การสร้างต้นแบบระบบ MIMO โดยใช้บอร์ด FPGA

(Implementation of MIMO System on FPGA Board)

โดย

นาย[illegible] เจริญสุข รหัสนักศึกษา [illegible]

นางสาว[illegible] [illegible] รหัสนักศึกษา [illegible]

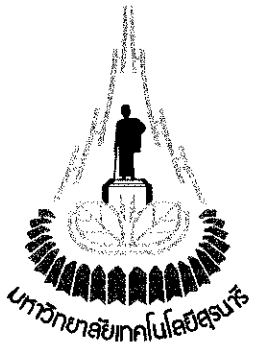
รายงานนี้เป็นส่วนหนึ่งของการศึกษาวิชา 427499 โครงงานวิศวกรรมโทรคมนาคม

และวิชา 427494 โครงงานศึกษาวิศวกรรมโทรคมนาคม ประจำภาคการศึกษาที่ 1 และ 2 ปีการศึกษา [illegible]

หลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมโทรคมนาคม หลักสูตรปรับปรุง พ.ศ.2545

สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

CONTRIBUTION



การสร้างต้นแบบระบบ MIMO โดยใช้บอร์ด FPGA

(Implementation of MIMO System on FPGA Board)

โดย

นางสาวทรัสตี เจริญคร รหัสนักศึกษา B4708001

นางสาวศิริวรรณ ชนะศิลป์ รหัสนักศึกษา B4704522

รายงานนี้เป็นส่วนหนึ่งของการศึกษาวิชา 427499 โครงงานศึกษาวิศวกรรมโทรคมนาคม
และวิชา 427494 โครงงานวิศวกรรมโทรคมนาคม ประจำภาคการศึกษาที่ 1 และ 2 ปีการศึกษา 2550
หลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมโทรคมนาคม หลักสูตรปรับปรุง พ.ศ.2545
สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

การสร้างต้นแบบระบบ MIMO โดยใช้บอร์ด FPGA

คณะกรรมการสอบโครงงาน

(อาจารย์ ดร. พีระพงษ์ อุฑารสกุล)
อาจารย์ที่ปรึกษาโครงงาน

(ผู้ช่วยศาสตราจารย์ ดร. ชุติมา พรหมมาก)
กรรมการ

(ผู้ช่วยศาสตราจารย์ ร. อ.ดร. ประโยชน์ คำสวัสดิ์)
กรรมการ

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับรายงานโครงงานฉบับนี้ เป็นส่วนหนึ่งของการศึกษาระดับปริญญาตรี สาขาวิชาวิศวกรรมโทรคมนาคม วิชา 427 494 โครงงานศึกษาวิศวกรรมโทรคมนาคม และวิชา 427 499 โครงงานวิศวกรรมโทรคมนาคม ประจำปีการศึกษา 2550

โครงงาน	การสร้างต้นแบบระบบ MIMO โดยใช้ FPGA Board
	(Implementation of MIMO System on FPGA Board)
โดย	นางสาวทรัสตี เจริญคร รหัส B4708001
	นางสาวศิริวรรณ ชนะศิลป์ รหัส B4704522
อาจารย์ที่ปรึกษา	ดร. พีระพงษ์ อุฑารสกุล
สาขาวิชา	วิศวกรรมโทรคมนาคม
ภาคการศึกษาที่	1 และ 2/2550

บทคัดย่อ

(Abstract)

โครงงานนี้เป็นการศึกษา วิเคราะห์ ออกแบบ และสร้างต้นแบบระบบ MIMO ลงบน FPGA Board โดยใช้เทคนิคการเข้ารหัสด้วยวิธี 2x2 Alamouti และผสมสัญญาณแบบ BPSK ผ่านทางสายเคเบิลสองเส้นแทนการใช้สายอากาศจริง โดยให้ FPGA Board เป็นระบบภาครับและระบบภาคส่ง นอกจากนี้ยังมีการใช้ซอฟแวร์ในการจำลองการทำงานเพื่อตรวจสอบความถูกต้องของต้นแบบที่พัฒนาขึ้น

กิตติกรรมประกาศ

(Acknowledgement)

จากการที่คณะจัดทำรายงานได้รับมอบหมายให้ทำโครงงานเรื่อง "การสร้างต้นแบบระบบ MIMO โดยใช้บอร์ด FPGA" ส่งผลให้คณะจัดทำรายงานได้รับความรู้และประสบการณ์ต่างๆ เกี่ยวกับการออกแบบรวมทั้งการทำงานของระบบ MIMO และบอร์ด FPGA บัดนี้โครงงานดังกล่าวพร้อมทั้งรายงานได้สำเร็จลงแล้ว ทั้งนี้ด้วยความร่วมมือและสนับสนุนจาก ดร.พีระพงษ์ อุฑารสกุล อาจารย์ที่ปรึกษาโครงงานเป็นอย่างดียิ่ง

ข้าพเจ้าใคร่ขอขอบพระคุณในการให้ข้อมูลและเป็นที่ปรึกษาในการทำรายงานฉบับนี้จนเสร็จสมบูรณ์ และหวังเป็นอย่างยิ่งว่าโครงงานนี้จะเป็นประโยชน์แก่การศึกษาหรือนำไปอ้างอิงการทำงานของระบบ MIMO ได้ไม่มากก็น้อย หากมีข้อผิดพลาดประการใด ข้าพเจ้ากราบขออภัยมา ณ ที่นี้ด้วย

นางสาวทรัสตี เจริญคร

นางสาวศิริวรรณ ชนะศิลป์

คณะผู้จัดทำโครงงาน

สารบัญ

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมา

ปัจจุบันการสื่อสารด้วยระบบไร้สายได้เติบโตอย่างรวดเร็วเนื่องจากระบบไร้สายถูกนำมาใช้อย่างแพร่หลาย ดังนั้นเพื่อรองรับต่อความต้องการสำหรับการส่งข้อมูลด้วยอัตราเร็วที่สูงขึ้นจึงมีการพัฒนาให้ระบบมีประสิทธิภาพที่ดียิ่งขึ้น การนำสายอากาศหลายตัวเข้ามาใช้ในระบบการสื่อสารไร้สายเป็นอีกวิธีหนึ่งที่สามารถเพิ่มประสิทธิภาพของระบบไร้สายได้ ซึ่งในปัจจุบันนี้ระบบ MIMO (Multiple Input Multiple Output) ที่มีการใช้เทคโนโลยีสายอากาศหลายตัวกำลังได้รับความสนใจอย่างมากต่อการสื่อสารด้วยระบบไร้สาย อันจะเห็นได้จากการนำระบบ MIMO ไปเป็นส่วนหนึ่งของมาตรฐานต่างๆ เช่น WiMax, IEEE 802.16 และ EDGE เป็นต้น

ปัจจุบันระบบ MIMO ถูกนำมาใช้กับระบบการสื่อสารไร้สายในต่างประเทศ แต่ยังไม่ได้ถูกใช้อย่างแพร่หลายในประเทศไทย เพราะยังเป็นเทคโนโลยีที่ใหม่ ดังนั้นทางคณะผู้จัดทำโครงการจึงทำการจำลองระบบการทำงานของระบบ MIMO เพื่อทดสอบและแสดงให้เห็นว่าระบบ MIMO สามารถใช้งานจริงได้

นอกจากนี้ได้มีการนำวงจรรวมชนิด FPGA (Field Programmable Gate Array) มาเป็นองค์ประกอบในการออกแบบวงจรดิจิตอล เนื่องจากวงจรที่ออกแบบสามารถนำไปเชื่อมต่อบนโปรแกรมคอมพิวเตอร์แล้วจำลองการทำงานเพื่อวิเคราะห์ผล และทำการโปรแกรมวงจรที่ออกแบบไว้ลงบนชิพที่ใช้ทำงานได้จริงทันที หรือที่เรียกว่า ระบบ Embedded System

ระบบ MIMO ที่ได้ทำการสร้างแบบจำลองขึ้นมานี้มีการเข้ารหัสและถอดรหัสแบบ Alamouti code ซึ่งเป็นวิธีการหนึ่งของการส่งข้อมูลโดยใช้สายอากาศหลายตัว โดยโครงการนี้จะพัฒนาตัวต้นแบบระบบ MIMO บนบอร์ด FPGA ที่ทำหน้าที่เป็นภาคส่งและภาครับตามการทำงานของการเข้ารหัสแบบ Alamouti code เพื่อเป็นประโยชน์ในการนำระบบ MIMO มาใช้งานจริงในอนาคต

โครงงานนี้จึงนำเสนอการสร้างต้นแบบระบบ MIMO ลงบนบอร์ด FPGA เพื่อทดสอบประสิทธิภาพการทำงานระบบ MIMO โดยใช้บอร์ด FPGA 2 ชุดในการสร้างต้นแบบระบบ MIMO โดยบอร์ดชุดที่ 1 ทำหน้าที่เป็นระบบภาคส่ง และบอร์ดชุดที่ 2 ทำหน้าที่เป็นระบบภาครับ

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาการทำงานของระบบ MIMO
2. เพื่อศึกษาและออกแบบการทำงานบนบอร์ด FPGA
3. เพื่อศึกษาและเขียนโปรแกรมระบบส่งข้อมูลโดยการเข้ารหัสแบบ Alamouti code
4. จำลองการทำงานของระบบ MIMO ลงบนบอร์ด FPGA เพื่อทดสอบว่าระบบ MIMO สามารถใช้งานจริงได้

1.3 ขอบเขตการทำงาน

1. ศึกษาการเข้ารหัส – ถอดรหัสแบบ 2x2 Alamouti Code
2. ศึกษาโครงสร้างสถาปัตยกรรมการรับส่งข้อมูลบนบอร์ด FPGA
3. ศึกษาการเขียนโปรแกรมของบอร์ด FPGA
4. ออกแบบและเขียนโปรแกรมทดสอบระบบ MIMO ด้วยบอร์ด FPGA
5. ทำการส่งและรับข้อมูลระบบ MIMO ด้วยสายเคเบิล

1.4 ขั้นตอนการดำเนินงาน

1. ศึกษา ค้นคว้าหาข้อมูล
2. เขียนโครงการและเสนอโครงการกับอาจารย์ที่ปรึกษา
3. จัดหาซื้ออุปกรณ์ที่ใช้ในโครงการนี้
4. ออกแบบโครงสร้างการทำงานของระบบและโปรแกรม
5. เขียนโปรแกรม รวมทั้งทดสอบการทำงานจริงของระบบ MIMO จำลองที่สร้างขึ้น
6. สรุปผลโครงงานและเขียนรายงาน
7. นำเสนอโครงงาน

บทที่ 2

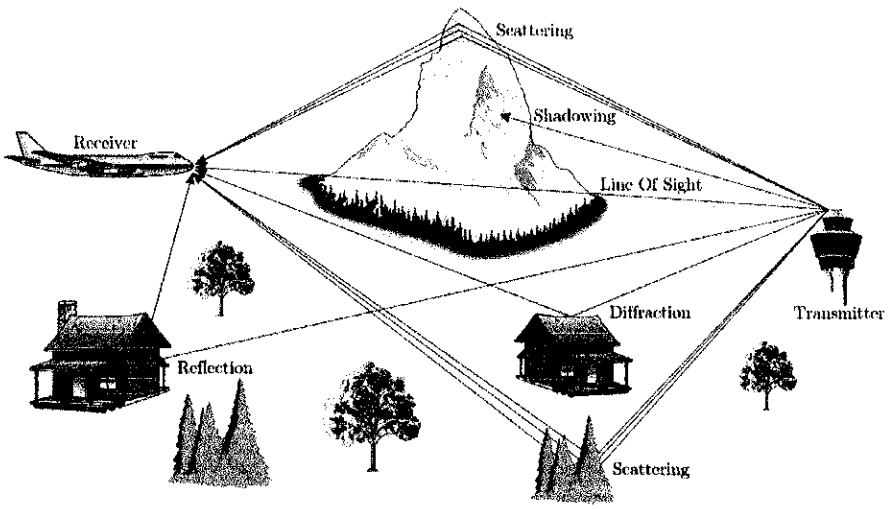
ระบบ MIMO

(Multiple Input Multiple Output)

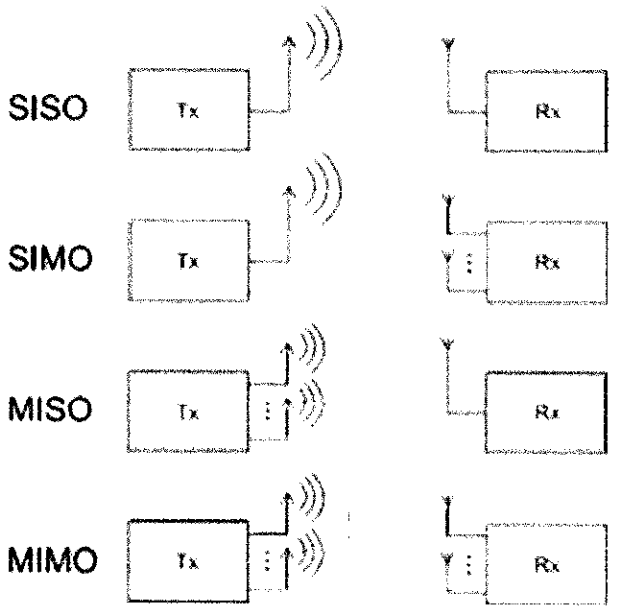
2.1 ระบบ MIMO

ระบบการสื่อสารไร้สายแม้ว่าจะเจริญเติบโตอย่างรวดเร็ว แต่ก็ยังมีข้อจำกัดทาง Radio Spectrum ซึ่งเป็นทรัพยากรจำกัด ทำให้มีการใช้อากาศหลายตัวแทนสายอากาศตัวเดียวในภาคส่งเพื่อเพิ่มประสิทธิภาพในการใช้ความถี่ พัฒนาเป็นเทคโนโลยีสายอากาศอัจฉริยะ หรือ Smart Antenna แต่ระบบก็ยังมีข้อเสียเรื่อง Multipath ทำให้เกิดการยอมรับที่จะใช้สายอากาศหลายตัวในภาครับด้วย เป็นแรงผลักดันให้เกิดระบบ MIMO ขึ้น เพื่อให้ใช้ความถี่ให้เกิดประโยชน์สูงสุด

Multipath Propagation คือการแพร่ของสัญญาณที่สะท้อนและเบี่ยงเบนของคลื่นกับสิ่งแวดล้อมของระบบ (Transmission Environment) มายังสายอากาศ ทำให้สัญญาณที่รับได้ซึ่งเป็นผลรวมของคลื่นทั้งหมดเกิดการเสริมและหักล้างกันของสัญญาณจนบางครั้งระดับสัญญาณที่รับได้ต่ำมาก ทำให้เกิดการขาดหายของสัญญาณ (fading) ซึ่งเป็นข้อเสียของระบบสื่อสารไร้สาย



รูปที่ 2-1 การเกิดสัญญาณสะท้อน (Multipath)



