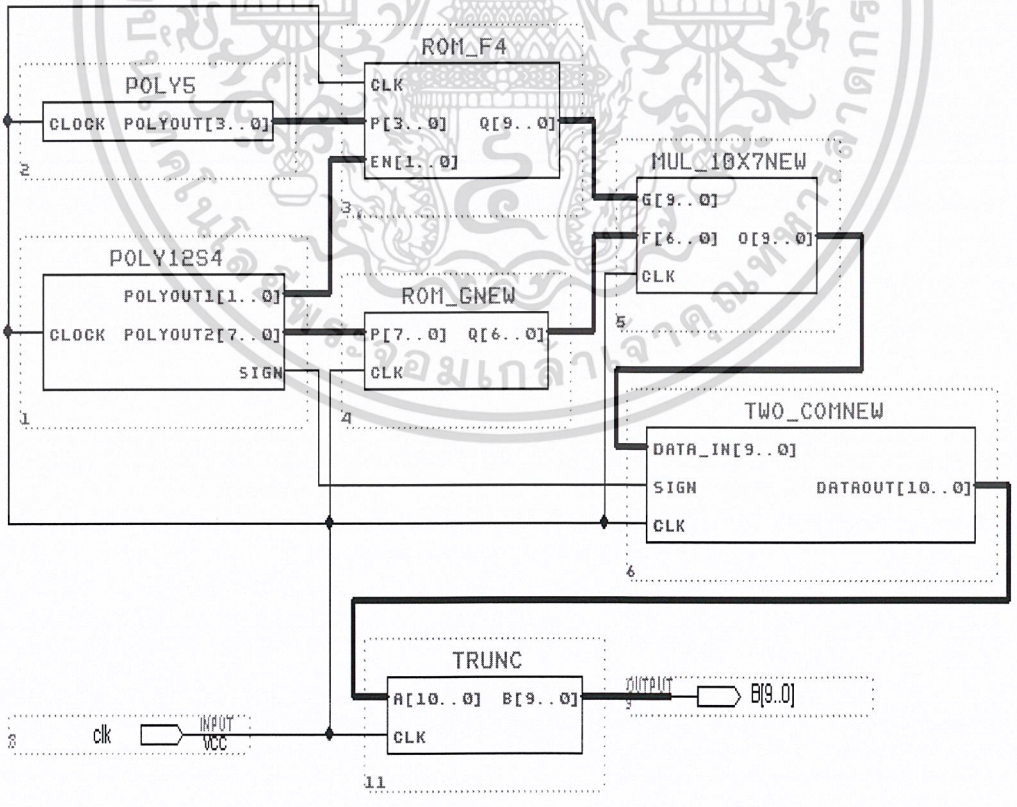
ทำหน้าที่ในการสร้างสัญญาณสุ่มที่มีพีดีเอฟแบบเกาส์เซียน ซึ่งประกอบไปด้วย วงจร LFSR ที่มีโพลิโนเมียล m = 5 วงจร LFSR ที่มีโพลิโนเมียล m = 12 และm =14 หน่วยความจำของฟังก์ชั่นเอฟ (ROM_f(x)) หน่วยความจำของฟังก์ชั่นจี(ROM_g(x)) วงจรการคูณขนาด(10x7)บิต วงจรทูคอมพลีเมนท์ วงจรตัดบิต ทำการจำลองการทำงานของทุกส่วน แล้วนำมาต่อกันทางซอฟต์แวร์ ทดสอบการทำงานอีกครั้ง สามารถเขียนโปรแกรมซึ่งสังเคราะห์ได้ และอุปกรณ์มีสัญลักษณ์(Symbol) ดังนี้



รูปที่ 4.19 แสดงสัญลักษณ์ของBox-Muller ที่ได้จากการ Compile โปรแกรมที่เขียนขึ้น

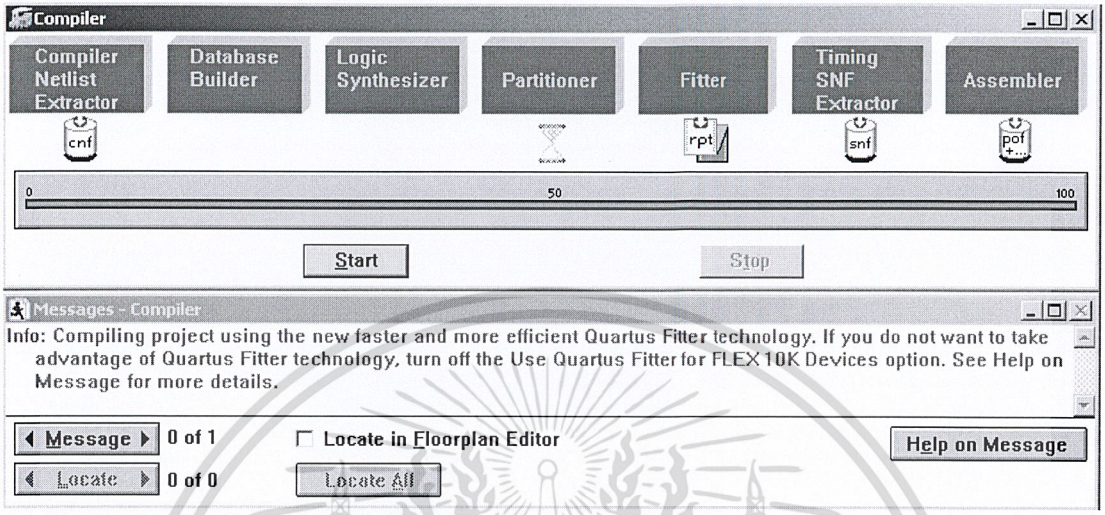
4.8 แสดงรูปวงจรภายในBox-Mullerและกราฟแสดงผล

4.8.1 การสร้างวงจร Box-Muller1 ที่มี LFSRโพลิโนเมียล m=12 ใช้ Rom_f(x) 4 ตัว ซึ่งเมื่อนำมาผลลัพธ์ของข้อมูลมาคำนวณหาค่า พีดีเอฟ ฮีสโตรแกรม ค่าออโตคอรีเลชั่นและความหนาแน่นกำลังเชิงสเปกตรัม จะสามารถแสดงได้ดังนี้



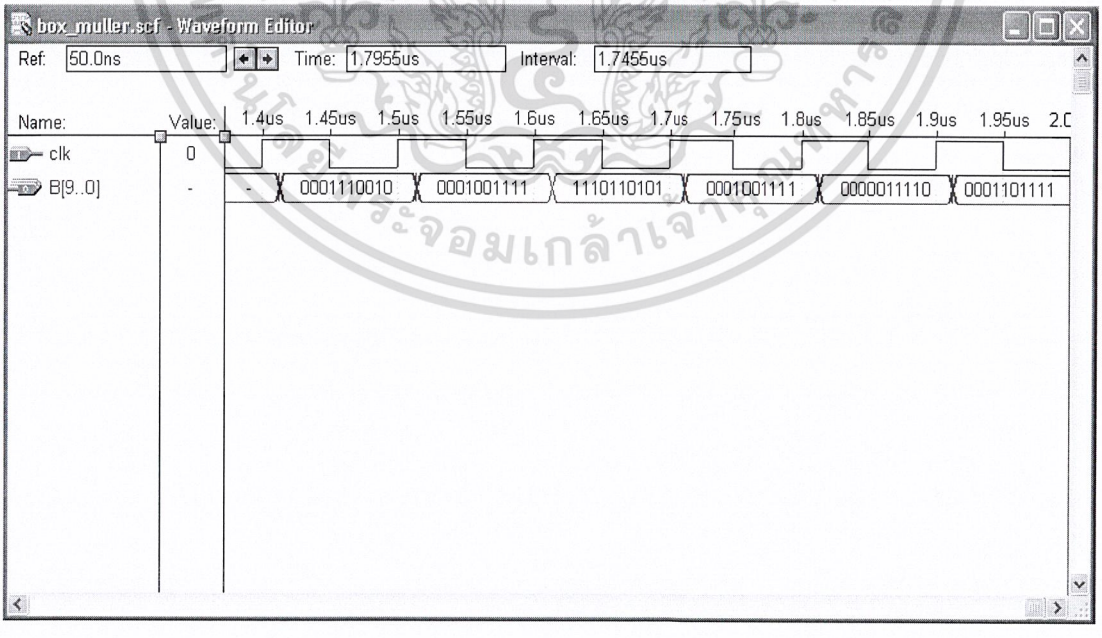
รูปที่ 4.20 แสดงการต่อสัญลักษณ์ของวงจรต่างๆเป็น Box-Muller1

แสดงการคอมไพล์วงจรที่ออกแบบขึ้นมาเพื่อให้ซอร์ฟแวร์ที่ใช้ในการสังเคราห์แสดงค่าอุปกรณ์ที่สามารถรองรับการทำงานและรายละเอียดต่างๆของสิ่งที่เราออกแบบขนาดของเกทที่เราใช้งานโดยแสดงออกมาเป็นเปอร์เซ็นต์การใช้งาน



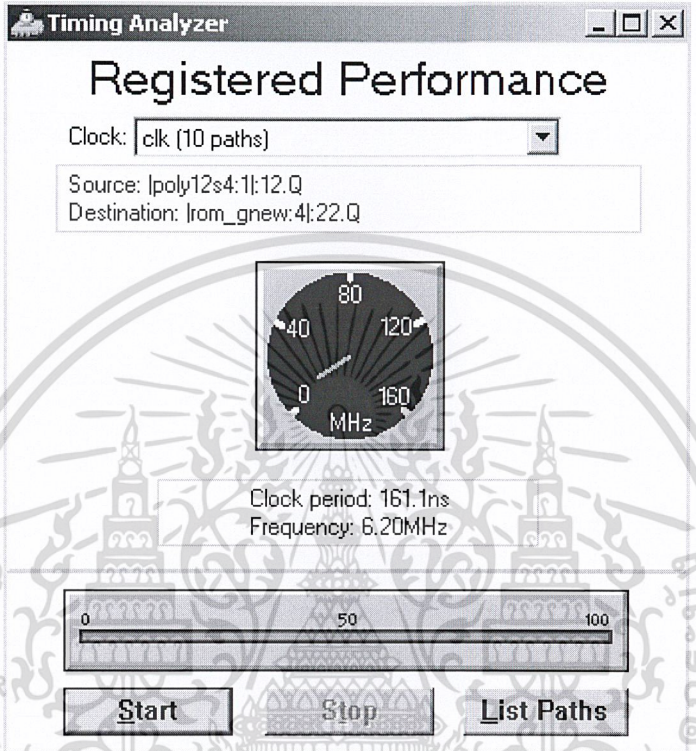
รูปที่ 4.21 แสดงการคอมไพล์ Box-Muller1เพื่อแสดงรายละเอียดการใช้อุปกรณ์

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานและตรวจสอบผลการทำงานเพื่อให้ได้ตามเงื่อนไขการออกแบบวงจรสามารถแสดงได้ดังนี้



รูปที่ 4.22 แสดงการจำลองการทำงานของ Box-Muller1

การสังเคราะห์วงจรที่เราออกแบบมานั้นจะมีข้อกำหนดทางด้านความถี่ที่นำมาใช้งานและระดับความเร็วของสัญญาณนาฬิกาซึ่งซอร์ฟแวร์ที่นำมาสังเคราะห์นั้นจะบอกรายระเอียดแสดงได้ดังรูปข้างล่างนี้



รูปที่ 4.23 แสดงค่าเวลาและความถี่ที่ใช้งานของ Box-Muller1

ผลของการพรอตค่าจากวงจรที่รูป 4.20 มีรายละเอียดดังนี้

จำนวนของข้อมูล(N)= 126,946

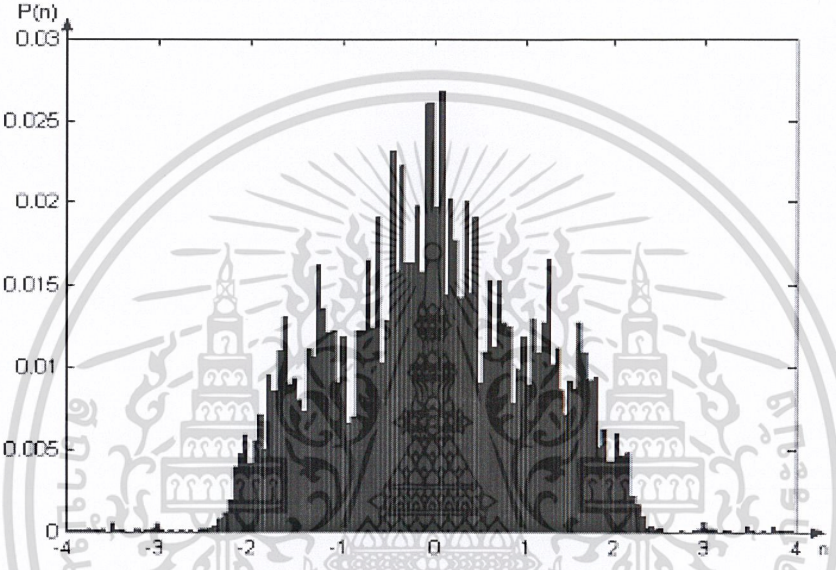
ค่าเบี่ยงเบนมาตรฐาน (σ) =1.1497

ค่าเฉลี่ย (μ)= -0.0112

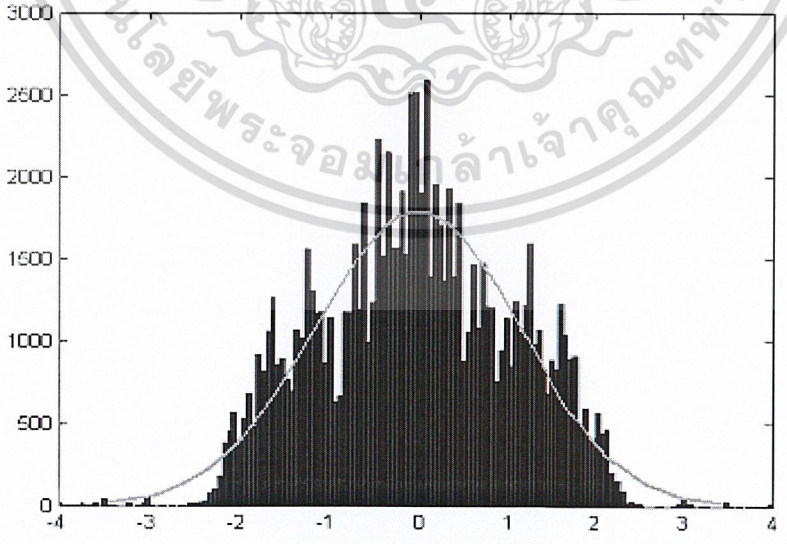
ค่าความแปรปรวน (σ^2) = 1.3217

ค่าสูงสุดของข้อมูล = 3.9844

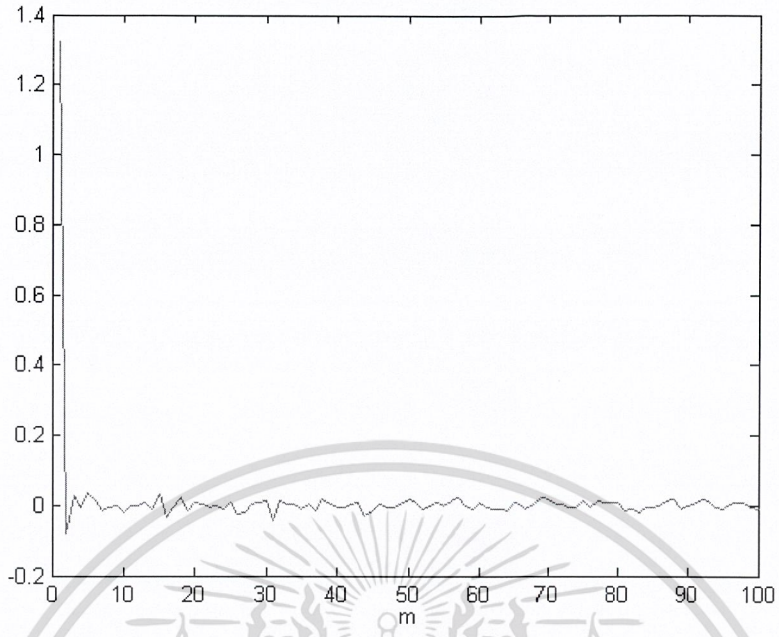
ค่าต่ำสุดของข้อมูล = -4



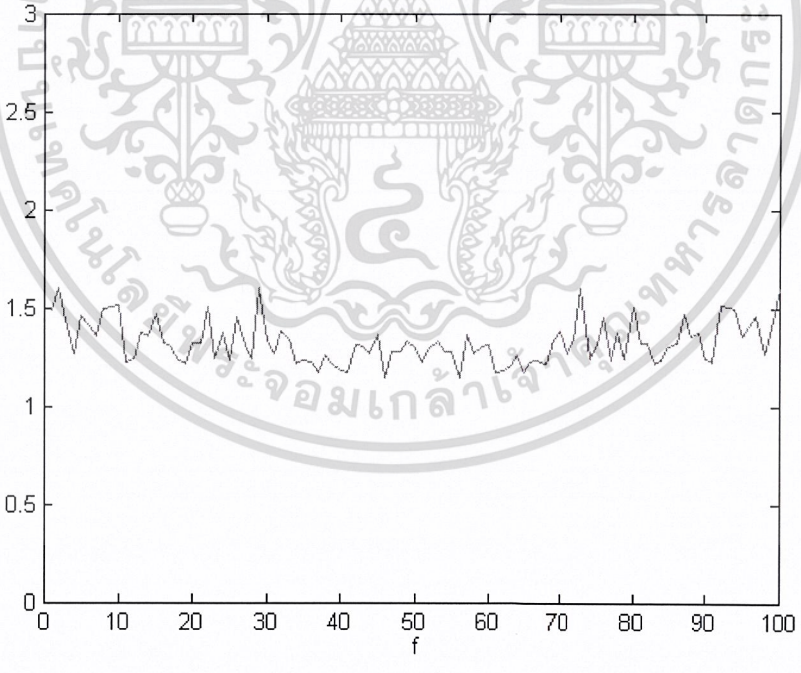
รูปที่4.24 แสดงค่าพีดีเอฟของ Box-Muller1



