การออกแบบตัวควบคุมเทอร์มินัลโนดโดยใช้เอฟพีจีเอ

นายอุดมศักดิ์ ใช้ศรีทอง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2545

ISBN 974-17-3373-9

DESIGN OF A TERMINAL NODE CONTROLLER USING FPGA

Mr. Udomsak Chaisrithong

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2002

ISBN 974-17-3373-9

หัวข้อวิทยานิพนธ์	การออกแบบตัวควบคุมเทอร์มินัลโนดโดยใช้เอฟพีจีเอ
โดย	นายอุดมศักดิ์ ใช้ศรีทอง
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา	รองศาสตราจารย์กฤษดา วิศวธีรานนท์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดี คณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ศาสตราจารย์ ดร.มงคล เดชนครินทร์)

..... อาจารย์ที่ปรึกษา
(รองศาสตราจารย์กฤษดา วิศวธีรานนท์)

..... กรรมการ
(รองศาสตราจารย์ ดร.เอกชัย ลีลารัศมี)

..... กรรมการ
(อาจารย์สุวิทย์ นาคพีระยุทธ)

อุดมศักดิ์ ใช้ศรีทอง : การออกแบบตัวควบคุมเทอร์มินัลโนดโดยใช้เอฟพีจีเอ
(DESIGN OF A TERMINAL NODE CONTROLLER USING FPGA)
อาจารย์ที่ปรึกษา : รศ.กฤษดา วิศวธีรานนท์, 149 หน้า, ISBN 974-17-3373-9

วิทยานิพนธ์ฉบับนี้กล่าวถึงการออกแบบและสร้างตัวควบคุมเทอร์มินัลโนดสำหรับเครือข่ายวิทยุกลุ่มข้อมูล เพื่อใช้สื่อสารข้อมูลระหว่างคอมพิวเตอร์กับคอมพิวเตอร์ ผ่านระบบสื่อสารแบบคลื่นวิทยุย่านความถี่ UHF โดยใช้โพรโทคอล AX.25 ด้วยวิธีการเข้าใช้ช่องสัญญาณแบบซีเอสเอ็มเอ (Carrier Sense Multiple Access: CSMA) อินเตอร์เฟสกับคอมพิวเตอร์ หรืออุปกรณ์ปลายทางผ่านการสื่อสารแบบอนุกรม RS-232-C การออกแบบเน้นที่การนำเอฟพีจีเอ (Field Programmable Gate Array: FPGA) มาใช้ในการสร้างส่วนควบคุมหลักของตัวควบคุมเทอร์มินัลโนด ได้แก่ วงจรแปลงสัญญาณ RS-232, ตัวควบคุมการแบ่งเฟรมข้อมูล (High-level Data Link Controller: HDLC) และวงจรแปลงสัญญาณแบบ NRZI (Non-Return to Zero Invert) เพื่อเพิ่มประสิทธิภาพการทำงาน และลดขนาดของตัวควบคุมเทอร์มินัล

ภาควิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต .....
สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่ออาจารย์ที่ปรึกษา .....
ปีการศึกษา 2545

4270671921 : MAJOR ELECTRICAL ENGINEERING

KEYWORD : TERMINAL NODE CONTROLLER (TNC) / AX.25 / FPGA

UDOMSAK CHAISRITHONG : DESIGN OF A TERMINAL NODE CONTROLLER USING FPGA. THESIS ADVISOR : KRISADA VISAVATEERANON, ASSOC. PROF., 149 pp.

ISBN 974-17-3373-9

This thesis describes a design and construction of a terminal node controller (TNC) for packet radio network that can be used for computer-to-computer communication via a UHF radio transmission system, by using the AX.25 protocol with CSMA (carrier sense multiple access) channel control. This terminal node controller is designed to interface with a computer or data terminal equipment (DTE) via the EIA-standard: RS-232-C. The design emphasizes in increasing performance and reducing circuit size of terminal node controller by using an advantage of field programmable gate array (FPGA). This FPGA is used to implement a central control unit of terminal node controller, including an RS-232 converter, a high-level data link controller (HDLC) and a non-return to zero invert (NRZI) converter.