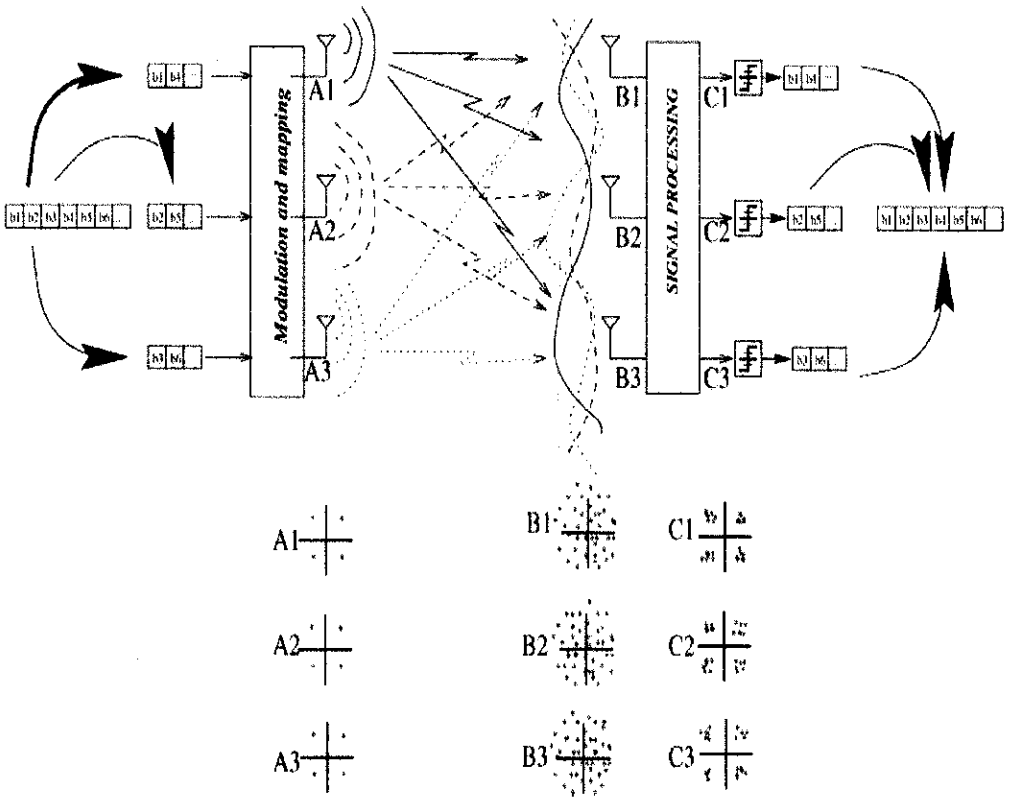
รูปที่ 2-2 การสื่อสารไร้สายระบบต่างๆ

จากยุคแรกเริ่มของการสื่อสารไร้สาย ระบบ SISO (Single Input Single Output) เป็นระบบส่งข้อมูลไร้สายแบบพื้นฐาน โดยภาครับและภาคส่งมีสายอากาศเพียงตัวเดียว เมื่อเจอสัญญาณสะท้อนมากๆ ระบบจะไม่สามารถรับสัญญาณได้ จึงมีการให้ภาครับใช้สายอากาศหลายตัวเป็นระบบ SIMO (Single Input Multiple Output) เพื่อให้รับสัญญาณได้มากขึ้น แต่ก็มีปัญหาเรื่องระดับสัญญาณในระยะที่ไกลออกไป อีกทั้งความยุ่งยากและความไม่เหมาะสมในการทำให้ภาครับซึ่งโดยทั่วไปก็คือผู้ใช้บริการใช้สายอากาศหลายตัวทำให้ระบบ SIMO ไม่ได้รับความนิยมมากนัก

การใช้สายอากาศที่ภาคส่งหลายตัวหรือระบบ MISO (Multiple Input Single Output) จึงถูกเข้ามาใช้แทนระบบ SIMO ซึ่งมีเทคโนโลยี Smart Antenna ในการสร้างแบบรูปการแผ่กระจายคลื่นแบบอัตโนมัติเพื่อหลีกเลี่ยงการขาดหายของสัญญาณด้วย แต่นั่นก็ไม่สามารถหลีกเลี่ยงได้ร้อยเปอร์เซ็น เพราะบางครั้งการ Multipath ก็ทำให้ภาคส่งไม่สามารถส่งสัญญาณออกไปได้ รวมถึงเมื่อมีการใช้บริการพร้อมกันหลายคนการจ่ายสัญญาณให้ทั่วถึงผู้ใช้บริการทุกคนจำเป็นต้องใช้กำลังส่งที่สูง และก็ยังมีข้อจำกัดเรื่องการลงทุนที่สูงด้วย เนื่องมาจากการพัฒนาเทคโนโลยีทางด้านสายอากาศทำให้สายอากาศมีขนาดเล็กลงและมีประสิทธิภาพสูงขึ้นเป็นแรงผลักดันให้มีการใช้สายอากาศหลายตัวที่ภาครับด้วย

การใช้สายอากาศหลายตัวทั้งภาคส่งและภาครับของระบบ MIMO บวกกับการคำนวณทางคณิตศาสตร์และสมการใหม่ ๆ ในการเข้ารหัสถอดรหัสที่ช่วยสร้างรูปคลื่นที่สมบูรณ์จากการรวมสัญญาณสะท้อนทั้งหมดเข้าด้วยกัน เป็นการนำสัญญาณสะท้อนที่เป็นข้อเสียของระบบสื่อสารไร้

สายมาใช้ให้เกิดประโยชน์ ระบบจึงสามารถรับสัญญาณได้แม้ระยะที่ไกลออกไปหรือมีสัญญาณสะท้อนมาก และนี่จึงเป็นสาเหตุสำคัญที่ทำให้ระบบ MIMO รับส่งสัญญาณได้อย่างมีประสิทธิภาพ



รูปที่ 2-3 ระบบ MIMO

2.2 ทฤษฎีระบบ MIMO

2.2.1 ระบบสายอากาศหลายตัว (Multi-Antenna System)

ระบบสายอากาศหลายตัวเป็นระบบที่มีการนำสายอากาศพื้นฐานมาวางเรียงกันหลายตัว เพื่อใช้ในระบบสื่อสารไร้สาย หรือที่เรียกว่า Array Antenna ทำให้สามารถส่งข้อมูลได้ดีขึ้น สายอากาศมีอัตราขยายที่สูงขึ้น การรับส่งข้อมูลมีประสิทธิภาพดีขึ้น แต่นั่นก็ต้องแลกมาด้วยการที่สายอากาศมี Side Lobe ที่สูงเช่นกัน ซึ่งมีผลต่อทิศทางการส่งที่ไปรบกวนระบบสื่อสารอื่น

เนื้อหาในส่วนนี้อยู่นอกเหนือขอบเขตของโครงการ ดังนั้นจึงขอกล่าวเพียงสั้นๆ เพื่อให้เข้าใจถึงระบบสายอากาศหลายตัวเท่านั้น ซึ่งในการทำโครงการเราใช้สายเคเบิลสองสายส่งข้อมูลแทนระบบสายอากาศสองต้นในภาคส่งและภาครับ จึงไม่จำเป็นต้องคำนึงถึง Channel Estimation

ในการใช้ระบบสายอากาศหลายตัวนั้น จะต้องคำนึงถึงตำแหน่งการเรียงตัวของสายอากาศ, การเลื่อนเฟส และอื่นๆ ซึ่งปัจจัยเหล่านี้ส่งผลต่อแบบรูปการแผ่กระจายคลื่น ซึ่งเป็นเรื่องที่สำคัญมากในระบบสื่อสารไร้สายที่จำเป็นต้องคำนึงถึงทิศทางการรับส่งสัญญาณเป็นสำคัญ เช่น ในระบบเรดาห์ เป็นต้น

2.2.2 การเข้ารหัสและถอดรหัสแบบ 2x2 Alamouti

การเข้ารหัส 2x2 Alamouti (ภาคส่ง)

ระบบ MIMO มีการทำ Space Time Coding (STC) โดยการเข้ารหัสแบบ Alamouti ซึ่งในโครงงานนี้ใช้การเข้ารหัส Alamouti แบบเมตริกซ์ 2x2 ซึ่งมีรูปแบบการส่งดังนี้

$$X = \begin{pmatrix} s_1 & -s_2^* \\ s_2 & s_1^* \end{pmatrix} \qquad (2\text{-}1)$$

จากสมการ Alamouti Code แถวของเมตริกซ์แทนสายอากาศแต่ละตัว และคอลัมน์ของเมตริกซ์แทนช่วงเวลาในการส่ง

เมื่อ X คือสัญญาณที่ส่งออกจากภาคส่ง

S_1 คือสัญญาณที่ส่งออกจากสายอากาศตัวที่ 1

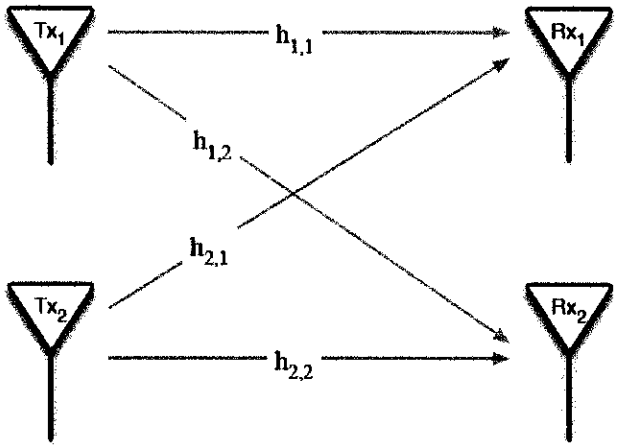
S_2 คือสัญญาณที่ส่งออกจากสายอากาศตัวที่ 2

เมื่อส่งสัญญาณ binary 8 bits จะได้รูปแบบเมตริกซ์ดังนี้

$$X=\begin{bmatrix} s_1 & -s_2^* \\ s_2 & s_1^* \end{bmatrix}\begin{bmatrix} s_3 & -s_4^* \\ s_4 & s_3^* \end{bmatrix}\begin{bmatrix} s_5 & -s_6^* \\ s_6 & s_5^* \end{bmatrix}\begin{bmatrix} s_7 & -s_8^* \\ s_8 & s_7^* \end{bmatrix} \qquad (2\text{-}2)$$

ค่าในแถวที่หนึ่งคือสัญญาณที่สายอากาศตัวที่ 1 ส่งออก และในแถวที่สองคือสัญญาณที่สายอากาศตัวที่สองทำการส่งออก โดยที่คอมลัมน์ของเมตริกซ์คือเวลาที่ทำการส่ง จะเห็นว่าเมื่อเวลาช่วงที่หนึ่ง (คอลัมน์ที่ 1) สายอากาศตัวที่หนึ่งทำการส่ง s_1 และสายอากาศตัวที่สองทำการส่ง s_2 จากนั้นเวลาช่วงที่สองสายอากาศตัวที่หนึ่งจะส่งสัญญาณ $-s_2^*$ และสายอากาศตัวที่สองส่ง s_1^*

การถอดรหัส 2x2 Alamouti (ภาครับ)



รูปที่ 2-4 แสดงช่องสัญญาณในการส่งข้อมูล

จากรูปที่ 4 การส่งข้อมูลผ่านสายอากาศในส่วนของภาคส่ง สายอากาศภาครับแต่ละตัวจะรับข้อมูลได้ทั้งจากสายอากาศตัวที่หนึ่งและตัวที่สอง อีกทั้งในการส่งข้อมูลยังมีค่าความผิดพลาดอีกด้วย ดังนั้นในการถอดรหัสต้องพิจารณาถึงทั้งช่องสัญญาณและความผิดพลาดของข้อมูล

สัญญาณที่ภาครับรับได้จะมีสมการดังนี้

$$Y=\begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}\begin{pmatrix} s_1 & -s_2^* \\ s_2 & s_1^* \end{pmatrix}+\begin{pmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{pmatrix} \qquad (2\text{-}3)$$

โดยให้

$$r_{11} \triangleq h_{11}s_1 + h_{12}s_2 + e_{11} \qquad (2\text{-}4)$$

$$r_{12} \triangleq -h_{11}s_2^* + h_{12}s_1^* + e_{12} \qquad (2\text{-}5)$$

$$r_{21} \triangleq h_{21}s_1 + h_{22}s_2 + e_{21} \qquad (2\text{-}6)$$

$$r_{22} \triangleq -h_{21}s_2^* + h_{22}s_1^* + e_{22} \qquad (2\text{-}7)$$

เราจึงได้สมการ Y เป็น

$$Y = \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix} \qquad (2\text{-}8)$$

ซึ่งในการเข้ารหัสแบบ Alamouti สามารถประมาณค่า s_1 และ s_2 ได้เป็น

$$\widetilde{s_1} = h_{11}^* r_{11} + h_{12} r_{12}^* + h_{21}^* r_{21} + h_{22} r_{22}^* \qquad (2\text{-}9)$$

$$\widetilde{s_2} = h_{12}^* r_{11} - h_{11} r_{12}^* + h_{22}^* r_{21} - h_{21} r_{22}^* \qquad (2\text{-}10)$$

เมื่อแทนค่า r_{11}, r_{12}, r_{21} และ r_{22} ลงในสมการข้างบนจะได้สัญญาณ S1 และ S2 ที่ภาคส่งทำการส่งออกอากาศ

$$\begin{aligned}\widetilde{s_1} &= h_{11}^* \left(h_{11}s_1 + h_{12}s_2 + e_{11}\right) \\ &\quad + h_{12}\left(-h_{11}^* s_2 + h_{12}^* s_1 + e_{12}^*\right) \\ &\quad + h_{21}^* \left(h_{21}s_1 + h_{22}s_2 + e_{21}\right) \\ &\quad + h_{22}\left(-h_{21}^* s_2 + h_{22}^* s_1 + e_{22}^*\right) \\ &= s_1\left(|h_{11}| + |h_{12}| + |h_{21}| + |h_{22}|\right) \\ &\quad + h_{11}^* e_{11} + h_{12} e_{12}^* + h_{21}^* e_{21} + h_{22} e_{22}^*\end{aligned}$$

$$\begin{aligned}\widetilde{s_2} &= s_2\left(|h_{11}| + |h_{12}| + |h_{21}| + |h_{22}|\right) \\ &\quad - h_{11} e_{12}^* + h_{12}^* e_{11} - h_{21} e_{22}^* + h_{22}^* e_{21}\end{aligned}$$

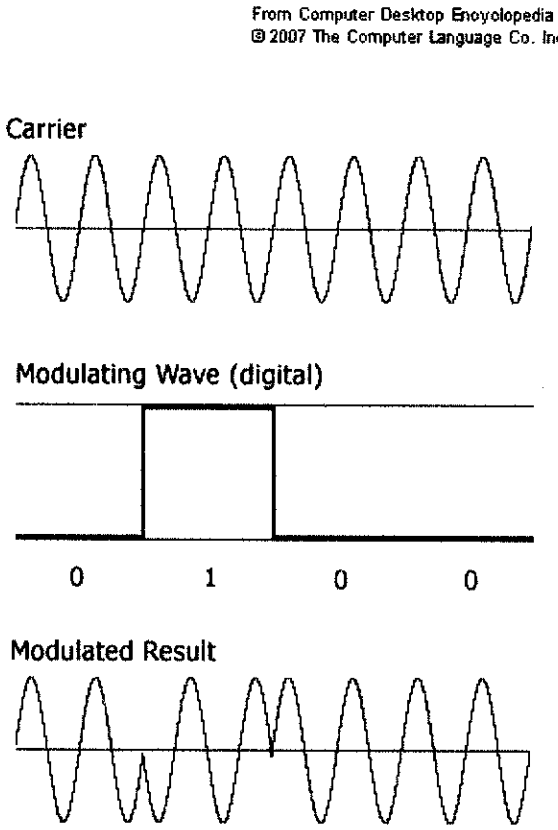
ด้วยการเข้ารหัสแบบ Alamouti ทำให้ระบบ MIMO สามารถนำ Multipath Propagation มาใช้ให้เกิดผลดีต่อระบบ ซึ่งทำให้ประสิทธิภาพโดยรวมของระบบดีขึ้น นอกจากจะทำให้ปัญหาเรื่อง fading หมดไปแล้ว ยังให้คุณภาพสัญญาณในระยะไกลดีกว่าระบบ SISO อีกด้วย และที่สำคัญระบบ MIMO ยังใช้ความถี่ซึ่งเป็นทรัพยากรจำกัดในระบบสื่อสารได้อย่างคุ้มค่า