รูปที่ 4.25 แสดงฮีสโตรแกรมของ Box-Muller1เทียบกับเงื่อนไขเดียวกันตามทฤษฎี

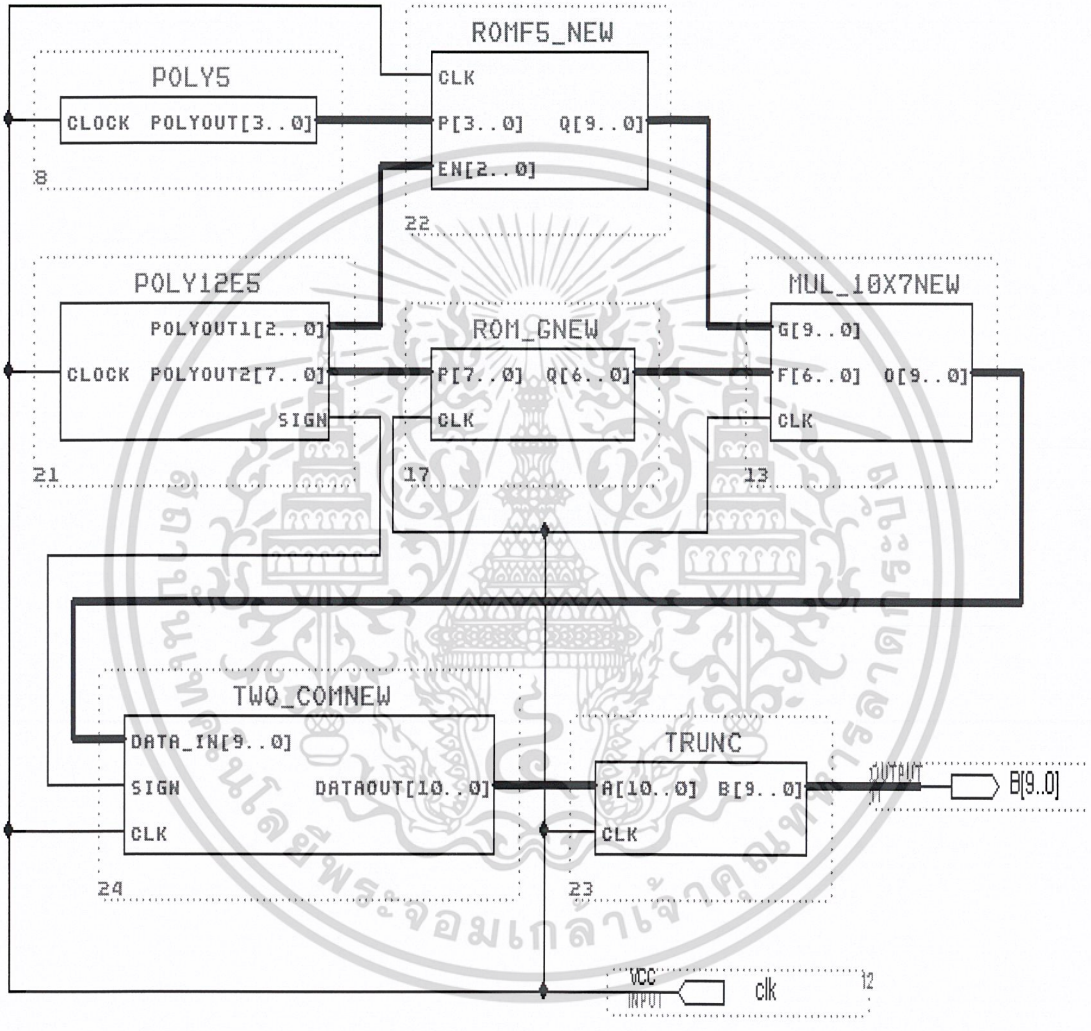


รูปที่ 4.26 แสดงกราฟ ออโตคอรีเลชั่นของ Box-Muller1



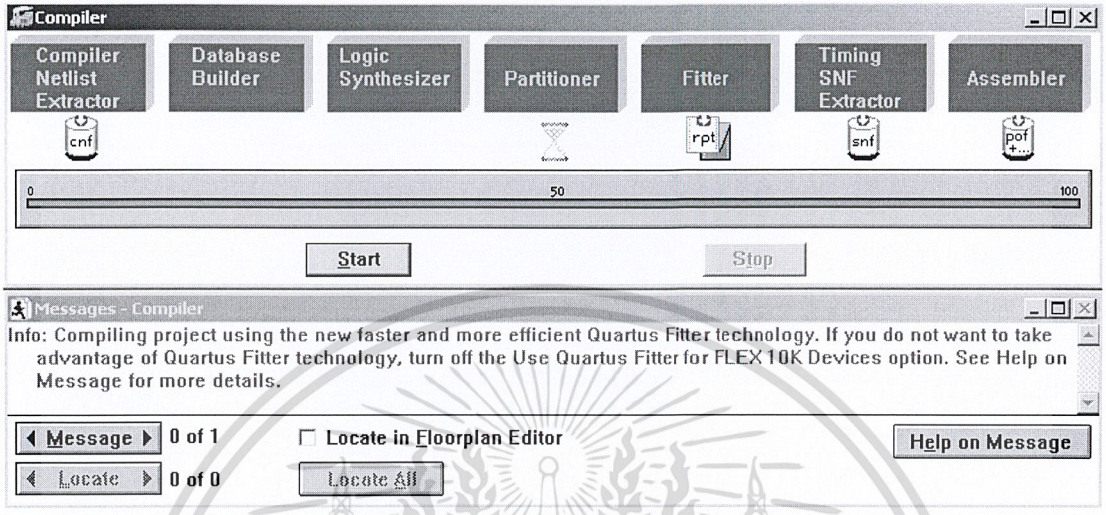
รูปที่ 4.27 แสดงกราฟความหนาแน่นกำลังเชิงสเปกตรัมของ Box-Muller1

4.8.2 การสร้างวงจร Box-Muller2 ที่มี LFSR โพลิโนเมียล m= 12 ใช้ Rom_f(x) 5 ตัว ซึ่งเมื่อนำมาผลลัพธ์ของข้อมูลมาคำนวณหาค่า พีดีเอฟ ฮีสโตรแกรม ค่าออโต้คอรีเลชั่นและความหนาแน่นกำลังเชิงสเปกตรัม จะสามารถแสดงได้ดังนี้



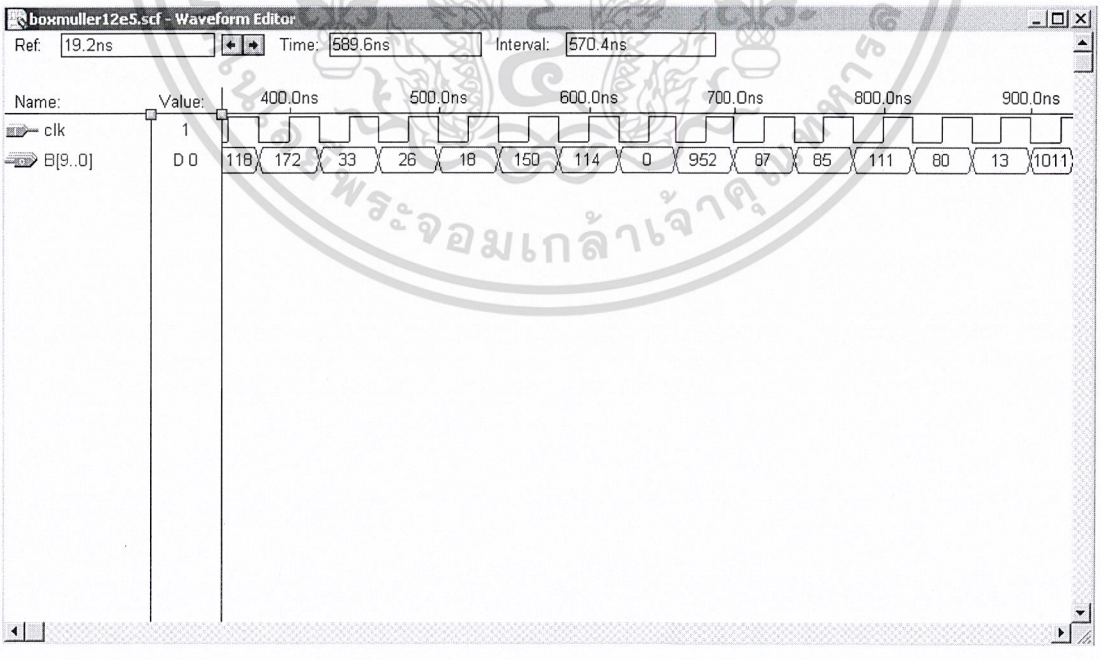
รูปที่ 4.28 แสดงการต่อสัญลักษณ์ของวงจรต่างๆเป็น Box-Muller2

แสดงการคอมไพล์วงจรที่ออกแบบขึ้นมาเพื่อให้ซอร์ฟแวร์ที่ใช้ในการสังเคราห์แสดงค่าอุปกรณ์ที่สามารถรองรับการทำงานและรายละเอียดต่างๆของสิ่งที่เราออกแบบขนาดของเกทที่เราใช้งานโดยแสดงออกมาเป็นเปอร์เซ็นต์การใช้งาน



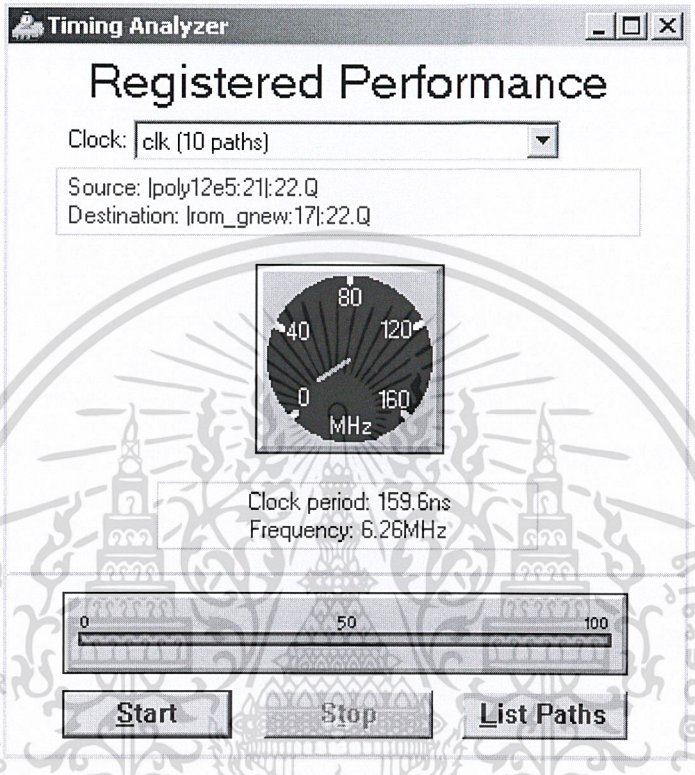
รูปที่ 4.29 แสดงการคอมไพล์ Box-Muller2เพื่อแสดงรายละเอียดการใช้อุปกรณ์

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานและตรวจสอบผลการทำงานเพื่อให้ได้ตามเงื่อนไขการออกแบบวงจรสามารถแสดงได้ดังนี้



รูปที่ 4.30 แสดงการจำลองการทำงานของ Box-Muller2

การสังเคราะห์วงจรที่เราออกแบบมานั้นจะมีข้อกำหนดทางด้านความถี่ที่นำมาใช้งานและระดับความเร็วของสัญญาณนาฬิกาซึ่งซอร์ฟแวร์ที่นำมาสังเคราะห์นั้นจะบอกรายระเอียดแสดงได้ดังรูปข้างล่างนี้