Department Electrical Engineering Student's signature .....

Field of study Electrical Engineering Advisor's signature .....

Academic year 2002

กิตติกรรมประกาศ

ผู้วิจัยขอขอบคุณท่านอาจารย์ที่ปรึกษา รองศาสตราจารย์กฤษดา วิศวธีรานนท์ เป็นอย่างยิ่ง ที่ได้กรุณาสละเวลาให้คำปรึกษา แนะนำ พร้อมทั้งจัดหาอุปกรณ์ที่ใช้ในการวิจัยด้วยดีตลอดมา ช่วยให้วิทยานิพนธ์นี้สำเร็จไปด้วยดี

ขอขอบคุณ ศาสตราจารย์ ดร.มงคล เดชนครินทร์, รองศาสตราจารย์ ดร.เอกชัย ลีลารัศมี และอาจารย์สุวิทย์ นาคพีระยุทธ ที่กรุณาสละเวลาอันมีค่าในการเป็นกรรมการในการสอบวิทยานิพนธ์

ขอขอบคุณ คุณครูและอาจารย์ทุกท่าน ที่ให้การศึกษาและอบรมสั่งสอนผู้วิจัยด้วยดีตลอดมาจนสำเร็จการศึกษา

ขอขอบคุณ เพื่อนพี่น้องนิสิตห้องปฏิบัติการวิจัยระบบเชิงเลข และห้องปฏิบัติการออกแบบอิเล็กทรอนิกส์ทุกท่าน ที่มีส่วนช่วยให้คำแนะนำ ข้อคิดเห็น และกำลังใจแก่ผู้วิจัยตลอดระยะเวลาการศึกษา

ขอขอบคุณ คุณนวพรรณ สรหงษ์ ที่ช่วยพิมพ์และเตรียมต้นฉบับวิทยานิพนธ์นี้, คุณโสรัตย์ อุนหะวรากร ที่ให้คำแนะนำด้านโปรแกรมทดสอบบนคอมพิวเตอร์ และบริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) ที่ให้สถานที่ และเครื่องมือที่ใช้ในการทำวิจัย

ขอขอบคุณ บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย ที่ให้ทุนสนับสนุนการวิจัย

ท้ายนี้ ผู้วิจัยขอกราบขอบพระคุณบิดามารดา ที่ได้อบรมเลี้ยงดู สนับสนุนด้านการศึกษา และให้กำลังใจจนผู้วิจัยมีวันนี้ได้

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

หน้า

สารบัญภาพ (ต่อ)

หน้า

บทที่ 1

บทนำ

1.1 แนวเหตุผลในการทำวิทยานิพนธ์

เครือข่ายวิทยุกลุ่มข้อมูล (Packet radio network) เป็นเครือข่ายที่มีการรับส่งข้อมูลข่าวสารผ่านคลื่นวิทยุ โดยนำข้อมูลที่จะส่งนั้นมาจัดเป็นกลุ่มย่อยๆ และส่งไปทีละกลุ่มจนครบ ทางผู้รับจะนำข้อมูลที่ได้รับมาเป็นกลุ่มๆ มาเรียงต่อกันเป็นข้อมูลที่ถูกต้อง

ประวัติของเครือข่ายวิทยุกลุ่มข้อมูลนั้นเริ่มต้นขึ้นเมื่อปี ค.ศ.1971 เป็นเครือข่ายวิทยุกลุ่มข้อมูลของมหาวิทยาลัยฮาวาย (Hawaii university) ประเทศสหรัฐอเมริกา เนื่องจากคอมพิวเตอร์ที่วิทยาเขต 7 แห่ง ซึ่งตั้งกระจายอยู่บนเกาะ 4 เกาะ มีปัญหาในการติดต่อกับคอมพิวเตอร์แม่ข่ายที่ Oaha โดยที่ระบบส่งสัญญาณไม่สามารถใช้สายโทรศัพท์ได้ เนื่องจากค่าเดินสายโทรศัพท์ผ่านทะเลมีมูลค่าสูงมากและระบบมีความเชื่อถือได้น้อย ในสมัยนั้นระบบส่งสัญญาณจึงถูกเลือกใช้ในรูปของระบบวิทยุสื่อสาร มีระยะระหว่างจุดรับและจุดส่งห่างกันไม่เกิน 30 กิโลเมตร