2.2.3 Phase Shift Keying Modulation (PSK)

เป็นการมอดูเลทสัญญาณข่าวสารดิจิตอลเข้ากับคลื่นพาห์(carrier) โดยมีเฟสเปลี่ยนแปลงไปตามการเปลี่ยนแปลงของข้อมูล สัญญาณจะมีความถี่และแอมพลิจูดคงที่ หรือเรียกอีกอย่างว่า "การมอดูเลทเชิงเฟส" แบ่งตามจำนวนองศาเฟสที่เปลี่ยนแปลงเป็น BPSK, 4PSK, QPSK และ M-PSK

ยกตัวอย่าง BPSK จะมีการเลื่อนเฟสเมื่อมีการเปลี่ยนแปลงของข้อมูลข่าวสารเริ่มต้น เช่น เมื่อเปลี่ยนจากบิต 0 เป็นบิต 1 จะมีการเลื่อนเฟสจากเริ่มต้นที่ 0 องศาไปเป็น 180 องศา

ในทางกลับกัน เมื่อข้อมูลข่าวสารดิจิตอลมีการเปลี่ยนแปลงจากบิต 1 เป็นบิต 0 สัญญาณมอดูเลทจะมีการเลื่อนเฟสจากเริ่มที่ 180 องศาไปเป็น 0 องศา ดังแสดงในรูปที่ 5



รูปที่ 2-5 การมอดูเลทแบบ BPSK

ตัวอย่างการเข้ารหัสแบบ Alamouti และมอดูเลทแบบ BPSK

สัญญาณที่เข้ามาเป็น 8 บิต คือ 10111011

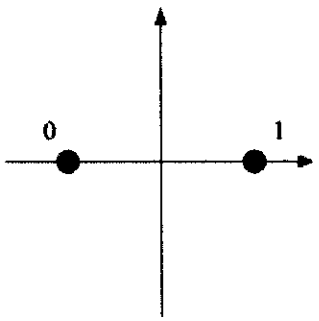
จากสมการการเข้ารหัสแบบ Alamouti

$$X = \begin{bmatrix} s_1 & -s_2^* \\ s_2 & s_1^* \end{bmatrix} \qquad (2\text{-}11)$$

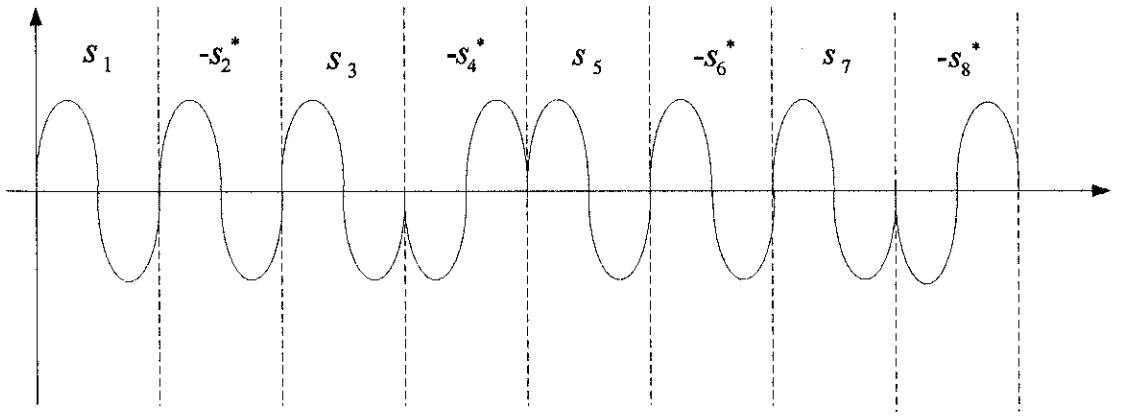
จะได้

$$X = \begin{bmatrix} s_1 & -s_2^* \\ s_2 & s_1^* \end{bmatrix}\begin{bmatrix} s_3 & -s_4^* \\ s_4 & s_3^* \end{bmatrix}\begin{bmatrix} s_5 & -s_6^* \\ s_6 & s_5^* \end{bmatrix}\begin{bmatrix} s_7 & -s_8^* \\ s_8 & s_7^* \end{bmatrix} \qquad (2\text{-}12)$$

$$\begin{matrix} T_{X1} \rightarrow \\ T_{X2} \rightarrow \end{matrix}\begin{bmatrix} 1+0j & 1+0j \\ -1+0j & 1-0j \end{bmatrix}\begin{bmatrix} 1+0j & -1+0j \\ 1+0j & 1-0j \end{bmatrix}\begin{bmatrix} 1+0j & 1+0j \\ -1+0j & 1-0j \end{bmatrix}\begin{bmatrix} 1+0j & -1+0j \\ 1+0j & 1-0j \end{bmatrix} \qquad (2\text{-}13)$$

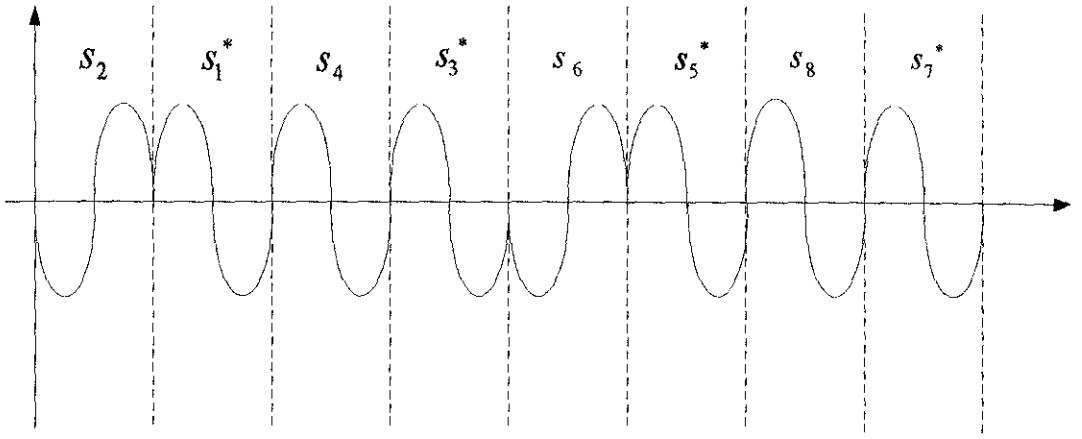


ได้รูปสัญญาณของ T_{X1} ดังนี้



รูปที่ 2-6 การมอดูเลทแบบ BPSK ของสายอากาศตัวที่ 1

ได้รูปสัญญาณของ T_{X2} ดังนี้



รูปที่ 2-7 การมอดูเลทแบบ BPSK ของสายอากาศตัวที่ 2

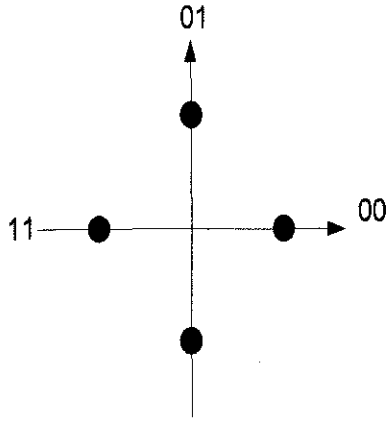
ตัวอย่างการเข้ารหัสแบบ Alamouti และมอดูเลทแบบ 4PSK

สัญญาณที่เข้ามาเป็น 8 บิต คือ 10111011

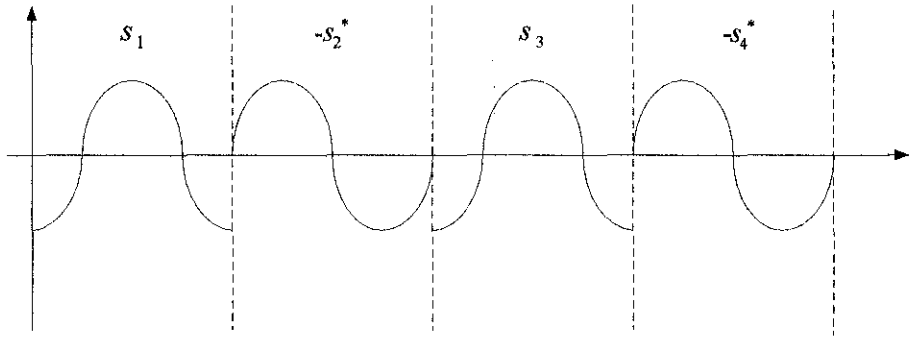
จะสมการเมตริกซ์การเข้ารหัส Alamouti เป็น

$$X=\begin{bmatrix} s_1 & -s_2^* \\ s_2 & s_1^* \end{bmatrix}\begin{bmatrix} s_3 & -s_4^* \\ s_4 & s_3^* \end{bmatrix} \qquad (2\text{-}14)$$

$$\begin{matrix} T_{X1} \rightarrow \\ T_{X2} \rightarrow \end{matrix}\begin{bmatrix} 0\text{-}1j & 1+0j \\ -1+0j & 0+1j \end{bmatrix}\begin{bmatrix} 0\text{-}1j & 1+0j \\ -1+0j & 0+1j \end{bmatrix} \qquad (2\text{-}15)$$

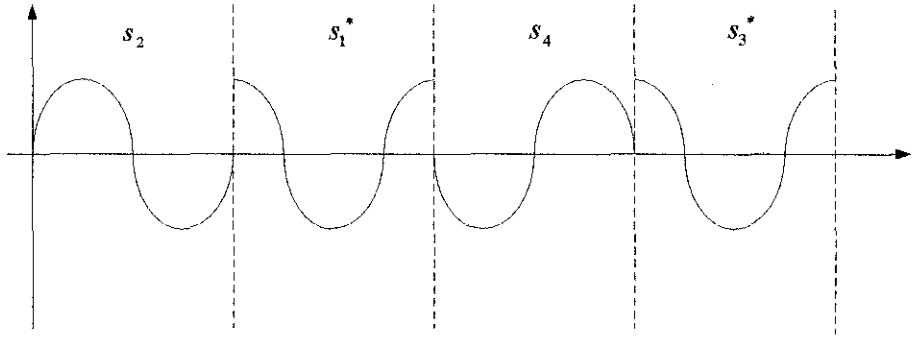


ได้รูปสัญญาณของ T_{X1} ดังนี้



รูปที่ 2-8 การมอดูเลทแบบ 4PSK ของสายอากาศตัวที่ 1

ได้รูปสัญญาณของ T_{X2} ดังนี้



รูปที่ 2-9 การมอดูเลทแบบ 4PSK ของสายอากาศตัวที่ 2

บทที่ 3
FPGA

3.1 FPGA

ในช่วงก่อนทศวรรษ 1970 อุตสาหกรรมเซมิคอนดัคเตอร์ได้ถูกปลุกให้ตื่นตัวขึ้นหลังจากที่มีการประดิษฐ์วงจรรวมหรือไอซีขนาดเล็กหรือ SSI (Small-Scale Integration) ประกอบไปด้วยเกทดิจดตอลจำนวนไม่มากนัก (ประมาณ 1 ถึง 10 ตัว) ต่อมาได้มีการเพิ่มปริมาณของเกทดิจิตอลและฟังก์ชันทางลอจิกให้มากขึ้นจนเป็น MSI (Medium-Scale Integration)

การพัฒนาไอซีเป็นไปอย่างต่อเนื่องจนมาถึงยุคของ LSI(Large-Scale Integration) ซึ่งเป็นยุคที่มีการสร้างไมโครโปรเซสเซอร์ตัวแรกขึ้นและในปัจจุบันเป็นยุคของVLSI(Very Large-Scale Integration) ซึ่งเป็นเทคโนโลยีในการสร้างไอซีรุดหน้าจนสามารถสร้างไมโครโปรเซสเซอร์ขนาด 64 บิตที่มีหน่วยความจำแคชกับหน่วยคำนวณทางคณิตศาสตร์ของโฟล์ทติงพอยน์(Floating-Point Arithmetic Units) รวมอยู่ภายในตัวมัน และเนื่องจากการปรับปรุงเทคโนโลยีของกระบวนการสร้างซีมอส ที่มีมาอย่างต่อเนื่องทำให้ขนาดของทรานซิสเตอร์ที่บรรจุอยู่ในไอซีมีขนาดเล็กลงมาเรื่อยๆ จนบางคนโดยเฉพาะชาวญี่ปุ่นใช้คำว่า ULSI(Ultra large Scale Integration) เพื่อใช้เรียกระดับของไอซีในปัจจุบันแต่คนส่วนมาก ยังนิยมเรียกเพียงแค่ VLSI จากการปรากฏณ์ตัวของ VLSI ในช่วงทศวรรษ 1980 ทำให้วิศวกรเริ่มมีการออกแบบไอซีตามความต้องการของลูกค้าซึ่งใช้ในระบบที่เจาะจงนอกเหนือจากการใช่ไอซีมาตรฐานเพียงอย่างเดียว โดยไอซีเหล่านี้มีชื่อเรียกว่า ASIC: Application Specific Integrated Circuit(ออกเสียงว่า เอ-ซิก) ซึ่งตัวอย่างของ ASIC ได้แก่ชิพไอซีที่ใช้ สำหรับตุ๊กตาของเล่นที่พูดได้ ดาวเทียม และชิพที่ภายในบรรจุด้วยไมโครโปรเซสเซอร์กับอุปกรณ์ทางลอจิกอื่นๆ