รูปที่ 4.31 แสดงค่าเวลาและความถี่ที่ใช้งานของ Box-Muller2

ผลของการพรอตค่าจากวงจรที่รูป 4.28 มีรายละเอียดดังนี้

จำนวนของข้อมูล(N)= 126,946

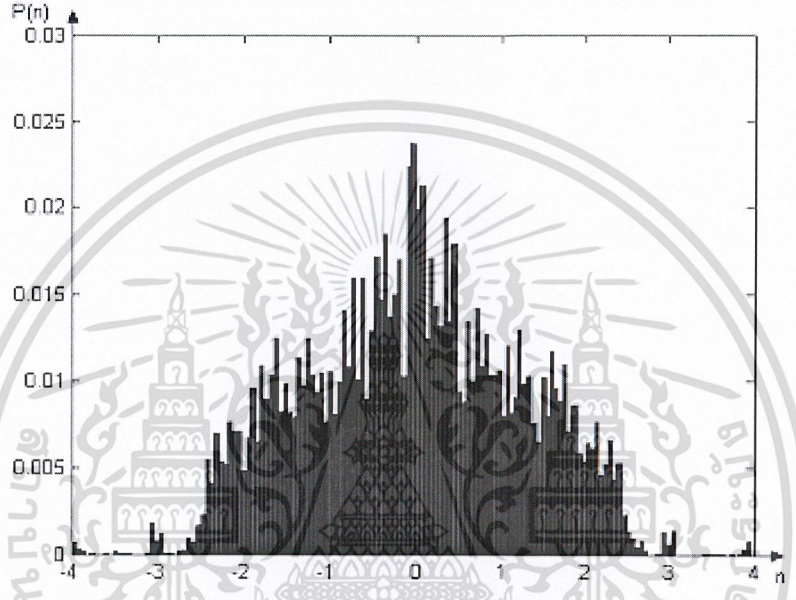
ค่าเบี่ยงเบนมาตรฐาน (σ) =1.2626

ค่าเฉลี่ย (μ)= -0.0279

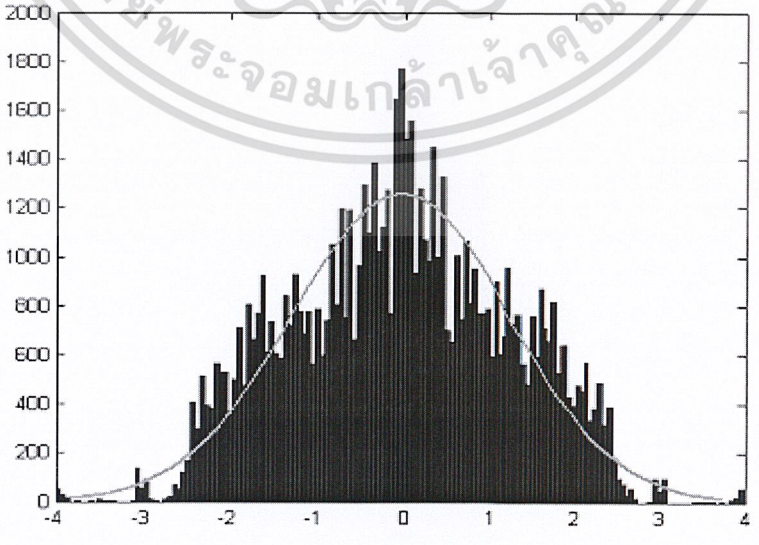
ค่าความแปรปรวน (σ^2) = 1.5942

ค่าสูงสุดของข้อมูล = 3.9844

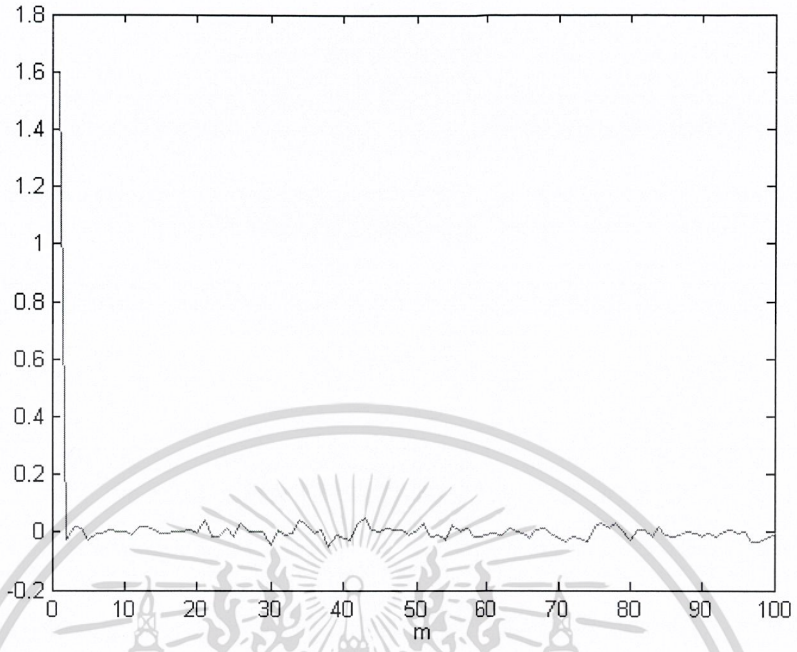
ค่าต่ำสุดของข้อมูล = -4



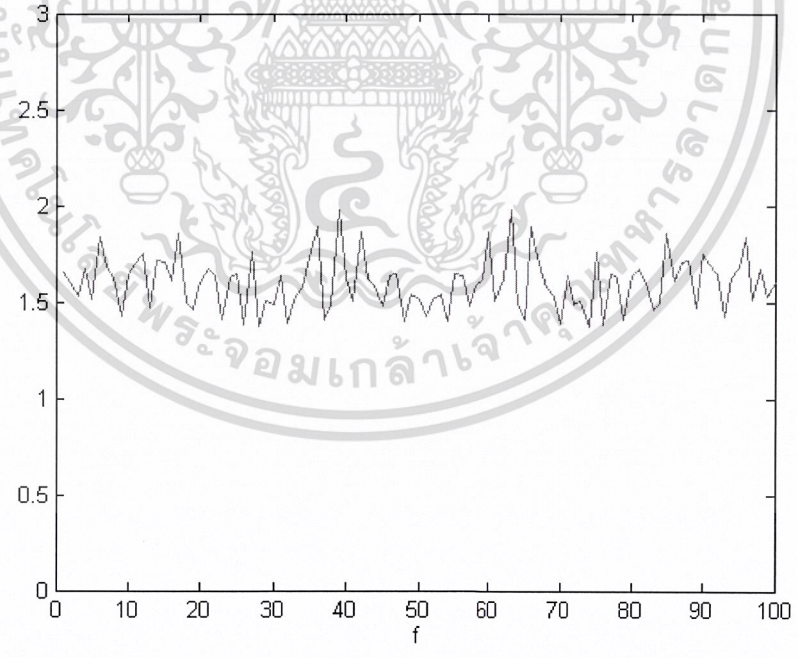
รูปที่4.32แสดงค่าพีดีเอฟของ Box-Muller2



รูปที่ 4.33แสดงฮีสโตรแกรมของ Box-Muller2เทียบกับเงื่อนไขเดียวกันตามทฤษฎี

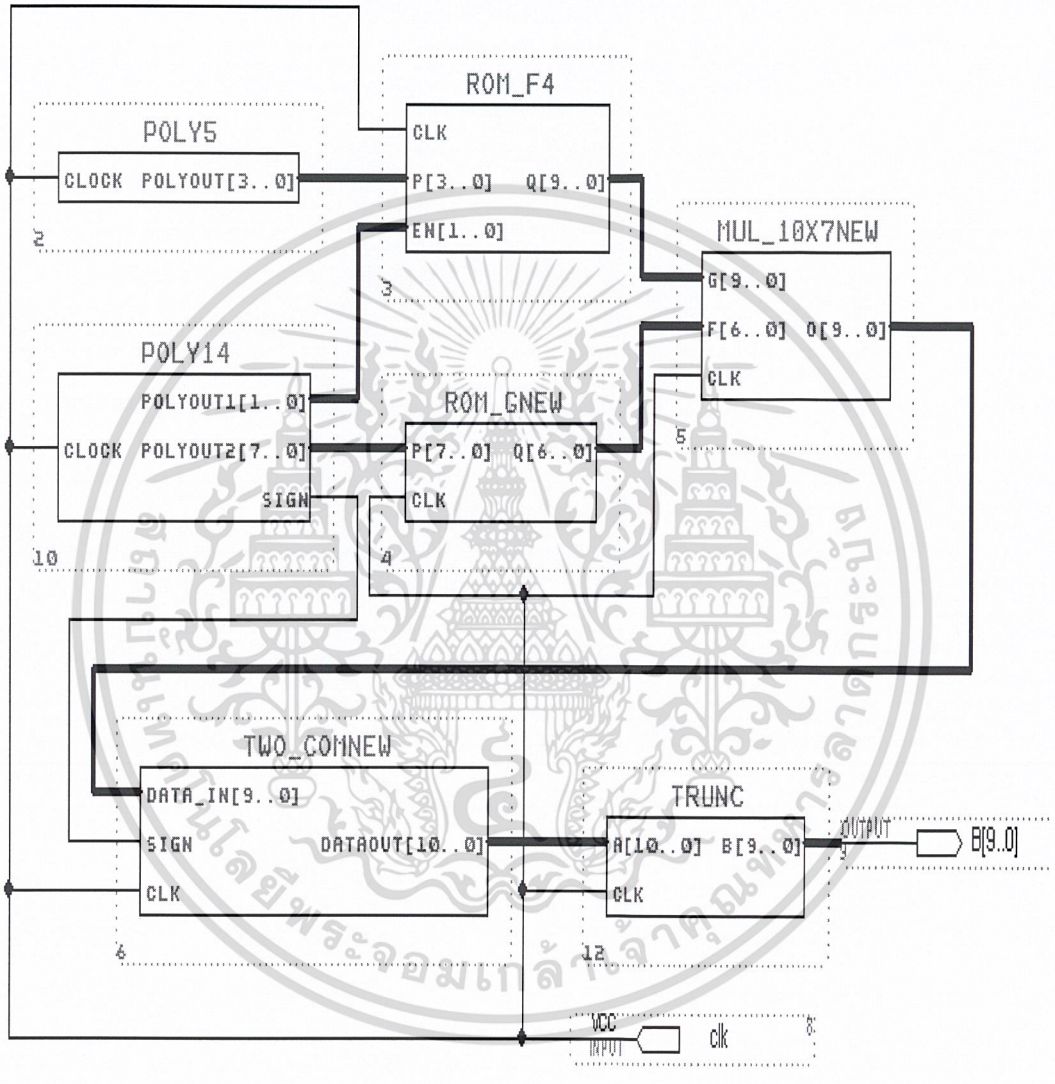


รูปที่ 4.34 แสดงกราฟออโต้คอรีเลชั่นของ Box-Muller2



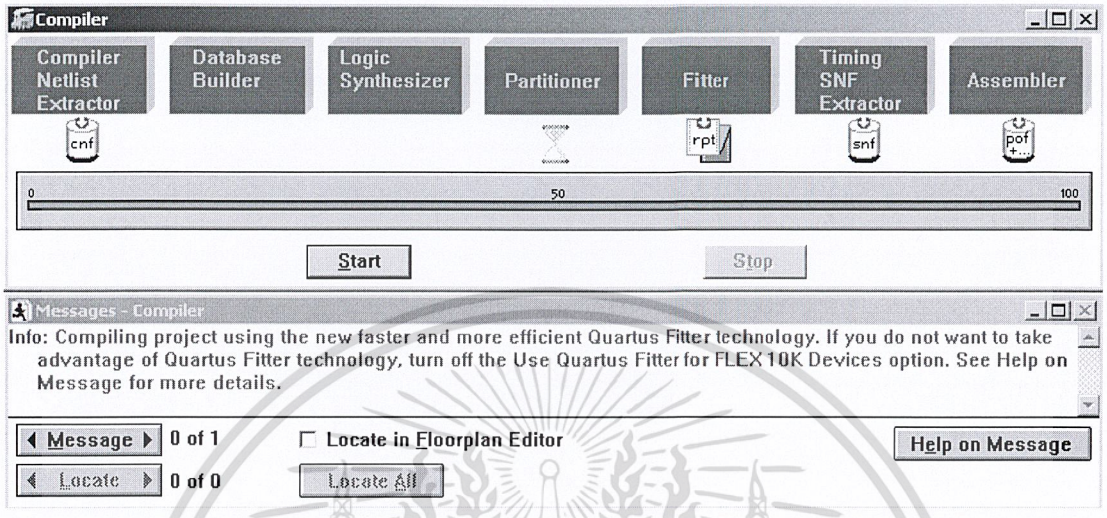
รูปที่ 4.35 แสดงกราฟความหนาแน่นกำลังเชิงสเปกตรัมของ Box-Muller2

4.8.3 การสร้างวงจร Box-Muller3 ที่มี LFSR โพลิโนเมียล m=14 ใช้ Rom_f(x) 4 ตัว ซึ่งเมื่อนำมาผลลัพธ์ของข้อมูลมาคำนวณหาค่า พีดีเอฟ ฮีสโตรแกรม ค่าออโต้คอรีเลชั่นและความหนาแน่นกำลังเชิงสเปกตรัม จะสามารถแสดงได้ดังนี้



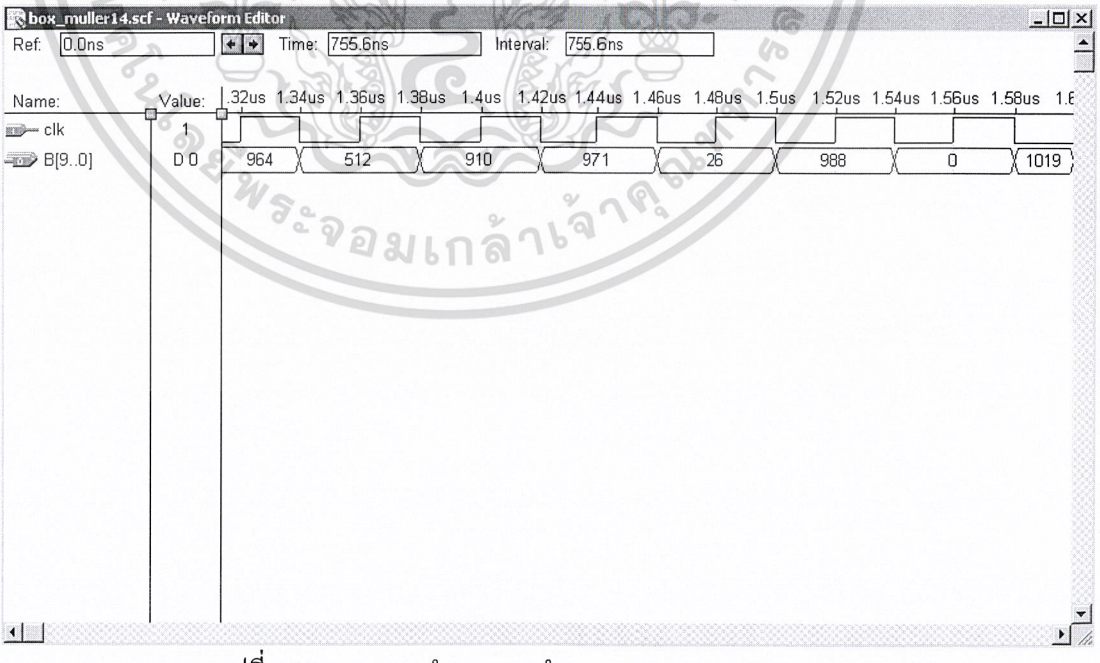
รูปที่ 4.36แสดงการต่อสัญลักษณ์ของวงจรต่างๆเป็น Box-Muller3

แสดงการคอมไพล์วงจรที่ออกแบบขึ้นมาเพื่อให้ซอร์ฟแวร์ที่ใช้ในการสังเคราห์แสดงค่าอุปกรณ์ที่สามารถรองรับการทำงานและรายละเอียดต่างๆของสิ่งที่เราออกแบบขนาดของเกทที่เราใช้งานโดยแสดงออกมาเป็นเปอร์เซ็นต์การใช้งาน



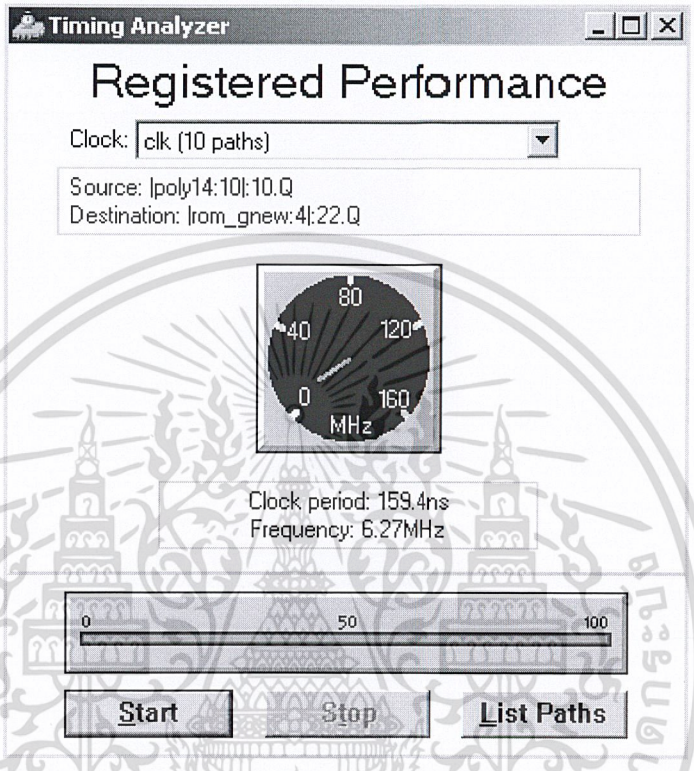
รูปที่ 4.37แสดงการคอมไพล์ Box-Muller3 เพื่อแสดงรายละเอียดการใช้อุปกรณ์

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานและตรวจสอบผลการทำงานเพื่อให้ได้ตามเงื่อนไขการออกแบบวงจรสามารถแสดงได้ดังนี้



รูปที่ 4.38 แสดงการจำลองการทำงานของ Box-Muller3

การสังเคราะห์วงจรที่เราออกแบบมานั้นจะมีข้อกำหนดทางด้านความถี่ที่นำมาใช้งานและระดับความเร็วของสัญญาณนาฬิกาซึ่งซอร์ฟแวร์ที่นำมาสังเคราะห์นั้นจะบอกรายระเอียดแสดงได้ดังรูปข้างล่างนี้