หลังจากนั้นได้มีการพัฒนาระบบเครือข่ายที่ใช้หลักการแบบนี้เรื่อยมา เช่น ระบบเก็บข้อมูลอัตโนมัติแบบไร้สาย (Wireless data collection system) และระบบโทรมาตร (Telemeter system) เป็นต้น สำหรับเครือข่ายวิทยุกลุ่มข้อมูลในกิจการวิทยุสมัครเล่นนั้น ได้ใช้ระเบียบวิธีการสื่อสารหรือโพรโทคอล (Protocol) แบบ AX.25 ซึ่งเป็นโพรโทคอลที่ถือกำเนิดขึ้นอย่างเป็นทางการเมื่อเดือนตุลาคม ปี ค.ศ. 1984 โดยสหภาพวิทยุสมัครเล่นระหว่างประเทศ (International amateur radio union: IARU) มีต้นแบบมาจากมาตรฐาน CCITT X.25 ที่ใช้ในเครือข่ายข้อมูลสาธารณะแบบแพ็กเกตสวิตช์ (Packet-switched public data network)

เริ่มแรกอุปกรณ์ที่ใช้เชื่อมต่อเข้ากับเครือข่ายวิทยุกลุ่มข้อมูลแบ่งได้เป็น 3 ส่วนหลัก คือ

1. อุปกรณ์ PAD (Packet assembler disassembler) ทำหน้าที่แบ่งกลุ่มและรวมกลุ่มข้อมูลตามรูปแบบของโพรโทคอล AX.25
2. อุปกรณ์โมเด็มทำหน้าที่เชื่อมโยง PAD เข้ากับเครื่องรับส่งวิทยุสื่อสาร
3. เครื่องรับส่งวิทยุสื่อสาร

ต่อมาได้รวมอุปกรณ์ PAD และอุปกรณ์โมเด็มเข้าไว้ด้วยกัน และเรียกอุปกรณ์นี้ว่าตัวควบคุมเทอร์มินัลโนด (Terminal node controller: TNC)

การพัฒนาตัวควบคุมเทอร์มินัลโนดที่ผ่านมา ส่วนใหญ่จะพัฒนาโดยใช้ไมโครคอนโทรลเลอร์เป็นหลัก หรือพัฒนาโดยใช้ซอฟต์แวร์บนคอมพิวเตอร์เพื่อควบคุมการทำงานตามโพรโทคอล [14]

สำหรับประเทศไทยได้มีผู้สนใจระบบเครือข่ายวิทยุกลุ่มข้อมูลสำหรับนักวิทยุสมัครเล่นอยู่ไม่มากนัก เช่น การพัฒนาอุปกรณ์โมเด็มวิทยุ (Radio modem) โดยใช้ซอฟต์แวร์ทำงานเป็นตัวควบคุมเทอร์มินัลโนด (Soft TNC) [15,16] คือ ให้ซอฟต์แวร์จัดการกับข้อมูลตามรูปแบบโพรโทคอล หรือการออกแบบตัวควบคุมเทอร์มินัลโนดในเครือข่ายวิทยุกลุ่มข้อมูล ออกแบบโดยใช้ไมโครคอนโทรลเลอร์และวงจรรวมมาตรฐาน [17] ซึ่งอัตราเร็วในการส่งข้อมูลที่ได้ค่อนข้างต่ำและใช้อุปกรณ์ค่อนข้างมาก