3.1.1 ประเภทของ ASIC

ASIC แบ่งเป็น 3 ประเภทใหญ่ๆ คือ Full-custom, Semi-custom, Programmable

3.1.1-1 Full-custom

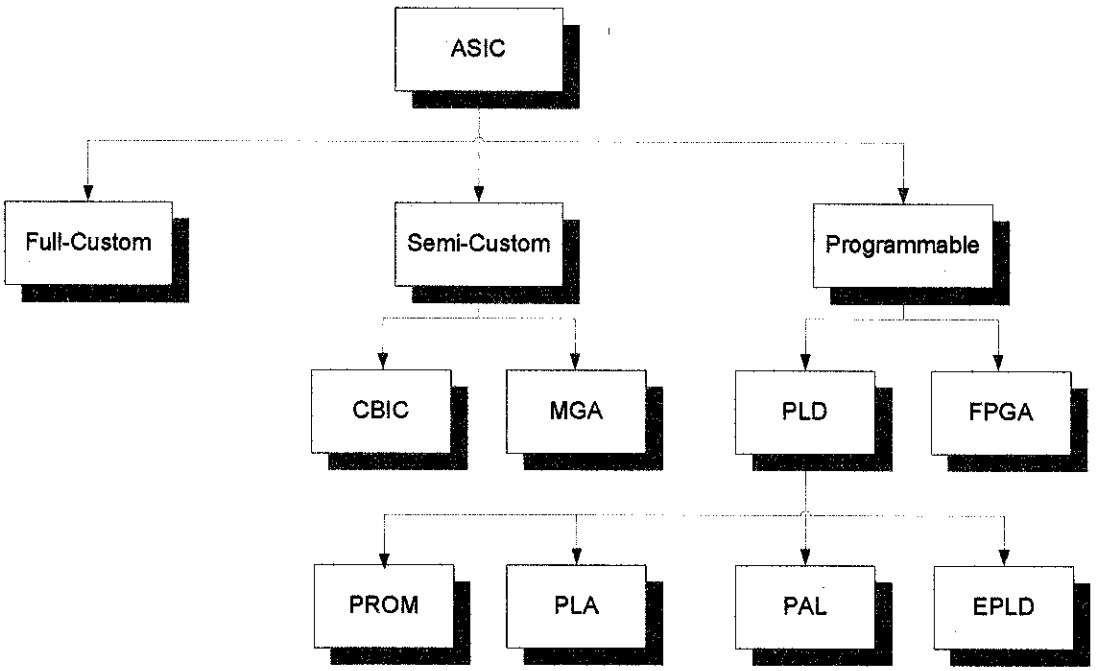
ASIC ประเภทนี้ลูกค้าจะเป็นผู้ออกแบบเซลล์ลอจิก (เช่น แอนด์เกท ออร์เกท มัลติเพล็กเซอร์ และฟลิปฟล็อบ) และลักษณะ การจัดวางอุปกรณ์บนตัวไอซีรวมถึงหน้ากากสำหรับควาบคุมการเจือและสร้างชั้นสาร (Mask) ต่างๆที่ใช้ในการทำไอซีเอง ดังนั้นค่าใช้จ่ายในการออกแบบและการผลิตจะสูงมาก

3.1.1-2 Semi-custom

ASIC ประเภทนี้เซลล์ลอจิกจะถูกออกแบบเอาไว้ก่อนแล้วในรูปแบบของไลบรารีและลูกค้าจะเป็นผู้ออกแบบ Mask ต่างๆเอง ตัวอย่างของไอซีประเภทนี้ได้แก่ Standard-Cell-Based ASIC และ Mask Gate-Array-Based ASIC

3.1.1-3 Programmable

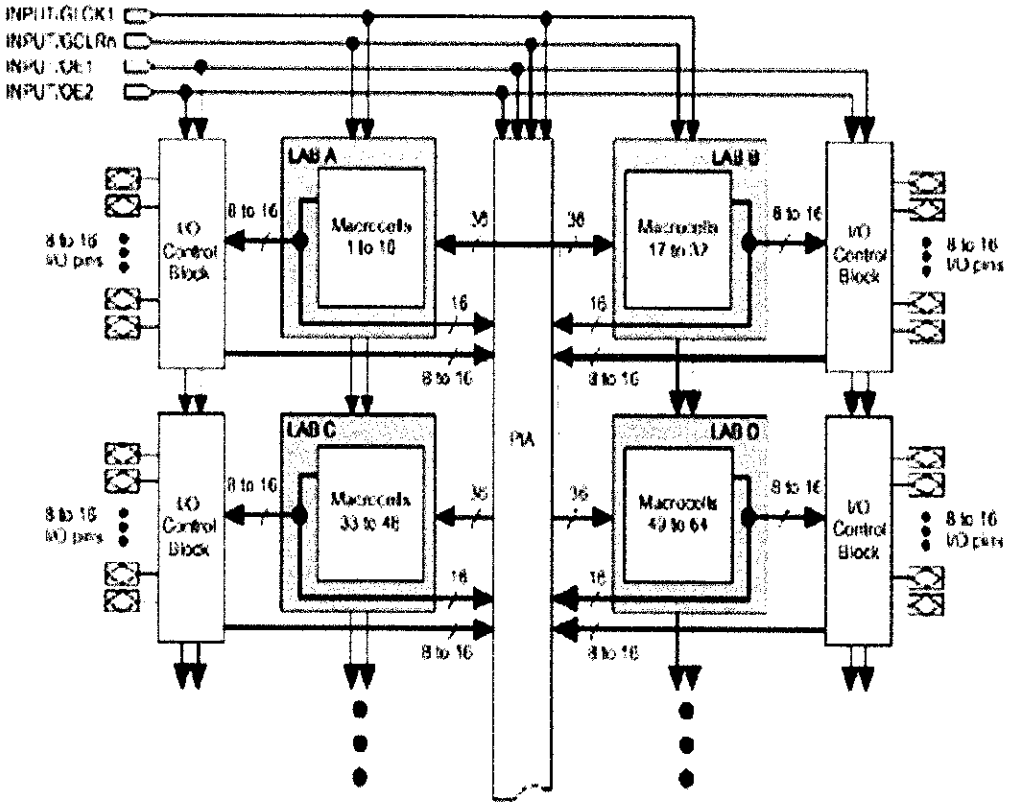
ASIC ประเภทนี้เซลล์ลอจิกจะถูกออกแบบไว้ก่อนเช่นเดียวกับ Semi-custom แต่ชั้นของ Mask จะไม่สามารถเปลี่ยนแปลงได้ ตามความต้องของผู้ออกแบบ ไอซีประเภทนี้ยังแบ่งออกเป็น 2 ชนิดคือ Programmable Logic Device (PLD) และ Field Programmable Gate Array (FPGA)



รูปที่ 3-1 แผนผัง ASIC

3.2 โครงสร้างภายในของ FPGA

ลักษณะโครงสร้างภายในของ FPGA จะเป็นอะเรย์ของบล็อคลอจิกที่สามารถทำการโปรแกรมได้



รูปที่ 3-2 แผนผังโครงสร้างภายในของ FPGA

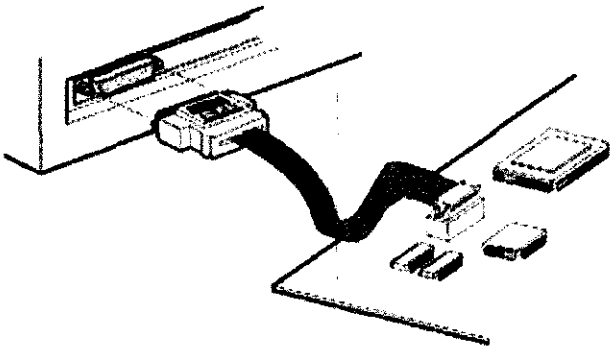
3.3 ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว

1. ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพ เพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออก แบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึง ภาษา Assembly ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2. มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจร หรือ HDL (Hardware Description Language) เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็น ต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่

กำหนดลักษณะการทำงานให้มัน จากนั้นตัวซอฟต์แวร์จะทำ Synthesis and Optimize ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตราฐาน เดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3. การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของ คอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้ โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอก ดังรูปที่ 3-3 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสีย ค่าใช้จ่ายเพิ่มแต่อย่างใด



รูปที่ 3-3 การโปรแกรมโดยส่งข้อมูลผ่านสาย JTAG

3.4 ภาษาที่ใช้เขียนลงบน FPGA

การโปรแกรมการทำงานของ FPGA ของแต่ละผู้ผลิตจะมีการใช้ซอฟต์แวร์ของตัวเองซึ่งออกแบบให้ทำงานร่วมกับการประมวลผลของชิพอย่างเหมาะสมได้เป็นอย่างดี ส่วนภาษาที่ใช้ในการเขียนฟังก์ชันการทำงานต่างๆ ของ FPGA นั้นจะใช้ VHDL เป็นมาตรฐานเดียวกันทั่วทั้งโลก

VHDL ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูงที่ใช้สำหรับการออกแบบฮาร์ดแวร์ในระบบดิจิตอลซึ่งสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และสามารถเขียนได้หลายรูปแบบ โครงสร้างภาษาอ่านเข้าใจง่าย การเปลี่ยนแปลงแก้ไขวงจรทำได้ง่าย อีกทั้งยังมีซอฟต์วัทูลช่วยในการตรวจสอบความถูกต้องของวงจรที่ออกแบบ โดยการจำลองการทำงานของวงจร ทั้งนี้ผู้ออกแบบวงจรสามารถออกแบบและจำลองการทำงานของฟังก์ชันโดยยังไม่ต้องคำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง

VHDL พัฒนาโดยกระทรวงกลาโหมสหรัฐอเมริกาและประกาศเป็นมาตรฐาน IEEE 1076-1987 ในปี คศ.1987 แต่อย่างไรก็ตามชนิดของข้อมูลใน IEEE 1076 ก็ไม่ครอบคลุมบางสถานะของวงจรดิจิตอล จึงได้มีการนิยาม Standard logic package คือ IEEE 1164 เพิ่มเติมให้มาตรฐาน VHDL

มีความสมบูรณ์ยิ่งขึ้น ต่อมา IEEE 1076-1987 ได้ปรับปรุงเป็น IEEE1076-1993 และมีการนิยาม Numeric Standard หรือ Synthesis standard เพิ่มเติมเป็นมาตรฐาน IEEE1076.3 ในปลายปี คศ.1997

3.4.1 องค์ประกอบของภาษา VHDL

องค์ประกอบภาษา VHDL (VHDL Structural elements) มีหน่วยออกแบบ หรือ Design Unit ได้แก่

3.4.1-1 Entity declaration เป็นหน่วยการออกแบบที่ใช้สำหรับติดต่อกับวงจรภายนอก รวมทั้งการส่งค่าพารามิเตอร์บางอย่างระหว่างอุปกรณ์หรือวงจรภายนอก

3.4.1-2 Architecture เป็นหน่วยการออกแบบส่วนที่ใช้เขียนบรรยายการทำงานของวงจร โดยมีความสัมพันธ์กับสิ่งที่กำหนดใน entity ซึ่งในส่วนนี้สามารถเขียนได้หลายรูปแบบ ดังต่อไปนี้

-Behavioral style

-Data flow style

-Structural style

-Mixed model style

3.4.1-3 Package เป็นหน่วยการออกแบบที่ใช้เก็บข้อมูลต่างๆ ตลอดจนโปรแกรมย่อยที่เป็นประโยชน์ในการเขียนบรรยายวงจรดิจิตอล โดยที่ข้อมูลใน Package สามารถเรียกใช้ได้โดย Entity, Architecture หรือด้วย Package อื่นๆ ด้วยคำสั่ง Use Statement

3.4.1-4 Configuration เป็นส่วนที่ใช้กำหนดว่าใน entity ที่มีหลายArchitecture จะใช้ Architecture ใดในการจำลองการทำงานของวงจรที่ออกแบบ เพื่อให้ซอฟต์แวร์ Simulator นำไปจำลองการทำงานของวงจร

3.4.2 การเขียนภาษา VHDL เบื้องต้น

ปกติภาษา VHDL เป็นภาษาแบบ Case insensitive คือ ไม่มีความแตกต่างกันในการเขียนอักษรพิมพ์เล็กหรือพิมพ์ใหญ่ กล่าวคือ ไม่ว่าจะเขียนด้วยตัวอักษรพิมพ์เล็กหรือพิมพ์ใหญ่มีก็ความหมายเหมือนกัน และนอกจากนี้ภาษา VHDL ยังมีลักษณะเป็น Concurrent คือคำสั่งทุกชุดคำสั่งจะทำงานพร้อมกันไม่ขึ้นอยู่กับลำดับก่อนหลัง มีกฎเกณฑ์ทั่วๆ ไปในการเขียนดังนี้

การตั้งชื่อ

การตั้งชื่อในภาษา VHDL ประกอบไปด้วย พยัญชนะ, สระ, ตัวเลข และเครื่องหมายขีดเส้นใต้ (_) โดยที่มีข้อจำกัดว่า ต้องขึ้นต้นด้วยพยัญชนะและเขียนติดกันโดยไม่เว้นช่องว่าง, ห้ามใช้

เครื่องหมายขีดเส้นใต้ติดกัน, ห้ามจบด้วยเครื่องหมายขีดเส้นใต้ และที่สำคัญที่สุดก็คือ "ห้ามใช้คำสงวน" ซึ่งคำสงวนในภาษา VHDL มีดังต่อไปนี้

abs	configuration	impure	null	rem	type
access	constant	in	of	report	unaffected
after	disconnect	inertial	on	return	units
alias	downto	inout	open	rol	until
all	else	is	or	ror	use
and	elsif	label	others	select	variable
architecture	end	library	out	severity	wait
array	entity	linkage	package	signal	when
assert	exit	literal	port	shared	while
attribute	file	loop	postponed	sla	with
begin	for	map	procedure	sll	xnor
block	function	mod	process	sra	xor
body	generate	nand	pure	srl	
buffer	generic	new	range	subtype	
bus	group	next	record	then	
case	guarded	nor	register	to	
component	if	not	reject	transport	

รูปที่ 3-4 คำสงวนในภาษา VHDL

ประเภทของพอร์ต

ในการประกาศ Entity มีความจำเป็นอย่างยิ่งที่จะต้องระบุประเภทของพอร์ตหรือ interface ที่ใช้ติดต่อกับวงจรภายนอกของวงจรที่เราออกแบบ ซึ่งประเภทของพอร์ตที่ใช้ใน Entity declaration มีดังต่อไปนี้

- In คือ อินพุต (input) เป็นพอร์ตรับสัญญาณจากวงจรภายนอกเข้ามาในวงจรที่ออกแบบ

- Out คือ เอาท์พุต (output) เป็นพอร์ตส่งสัญญาณจากวงจรที่ออกแบบออกไปยังวงจรภายนอก

- Inout คือ อินพุต/เอาท์พุต (I/O) เป็นพอร์ตรับหรือส่งสัญญาณเข้าหรือออกที่ใช้ติดต่อกับวงจรภายนอก

- Buffer เป็นเอาท์พุตพอร์ตที่สามารถอ่านค่ากลับเข้ามาภายในวงจรที่ออกแบบได้ เช่น วงจรนับ เป็นต้น

ชนิดข้อมูล

ชนิดข้อมูลต่างๆ ที่ใช้ในภาษา VHDL โดยเบื้องต้นที่ผู้ออกแบบวงจรควรทราบมีดังต่อไปนี้

bit มีค่าเป็นลอจิค '0' และ '1'

bit_vector เป็นข้อมูลแบบหลายบิต หรือ bus หรือ array ของชนิดข้อมูลบิต

std_logic มีค่าทางลอจิกได้ 9 ค่า ได้แก่ 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H' และ '-'

std_logic_vector เป็นข้อมูลแบบหลายบิตหรือ array ของข้อมูลชนิด std_logic

boolean มีค่าเป็น TRUE หรือ FALSE เป็นจริงหรือเท็จเท่านั้น

integer มีค่าอยู่ในช่วง $-(2^{31}-1)$ ถึง $+(2^{31}-1)$ หรือ -2,147,483,647 ถึง 2,147,483,647 โดยปกติจะถูกกำหนดไว้ล่วงหน้าเป็นเลขฐานสิบ

real เป็นค่าเลขทศนิยมฐานสิบ

signed,unsigned เป็นข้อมูลที่อยู่ในรูปฟอร์ม std_logic_vector แต่สามารถใช้กับตัวดำเนินการด้านคณิตศาสตร์ได้