รูปที่ 4.39 แสดงค่าเวลาและความถี่ที่ใช้งานของ Box-Muller3

ผลของการพรอตค่าจากวงจรที่รูป 4.36 มีรายละเอียดดังนี้

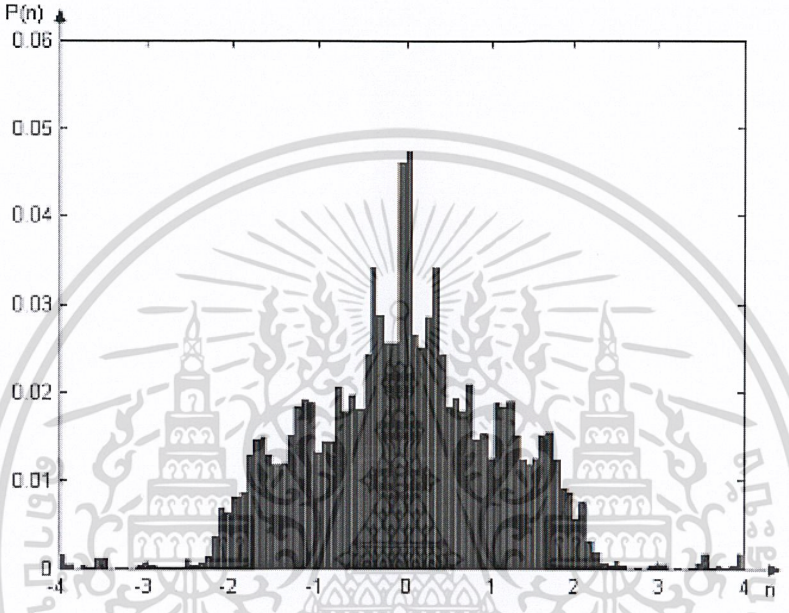
จำนวนของข้อมูล(N)= 305,548

ค่าเบี่ยงเบนมาตรฐาน (σ) =1.1218

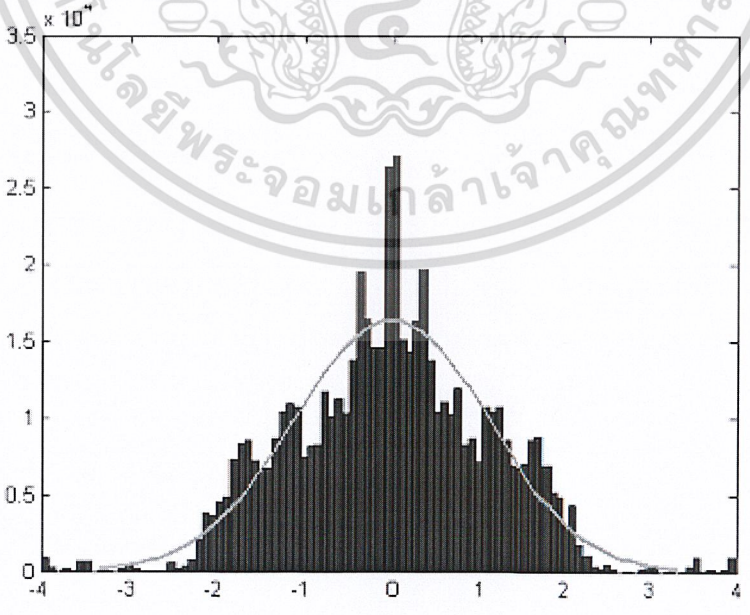
ค่าเฉลี่ย (μ)= - 0.0042

ค่าความแปรปรวน (σ^2) = 1.2584

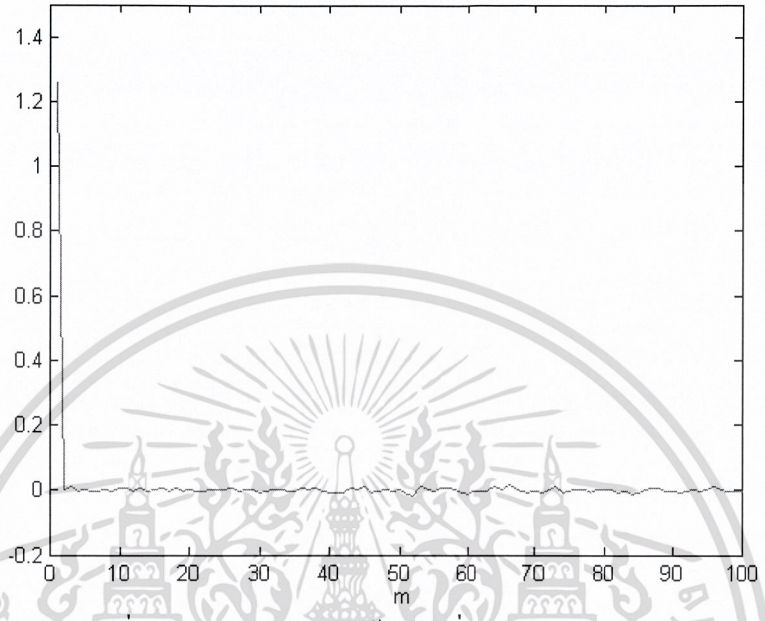
ค่าสูงสุดของข้อมูล = 3.9844 ค่าต่ำสุดของข้อมูล = -4



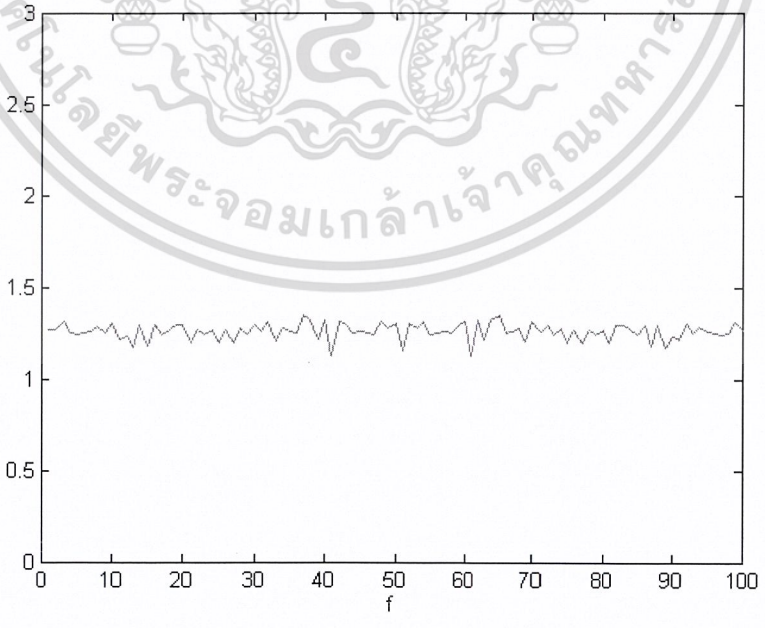
รูปที่4.40 แสดงค่าพีดีเอฟของ Box-Muller3



รูปที่ 4.41 ฮีสโตรแกรมของ Box-Muller3เทียบกับเงื่อนไขเดียวกันตามทฤษฎี

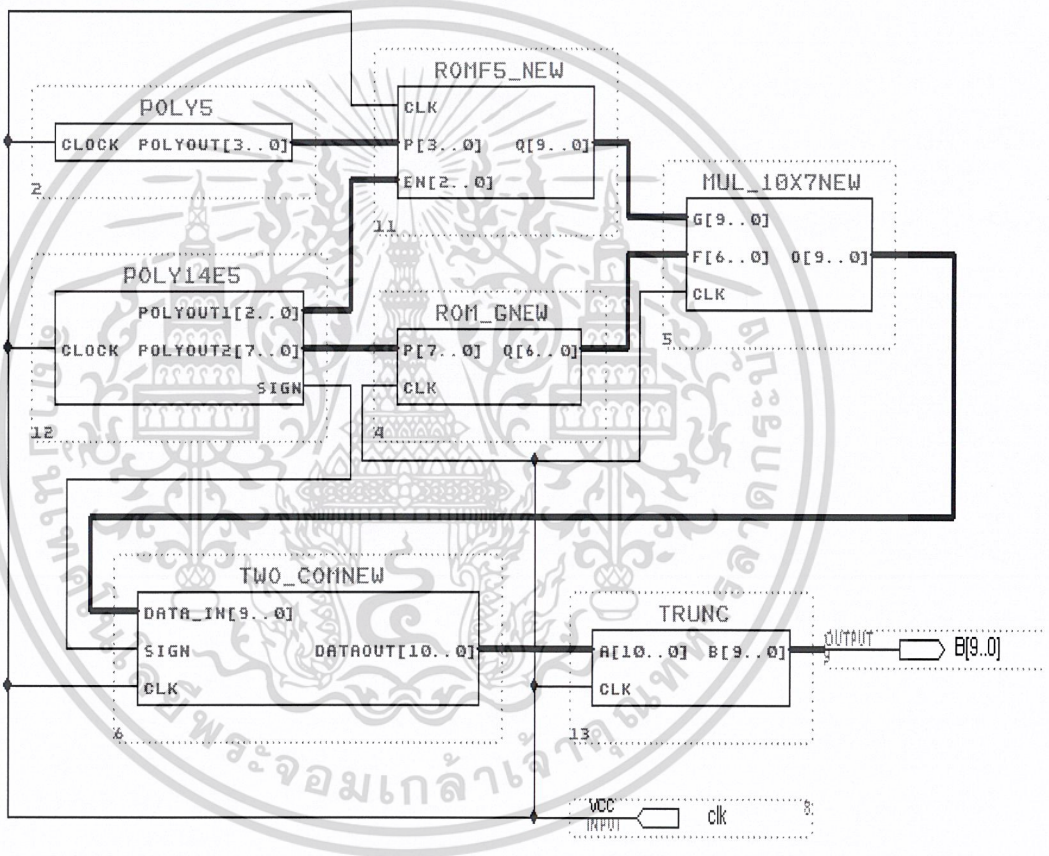


รูปที่ 4.42 แสดงกราฟ ออโต้คอรีเลชั่น Box-Muller3



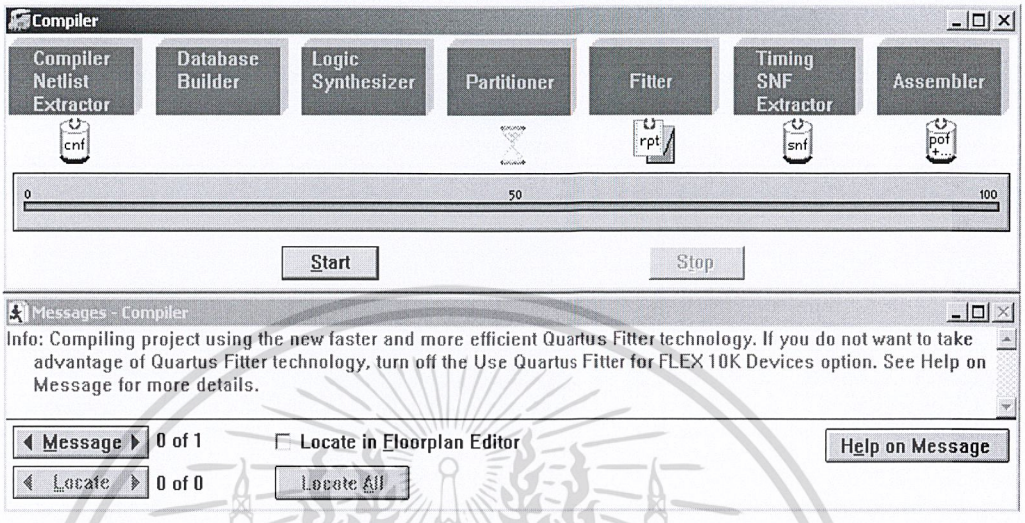
รูปที่ 4.43 แสดงกราฟความหนาแน่นกำลังเชิงสเปกตรัมของ Box-Muller3

4.8.4 การสร้างวงจร Box-Muller4 ที่มี LFSRโพลิโนเมียล m= 14 ใช้ Rom_f(x) 5ตัว ซึ่งเมื่อนำมาผลลัพธ์ของข้อมูลมาคำนวณหาค่า พีดีเอฟ ฮีสโตรแกรม ค่าออโต้คอรีเลชั่นและความหนาแน่นกำลังเชิงสเปกตรัม จะสามารถแสดงได้ดังนี้



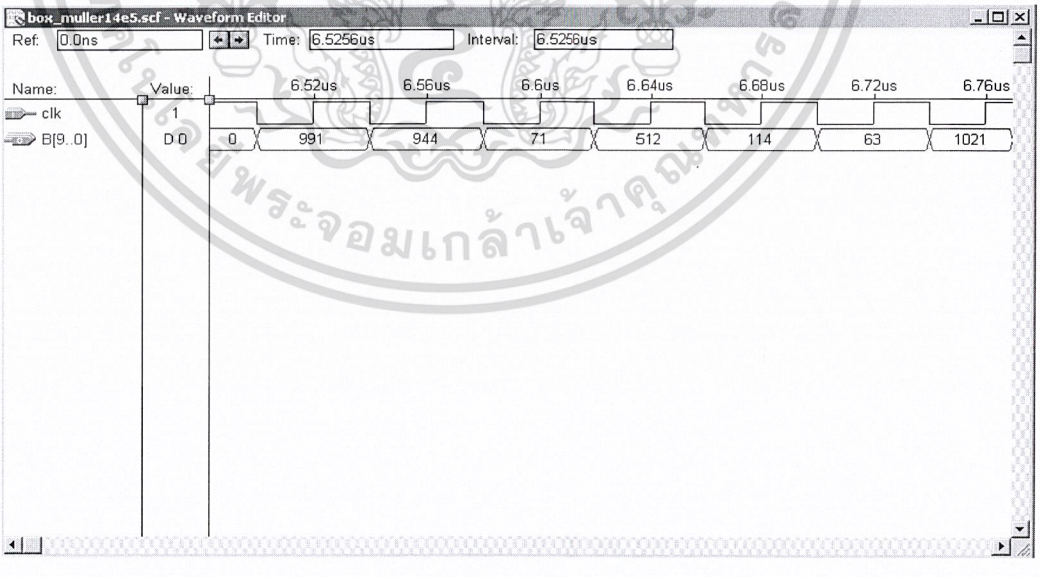
รูปที่ 4.44 แสดงการต่อสัญลักษณ์ของวงจรต่างๆเป็น Box-Muller4

แสดงการคอมไพล์วงจรที่ออกแบบขึ้นมาเพื่อให้ซอร์ฟแวร์ที่ใช้ในการสังเคราห์แสดงค่าอุปกรณ์ที่สามารถรองรับการทำงานและรายละเอียดต่างๆของสิ่งที่เราออกแบบขนาดของเกทที่เราใช้งานโดยแสดงออกมาเป็นเปอร์เซ็นต์การใช้งาน



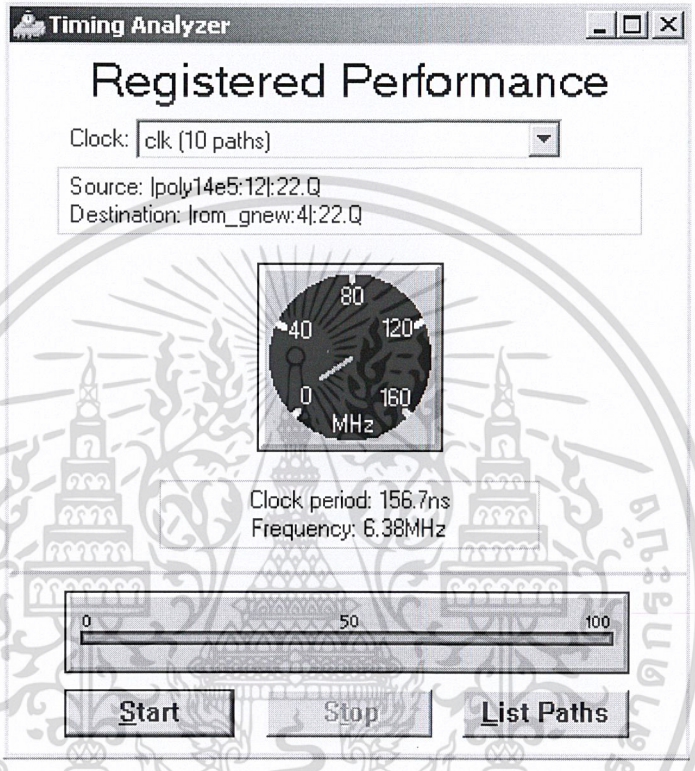
รูปที่ 4.45 แสดงการคอมไพล์ Box-Muller4เพื่อแสดงรายละเอียดการใช้อุปกรณ์

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานและตรวจสอบผลการทำงานเพื่อให้ได้ตามเงื่อนไขการออกแบบวงจรสามารถแสดงได้ดังนี้



รูปที่ 4.46 แสดงการจำลองการทำงานของ Box-Muller4

การสังเคราะห์วงจรที่เราออกแบบมานั้นจะมีข้อกำหนดทางด้านความถี่ที่นำมาใช้งานและระดับความเร็วของสัญญาณนาฬิกาซึ่งซอร์ฟแวร์ที่นำมาสังเคราะห์นั้นจะบอกรายระเอียดแสดงได้ดังรูปข้างล่างนี้