ปัจจุบันได้มีการพัฒนาวงจรรวมดิจิทัลแบบโปรแกรมได้ที่มีประสิทธิภาพสูง จึงมีแนวคิดที่จะนำวงจรรวมนี้มาใช้ในการออกแบบตัวควบคุมเทอร์มินัลโนด แทนการใช้อุปกรณ์ไมโครคอนโทรลเลอร์และวงจรรวมมาตรฐาน โดยเลือกใช้ FPGA (Field programmable gate array) เป็นอุปกรณ์หลักในการออกแบบตัวควบคุมเทอร์มินัลโนด เพื่อรวมฟังก์ชันการทำงานต่างๆ ไว้ในวงจรรวมเพียงตัวเดียว ช่วยลดจำนวนอุปกรณ์ และเพิ่มประสิทธิภาพการทำงานของระบบให้ดีขึ้น

1.2 วัตถุประสงค์

เพื่อศึกษาระเบียบวิธีการสื่อสารที่ใช้ในเครือข่ายวิทยุกลุ่มข้อมูล และออกแบบตัวควบคุมเทอร์-มินัลโนดในเครือข่ายวิทยุกลุ่มข้อมูลโดยใช้วงจรรวม FPGA และทำงานร่วมกับระบบรับส่งวิทยุสื่อสาร

1.3 ขอบเขตของวิทยานิพนธ์

1. ออกแบบและสร้างตัวควบคุมเทอร์มินัลโนดจำนวน 3 ชุด โดยใช้อุปกรณ์ FPGA เป็นอุปกรณ์ควบคุมการทำงานหลักตามระเบียบวิธีการสื่อสาร AX.25 ร่วมกับระบบรับส่งวิทยุสื่อสารแบบ 2 ทิศทางย่านความถี่ UHF มีอัตราส่งข้อมูลขั้นต่ำ 1,200 บิตต่อวินาที และอัตราบิตผิดพลาดไม่เกิน $1X10^{-3}$ สามารถใช้งานกับคอมพิวเตอร์ IBM compatible ได้ โดยเชื่อมต่อทางพอร์ตอนุกรม
2. ทดสอบการทำงานของตัวควบคุมเทอร์มินัลโนด และระบบเครือข่ายข้อมูลโดยใช้ตัวควบคุมเทอร์มินัลโนดที่สร้างขึ้น

1.4 ขั้นตอนและวิธีดำเนินงาน

1. ศึกษาทฤษฎีของระบบเครือข่ายวิทยุกลุ่มข้อมูล
2. ศึกษาข้อมูลที่เกี่ยวกับการออกแบบอุปกรณ์ตัวควบคุมเทอร์มินัลโนด
3. กำหนดข้อกำหนดของตัวควบคุมเทอร์มินัลโนดเพื่อออกแบบ
4. ออกแบบและจำลองการทำงานในส่วนต่างๆ ของวงจร FPGA โดยอาศัยซอฟต์แวร์

5. ออกแบบสร้างและทดสอบวงจรตัวควบคุมเทอร์มินัลโนดในส่วนต่าง ๆ
6. ทดสอบระบบทั้งหมด และแก้ไขข้อบกพร่องที่เกิดขึ้น
7. วิเคราะห์และสรุปผลการดำเนินงาน
8. เขียนวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. อุปกรณ์ตัวควบคุมเทอร์มินัลโนดที่นำไปใช้งานได้จริงในทางปฏิบัติ
2. เรียนรู้การออกแบบและสร้างอุปกรณ์รับส่งข้อมูลในเครือข่ายวิทยุกลุ่มข้อมูล

บทที่ 2

ระบบเครือข่ายวิทยุกลุ่มข้อมูล และโพรโทคอล AX.25

2.1 แนวคิดพื้นฐานของระบบเครือข่ายวิทยุกลุ่มข้อมูล

ระบบสื่อสารข้อมูลในลักษณะเชื่อมโยงกันเป็นเครือข่ายในปัจจุบันนี้ สื่อที่ใช้ในการสื่อสารมีทั้งแบบที่ใช้สาย เช่นสายทองแดง และสายนำใยแสง และแบบที่ไม่ใช้สาย โดยใช้สื่อทางคลื่นวิทยุ ในที่นี้จะอธิบายโครงสร้างของเครือข่ายวิทยุกลุ่มข้อมูล