ตัวดำเนินการ (Operator)

ภาษา VHDL จะมีตัวดำเนินการที่ใช้สำหรับการเปรียบเทียบ, การกระทำการทางคณิตศาสตร์ หรือการกระทำทางตรรกะ (Boolean) โดยชุดของ Operator ทั้งหมดที่มีในภาษา VHDL มีดังต่อไปนี้

NOT	inversion
AND	and function
NAND	not-and function
OR	or function
NOR	not-or function
XOR	exclusive-or function
XNOR	exclusive-nor function
=	equality
/=	inequality
>=	greater-than or equal
>	greater-than
<=	less-than equal

<	less-than
SLL	shift-left logical
SRL	shift-right logical
SLA	shift-left arithmetic
SRA	shift-right arithmetic
ROL	rotate left
ROR	rotate right
+	addition
-	minus sign
*	multiplication
/	division
MOD	modulo arithmetic
REM	remainder after division
**	exponentiation
ABS	absolute value
&	concatenation

ลำดับการทำงานของตัวดำเนินการจะทำงานจากลำดับสูงสุดไปหาต่ำสุด สำหรับตัวดำเนินการลำดับเดียวกันจะทำงานโดยเรียงจากซ้ายไปขวาของคำสั่ง ซึ่งตัวดำเนินการในภาษา VHDL สามารถแบ่งกลุ่มละมีลำดับการทำงานจากสูงไปต่ำ ดังนี้

กลุ่มพิเศษ	**, ABS, NOT
กลุ่มการคูณ	*, /, MOD, REM
กลุ่มเครื่องหมาย	+, -
กลุ่มการบวก	+, -, &
กลุ่มการเลื่อนบิต	SLL, SRL, SLA, SRA, ROL, ROR
กลุ่มความสัมพันธ์	=, /=, <, <=, >, >=
กลุ่มตรรกะ	AND, OR, NAND, NOR, XOR, XNOR

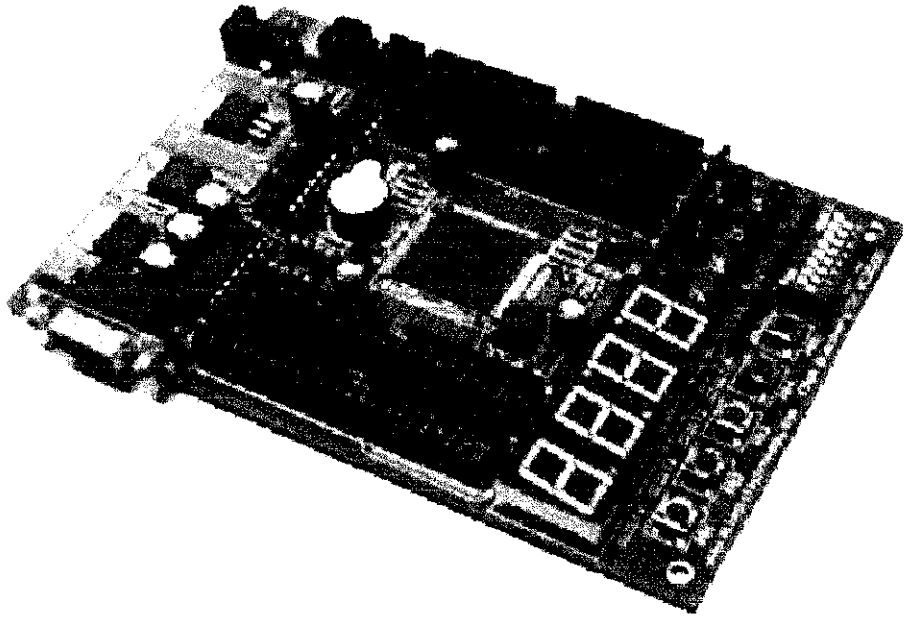
การเขียนคำสั่งแบบลำดับก่อนหลัง (Sequential)

ในการเขียนคำสั่งต่างๆ ที่เป็นแบบ Sequential จะต้องอยู่ภายใต้คำสั่งพิเศษเท่านั้น นั่นคือ คำสั่ง Process นั่นเอง ซึ่งคำสั่งที่เป็นรูปแบบ Sequential ที่มีให้ใช้ในภาษา VHDL มีดังนี้

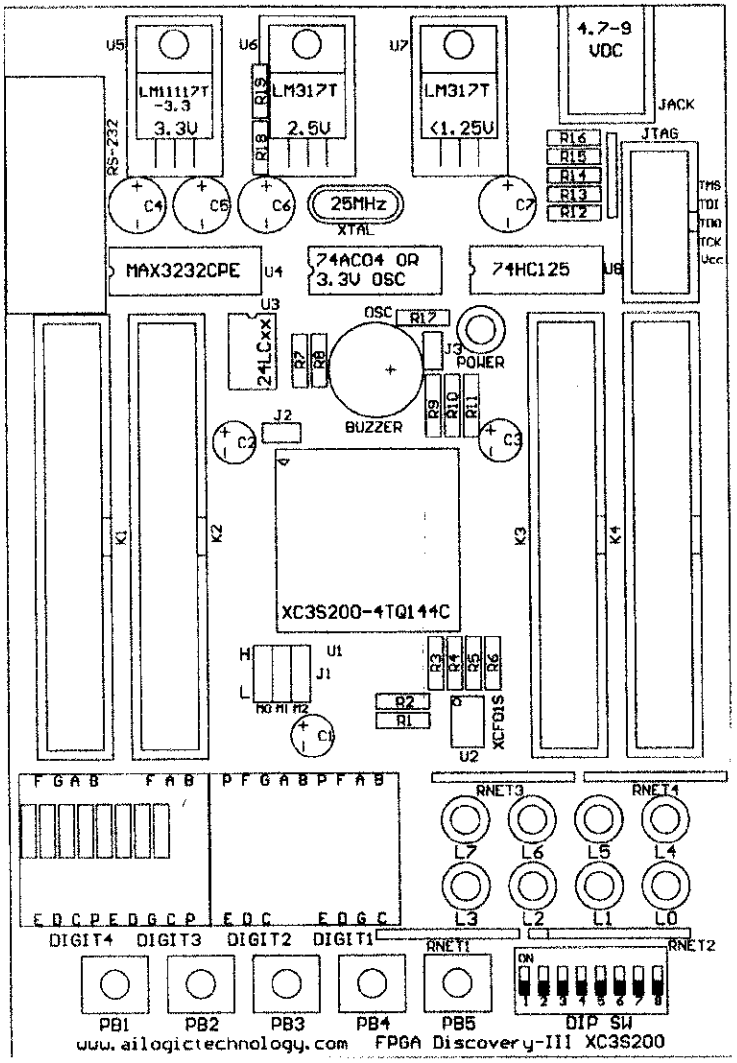
- IF Statement
- CASE Statement
- LOOP Statement
- Wait Statement

3.5 บอร์ด FPGA ที่ใช้ในการทำโครงการ

บอร์ดทดลอง FPGA Discovery-III XC3S400 มีรายละเอียดที่แสดงตามรูป โดยที่บอร์ดนี้จะเป็นได้ทั้งบอร์ดทดลองและบอร์ดพัฒนา FPGA ในบอร์ดเดียวกันที่มีความจุมากถึง 400,000 เกต และใช้ Platform Flash PROM สำหรับเก็บข้อมูลวงจร ซึ่งสามารถโปรแกรมวงจรลง Platform Flash PROM ผ่านทางสายดาวน์โหลดแบบ JTAC ได้โดยตรงและสามารถโปรแกรมได้ซ้ำถึง 20,000 ครั้ง บอร์ดนี้มีอุปกรณ์ที่อำนวยความสะดวกที่เพียบพร้อมด้วยอุปกรณ์อินพุตเอาท์พุตอย่างครบครันเพื่อให้ผู้ทดลองได้เรียนรู้การออกแบบวงจรรวมดิจิตอลตั้งแต่วงจรขั้นพื้นฐานจนถึงขั้นนำไปพัฒนาออกแบบสร้างวงจรขนาดใหญ่ด้วยตัวเอง



รูปที่ 3-5 บอร์ดทดลอง FPGA Discovery-III XC3S400



รูปที่ 3-6 โครงสร้างตัวบอร์ดทดลอง FPGA Discovery-III XC3S400

3.5.1 Connector and Jumper

3.5.1-1 Expansion connector (K1 – K4)

เป็นหัวต่อที่ใช้เชื่อมสัญญาณ I/O จาก FPGA ไปยังบอร์ดหรืออุปกรณ์ภายนอกที่มี I/O เป็น 3.3V โดยจะต่ออยู่กับขา CPLD ดังตารางด้านท้าย ในกรณีที่ I/O ของ FPGA เป็น Output สามารถต่อออกจากบอร์ดไปขับ Input ของอุปกรณ์ที่เป็นระบบ 3.3V และ 5V ได้โดยตรง แต่ถ้า I/O ของ FPGA เป็น Input นั้นจะรับได้เฉพาะ Inputที่เป็นระบบ 3.3V เท่านั้น (ถ้ารับมาจากระบบ 2.5V ต้องใช้ตัวความต้านทาน (R) มา Pull up) แต่ถ้า Inputเป็นระบบ 5V จะต้องใช้บัฟเฟอร์ที่เป็นระบบ 3.3V มากั้นเพื่อป้องกันไม่ให้ I/O ของ FPGA ได้รับความเสีย บัฟเฟอร์ที่เป็นระบบ 3.3V อาจใช้ไอซีตระกูล 74HCxx หรือ 74ACxx (เร็วกว่า) มากินไฟเลี้ยง (Vcc) 3.3 V_{DC} และต้องต่อความต้านทาน 200

โอห์มที่ Input ของบัฟเฟอร์ทุกตัว (เพื่อจำกัดกระแสไม่ให้ Input ของบัฟเฟอร์เสียหาย คือ < 10 mA)

3.5.1-2 JTAG connector

เป็นหัวต่อที่ใช้ต่อกับสายที่ใช้โปรแกรมข้อมูลลงตัว FPGA และ PROM โดยผ่ายทาง JTAG Cable โดยจะมีจัมเปอร์ต่างๆ ดังต่อไปนี้

J1 เป็นจัมเปอร์ J1 ประกอบด้วย M0 , M1 , M2 โดยปกติให้เซทไว้ที่ลอจิก "L" หรืออยู่ในโหมด Master serial เนื่องจากเราสามารถโปรแกรม PROM หรือ FPGA โดยใช้สาย JTAG ได้อยู่แล้วโดยไม่ต้องสนใจตำแหน่งของจัมเปอร์แต่อย่างใด

J2 เป็นจัมเปอร์ที่ใช้ควบคุมให้ FPGA ทำการ Pull up I/O ของ FPGA ทุกขาเมื่อใส่จัมเปอร์ และจะเป็น Hi Impedance เมื่อถอดจัมเปอร์ออก

J3 เป็นจัมเปอร์ที่ใช้ตัด BUZZER ออกจาก FPGA เมื่อถอดจัมเปอร์ออก

3.5.2 Input

3.5.2-1 DIP switch (DIP SW) เป็นชุดของสวิทช์เลื่อนขนาดเล็กที่ใช้ป้อนข้อมูลเข้าสู่ FPGA โดยถ้าเลื่อนลง (Off) จะเป็น "1" ถ้าเลื่อนขึ้น (On) จะเป็น "0" โดยเชื่อมต่อกับขาของ FPGA DIP SW ทุกตัวจึงทำงานแบบ Active Low

Dip SW	FPGA Pinout	Description
1	p52	Dip Switch No.1
2	p53	Dip Switch No.2
3	p55	Dip Switch No.3
4	p56	Dip Switch No.4
5	p59	Dip Switch No.5
6	p60	Dip Switch No.6
7	p63	Dip Switch No.7
8	p68	Dip Switch No.8

ตารางที่ 3-1 แสดงการเชื่อมต่อขาของ FPGA กับ DIP Switch

3.5.2-2 Push button switch (PB1 – PB5) เป็นสวิทซ์กดติดปล่อยดับที่ให้สัญญาณเอ้าท์พุทเป็นระดับลอจิก "0" เมื่อกดสวิทซ์ และเป็นระดับลอจิก "1" เมื่อปล่อยสวิทซ์ โดยจะต่ออยู่กับขา FPGA ดังตารางด้านท้าย Push button switch ทุกตัวจึงทำงานแบบ Active Low

Push Botton	FPGA Pinout	Descriptions
PB1	p44	Push Botton No. 1
PB2	p46	Push Botton No. 2
PB3	p47	Push Botton No. 3
PB4	p50	Push Botton No. 4
PB5	p51	Push Botton No. 5

ตารางที่ 3-2 แสดงการเชื่อมต่อขาของ FPGA กับ Push button switch

3.5.2-3 Changeable oscillator (OSC) เป็นตัวกำเนิดสัญญาณนาฬิกาที่สามารถเปลี่ยนค่าความถี่ที่ต้องการได้ โดยการถอดเปลี่ยนออสซิลเลเตอร์เดิม (3.3V) ที่ให้มาบนบอร์ดออก แล้วใส่ตัวใหม่ (3.3V) เข้าไปแทนที่ที่ซอคเก็ตไอซีเบอร์ 74AC04 โดยที่เอาท์พุทของ OSC จะต่ออยู่กับขา FPGA ซึ่งเป็นขา Global clock เหมาะสำหรับวงจรที่ต้องการความถี่ในการทำงานสูงๆ

Oscillator	FPGA Pinout	Descriptions
OSC	p127	25MHz , GCLK6

ตารางที่ 3-3 แสดงการเชื่อมต่อขาของ FPGA กับ OSC

3.5.3 Output

3.5.3-1 ตัวแสดงผล 7-Segment (DIGIT1 – DIGIT4) เป็นตัวแสดงผลเจ็ดส่วนจำนวน 4 หลักที่สามารถถอดออกได้ (หากต้องการใช้ I/O ที่ Connector K1และ K2 ส่วนที่แชร์ I/O อยู่กับตัวแสดงผลเจ็ดส่วนทั้ง 4 หลัก) โดยเรียงจากซ้ายไปขวาคือ DIGIT4, DIGIT3,DIGIT2 และ DIGIT1 โดยตัวที่ 2 และตัวที่ 1 จะทำการกลับตัวแสดงผลเจ็ดส่วนให้เพื่อใช้จุด (Dot) ในการทำนาฬิกาหรือแสดงองศาในการวัดอุณหภูมิ เช่น 11:39 หรือ 20° ตัวแสดงผลเจ็ดส่วนทั้งหมดจะต่อขาดาต้าเข้าด้วยกันโดยมีขาไฟร่วม (Common cathode) แยกกันสี่ขา ดังนั้นผู้ใช้จึงจำเป็นต้องใช้เทคนิคในการ

สแกน (Scan) เพื่อให้ตัวแสดงผลทั้งเจ็ดส่วนสามารถแสดงผลพร้อมกันได้ทั้งหมด และตัวแสดงผลทั้งหมดเป็นแบบไฟร่วม (Common cathode) โดยจะต่ออยู่กับขา FPGA ดังตารางด้านท้าย