รูปที่ 4.47 แสดงค่าเวลาและความถี่ที่ใช้งานของ Box-Muller4

ผลของการพรอตค่าจากวงจรที่รูป 4.44 มีรายละเอียดดังนี้

จำนวนของข้อมูล(N)= 305,904

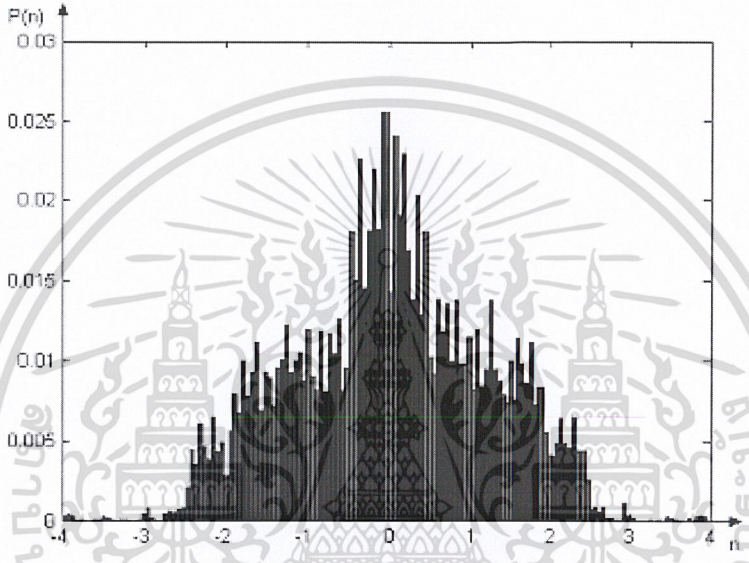
ค่าเบี่ยงเบนมาตรฐาน (σ) =1.1905

ค่าเฉลี่ย (μ)= 0.0144

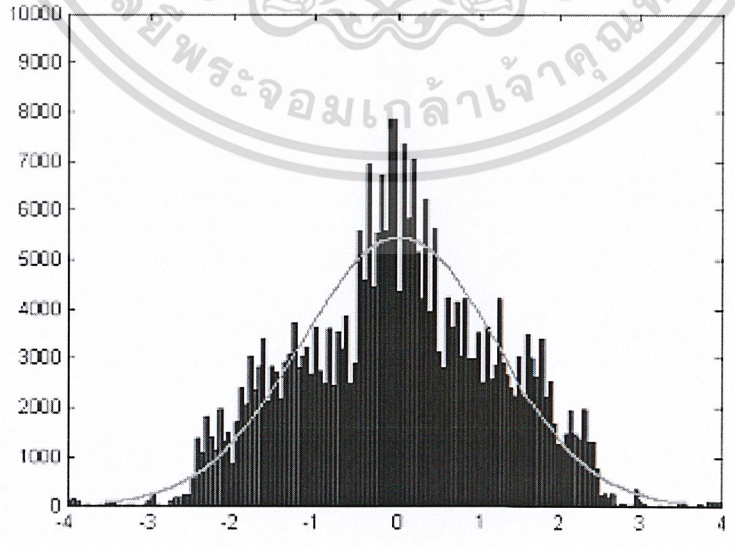
ค่าความแปรปรวน (σ^2) = 1.4174

ค่าสูงสุดของข้อมูล = 3.9844

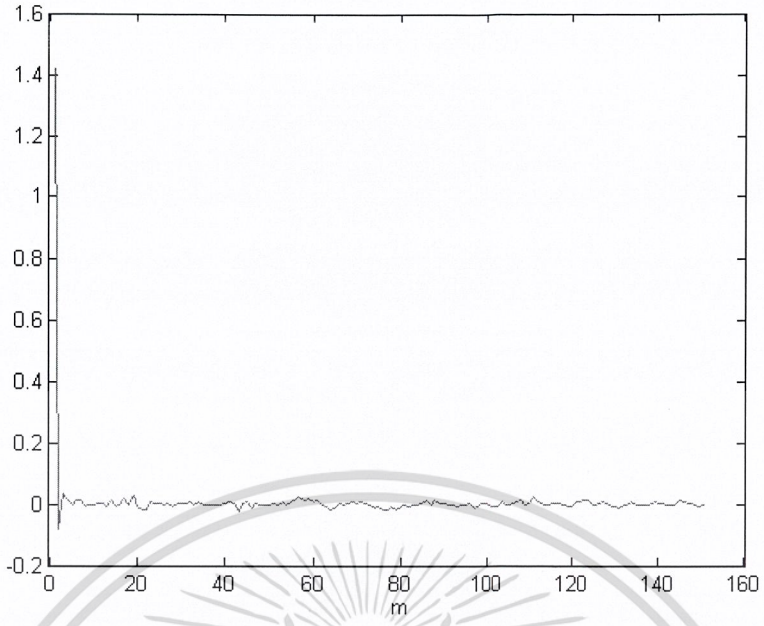
ค่าต่ำสุดของข้อมูล = -4



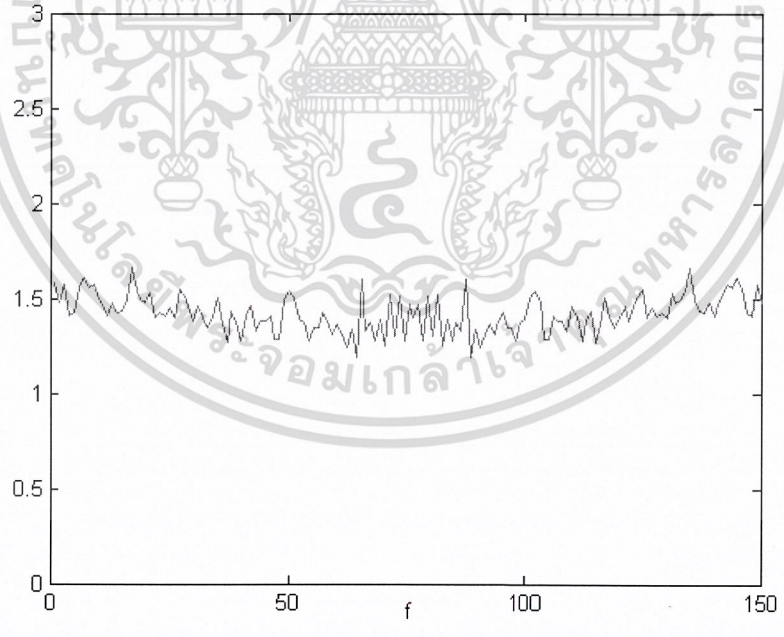
รูปที่4.48 แสดงค่าพีดีเอฟของ Box-Muller4



รูปที่ 4.49 ฮีสโตรแกรมของ Box-Muller4 เทียบกับเงื่อนไขเดียวกันตามทฤษฎี



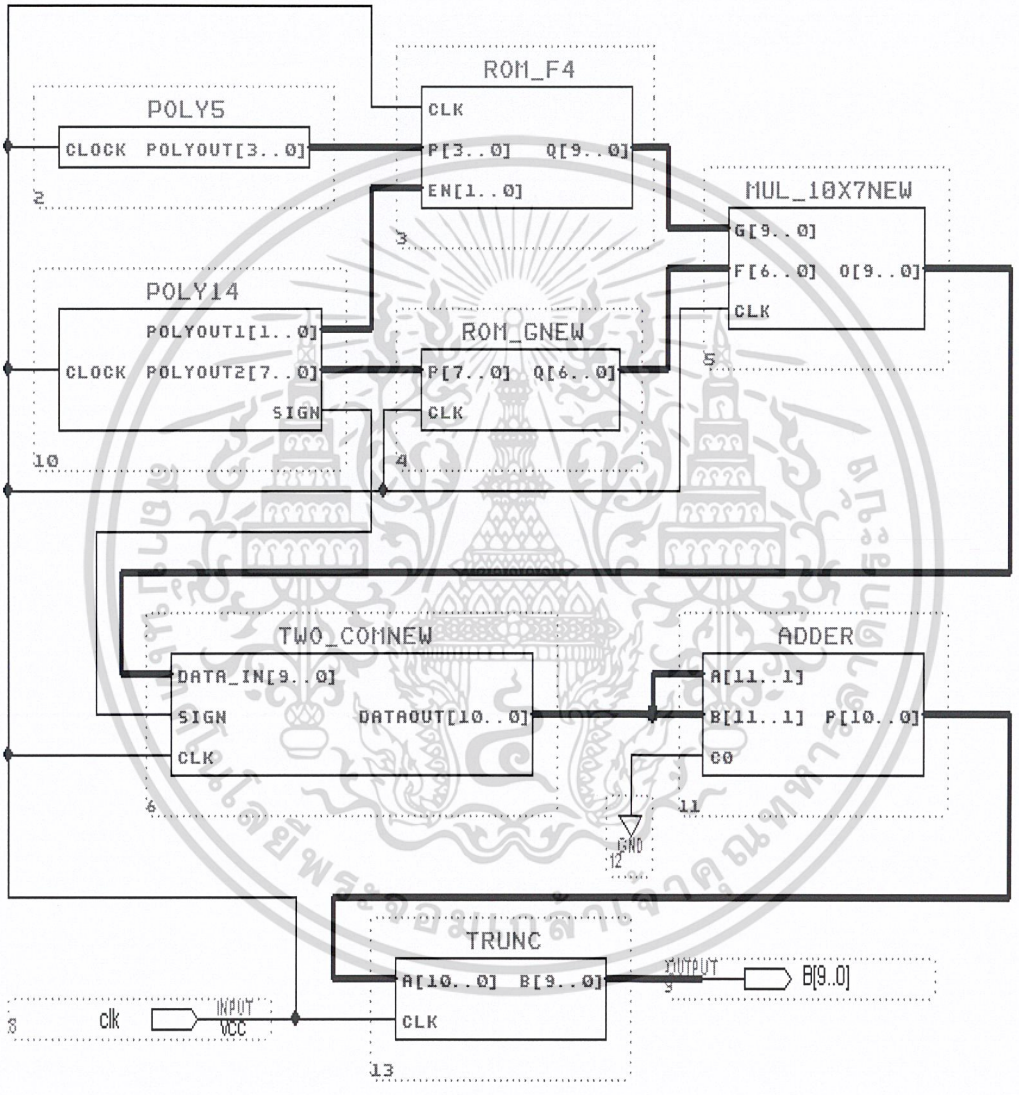
รูปที่ 4.50 แสดงกราฟออโต้คอรีเลชั่นของ Box-Muller4



รูปที่ 4.51 แสดงกราฟความหนาแน่นกำลังเชิงสเปกตรัมของ Box-Muller4

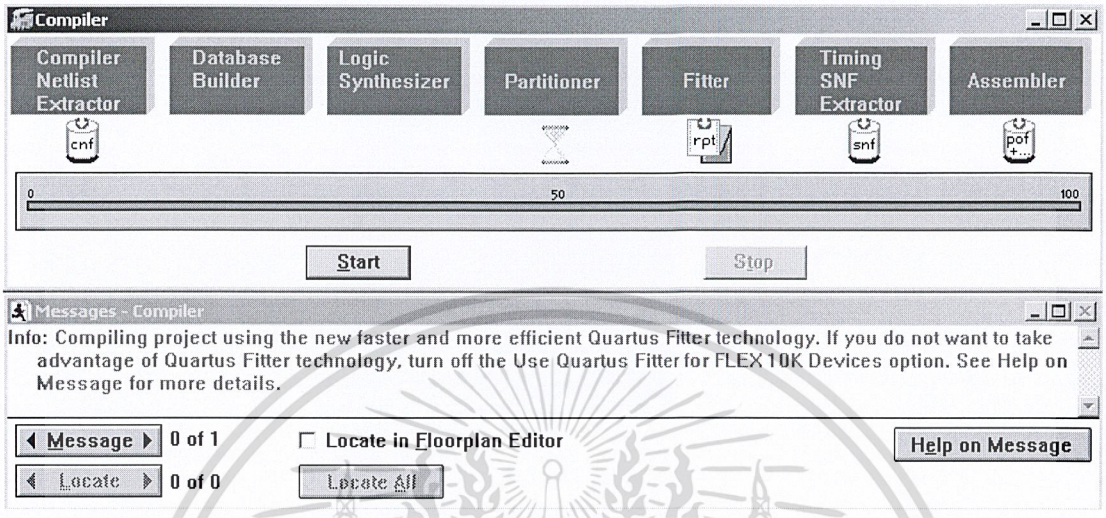
4.9 การเพิ่มแอคคิวมูเลเตอร์ลงใน Box-Muller3 และกราฟแสดงผล

วงจร Box-Muller3 ที่มี LFSRโพลิโนเมียล m=14 ใช้ Rom_f (x) 4 ตัว เมื่อนำแอคคิวมูเลเตอร์มาต่อร่วมด้วยจะมีผลลัพธ์ของข้อมูลที่นำมาคำนวณหาค่า พีดีเอฟ ฮีสโตรแกรม ค่าออโต้คอรีเลชั่นและความหนาแน่นกำลังเชิงสเปกตรัม จะสามารถแสดงได้ดังนี้



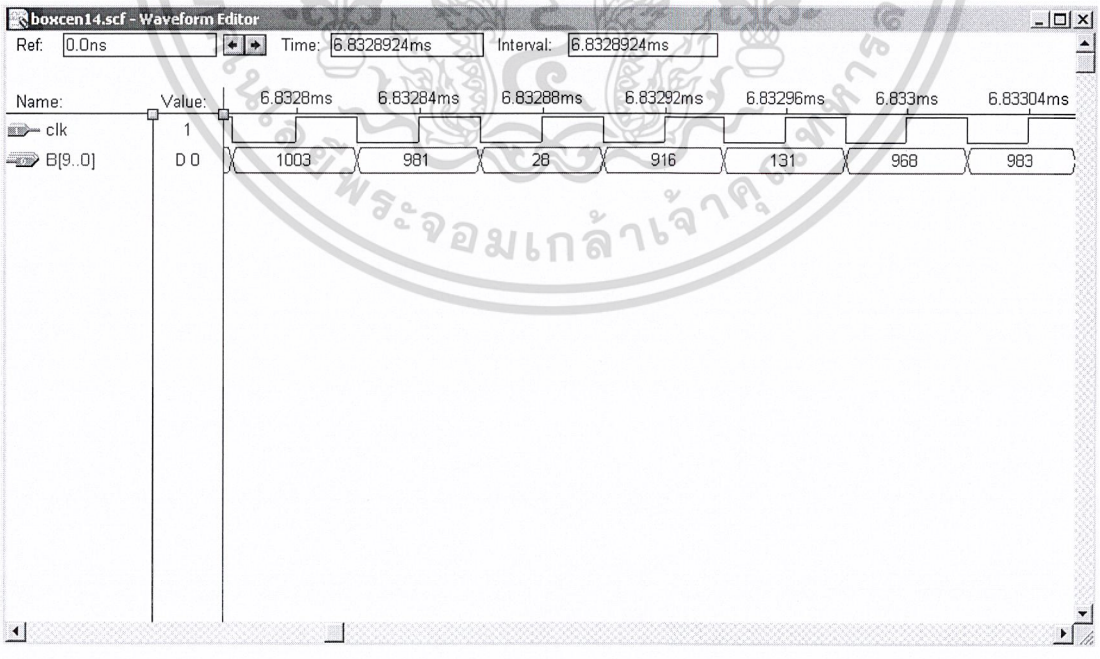
รูปที่ 4.52 แสดงสัญลักษณ์ของวงจร Box-Muller3 ที่ต่อร่วมกับแอคคิวมูเลเตอร์

แสดงการคอมไพล์วงจรที่ออกแบบขึ้นมาเพื่อให้ซอร์ฟแวร์ที่ใช้ในการสังเคราห์แสดงค่าอุปกรณ์ที่สามารถรองรับการทำงานและรายละเอียดต่างๆของสิ่งที่เราออกแบบขนาดของเกทที่เราใช้งานโดยแสดงออกมาเป็นเปอร์เซ็นต์การใช้งาน



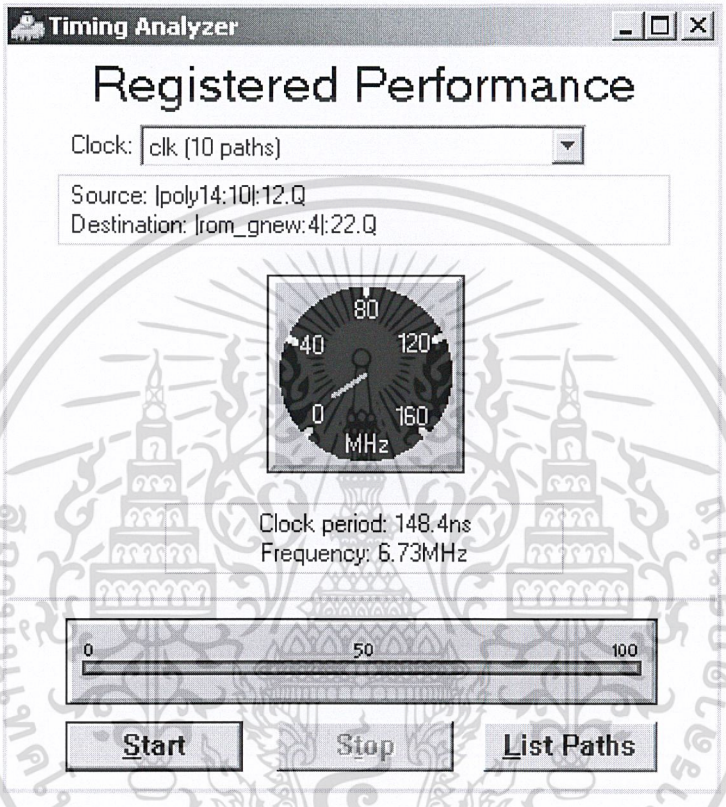
รูปที่ 4.53 แสดงการคอมไพล์ของ Box-Muller3 ที่ต่อร่วมกับแอคคิวมูเลเตอร์