โครงสร้างโดยทั่วไปของระบบเครือข่ายวิทยุกลุ่มข้อมูล จำแนกได้หลักๆ เป็น 2 แบบ คือ

2.1.1 **เครือข่ายแบบรวมศูนย์กลาง** (Centralized network)

เครือข่ายแบบรวมศูนย์กลางมีโครงสร้าง โครงสร้างเครือข่ายแบบรวมศูนย์กลางนี้ จะมีสถานีแม่ข่ายเป็นศูนย์ควบคุมอยู่ 1 สถานี ศูนย์ควบคุมนี้จะมีฐานข้อมูลและทรัพยากรส่วนกลางของระบบอยู่ และสถานีลูกข่ายทุกๆ สถานีจะติดต่อกับสถานีแม่ข่ายนี้ได้เท่านั้น การติดต่อระหว่างสถานีลูกข่ายกับสถานีลูกข่ายด้วยกันเองจะต้องติดต่อผ่านสถานีแม่ข่าย โดยสถานีแม่ข่ายจะเป็นสถานีจัดการถ่ายทอดให้อีกทอดหนึ่ง ในโครงสร้างของเครือข่ายแบบนี้ สถานีแม่ข่ายมักจะมีสายอากาศแบบรอบทิศทาง ทำให้ส่งกลุ่มข้อมูลให้สถานีลูกข่ายได้ทุกสถานี โครงสร้างแบบนี้เปรียบได้กับการส่งกลุ่มข้อมูลในสายแบบหลายจุด (Multipoint line) โดยมีสถานีปฐมภูมิเป็นสถานีศูนย์ควบคุม และมีสถานีทุติยภูมิหลายสถานีเป็นเสมือนลูกข่าย โครงสร้างเครือข่ายแบบรวมศูนย์กลางนี้ปัจจุบันมีใช้ในระบบต่างๆ มากมาย เช่น ระบบตรวจสอบและควบคุมระยะไกล (SCADA) และระบบเก็บข้อมูลอัตโนมัติแบบไร้สาย เป็นต้น

2.1.2 **เครือข่ายแบบกระจาย** (Distributed network)

ในโครงสร้างเครือข่ายแบบกระจายนี้สายอากาศของทุกสถานีจะเป็นแบบส่งได้รอบทิศทาง โดยทุกสถานีจะติดต่อกับสถานีใดก็ได้ภายในเครือข่าย โดยหลักการแล้วโครงสร้างเครือข่ายแบบนี้เทียบเท่ากับระบบเครือข่ายคอมพิวเตอร์ท้องถิ่น (Local area network: LAN) สำหรับเครือข่ายที่ออกแบบในงานวิจัยครั้งนี้ เป็นเครือข่ายแบบกระจาย

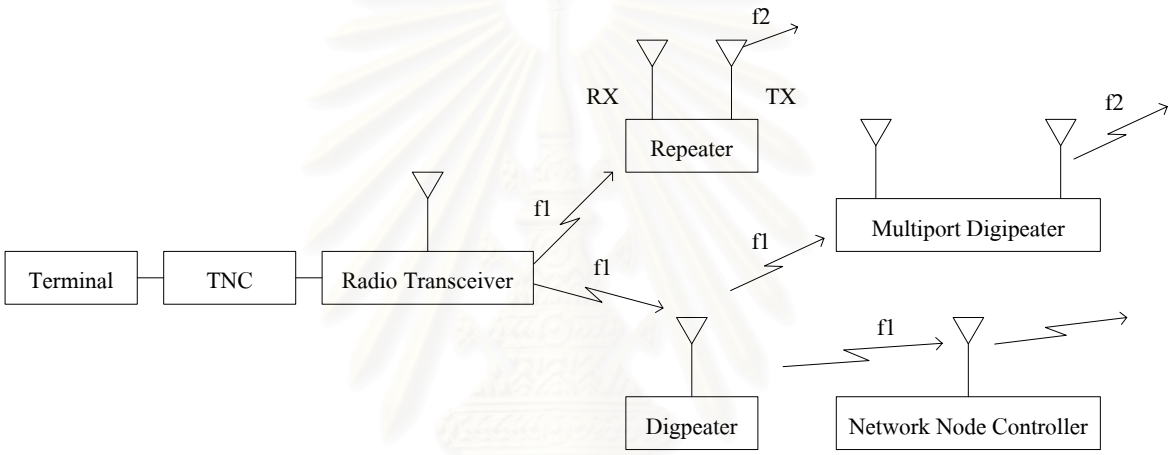
เนื่องจากระยะทางระหว่างจุดส่งและจุดรับในตัวกลางแบบคลื่นวิทยุนี้จะถูกจำกัดอยู่ที่ค่าหนึ่ง ทั้งนี้เกิดจากการสูญเสียของคลื่นวิทยุในตัวกลาง ภูมิประเทศ โดยทั่วไปจะใช้อุปกรณ์ทวนสัญญาณช่วยเพิ่มระยะทางที่ส่งได้