7-Segment	FPGA Pinout	Descriptions
a	p40	a
b	p35	b
c	p32	c
d	p30	d
e	p27	e
f	p25	f
g	p23	g
dp	p20	Decimal Point
DG1	p31	DIGIT1 , COMMON CATHODE
DG2	p33	DIGIT2 , COMMON CATHODE
DG3	p36	DIGIT3 , COMMON CATHODE
DG4	p41	DIGIT4 , COMMON CATHODE

ตารางที่ 3-4 แสดงการเชื่อมต่อขาของ FPGA กับตัวแสดงผล 7 Segment

3.5.3-2 ไฟ LED แสดงผล ไฟ LED แสดงผล L0 – L7 จะต่อแชร์กับ I/O ของ Connecter K3 และ K4 โดยที่ L2 , L3 , L6 และ L7 จะต่อแชร์กับ I/O ของ Connecter K3 โดยมี Resister " RNET3" 8P4R 470 Ohm จำกัดกระแส และ L0 , L1 , L4 และ L5 จะต่อแชร์กับ I/O ของ Connecter K4 โดยมี Resister "RNET3" 8P4R 470 Ohm จำกัดกระแส

3.5.3-3 Buzzer เป็นออดความถี่เสียง (Buzzer) โดยที่จะมีเสียงดังเมื่อป้อนสัญญาณเป็น High "1" โดยจะต่ออยู่กับขา FPGA ดังตารางด้านท้าย กรณีที่ต้องการใช้ I/O ของ Connecter K4 ที่แชร์อยู่กับออดหรือไม่ต้องการใช้ออดให้ถอดจัมเปอร์ J3 ออก

BUZZER	FPGA Pinout	Descriptions
BUZZER	p124	BUZZER

ตารางที่ 3-5 แสดงการเชื่อมต่อขาของ FPGA กับ Buzzer

LED	FPGA Pinout	Descriptions
L0	p70	L0
L1	p77	L1
L2	p69	L2
L3	p76	L3
L4	p74	L4
L5	p79	L5
L6	p73	L6
L7	p78	L7

ตารางที่ 3-6 แสดงการเชื่อมต่อขาของ FPGA กับไฟแสดงผล LED

3.5.4 Misc

3.5.4-1 Jack สำหรับ DC Adaptor เป็นหัวต่อไฟเลี้ยงเพื่อป้อนให้แก่บอร์ดในการทำงาน ต่ออยู่กับ Adapter ที่มีไฟออกมาเป็น 4.7V –9V โดยมีขั้วด้านในเป็นบวก "+" ด้านนอกเป็นลบ "-"

3.5.4-2 Power LED (POWER) เป็นไดโอดเปล่งแสงว่าในขณะนั้นๆ มีไฟเลี้ยงบอร์ดอยู่หรือไม่

3.5.4-3 RS-232C Port เป็นพอร์ต RS-232C ซึ่งหากไม่ต้องการใช้พอร์ต RS-232C แต่ต้องการใช้เป็น I/O ที่ Connector K1 และ K2 (ส่วนที่แชร์ I/O อยู่กับพอร์ต RS-232C) ให้ถอดไอซี MAX3232CPE ออกจาก Socket ซึ่ง ต้องบัดกรีขา 15 ของไอซี MAX3232CPE ลงกราวนด์ด้วย

3.5.4-4 Platform Flash PROM เบอร์ XCF01S เป็น Serial PROM ที่สามารถโปรแกรมได้โดยตรงผ่านทางสาย JTAG สามารถโปรแกรมซ้ำได้ประมาณ 20,000 ครั้ง

3.5.4-5 I2C Socket เป็น Socket สำหรับใส่ไอซีแบบ I2C Serial EEPROM เบอร์ 24LCxx เช่น 24LC256 เป็นต้น ซึ่งสามารถถอดออกได้ (หากต้องการใช้ I/O ที่ Connector K1 และ K2 ส่วนที่แชร์ I/O อยู่กับ I2C Serial EEPROM) I/O 2 ขานี้จะมี Pull up resister 4.7kOhm ต่ออยู่

3.5.5 ตาราง I/O ของ FPGA

K1 Pinout	FPGA Pinout	Descriptions
1	p40	I/O , a
2		3.3 V
3	p35	I/O , b
4		GND
5	p32	I/O , c
6		GND
7	p30	I/O , d
8		GND
9	p27	I/O , e
10		GND
11	p25	I/O , f
12		GND
13	p23	I/O , g
14		GND
15	p20	I/O , dp
16		GND
17	p17	I/O
18		GND
19	p14	I/O
20		GND

k1 Pinout	FPGA Pinout	Descriptions
21	p12	I/O
22		GND
23	p10	I/O
24		GND
25	p7	I/O
26		GND
27	p5	I/O
28		GND
29	p2	I/O
30		GND
31	p141	I/O
32		GND
33	p137	I/O
34		GND
35	p132	I/O, RS-232 (RX)
36		GND
37	p130	I/O
38		GND
39	p128	I/O , GCLK7 , I2C-SCL
40		GND

ตารางที่ 3-7 แสดงการเชื่อมต่อขาของ FPGA กับ K1

K1 Pinout	FPGA Pinout	Descriptions
1	p40	I/O , a
2		3.3 V
3	p35	I/O , b
4		GND
5	p32	I/O , c
6		GND
7	p30	I/O , d
8		GND
9	p27	I/O , e
10		GND
11	p25	I/O , f
12		GND
13	p23	I/O , g
14		GND
15	p20	I/O , dp
16		GND
17	p17	I/O
18		GND
19	p14	I/O
20		GND

k1 Pinout	FPGA Pinout	Descriptions
21	p12	I/O
22		GND
23	p10	I/O
24		GND
25	p7	I/O
26		GND
27	p5	I/O
28		GND
29	p2	I/O
30		GND
31	p141	I/O
32		GND
33	p137	I/O
34		GND
35	p132	I/O, RS-232 (RX)
36		GND
37	p130	I/O
38		GND
39	p128	I/O , GCLK7 , I2C-SCL
40		GND

ตารางที่ 3-8 แสดงการเชื่อมต่อขาของ FPGA กับ K2

K3 Pinout	FPGA Pinout	Descriptions
1		3.3 V
2	p69	I/O , L2
3		GND
4	p73	I/O , L6
5		GND
6	p76	I/O , L3
7		GND
8	p78	I/O , L7
9		GND
10	p80	I/O
11		GND
12	p83	I/O
13		GND
14	p85	I/O
15		GND
16	p87	I/O
17		GND
18	p90	I/O
19		GND
20	p93	I/O

k3 Pinout	FPGA Pinout	Descriptions
21		GND
22	p96	I/O
23		GND
24	p98	I/O
25		GND
26	p100	I/O
27		GND
28	p103	I/O
29		GND
30	p105	I/O
31		GND
32	p108	I/O
33		GND
34	p113	I/O
35		GND
36	p118	I/O
37		GND
38	p122	I/O
39		GND
40	p124	I/O , GCLK4 , BUZZER

ตารางที่ 3-9 แสดงการเชื่อมต่อขาของ FPGA กับ K3

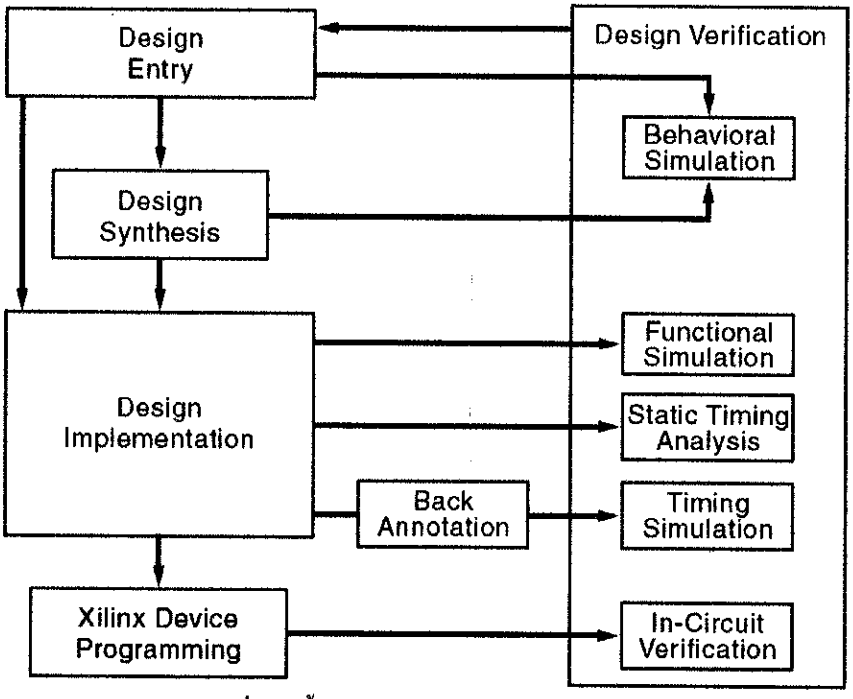
K4 Pinout	FPGA Pinout	Descriptions
1		3.3 V
2	p70	I/O , L0
3		GND
4	p74	I/O , L4
5		GND
6	p77	I/O , L1
7		GND
8	p79	I/O , L5
9		GND
10	p82	I/O
11		GND
12	p84	I/O
13		GND
14	p86	I/O
15		GND
16	p89	I/O
17		GND
18	p92	I/O
19		GND
20	p95	I/O

k4 Pinout	FPGA Pinout	Descriptions
21		GND
22	p97	I/O
23		GND
24	p99	I/O
25		GND
26	p102	I/O
27		GND
28	p104	I/O
29		GND
30	p107	I/O
31		GND
32	p112	I/O
33		GND
34	p116	I/O
35		GND
36	p119	I/O
37		GND
38	p123	I/O
39		GND
40	p125	I/O , GCLK5

ตารางที่ 3-10 แสดงการเชื่อมต่อขาของ FPGA กับ K4

3.6 โปรแกรมที่ใช้เขียนบน FPGA

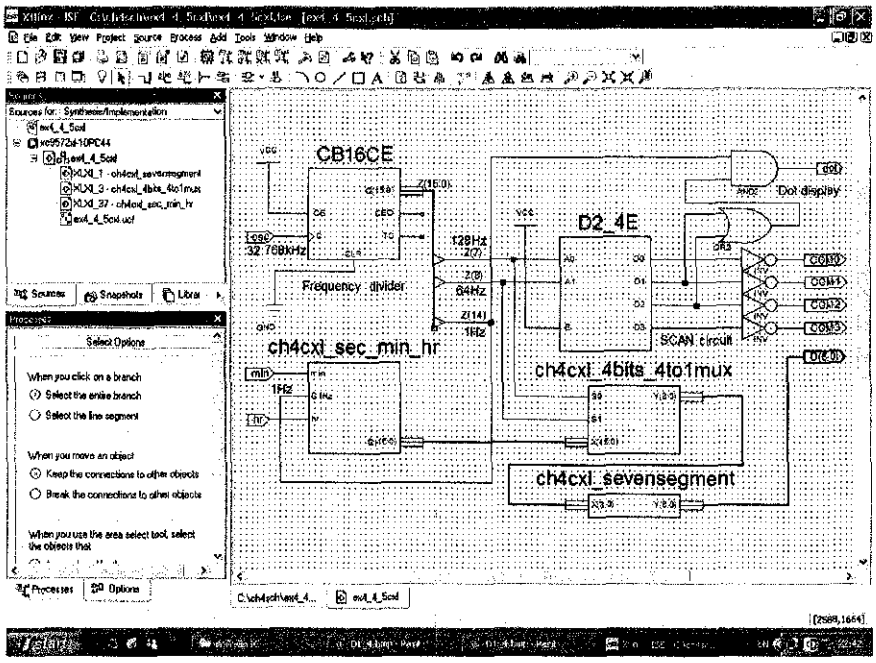
ตัวชิฟของบอร์ดที่ใช้ในโครงการนั้น เป็นชิฟของบริษัท Xillinx ซึ่งมีซอร์ฟแวร์ในการโปรแกรมคำสั่งลงบนตัว FPGA ในที่นี้โปรแกรมที่ใช้คือ ISE 8.1i ซึ่งขั้นตอนในการออกแบบมีดังนี้



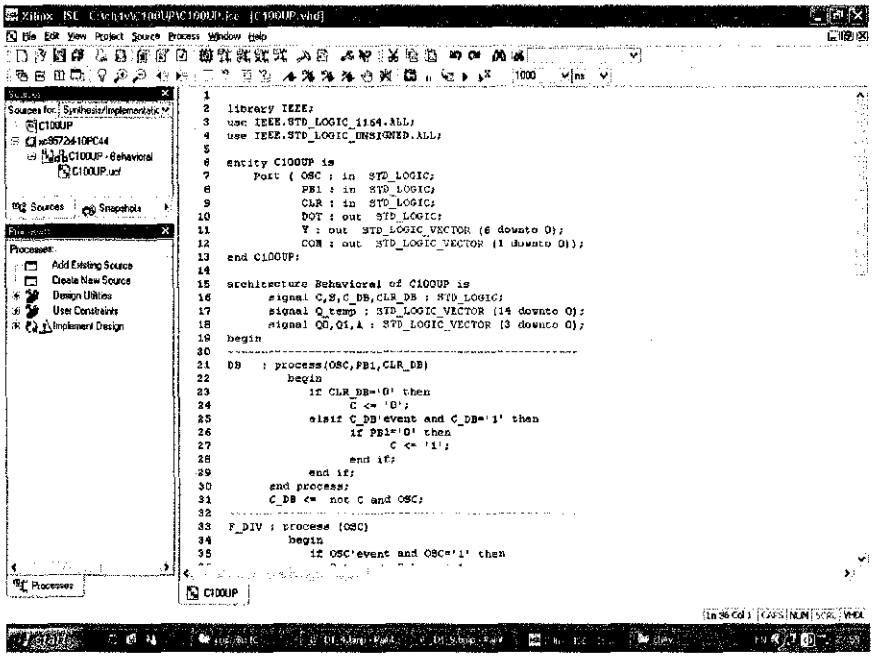
รูปที่ 3-7 ขั้นตอนการออกแบบวงจรดิจิตอลด้วย FPGA

3.6.1 การออกแบบวงจร

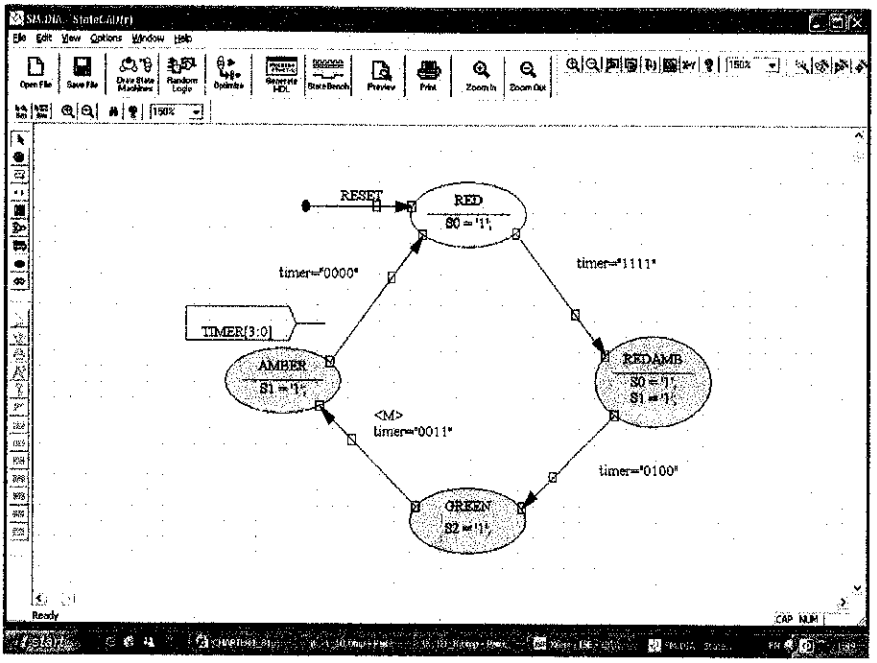
ในการออกแบบวงจรดิจิตอลใน ISE 8.1i สามารถทำได้หลายวิธีคือ วิธีการวาดผังวงจร, วิธีโปรแกรมด้วยภาษา VHDL หรือออกแบบด้วยการเขียนแผนภูมิสถานะเพื่อให้ได้วงจรตามที่ต้องการ นอกจากนี้ยังมีซอฟแวร์ทูลเพื่อจำลองผลการทำงานของโปรแกรมอีกด้วย โดยที่เราสามารถกำหนดค่าพารามิเตอร์ต่างๆ และการจำลองการทำงานของโปรแกรมเพื่อตรวจสอบความถูกต้องของโปรแกรมได้