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานและตรวจสอบผลการทำงานเพื่อให้ได้ตามเงื่อนไขการออกแบบวงจรสามารถแสดงได้ดังนี้



รูปที่ 4.54 แสดงการจำลองการทำงานของ Box-Muller3 ที่ต่อร่วมกับแอคคิวมูเลเตอร์

การสังเคราะห์วงจรที่เราออกแบบมานั้นจะมีข้อกำหนดทางด้านความถี่ที่นำมาใช้งานและระดับความเร็วของสัญญาณนาฬิกาซึ่งซอร์ฟแวร์ที่นำมาสังเคราะห์นั้นจะบอกรายระเอียดแสดงได้ดังรูปข้างล่างนี้



รูปที่ 4.55 แสดงค่าเวลาและความถี่ที่ใช้งานของ Box-Muller3 ที่ต่อร่วมกับแอคคิวมูเลเตอร์

ผลของการพรอตค่าจากวงจรที่รูป 4.52 มีรายละเอียดดังนี้

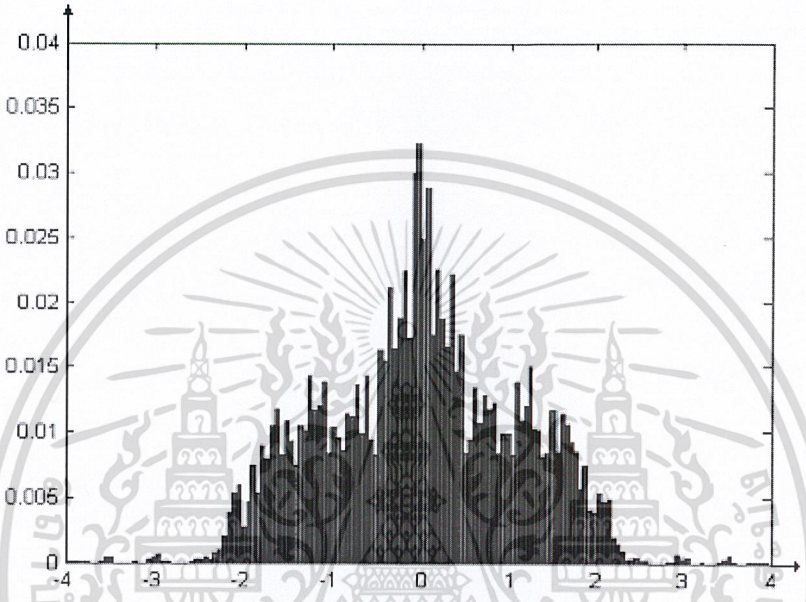
จำนวนของข้อมูล(N)= 563,118

ค่าเบี่ยงเบนมาตรฐาน (σ) =1.1211

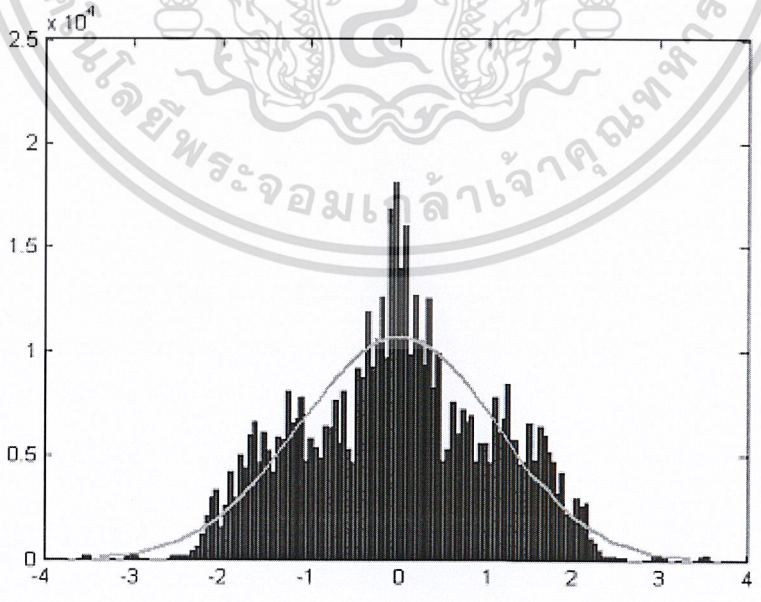
ค่าเฉลี่ย (μ)= -0.0036

ค่าความแปรปรวน (σ^2) = 1.2569

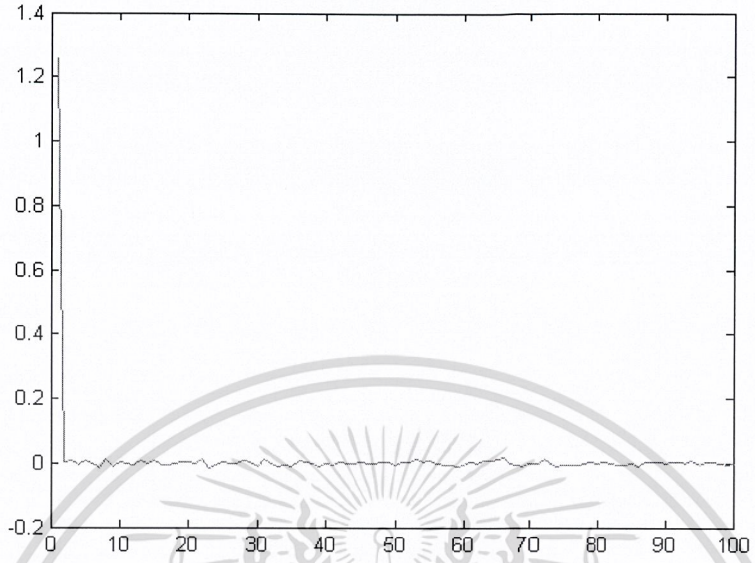
ค่าสูงสุดของข้อมูล = 3.9844 ค่าต่ำสุดของข้อมูล = -4



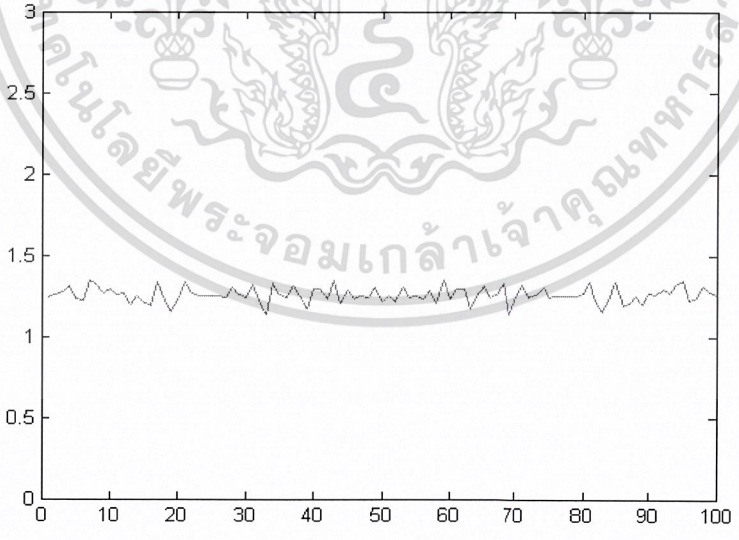
รูปที่4.56 แสดงค่าพีดีเอฟของ Box-Muller3 ที่ต่อร่วมกับแอคคิวมูเลเตอร์



รูปที่ 4.57 ฮีสโตรแกรมของ Box-Muller3 ที่ต่อร่วมกับแอคคิวมูเลเตอร์



รูปที่ 4.58 แสดงกราฟออโต้คอรีเลชั่นของ Box-Muller3 ที่ต่อร่วมกับแอคคิวมูเลเตอร์



รูปที่ 4.59 แสดงกราฟความหนาแน่นกำลังเชิงสเปกตรัมของ
Box-Muller3 ที่ต่อร่วมกับแอคคิวมูเลเตอร์

4.10 ค่าผิดพลาด(Error)

ค่าผลต่างที่เกิดจากจำนวนตัวแปรสุ่มที่เท่ากันระหว่างค่าที่ได้จาก Matlab และค่าจริงที่เกิดจากเอาท์พุตของFPGAs ในที่นี้ใช้การหา $P(X \leq 2)$ เป็นจุดอ้างอิงเพื่อหาค่าผิดพลาดที่เกิดขึ้นทางปฏิบัติเมื่อเปรียบเทียบกับค่าที่ฟังก์ชันของโปรแกรม Matlab สร้างขึ้น ซึ่งตัวแปรสุ่มและรายละเอียดต่างๆมีดังนี้

วิธีการคำนวณหาค่าผิดพลาด

คิดจากนิยามเมื่อตัวแปรสุ่ม X มีการแจกแจงปกติ ฟังก์ชันหนาแน่นความน่าจะเป็นของ X กำหนดโดย

$$F(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-(x-\mu)^2}{2\sigma^2}} \qquad ,-\infty<x<\infty$$

พิจารณาค่าจากตารางที่4.1แล้วหาค่า $P(X \leq 2)$

- **ค่าที่ได้จากการจำลองด้วย MATLAB**

  จะได้ฟังก์ชันความหนาแน่นเป็น

$$F(x)=\frac{1}{0.996\sqrt{2\pi}}e^{\frac{-(2-(-0.00093818))^2}{2(0.9996)^2}}$$

$\because P(X \leq 2)=F\left(\frac{x-\mu}{\sigma}\right)=F\left(\frac{2-(-0.00093818)}{0.9996}\right)=F(2.0017)$ นำค่าที่ได้ไปหาค่าจากตาราง $F(x)$ ที่ภาคผนวกจะได้ค่า

$F(2.0017)$ =0.977

- **ค่าที่ได้จากการทดลองเอาท์พุต FPGAs**

  จะได้ฟังก์ชันความหนาแน่นเป็น

$$F(x)=\frac{1}{1.1497\sqrt{2\pi}}e^{\frac{-(2-(-0.0112))^2}{2(1.1497)^2}}$$

$\because P(X \leq 2)=F\left(\frac{2-(-0.0112)}{1.1497}\right)=F(1.7493)$ นำค่าที่ได้ไปหาค่าจากตาราง

$F(1.7493)$ =0.9599

$\because$ จะได้ค่าผิดพลาดเท่ากับ 1.75%

จำนวนของข้อมูล(N)= 126,946	
ค่าจากเอาท์พุต FPGAs	**ค่าจาการจำลองด้วย MATLAB**
ค่าเบี่ยงเบนมาตรฐาน (σ) =1.1497	ค่าเบี่ยงเบนมาตรฐาน (σ) =0.9996
ค่าเฉลี่ย (μ)= -0.0112	ค่าเฉลี่ย (μ)= -0.00093818
ค่าความแปรปรวน (σ^2) = 1.3217	ค่าความแปรปรวน (σ^2) = 0.992016
ค่าสูงสุดของข้อมูล = 3.9844	ค่าสูงสุดของข้อมูล = 4.1002
ค่าต่ำสุดของข้อมูล = -4	ค่าต่ำสุดของข้อมูล = -5.1651
x=1.7493	x=2.0017
$F(x)$ =0.9599	$F(x)$ =0.977
ค่าผิดพลาด = 1.75%	

ตารางที่4.1 เมื่อใช้Box-Muller1 ที่มี LFSR โพลิโนเมียล m=12ใช้ Rom_f(x) 4ตัว

จำนวนของข้อมูล(N)= 126,946	
ค่าจากเอาท์พุต FPGAs	**ค่าจาการจำลองด้วย MATLAB**
ค่าเบี่ยงเบนมาตรฐาน (σ) =1.2626	ค่าเบี่ยงเบนมาตรฐาน (σ) =0.9996
ค่าเฉลี่ย (μ)= -0.0279	ค่าเฉลี่ย (μ)= -0.00093818
ค่าความแปรปรวน (σ^2) = 1.59415876	ค่าความแปรปรวน (σ^2) = 0.992016
ค่าสูงสุดของข้อมูล = 3.9844	ค่าสูงสุดของข้อมูล = 4.1002
ค่าต่ำสุดของข้อมูล = -4	ค่าต่ำสุดของข้อมูล = -5.1651
x=1.6061	x=2.0017
$F(x)$ =0.9459	$F(x)$ =0.977
ค่าผิดพลาด =3.18%	

ตารางที่4.2 เมื่อใช้Box-Muller2 ที่มี LFSR โพลิโนเมียล m=12ใช้ Rom_f(x) 5ตัว

จำนวนของข้อมูล(N)= 305,904	
ค่าจากเอาท์พุต FPGAs	**ค่าจาการจำลองด้วย MATLAB**
ค่าเบี่ยงเบนมาตรฐาน (σ) =1.1218	ค่าเบี่ยงเบนมาตรฐาน (σ) =1.0014
ค่าเฉลี่ย (μ)= - 0.0042	ค่าเฉลี่ย (μ)= 0.0029
ค่าความแปรปรวน (σ^2) = 1.2584	ค่าความแปรปรวน (σ^2) = 0.9973
ค่าสูงสุดของข้อมูล = 3.9844	ค่าสูงสุดของข้อมูล = 4.3094
ค่าต่ำสุดของข้อมูล = -4	ค่าต่ำสุดของข้อมูล = -4.1150
x=1.7866	x=2.0069
$F(x)$ =0.963	$F(x)$ =0.9776
ค่าผิดพลาด =1.49%	

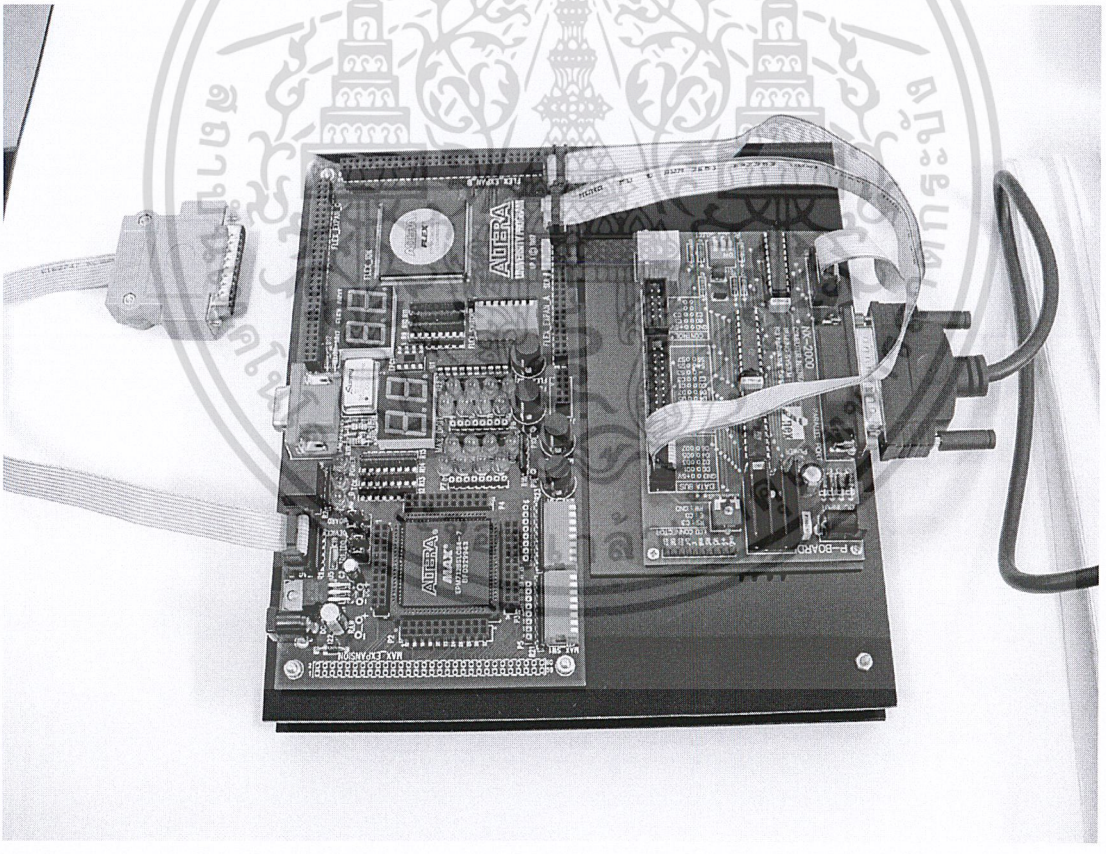
ตารางที่4.3 เมื่อใช้Box-Muller3 ที่มี LFSR โพลิโนเมียล m=14ใช้ Rom_f(x) 4ตัว