ในเครือข่ายแบบรวมศูนย์กลางนั้น อุปกรณ์ทวนสัญญาณจะรับกลุ่มข้อมูลสถานีแม่ข่าย และทวนสัญญาณไปยังสถานีลูกข่ายอีกกลุ่มหนึ่งในระยะทางที่ไกลขึ้น และอุปกรณ์ทวนสัญญาณจะรับกลุ่ม

ข้อมูลจากสถานีลูกข่าย และส่งกลุ่มข้อมูลไปยังสถานีแม่ข่ายเช่นเดียวกัน ส่วนในเครือข่ายแบบกระจายนั้นอุปกรณ์ทวนสัญญาณจะรับกลุ่มข้อมูล จากสถานีรับส่งกลุ่มข้อมูลใดๆ และส่งกลุ่มข้อมูลนั้นไปยังสถานีปลายทางได้ในระยะทางที่ไกลขึ้น

2.2 อุปกรณ์ส่วนฮาร์ดแวร์ของระบบ

อุปกรณ์ส่วนฮาร์ดแวร์ของเครือข่ายวิทยุกลุ่มข้อมูล โดยทั่วไปขึ้นอยู่กับโครงสร้างมาตรฐานของระบบเครือข่ายวิทยุกลุ่มข้อมูลที่ใช้โพลโทคอล AX.25 ซึ่งเป็นเครือข่ายแบบกระจาย มีอุปกรณ์ประกอบเครือข่ายดังรูปที่ 2.1



รูปที่ 2.1 อุปกรณ์ประกอบเครือข่าย

2.2.1 อุปกรณ์ปลายทาง (Terminal)

เป็นส่วนที่ผู้ใช้งานติดต่อกับระบบใช้เป็นทางผ่านของสัญญาณ และการแสดงผลการติดต่อรวมทั้งข้อมูลข่าวสารที่รับส่งภายในเครือข่ายจะแสดงผลบนจอภาพของอุปกรณ์ปลายทาง

2.2.2 ตัวควบคุมเทอร์มินัลโนด (Terminal node controller: TNC)

เป็นอุปกรณ์ที่สำคัญในระบบเครือข่ายวิทยุกลุ่มข้อมูล ทำหน้าที่สร้างกลุ่มข้อมูล และรับส่งกลุ่มข้อมูลตามกระบวนการในระเบียบวิธีสื่อสาร ซึ่งจะกล่าวโดยละเอียดในหัวข้อต่อไป

2.2.3 เครื่องรับส่งวิทยุสื่อสาร (Radio transceiver)

เป็นอุปกรณ์สื่อสัญญาณ โดยนำกลุ่มข้อมูลจากตัวควบคุมเทอร์มินัลโนดส่งผ่านสื่อกลางที่เป็นอากาศ ไปยังสถานีสื่อสารปลายทางที่ต้องการติดต่อ

2.2.4 **อุปกรณ์ทวนสัญญาณ** (Repeater)