รูปที่ 3-8 การออกแบบวงจรด้วยวิธีวาดผังวงจร



รูปที่ 3-9 การออกแบบวงจรด้วยภาษา VHDL



รูปที่ 3-10 การออกแบบวงจรด้วยการเขียน Stage Diagram

บทที่ 4
การดำเนินโครงการและผลการดำเนินโครงการ

4.1 ขอบเขตของโครงการ

โครงการนี้เป็นการสร้างต้นแบบของระบบ MIMO โดยใช้บอร์ด FPGA สองตัว ซึ่งบอร์ดตัวที่หนึ่งแทนระบบภาคส่งและตัวที่สองแทนระบบภาครับ ส่วนในการรับส่งข้อมูลในโครงการนี้จะใช้สายเคเบิลแทนการส่งสัญญาณจริงด้วยสายอากาศ ระบบ MIMO ที่สร้างขึ้นนี้มีการเข้ารหัสถอดรหัสด้วยวิธี 2x2 Alamouti และมอดูเลทสัญญาณแบบ BPSK

4.2 โครงสร้างการทำงานของโปรแกรม

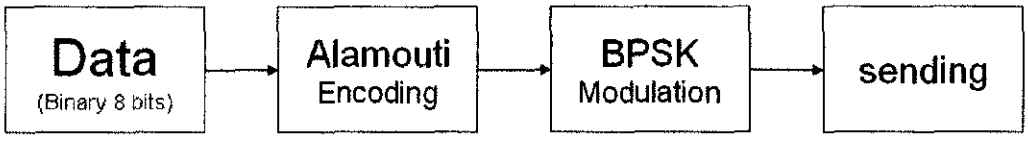
4.2.1 โครงสร้างการทำงานของโปรแกรมภาคส่ง ในภาคส่งจะมีลำดับขั้นตอนการทำงานดังต่อไปนี้นี้

1. รับข้อมูล 8 bit จาก DIP Switch

2. นำข้อมูลมาเข้ารหัสแบบ 2x2 Alamouti แยกเป็นข้อมูลสองชุด ซึ่งจำลองเป็นข้อมูลที่สายอากาศตัวที่ 1 และสายอากาศตัวที่ 2 จะต้องส่งออกอากาศ

3. ข้อมูลที่ได้จากการเข้ารหัสนำมาทำการ Modulate แบบ BPSK ซึ่งในโครงการนี้ทำการจำลองการส่งแบบ BPSK ด้วยการไล่ระดับบิตจากสูงไปต่ำและจากต่ำไปสูงแทนสัญญาณไซน์บิต '1' และบิต '0'

4. ส่งข้อมูลที่กล่าวไว้ในข้อ 3 ไปตามสายเคเบิลสองเส้น โดยเคเบิลแต่ละเส้นจะเชื่อมสายอากาศภาคส่งและรับโดยตรง



รูปที่ 4-1 แผนผังโครงสร้างโปรแกรมภาคส่ง

คำสั่งโปรแกรมภาคส่ง

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;


entity mimo_tx is
    Port ( clk,reset : in  STD_LOGIC;
           input : in  STD_LOGIC_VECTOR (7 downto 0);
           tx_1,tx_2,Data : out  STD_LOGIC_VECTOR (7 downto 0));
end mimo_tx;

architecture Behavioral of mimo_tx is
      signal Bclk,Cclk,res_x,Ccount1,Ccount2,p_out1,p_out2 : std_logic;
      signal Start1,start2 : bit;
      signal N : integer range 0 to 7:=7;
      signal Nx : integer range 0 to 9:=9;
      signal C_start1 : integer range 0 to 20800 := 20800;
      signal C_Bclk : integer range 0 to 2600 := 2600;
      signal C_start2: integer range 0 to 20800 := 20800;
      signal C_Cclk : integer range 0 to 260 := 260;
      signal Ant1,Ant2 : STD_LOGIC_VECTOR (7 downto 0);

begin
      Data<=input;
--------Alarmouti 2x2---------
      Ant1(7)<=input(7);
      Ant1(6)<=not input(6);
      Ant1(5)<=input(5);
      Ant1(4)<=not input(4);
      Ant1(3)<=input(3);
      Ant1(2)<=not input(2);
      Ant1(1)<=input(1);
      Ant1(0)<=not input(0);

      Ant2(7)<=input(6);
      Ant2(6)<=input(7);
      Ant2(5)<=input(4);
      Ant2(4)<=input(5);
      Ant2(3)<=input(2);
      Ant2(2)<=input(3);
      Ant2(1)<=input(0);
      Ant2(0)<=input(1);

 count1: Process(clk)
 begin
 res_x <= reset;
 Ccount1<= reset and (not res_x);
 if (rising_edge(clk)) then if Ccount1='1' then C_start1<=0;
                            else if C_start1/=20800 then C_start1<=C_start1+1; end if;
                            end if;
 end if;
 end process;

 count2: Process(clk)
 begin
 res_x <= reset;
 Ccount2<= reset and (not res_x);
 if (rising_edge(clk)) then if Ccount2='1' then C_start2<=0;
                            else if C_start2/=20800 then C_start2<=C_start2+1; end if;
                            end if;
```

```
end if;
end process;

count_start1: process(C_start1)
begin

case C_start1 is
   when 0 to 20799 =>
      start1 <= '1';
   when 20800 =>
      start1 <= '0';
   end case;
 end process;

count_start2: process(C_start2)
begin
case C_start2 is
   when 0 to 20799 =>
      start2 <= '1';
   when 20800 =>
      start2 <= '0';
   end case;
end process;


 GenClock: process(clk)
 begin
 if (rising_edge(clk)) then
   if start1 = '1' and C_Bclk = 2600 then
    C_Bclk <= 0;
    Bclk <= '1';
   elsif start1 = '1' and C_Bclk /= 2600 then
    C_Bclk <= C_Bclk+1;
    Bclk <= '0';
   elsif start1 = '0' then
    C_Bclk <= 2600;
   end if;
end if;
end process;
GenCclk: process(clk)
begin
 if (rising_edge(clk)) then
   if start2 = '1' and C_Cclk=260 then
    C_Cclk <=0;
    Cclk<='1';
   elsif start2 ='1' and C_Cclk/=260 then
    C_Cclk <= C_Cclk+1;
    Cclk<='0';
    elsif start2 = '0' then
    C_Cclk <= 260;
 end if;
 end if;
 end process;

-------------------->>>>sending<<<<--------------------
count_N: process(Bclk)
 begin
  if Bclk = '1' then
      if N = 7 then N <= 0;
      else N <= N+1;
      end if;
  end if;
 end process;
```

```
Count_Nx: process(Cclk)
 begin
  if Cclk = '1' then
      if Nx = 9 then Nx <= 0;
      else Nx <= Nx+1;
      end if;
  end if;
 end process;

-->>>>>pout 1 bit/8 times--->>
position_bit: process(N)
begin
if N=0 then p_out1<=Ant1(7); p_out2<=Ant2(7);
elsif N=1 then p_out1<=Ant1(6); p_out2<=Ant2(6);
elsif N=2 then p_out1<=Ant1(5); p_out2<=Ant2(5);
elsif N=3 then p_out1<=Ant1(4); p_out2<=Ant2(4);
elsif N=4 then p_out1<=Ant1(3); p_out2<=Ant2(3);
elsif N=5 then p_out1<=Ant1(2); p_out2<=Ant2(2);
elsif N=6 then p_out1<=Ant1(1); p_out2<=Ant2(1);
elsif N=7 then p_out1<=Ant1(0); p_out2<=Ant2(0);
end if;
end process;

tx : process (N,Nx)
begin
if p_out1<='0' then if Nx=0  then tx_1<="00000000";
         elsif Nx=1 then tx_1<="00000001";
         elsif Nx=2 then tx_1<="00000010";
         elsif Nx=3 then tx_1<="00000100";
         elsif Nx=4 then tx_1<="00001000";
         elsif Nx=5 then tx_1<="00010000";
         elsif Nx=6 then tx_1<="00100000";
         elsif Nx=7 then tx_1<="01000000";
         elsif Nx=8 then tx_1<="10000000";
         elsif Nx=9 then tx_1<="00000000";
         else tx_1<="11111111";
         end if;
else     if Nx=0    then tx_1<="00000000";
         elsif Nx=1 then tx_1<="10000000";
         elsif Nx=2 then tx_1<="01000000";
         elsif Nx=3 then tx_1<="00100000";
         elsif Nx=4 then tx_1<="00010000";
         elsif Nx=5 then tx_1<="00001000";
         elsif Nx=6 then tx_1<="00000100";
         elsif Nx=7 then tx_1<="00000010";
         elsif Nx=8 then tx_1<="00000001";
         elsif Nx=9 then tx_1<="00000000";
         else tx_1<="11111111";
        end if;
end if;

if p_out2<='0' then if Nx=0  then tx_2<="00000000";
         elsif Nx=1 then  tx_2<="00000001";
         elsif Nx=2 then  tx_2<="00000010";
         elsif Nx=3 then  tx_2<="00000100";
         elsif Nx=4 then  tx_2<="00001000";
         elsif Nx=5 then  tx_2<="00010000";
         elsif Nx=6 then  tx_2<="00100000";
         elsif Nx=7 then  tx_2<="01000000";
         elsif Nx=8 then  tx_2<="10000000";
         elsif Nx=9 then  tx_2<="00000000";
         else tx_2<="11111111";
         end if;

end if;
```

```
else      if Nx=0    then  tx_2<="00000000";
          elsif Nx=1 then  tx_2<="10000000";
          elsif Nx=2 then  tx_2<="01000000";
          elsif Nx=3 then  tx_2<="00100000";
          elsif Nx=4 then  tx_2<="00010000";
          elsif Nx=5 then  tx_2<="00001000";
          elsif Nx=6 then  tx_2<="00000100";
          elsif Nx=7 then  tx_2<="00000010";
          elsif Nx=8 then  tx_2<="00000001";
          elsif Nx=9 then  tx_2<="00000000";
          else tx_2<="11111111";
         end if;
end if;
end process;

end Behavioral;
```

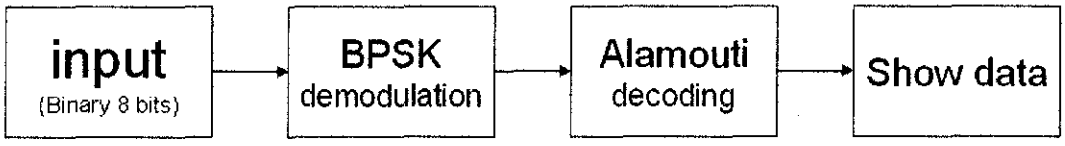
4.2.2 โครงสร้างการทำงานของโปรแกรมภาครับ ในภาครับจะมีลำดับขั้นตอนการทำงานดังต่อไปนี้

1. รับข้อมูล 8 bits สองชุดที่ส่งมาตามสายเคเบิล ซึ่งเป็นข้อมูลที่เข้ารหัสและถูกส่งมาโดยภาครับ

2. ข้อมูลที่ได้จากการเข้ารหัสนำมาทำการ Demodulate แบบ BPSK

3. ทำการถอดรหัสสัญญาณที่ได้จากการ Demodulate

4. แสดงค่าข้อมูลที่รับได้ทาง LED ของ FPGA Board



รูปที่ 4-2 แผนผังโครงสร้างโปรแกรมภาครับ

คำสั่งโปรแกรมภาครับ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity mimo_rx is
    Port ( Rx1,Rx2 : in  STD_LOGIC_VECTOR (7 downto 0);
           clk : in std_logic;
           Data : out  STD_LOGIC_VECTOR (7 downto 0));
end mimo_rx;

architecture Behavioral of mimo_rx is
      signal chk_bit,start,start_chk : std_logic:='0';
      signal N : integer range 0 to 7:=7;
      signal count : integer range 0 to 20800:=20800;
      signal count_chk : integer range 0 to 2600:=2600;
      signal Rx_1,Rx_2,Data_1,Data_2 : std_logic_vector (7 downto 0);

begin
--Data<=Data_1;

beginn : process (Rx1,Rx2)
begin
if Rx1/="00000000" or Rx2/="00000000" then start<='1';
else if Rx1="00000000" and Rx2="00000000" then start<='0'; end if;
end if;
end process;

countstart: process(clk)
begin
if (rising_edge(clk)) then if start='1' then
                              if count=20800 then count<=0;
                              else if count/=20800 then count<=count+1; end if;
                              end if;
                           end if;
end if;
end process;

start2: process(count)
begin
--if (rising_edge(clk)) then
if start='1' then
   if count>=520 and count<20800 then start_chk<='1';
   else start_chk<='0';
   end if;
else if start='0' then start_chk<='0'; end if;
end if;
end process;
```

```
check_bit: process(clk)
begin
if (rising_edge(clk)) then
   if start='1' and count_chk=2600 then
         count_chk<=0;
         chk_bit<='1';
   elsif start='1' and count_chk/=2600 then
         count_chk<=count_chk+1;
         chk_bit<='0';

   elsif start='0' then count_chk<=2600;
   end if;
end if;
end process;


count_N: process(chk_bit)
begin
if chk_bit='1' then
   if N = 7 then N <= 0;
      else N <= N+1;
      end if;
  end if;
 end process;
N_Rx1: process(N)
begin
if chk_bit='1' then if N=0  then if Rx1="00000001" then Rx_1(7)<='0'; Data_1(7)<='0';
                            else Rx_1(7)<='1'; Data_1(7)<='1';
                            end if;

                elsif N=1  then if Rx1="00000001" then Rx_1(6)<='0'; Data_1(6)<='1';
                            else Rx_1(6)<='1'; Data_1(6)<='0';
                            end if;

                elsif N=2  then if Rx1="00000001" then Rx_1(5)<='0'; Data_1(5)<='0';
                            else Rx_1(5)<='1'; Data_1(5)<='1';
                            end if;

                elsif N=3  then if Rx1="00000001" then Rx_1(4)<='0'; Data_1(4)<='1';
                            else Rx_1(4)<='1';  Data_1(4)<='0';
                            end if;

                elsif N=4  then if Rx1="00000001" then Rx_1(3)<='0'; Data_1(3)<='0';
                            else Rx_1(3)<='1';  Data_1(3)<='1';
                            end if;

                elsif N=5  then if Rx1="00000001" then Rx_1(2)<='0'; Data_1(2)<='1';
                            else Rx_1(2)<='1'; Data_1(2)<='0';
                            end if;

                elsif N=6  then if Rx1="00000001" then Rx_1(1)<='0'; Data_1(1)<='0';
                            else Rx_1(1)<='1'; Data_1(1)<='1';
                            end if;

                elsif N=7  then if Rx1="00000001" then Rx_1(0)<='0'; Data_1(0)<='1';
                            else Rx_1(0)<='1'; Data_1(0)<='0';
                            end if;
                end if;
else Rx_1<="00000000";
end if;
end process;
```

```
output: process(Data_1,Data_2)
begin
if  Data_1=Data_2 then Data<=Data_1;
else if Data_1/=Data_2 then Data<="00000000"; end if;
end if;
end process;
end Behavioral;
```