จำนวนของข้อมูล(N)= 305,904	
ค่าจากเอาท์พุต FPGAs	**ค่าจาการจำลองด้วย MATLAB**
ค่าเบี่ยงเบนมาตรฐาน (σ) =1.1905	ค่าเบี่ยงเบนมาตรฐาน (σ) =1.0014
ค่าเฉลี่ย (μ)= 0.0144	ค่าเฉลี่ย (μ)= 0.0029
ค่าความแปรปรวน (σ^2) = 1.4174	ค่าความแปรปรวน (σ^2) = 0.9973
ค่าสูงสุดของข้อมูล = 3.9844	ค่าสูงสุดของข้อมูล = 4.3094
ค่าต่ำสุดของข้อมูล = -4	ค่าต่ำสุดของข้อมูล = -4.1150
x=1.6679	x=2.0069
$F(x)$ =0.9523	$F(x)$ =0.9776
ค่าผิดพลาด =2.59%	

ตารางที่4.4 เมื่อใช้Box-Muller4 ที่มี LFSR โพลิโนเมียล m=14ใช้ Rom_f(x) 5ตัว

จำนวนของข้อมูล(N)= 563,118	
ค่าจากเอาท์พุต FPGAs	**ค่าจากการจำลองด้วย MATLAB**
ค่าเบี่ยงเบนมาตรฐาน (σ) =1.1211	ค่าเบี่ยงเบนมาตรฐาน (σ) =0.9996
ค่าเฉลี่ย (μ)= -0.0036	ค่าเฉลี่ย (μ)= 0.0014
ค่าความแปรปรวน (σ^2) = 1.2569	ค่าความแปรปรวน (σ^2) = 0.9991
ค่าสูงสุดของข้อมูล = 3.9844	ค่าสูงสุดของข้อมูล = 4.3094
ค่าต่ำสุดของข้อมูล = -4	ค่าต่ำสุดของข้อมูล = -4.1150
x=1.7871	x=1.9994
$F(x)$ =0.9631	$F(x)$ =0.996
ค่าผิดพลาด =3.30%	

ตารางที่4.5 แสดงค่าผิดพลาดของ Box-Muller3 ที่ต่อกับแอคคิวมูเลเตอร์



รูปที่ 4.60 แสดงบอร์ด FPGAs

บทที่ 5
บทวิจารณ์และบทสรุป

จากการออกแบบ เครื่องจำลองช่องสัญญาณรบกวนแบบ AWGN (AWGN-Channel Emulation) โดยการเขียนโปรแกรมด้วยภาษา VHDL และทำการ Implement ลงบนอุปกรณ์บอร์ด FPGAs ซึ่งผลของข้อมูลที่ได้จากเอาท์พุตเมื่อเรานำค่ามาคำนวณโดยโปรแกรม MATLAB จะได้กราฟแสดงผลลัพธ์ออกมาในระดับที่น่าพอใจถึงจะมีค่า error อยู่บ้างแต่นั่นเกิดจากข้อจำกัดของอุปกรณ์ FPGAs รุ่นที่นำมาใช้งานมีความจุของเกทและระดับความเร็วในการทำงานต่ำซึ่งถือว่าเป็นรุ่นที่อยู่ในระดับที่นำมาใช้เพื่อการศึกษาหรือทดลอง ยังไม่เหมาะที่จะนำไปใช้งานจริงในทางปฏิบัติจริง

ปัญหาของการทำงานครั้งนี้ส่วนมากเกิดจากปัญหาทางด้านข้อมูลทางเทคนิคของอุปกรณ์ FPGAs เนื่องจากเป็นอุปกรณ์ที่ถือว่าใหม่สำหรับประเทศไทยทำให้การหาเอกสารอ้างอิงที่ทำความเข้าใจนั้นหายากมาก และโปรแกรมภาษาที่เรานำมาเขียนก็เป็นภาษาใหม่ที่ถูกพัฒนาขึ้นมาใช้งานมาไม่นานนี้ทำให้เราหาข้อมูลเพื่อใช้ในการศึกษาทำความเข้าใจยากเช่นกัน แต่ก็สามารถทำให้สำเร็จลุล่วงไปด้วยดี ซึ่งทางผู้จัดทำก็หวังเป็นอย่างยิ่งว่างานชิ้นนี้จะเป็นแนวคิดในการพัฒนางานชิ้นอื่นที่มีคุณภาพสูงกว่านี้และมีประสิทธิภาพดียิ่งๆขึ้นไป

ภาคผนวก

ตาราง Cumulative Normal Distribution

$$F(x) = \int_{-\infty}^{X} \frac{1}{2\sqrt{\pi}} e^{-\frac{t^2}{2}} \, dt$$

x	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
.0	.5000	.5040	.5080	.5020	.5160	.5199	.5239	.5279	.5319	.5359
.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767
2.0	.9772	.9778	.9783	.9788	.9793	.9798	9803	.9808	.9812	.9817
2.1	.9821	.9826	.9830	.9834	.9838	.9842	.9846	.9850	.9854	.9857
2.2	.9861	.9864	.9868	.9871	.9875	.9878	.9881	.9884	.9887	.9890
2.3	.9893	.9896	.9898	.9901	.9904	.9906	.9909	9911	.9913	.9916
2.4	.9918	.9920	.9922	.9925	.9927	.9929	.9931	.9932	.9934	.9936
2.5	.9938	.9940	.9941	.9943	.9944	.9946	.9948	.9949	.9951	.9952
2.6	.9953	.9955	.9956	.9957	.9959	.9960	.9961	.9962	.9963	.9964
2.7	.9965	.9966	.9967	.9968	.9969	.9970	.9971	.9972	.9973	.9974
2.8	.9974	.9975	.9976	.9977	.9977	.9978	.9979	.9979	.9980	.9981
2.9	.9981	.9982	.9982	.9983	.9984	.9984	.9985	.9985	.9986	.9986
3.0	.9987	.9987	.9987	.9988	.9988	.9989	.9989	.9989	.9990	.9990
3.1	.9990	.9991	.9991	.9991	.9992	.9992	.9992	.9992	.9993	.9993
3.2	.9993	.9993	.9994	.9994	.9994	.9994	.9994	.9995	.9995	.9995
3.3	.9995	.9995	.9995	.9996	.9996	.9996	.9996	.9996	.9996	.9997
3.4	.9997	.9997	.9997	.9997	.9997	.9997	.9997	.9997	.9997	.9998