อุปกรณ์ทวนสัญญาณด้วยเสียง อุปกรณ์ทวนสัญญาณนี้จะทวนสัญญาณทันทีที่มีคลื่นวิทยุที่ตรงความถี่ขาเข้า (f1) จากนั้นจะถอดสัญญาณเสียงนั้นออกมา มอดูเลตไปเป็นอีกความถี่หนึ่ง (f2) แล้วส่งไปยังภาคส่งและสายอากาศส่ง

2.2.5 **อุปกรณ์ทวนสัญญาณแบบดิจิทัล** (Digipeater)

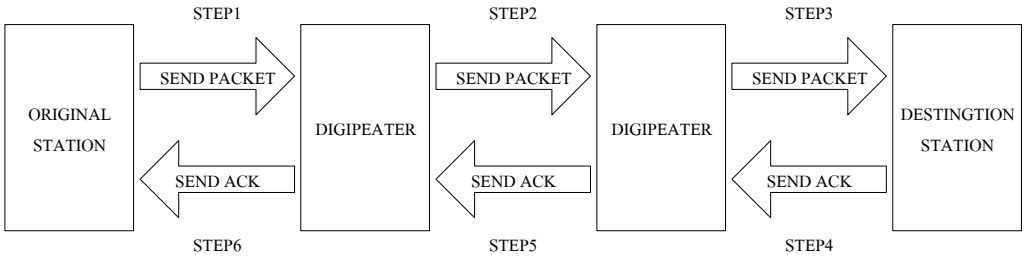
Digipeater มาจากคำว่า Digital repeater เป็นอุปกรณ์ที่รับข้อมูลมาเก็บไว้ชั่วระยะเวลาหนึ่งแล้วจึงทำการส่งต่อไป (Store and forward) โดยสัญญาณที่ส่งเข้า Digipeater ความถี่ f1 จะถูกดีมอดูเลตเป็นสัญญาณ FSK แล้วผ่านวงจรโมเด็ม เพื่อให้ได้ข่าวสารที่เป็นสัญญาณในระบบดิจิทัล Digipeater ทุกเครื่องในเครือข่ายวิทยุกลุ่มข้อมูลที่ใช้โพรโทคอล AX.25 จะมีนามเรียกขาน (Call sign) ประจำเครื่อง การส่งกลุ่มข้อมูลไปให้จะต้องอ้างนามเรียกขานของ Digipeater นั้นๆ ด้วย เมื่อ Digipeater ได้รับกลุ่มข้อมูล ก็จะเปรียบเทียบว่าใช่ข้อมูลของตนเองหรือไม่ โดยเทียบจากนามเรียกขาน ถ้าใช่ก็จะรอช่วงสัญญาณว่าง แล้วจึงส่งกลุ่มข้อมูลออกไปด้วยความถี่เดิมคือ f1

2.2.6 **อุปกรณ์ทวนสัญญาณดิจิทัลแบบมัลติพอร์ต** (Multiport digipeater)