4.3 การจำลองผลการทำงานของโปรแกรม

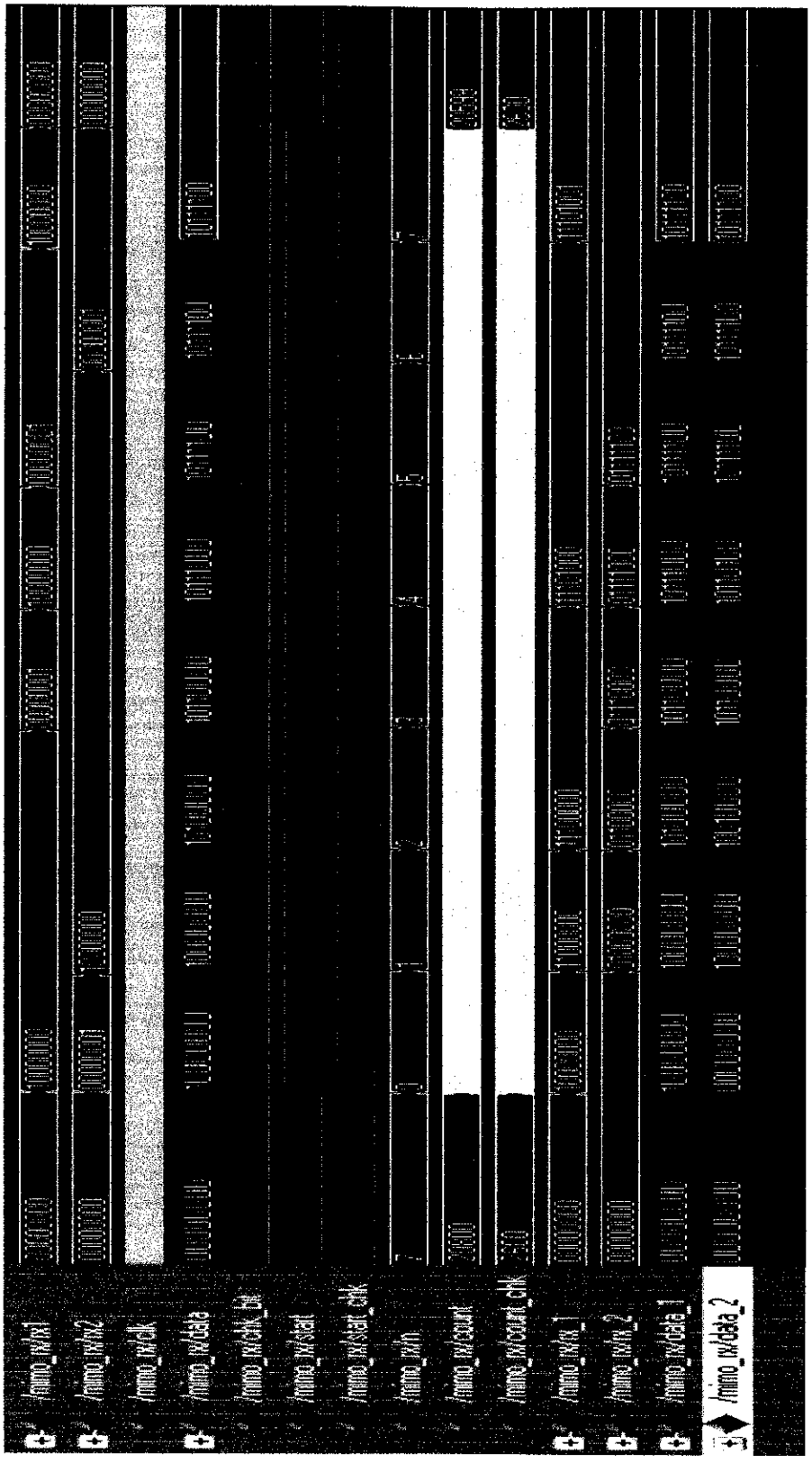
ในโครงการนี้มีการจำลองการทำงานของโปรแกรมโดยใช้ Modelsim XE ซึ่งผลของการจำลองโปรแกรมภาคส่งเป็นดังรูป 4-3 และการจำลองผลภาครับเป็นดังรูป 4-4

จากภาพการจำลองการทำงานของโปรแกรมภาคส่ง (รูปที่ 4-3) tx_1 ซึ่งแทนสายอากาศตัวที่ 1 มีการส่งข้อมูลที่เข้ารหัสแล้วออกไป จะเห็นได้ว่าโปรแกรมมีการส่งข้อมูลบิตไล่ระดับ เพื่อจำลองการส่งข้อมูลแบบ BPSK เช่นเดียวกันกับสายอากาศตัวที่ 2 (tx_2) ซึ่งมีการส่งบิตไล่ระดับเช่นเดียวกัน

มีการโปรแกรมให้ระบบภาคส่งมีการส่งข้อมูล 8 ครั้ง ซึ่งในแต่ละครั้งแสดงบิต 0 หรือ 1 ที่ได้จากการเข้ารหัส โดยบิต '0' หรือ '1' ที่ส่งออกมีการจำลองการส่ง BPSK ด้วย ในการควบคุมการทำงานโดยขึ้นกับเวลานี้จะมีสัญญาณนาฬิกาควบคุมคือตัวแปร clk ซึ่งเป็นสัญญาณจากตัว FPGA Board โดยที่มีความถี่ 25 MHz

จากภาพการจำลองการทำงานของโปรแกรมภาครับ (รูปที่ 4-4) จะเห็นได้ว่าโปรแกรมภาครับมีการใช้ clk ซึ่งเป็นจังหวะเดียวกันกับของโปรแกรมภาคส่ง เพื่อไม่ให้เกิดปัญหา synchronization มีการโปรแกรมให้ข้อมูลที่รับมาเป็น Rx1 และ Rx2 จากนั้นจะมีสัญญาณเพื่อตรวจสอบว่าข้อมูลจำลอง BPSK ที่ส่งมานั้นเป็นข้อมูลดิจิตอลบิตสูงหรือต่ำ (บิต 1 หรือบิต 0) เมื่อได้ข้อมูลมาแล้วนำข้อมูลมาเข้ารหัสด้วยวิธี 2x2 Alamouti และมีการเปรียบเทียบกันระหว่างข้อมูลถอดรหัสที่ได้จาก Rx1 และ Rx2 หากข้อมูลที่ถอดรหัสได้ตรงกันแสดงว่าข้อมูลที่ถอดรหัสมานั้นเป็นข้อมูลที่ต้นทางส่งมา

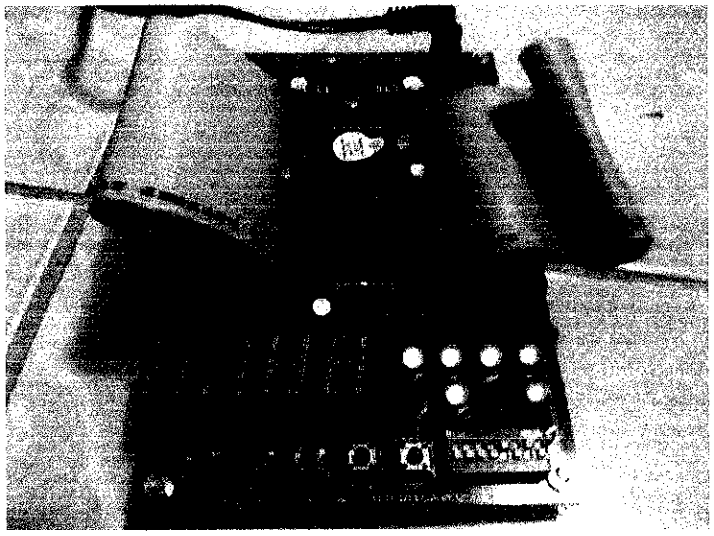
รูปที่ 4-3 ผลการจำลองโปรแกรมภาคส่ง



รูปที่ 4-4 ผลการจำลองโปรแกรมภาครับ

4.4 การทดลองและผลการทดลอง

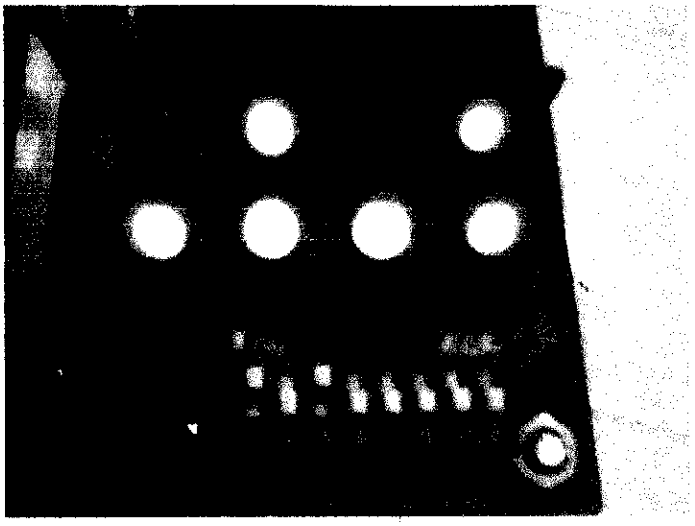
ในการทดลองส่งและรับข้อมูลพบว่า ระบบมีการทำงานเป็นไปตามทฤษฎี ข้อมูลที่รับได้มีความถูกต้อง ดังแสดงต่อไปนี้



รูปที่ 4-5 ต้นแบบระบบ MIMO ที่สร้างเสร็จแล้ว

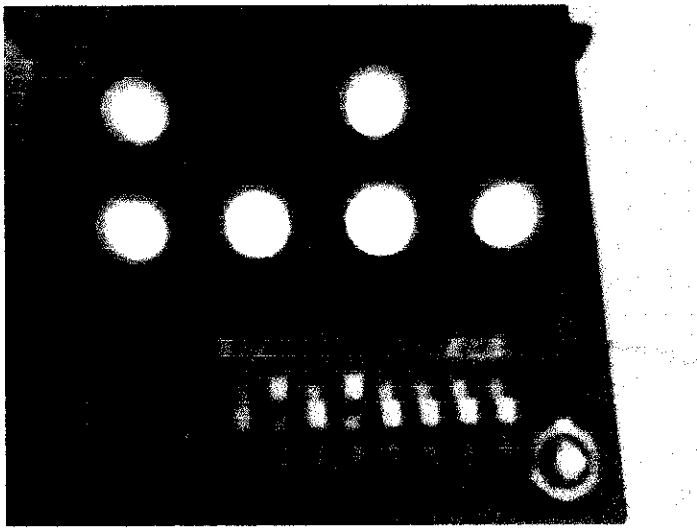
รูปที่ 4-6 Input ของระบบ

Input ของระบบ (ข้อมูลที่ต้องการส่งก่อนเข้าระบบ) เราให้เป็น DIP Switch แทนข้อมูลเลขฐานสอง 8 บิต และให้สัญญาณไฟ LED เป็น Output ของระบบ (สัญญาณที่รับมาและถอดรหัสแล้ว) จะเห็นผลดังต่อไปนี้



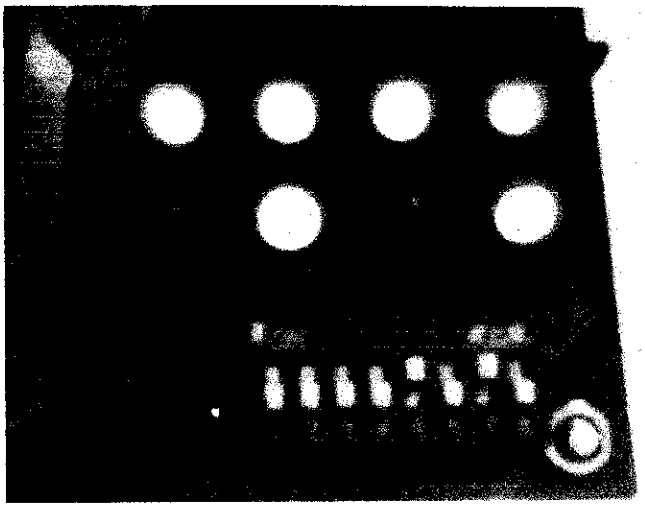
รูปที่ 4-7 ผลการทดสอบที่ 1

จากรูปข้อมูลเข้าเป็น 01011111 และผลที่ได้ที่ภาครับคือ 01011111 เช่นเดียวกัน แสดงว่าต้นแบบระบบ MIMO ที่สร้างขึ้นมีการทำงานที่ถูกต้อง



รูปที่ 4-8 ผลการทดสอบที่ 2

จากรูปข้อมูลเข้าเป็น 10101111 และผลที่ได้ที่ภาครับคือ 10101111 เช่นเดียวกัน แสดงว่าต้นแบบระบบ MIMO ที่สร้างขึ้นมีการทำงานที่ถูกต้อง



รูปที่ 4-9 ผลการทดสอบที่ 3

จากรูปข้อมูลเข้าเป็น 11110101 และผลที่ได้ที่ภาครับคือ 11110101 เช่นเดียวกัน แสดงว่าต้นแบบระบบ MIMO ที่สร้างขึ้นมีการทำงานที่ถูกต้อง

4.5 สรุปผลการดำเนินโครงการ

ระบบ MIMO ต้นแบบที่สร้างขึ้นในโครงการนี้ แสดงให้เห็นว่าการนำต้นแบบระบบที่สร้างขึ้นไปใช้งานจริงก็ยังมีความเป็นไปได้ แต่ต้องอาศัยการปรับปรุงและเพิ่มเติมในส่วนออกอากาศจริง

บรรณานุกรม

1. Ian Griffiths, "FPGA Implementation of MIMO Wireless Communications System", The University of Newcastle, 2005.
2. David Gesbert, "From Theory to Practice: An Overview of MIMO Space–Time Coded Wireless Systems", IEEE Journal, April 2003.
3. Sudhakar Yalamanchili, "VHDL Starter's Guide", Prentice Hall, Upper Saddle River, New Jersey 07458
4. Apex Instrument CO.,LTD, "FPGA Discovery-III XCS400 Board Manual", 2007
5. Apex Instrument CO.,LTD, "สร้างไอซีดิจิตอลสมัยใหม่ด้วย FPGA และ CPLD โดยใช้ ISE WebPack 8.1i-9.1i", 2007
6. ชำนาญ ปัญญาใส, วัชรากร หนูทอง, "ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล", ซีเอ็ดยูเคชั่น, 2005