x	1.282	1.645	1.960	2.326	2.576	3.090	3.291	3.891	4.417
F(x)	.90	.95	.975	.99	.995	.999	.9995	.99995	.999995
2[1-F(x)]	.20	.10	.05	.02	.01	.002	.001	.0001	.00001

```
-------------------%%% LFSR Polynomial m = 5 %%%--------------------

Library ieee;
Use ieee.std_logic_1164.all ;

--------------------------------------------------------------------------
Entity Poly5 is
Port(
              Clock    : in std_logic;
              Polyout : out std_logic_vector( 3 downto 0 )
      );
end Poly5;
--------------------------------------------------------------------------

Architecture APoly5 of Poly5 is

Signal Poly : std_logic_vector( 4 downto 0);

Begin
      Process(Clock)
      Begin
         if( Clock'event and Clock = '1' ) then
           if( Poly( 4 downto 0) =('0','0','0',
             '0','0')) then
                 Poly(0) <= '1';
           else
                 Poly( 3 downto 0) <= Poly( 4 downto 1);
                 Poly(4) <=  (Poly(3) xor Poly(0)) ;
                 Polyout <=   Poly( 4 downto 1 );

           end if;
         end if;
      end process;
end APoly5;

--------------------------------------------------------------------------
```

```
--------------%%%% LFSR Polynomial m = 12 %%%%-----------------------

library ieee;
use ieee.std_logic_1164.all;

---------------------------------------------------------------------


entity Poly12s4 is
port(
            Clock     : in bit;
            Polyout1  : out std_logic_vector( 1 downto 0 );
            Polyout2  : out std_logic_vector( 7 downto 0 );
            sign      : out std_logic
      );
end Poly12s4;

---------------------------------------------------------------------


architecture APoly12s of Poly12s4 is

signal Poly : std_logic_vector( 11 downto 0);

begin
process(Clock)
      begin
      if( Clock'event and Clock = '1' ) then
            if( Poly( 11 downto 0) =('0','0','0',
            '0','0','0','0','0','0','0','0','0')) then
                  Poly(0) <= '1';
            else
                  Poly( 10 downto 0) <= Poly( 11 downto 1);
                  Poly(11) <= (Poly(6) xor Poly(4)) xor
                              (Poly(1) xor Poly(0));
                  Polyout1 <= Poly( 10 downto 9 );
                  Polyout2 <= Poly( 8 downto 1 );
                  sign     <= Poly(0);
            end if;

      end if;
 end process;
end APoly12s;

---------------------------------------------------------------------
```

```
--------------------------%% ROM_f(x) %%--------------------------
Library ieee;
Use ieee.std_logic_1164.all;
-------------------------------------------------------------------------
Entity rom_f4 is
Port(
            clk : in std_logic;
            p   : in std_logic_vector(3 downto 0);
            q   : out std_logic_vector(9 downto 0);
            en  : in std_logic_vector( 1 downto 0)

      );
End rom_f4;
-------------------------------------------------------------------------
Architecture ROM2ent of rom_f4 is
Begin
-------------------%%%%Developed in 25/10/2002%%%%--------------

 Process (en,p,clk)
 Begin
  if( clk'event and clk = '1') then
   case en is
   when "00" => if          p = "0000" then q <= "0011110000";
                      elsif p = "0001" then q <= "0011000101";
                      elsif p = "0010" then q <= "0010101111";
                      elsif p = "0011" then q <= "0010011110";
                      elsif p = "0100" then q <= "0010010000";
                      elsif p = "0101" then q <= "0010000100";
                      elsif p = "0110" then q <= "0001111001";
                      elsif p = "0111" then q <= "0001101111";
                      elsif p = "1000" then q <= "0001100110";
                      elsif p = "1001" then q <= "0001011100";
                      elsif p = "1010" then q <= "0001010011";
                      elsif p = "1011" then q <= "0001001001";
                      elsif p = "1100" then q <= "0000111111";
                      elsif p = "1101" then q <= "0000110101";
                      elsif p = "1110" then q <= "0000101000";
                      elsif p = "1111" then q <= "0000010111";
                end if;

   when "01" => if          p = "0000" then q <= "0101000001";
                      elsif p = "0001" then q <= "0100100010";
                      elsif p = "0010" then q <= "0100010011";
                      elsif p = "0011" then q <= "0100001001";
                      elsif p = "0100" then q <= "0100000001";
                      elsif p = "0101" then q <= "0011111011";
                      elsif p = "0110" then q <= "0011110101";
                      elsif p = "0111" then q <= "0011110000";
                      elsif p = "1000" then q <= "0011101100";
                      elsif p = "1001" then q <= "0011101000";
                      elsif p = "1010" then q <= "0011100100";
                      elsif p = "1011" then q <= "0011100001";
                      elsif p = "1100" then q <= "0011011110";
                      elsif p = "1101" then q <= "0011011011";
                      elsif p = "1110" then q <= "0011011000";
                      elsif p = "1111" then q <= "0011010110";
                end if;
```

```
    when "10" => if              p = "0000" then q <= "0110000000";
                          elsif p = "0001" then q <= "0101101000";
                          elsif p = "0010" then q <= "0101011100";
                          elsif p = "0011" then q <= "0101010100";
                          elsif p = "0100" then q <= "0101001110";
                          elsif p = "0101" then q <= "0101001001";
                          elsif p = "0110" then q <= "0101000101";
                          elsif p = "0111" then q <= "0101000001";
                          elsif p = "1000" then q <= "0100111110";
                          elsif p = "1001" then q <= "0100111011";
                          elsif p = "1010" then q <= "0100111000";
                          elsif p = "1011" then q <= "0100110110";
                          elsif p = "1100" then q <= "0100110100";
                          elsif p = "1101" then q <= "0100110010";
                          elsif p = "1110" then q <= "0100110000";
                          elsif p = "1111" then q <= "0100101110";
                      end if;

    when "11" => if              p = "0000" then q <= "0110111000";
                          elsif p = "0001" then q <= "0110100010";
                          elsif p = "0010" then q <= "0110011000";
                          elsif p = "0011" then q <= "0110010001";
                          elsif p = "0100" then q <= "0110001100";
                          elsif p = "0101" then q <= "0110001000";
                          elsif p = "0110" then q <= "0110000100";
                          elsif p = "0111" then q <= "0110000001";
                          elsif p = "1000" then q <= "0101111111";
                          elsif p = "1001" then q <= "0101111100";
                          elsif p = "1010" then q <= "0101111010";
                          elsif p = "1011" then q <= "0101111000";
                          elsif p = "1100" then q <= "0101110110";
                          elsif p = "1101" then q <= "0101110100";
                          elsif p = "1110" then q <= "0101110011";
                          elsif p = "1111" then q <= "0101110001";
                      end if;
      when others => q <= "0000000000";
     end case;
    end if;
  end process;
end ROM2ent;
```

```
--------------------------%% ROM_g(x) %%-----------------------
library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------------
entity rom_gNEW is
port(
            p   : in std_logic_vector(7 downto 0);
            clk : in std_logic;
            q   : out std_logic_vector(6 downto 0));
end rom_gNEW;

-------------------------------------------------
architecture ROM_G of rom_gNEW is
begin
      process(p,clk)
      begin
            if( clk'event and clk = '1') then
            case p is
                  when "00000000" => q <= "1011011";        --add 0
                  when "00000001" => q <= "1011011";        --add 1
                  when "00000010" => q <= "1011011";
                  when "00000011" => q <= "1011011";
                  when "00000100" => q <= "1011011";
                  when "00000101" => q <= "1011011";        --add 5
                  when "00000110" => q <= "1011011";
                  when "00000111" => q <= "1011011";
                  when "00001000" => q <= "1011011";
                  when "00001001" => q <= "1011011";
                  when "00001010" => q <= "1011011";        --add 10
                  when "00001011" => q <= "1011010";
                  when "00001100" => q <= "1011010";
                  when "00001101" => q <= "1011010";
                  when "00001110" => q <= "1011010";
                  when "00001111" => q <= "1011010";        --add 15
                  when "00010000" => q <= "1011010";
                  when "00010001" => q <= "1011010";
                  when "00010010" => q <= "1011010";
                  when "00010011" => q <= "1011010";
                  when "00010100" => q <= "1011010";        --add 20
                  when "00010101" => q <= "1011001";
                  when "00010110" => q <= "1011001";
                  when "00010111" => q <= "1011001";
                  when "00011000" => q <= "1011001";
                  when "00011001" => q <= "1011001";        --add 25
                  when "00011010" => q <= "1011001";
                  when "00011011" => q <= "1011001";
                  when "00011100" => q <= "1011001";
                  when "00011101" => q <= "1011001";
                  when "00011110" => q <= "1011001";        --add 30
                  when "00011111" => q <= "1011001";
                  when "00100000" => q <= "1011000";
                  when "00100001" => q <= "1011000";
                  when "00100010" => q <= "1011000";
                  when "00100011" => q <= "1011000";        --add 35
                  when "00100100" => q <= "1011000";
                  when "00100101" => q <= "1011000";
                  when "00100110" => q <= "1011000";
                  when "00100111" => q <= "1011000";
                  when "00101000" => q <= "1010111";        --add 40
```

```
when "00101001" => q <= "1010111";
when "00101010" => q <= "1010111";
when "00101011" => q <= "1010111";
when "00101100" => q <= "1010111";
when "00101101" => q <= "1010111";          --add 45
when "00101110" => q <= "1010111";
when "00101111" => q <= "1010110";
when "00110000" => q <= "1010110";
when "00110001" => q <= "1010110";
when "00110010" => q <= "1010110";          --add 50
when "00110011" => q <= "1010110";
when "00110100" => q <= "1010110";
when "00110101" => q <= "1010101";
when "00110110" => q <= "1010101";
when "00110111" => q <= "1010101";          --add 55
when "00111000" => q <= "1010101";
when "00111001" => q <= "1010101";
when "00111010" => q <= "1010100";
when "00111011" => q <= "1010100";
when "00111100" => q <= "1010100";          --add 60
when "00111101" => q <= "1010100";
when "00111110" => q <= "1010100";
when "00111111" => q <= "1010011";
when "01000000" => q <= "1010011";
when "01000001" => q <= "1010011";          --add 65
when "01000010" => q <= "1010011";
when "01000011" => q <= "1010011";
when "01000100" => q <= "1010010";
when "01000101" => q <= "1010010";
when "01000110" => q <= "1010010";          --add 70
when "01000111" => q <= "1010010";
when "01001000" => q <= "1010001";
when "01001001" => q <= "1010001";
when "01001010" => q <= "1010001";
when "01001011" => q <= "1010001";          --add 75
when "01001100" => q <= "1010000";
when "01001101" => q <= "1010000";
when "01001110" => q <= "1010000";
when "01001111" => q <= "1010000";
when "01010000" => q <= "1001111";          --add 80
when "01010001" => q <= "1001111";
when "01010010" => q <= "1001111";
when "01010011" => q <= "1001111";
when "01010100" => q <= "1001110";
when "01010101" => q <= "1001110";          --add 85
when "01010110" => q <= "1001110";
when "01010111" => q <= "1001110";
when "01011000" => q <= "1001101";
when "01011001" => q <= "1001101";
when "01011010" => q <= "1001101";          --add 90
when "01011011" => q <= "1001100";
when "01011100" => q <= "1001100";
when "01011101" => q <= "1001100";
when "01011110" => q <= "1001011";
when "01011111" => q <= "1001011";          --add 95
when "01100000" => q <= "1001011";
when "01100001" => q <= "1001011";
when "01100010" => q <= "1001010";
when "01100011" => q <= "1001010";
when "01100100" => q <= "1001010";          --add 100
when "01100101" => q <= "1001001";
```

```
when "01100110" => q <= "1001001";
when "01100111" => q <= "1001001";
when "01101000" => q <= "1001000";
when "01101001" => q <= "1001000";          --add 105
when "01101010" => q <= "1001000";
when "01101011" => q <= "1000111";
when "01101100" => q <= "1000111";
when "01101101" => q <= "1000111";
when "01101110" => q <= "1000110";          --add 110
when "01101111" => q <= "1000110";
when "01110000" => q <= "1000110";
when "01110001" => q <= "1000101";
when "01110010" => q <= "1000101";
when "01110011" => q <= "1000100";          --add 115
when "01110100" => q <= "1000100";
when "01110101" => q <= "1000100";
when "01110110" => q <= "1000011";
when "01110111" => q <= "1000011";
when "01111000" => q <= "1000011";          --add 120
when "01111001" => q <= "1000010";
when "01111010" => q <= "1000010";
when "01111011" => q <= "1000001";
when "01111100" => q <= "1000001";
when "01111101" => q <= "1000001";          --add 125
when "01111110" => q <= "1000000";
when "01111111" => q <= "1000000";
when "10000000" => q <= "1000000";
when "10000001" => q <= "0111111";
when "10000010" => q <= "0111111";          --add 130
when "10000011" => q <= "0111110";
when "10000100" => q <= "0111101";
when "10000101" => q <= "0111101";
when "10000110" => q <= "0111101";
when "10000111" => q <= "0111101";          --add 135
when "10001000" => q <= "0111100";
when "10001001" => q <= "0111100";
when "10001010" => q <= "0111011";
when "10001011" => q <= "0111011";
when "10001100" => q <= "0111011";          --add 140
when "10001101" => q <= "0111010";
when "10001110" => q <= "0111010";
when "10001111" => q <= "0111001";
when "10010000" => q <= "0111001";
when "10010001" => q <= "0111001";          --add 145
when "10010010" => q <= "0111000";
when "10010011" => q <= "0111000";
when "10010100" => q <= "0110111";
when "10010101" => q <= "0110111";
when "10010110" => q <= "0110110";          --add 150
when "10010111" => q <= "0110110";
when "10011000" => q <= "0110101";
when "10011001" => q <= "0110101";
when "10011010" => q <= "0110101";
when "10011011" => q <= "0110100";          --add 155
when "10011100" => q <= "0110100";
when "10011101" => q <= "0110011";
when "10011110" => q <= "0110011";
when "10011111" => q <= "0110010";
when "10100000" => q <= "0110010";          --add 160
when "10100001" => q <= "0110001";
when "10100010" => q <= "0110001";
```

```
when "10100011" => q <= "0110000";
when "10100100" => q <= "0110000";
when "10100101" => q <= "0101111";          --add 165
when "10100110" => q <= "0101111";
when "10100111" => q <= "0101111";
when "10101000" => q <= "0101110";
when "10101001" => q <= "0101110";
when "10101010" => q <= "0101101";        --add 170
when "10101011" => q <= "0101101";
when "10101100" => q <= "0101100";
when "10101101" => q <= "0101100";
when "10101110" => q <= "0101011";
when "10101111" => q <= "0101011";          --add 175
when "10110000" => q <= "0101010";
when "10110001" => q <= "0101010";
when "10110010" => q <= "0101001";
when "10110011" => q <= "0101001";
when "10110100" => q <= "0101000";          --add 180
when "10110101" => q <= "0101000";
when "10110110" => q <= "0100111";
when "10110111" => q <= "0100111";
when "10111000" => q <= "0100110";
when "10111001" => q <= "0100110";          --add 185
when "10111010" => q <= "0100101";
when "10111011" => q <= "0100101";
when "10111100" => q <= "0100100";
when "10111101" => q <= "0100100";
when "10111110" => q <= "0100011";          --add 190
when "10111111" => q <= "0100011";
when "11000000" => q <= "0100010";
when "11000001" => q <= "0100001";
when "11000010" => q <= "0100001";
when "11000011" => q <= "0100001";          --add 195
when "11000100" => q <= "0100000";
when "11000101" => q <= "0100000";
when "11000110" => q <= "0011111";
when "11000111" => q <= "0011110";
when "11001000" => q <= "0011110";          --add 200
when "11001001" => q <= "0011101";
when "11001010" => q <= "0011101";
when "11001011" => q <= "0011100";
when "11001100" => q <= "0011100";
when "11001101" => q <= "0011011";          --add 205
when "11001110" => q <= "0011011";
when "11001111" => q <= "0011010";
when "11010000" => q <= "0011010";
when "11010001" => q <= "0011001";
when "11010010" => q <= "0011001";          --add 210
when "11010011" => q <= "0011000";
when "11010100" => q <= "0011000";
when "11010101" => q <= "0010111";
when "11010110" => q <= "0010111";
when "11010111" => q <= "0010110";          --add 215
when "11011000" => q <= "0010101";
when "11011001" => q <= "0010101";
when "11011010" => q <= "0010100";
when "11011011" => q <= "0010100";
when "11011100" => q <= "0010011";          --add 220
when "11011101" => q <= "0010011";
when "11011110" => q <= "0010010";
when "11011111" => q <= "0010010";
```

```
                when "11100000" => q <= "0010001";
                when "11100001" => q <= "0010001";          --add 225
                when "11100010" => q <= "0010000";
                when "11100011" => q <= "0001111";
                when "11100100" => q <= "0001111";
                when "11100101" => q <= "0001110";
                when "11100110" => q <= "0001110";          --add 230
                when "11100111" => q <= "0001101";
                when "11101000" => q <= "0001101";
                when "11101001" => q <= "0001100";
                when "11101010" => q <= "0001100";
                when "11101011" => q <= "0001011";          --add 235
                when "11101100" => q <= "0001011";
                when "11101101" => q <= "0001010";
                when "11101110" => q <= "0001001";
                when "11101111" => q <= "0001001";
                when "11110000" => q <= "0001000";          --add 240
                when "11110001" => q <= "0001000";
                when "11110010" => q <= "0000111";
                when "11110011" => q <= "0000111";
                when "11110100" => q <= "0000110";
                when "11110101" => q <= "0000110";          --add 245
                when "11110110" => q <= "0000101";
                when "11110111" => q <= "0000101";
                when "11111000" => q <= "0000100";
                when "11111001" => q <= "0000100";
                when "11111010" => q <= "0000011";          --add 250
                when "11111011" => q <= "0000011";
                when "11111100" => q <= "0000010";
                when "11111101" => q <= "0000001";
                when "11111110" => q <= "0000001";
                when "11111111" => q <= "0000000";          --add 255
                when others => q <= "0000000";
            end case;
        end if;
    end process;
end ROM_G;
```

```
----------%%%  MULTIPLIER %%%----------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

--------------------------------------------------

entity mul_10x7NEW is
      port (
                  G   : in UNSIGNED (9 downto 0);
                  F   : in UNSIGNED (6 downto 0);
                  clk : in STD_LOGIC;
                  O   : out UNSIGNED (9 downto 0)
            );
end mul_10x7NEW;
--------------------------------------------------

architecture test of mul_10x7NEW is
      signal tmp : unsigned( 16 downto 0 );
begin
      process
      begin

            wait until ( rising_edge(clk));
                  tmp <= G*F;

      end process;
      O <= tmp( 16 downto 7 );
end test;

--------------------------------------------------
```

```
------------------------%% TWO COMPLEMENT %%--------------------------

Library IEEE;
Use IEEE.std_logic_1164.all;

-----------------------------------------------------------------------

Entity Two_comNEW is
      Port(
            Data_in : in Std_logic_vector (9 downto 0);
            Dataout : out Std_logic_vector(10 downto 0);
            sign    : in bit;
            clk     : in std_logic
            );
End Two_comNEW;

-----------------------------------------------------------------------

Architecture Behave of Two_comNEW is
begin
      process(Data_in,sign,clk)

      variable one_com : Std_logic_vector (9 downto 0);
      variable buff    : Std_logic_vector (10 downto 0);

      begin
            if( clk'event and clk = '1') then
              if ( sign ='1') then
                one_com := Data_in xor "1111111111";
                if one_com(0)='0' then
                      buff:='1' & one_com(9 downto 1) & '1';
                      Dataout <= buff;
                elsif one_com(1)='0' then
                      buff:='1' & one_com(9 downto 2) & "10";
                      Dataout <= buff;
                elsif one_com(2)='0' then
                      buff:='1' & one_com(9 downto 3) & "100";
                      Dataout <= buff;
                elsif one_com(3)='0' then
                      buff:='1' & one_com(9 downto 4) & "1000";
                      Dataout <= buff;
                elsif one_com(4)='0' then
                      buff:='1' & one_com(9 downto 5) & "10000";
                      Dataout <= buff;
                elsif one_com(5)='0' then
                      buff:='1' & one_com(9 downto 6) & "100000";
                      Dataout <= buff;
                elsif one_com(6)='0' then
                      buff:='1' & one_com(9 downto 7) & "1000000";
                      Dataout <= buff;
                elsif one_com(7)='0' then
                      buff:='1' & one_com(9 downto 8) & "10000000";
                      Dataout <= buff;
                elsif one_com(8)='0' then
                      buff:='1' & one_com(9) & "100000000";
                      Dataout <= buff;
                elsif one_com(9)='0' then
                      buff:='1' & "1000000000";
                      Dataout <= buff;
```

```
                    else
                          Dataout<="10000000000";
                    end if;
                elsif ( sign ='0') then
                      Dataout<='0' & Data_in;
            end if;
          end if;
      end process;
end Run;
```

---------------------%%% TRUNCATION %%%---------------------

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity trunc is
port(
          a   : in std_logic_vector( 10 downto 0 );
          clk : in std_logic;
          b   : out std_logic_vector( 9 downto 0 ));
end trunc;
```

```
architecture behave of trunc is
begin
      process( clk,a )
      begin
            if ( clk'event and clk = '1') then
                b <= a( 10 downto 1 );
            end if ;
      end process;
end behave;
```

```
-----------------------------%% ADDER %%-------------------------------
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------------------------------

entity adder is
port(
            a,b : in std_logic_vector( 11 downto 1);
            c0  : in std_logic;
            p   : out std_logic_vector( 11 downto 0));
end adder;
-----------------------------------------------------------------------


architecture behave of adder is

 ----------------Component declaration-------------------
      component full_add
            port(
                  a,b,c_in    : in std_logic;
                  c_out, sum  : out std_logic);
      end component;
--------Define a signal for internal carry bits--------
      signal   c :  std_logic_vector( 10 downto 1);
      signal c11 :  std_logic ;
      signal sum :  std_logic_vector( 11 downto 1);
      signal d   :  std_logic_vector( 11 downto 0);

begin


      ---%%%%%%%% 10 component instantiation statements  %%%%%%%%---

      adder1: full_add
            port map( a    => a(1), b   => b(1), c_in => c0,
                      c_out => c(1), sum  => sum(1));
      adder2: full_add
            port map( a    => a(2), b   => b(2), c_in => c(1),
                      c_out => c(2), sum  => sum(2));
      adder3: full_add
            port map( a    => a(3), b   => b(3), c_in => c(2),
                      c_out => c(3), sum  => sum(3));
      adder4: full_add
            port map( a    => a(4), b   => b(4), c_in => c(3),
                      c_out => c(4), sum  => sum(4));
      adder5: full_add
            port map( a    => a(5), b   => b(5), c_in => c(4),
                      c_out => c(5), sum  => sum(5));
      adder6: full_add
            port map( a    => a(6), b   => b(6), c_in => c(5),
                      c_out => c(6), sum  => sum(6));
      adder7: full_add
            port map( a    => a(7), b   => b(7), c_in => c(6),
                      c_out => c(7), sum  => sum(7));
      adder8: full_add
            port map( a    => a(8), b   => b(8), c_in => c(7),
                     c_out => c(8), sum  => sum(8));
      adder9: full_add
            port map( a    => a(9), b   => b(9), c_in => c(8),
                      c_out => c(9), sum  => sum(9));
```

```
      adder10: full_add
            port map( a     => a(10), b  => b(10), c_in => c(9),
                      c_out => c(10), sum  => sum(10));
      adder11: full_add
            port map( a     => a(11), b  => b(11), c_in => c(10),
                      c_out => c11, sum  => sum(11));
      d <= c11 & sum;
      p <= d(11 downto 0);

end behave;
-------------------------------------------------------------------------
```

```
----------------------%%% Full Adder %%%--------------------------

library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------------------------------

entity full_add is
port(
           a,b,c_in  : in std_logic;
           c_out,sum : out std_logic);
end full_add;

-------------------------------------------------------------------

architecture adder of full_add is
begin
      c_out <= (( a xor b ) and c_in) or ( a and b );
      Sum   <= ( a xor b ) xor c_in ;
end adder;

-------------------------------------------------------------------
```

```
% MATLAB for autocorrelation and power spectrum %

echo on

M=100;
Rx_av=zeros(1,M+1);
Sx_av=zeros(1,M+1);
for j=1:10
    X=importdata('dataconVBoxcen2.dat');

    Rx=Rx_est(X,M);                     % autocorrelation of the realization
    Sx=fftshift(abs(fft(Rx)));          % of the realization
    Rx_av=Rx_av+Rx;                     % sum of the autocorrelations
    Sx_av=Sx_av+Sx;                     % sum of the spectrums
    echo off;
end;

echo on;
Rx_av=Rx_av/10;                         % ensemble average autocorrelation
Sx_av=Sx_av/10;                         % ensemble average spectrum
figure(3)
plot(Rx_av)
figure(4)
plot(Sx_av)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [Rx]=Rx_est(X,M)
%[Rx]=Rx_est(X,M)
%                   RX_EST Estimates the autocorrelation of the
%                   sequence of random variables given in X.
N=length(X);
Rx=zeros(1,M+1);
for m=1:M+1,
    for n=1:N-m+1,
        Rx(m)=Rx(m)+X(n)*X(n+m-1);
    end;
    Rx(m)=Rx(m)/(N-m+1);
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

หนังสืออ้างอิง

[1] ชัลวาล ธรรมราช ชูชาติ มูลคำ ศุภศิลป์ คำสร้อย " ตัวแปลงผลฟาสท์ฟูเรียร์ทรานส์ฟอร์ม " ปริญญานิพนธ์วิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมโทรคมนาคม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2544

[2] วิวัฒน์ กิรานนท์ "วิศวกรรมสื่อสาร" สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง พิมพ์ครั้งที่ 2 พ.ศ. 2542

[3] S.Ross, "A First Course in Probability, " Prentice Hall lnc, 2002

[4] S.Sjoholm, L.Lindh, "VHDL for Designer," Prentice Hall, 1997

[5] E.Boutill , J.Danger , A.Ghazel "Design of High Speed AWGN COMMUNICATION Channel Emulator," 2000

[6] ALTERA Data Book , 1998

[7] A.Papoulis, "Probability & Statistics," Prentice Hall Inc, 1990

[8] J.K.Proakis, M.Salehi, "Contemporary Communication Systems Using MATLAB," Books/Cole 2000

[9] S.Brown, Z.Vranesic, "Fundamentals of DIGITAL LOGIC with VHDL design," McGraw-Hill Book, 2000

[10] V.N.Yarmolik, S.N.Demidenko, "Generation and Application of Pseudorandom Sequences For Random Testing," John Wiley and Sons, 1988