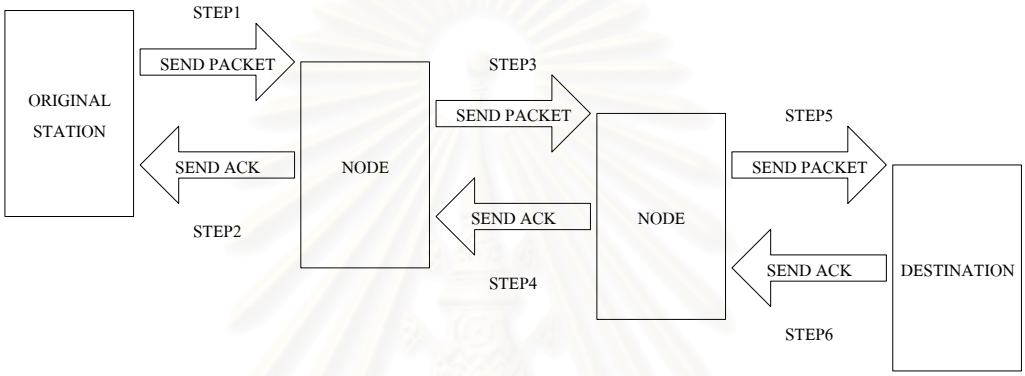
ทำงานแบบเดียวกับอุปกรณ์ Digipeater แต่สัญญาณด้านขาออกสามารถส่งความถี่ที่ต่างจากความถี่ขาเข้าได้ รวมทั้งอัตราเร็วในการส่งข้อมูลก็สามารถกำหนดให้ต่างจากอัตราเร็วการส่งข้อมูลขาเข้าได้ เช่น สัญญาณขาเข้า 300 บิต/วินาที ทำงานที่ความถี่ 150 MHz, สัญญาณขาออก 1,200 บิต/วินาที ทำงานที่ความถี่ 420 MHz เป็นต้น อุปกรณ์ทวนสัญญาณดิจิทัลแบบมัลติพอร์ตไม่สามารถใช้กับ TNC แบบมาตรฐานได้ การใช้งานกับ TNC จะต้องมีอุปกรณ์พิเศษต่อเพิ่มให้กับ TNC

2.2.7 **ตัวควบคุมเน็ตเวิร์กโนด** (Network node controller: NNC)

เป็นอุปกรณ์ที่สร้างขึ้นเพื่อควบคุมจัดการกับเส้นทางภายในเครือข่ายวิทยุกลุ่มข้อมูล ซึ่งเป็นอุปกรณ์ที่ยังไม่เป็นมาตรฐานนานาชาติ แต่ปัจจุบันนี้ในเครือข่ายนักวิทยุสมัครเล่นประเทศสหรัฐอเมริกาเป็นที่นิยมกันมาก ความแตกต่างของ Digipeater และตัวควบคุมเน็ตเวิร์กโนด คือการทำงานของ TNC ที่ส่งกลุ่มข้อมูลผ่าน Digipeater จะเป็นไปในลักษณะ End-to-end acknowledgment ดังรูปที่ 2.2 แต่ถ้า TNC ส่งข้อมูลผ่านตัวควบคุมเน็ตเวิร์กโนดการทำงานจะเป็นไปในลักษณะ Node-to-node acknowledgment ดังรูปที่ 2.3



รูปที่ 2.2 การส่งกลุ่มข้อมูลจากสถานีต้นทางไปยังสถานีปลายทางผ่าน Digipeater ทำงานในลักษณะ End-to-end acknowledgment



รูปที่ 2.3 การส่งกลุ่มข้อมูลจากสถานีต้นทางไปยังสถานีปลายทางผ่าน Network Node Controller ทำงานในลักษณะ Node-to-node acknowledgment

จากรูปที่ 2.2 และ 2.3 การทำงานของเครือข่ายโดย TNC ส่งกลุ่มข้อมูลผ่าน Digipeater แบบโพรโทคอล AX.25 นั้น สถานีต้นทางจะต้องรู้ว่ากลุ่มข้อมูลที่ส่งผ่านจะส่งผ่าน Digipeater มีนามเรียกขานใดบ้าง และต้องรู้ว่าลำดับการส่งผ่านเป็นอย่างไร กลุ่มข้อมูลจะถูกส่งไปยังสถานีปลายทาง และส่งสัญญาณตอบรับกลับมาผ่าน Digipeater ชุดเดิม ซึ่งลำดับการส่งผ่านจะต้องเป็นไปตามลำดับย้อนกลับไปยังสถานีต้นทาง ถ้าสัญญาณตอบรับถูก รบกวน และเกิดความผิดพลาดของสัญญาณระหว่างทาง สถานีต้นทางจะต้องส่งกลุ่มข้อมูลมาใหม่ โดยที่สถานีต้นทางจะไม่ทราบเลยว่า เกิดความผิดพลาดของสัญญาณในส่วนใดของเส้นทาง ส่วนการทำงานของเครือข่ายโดย TNC ส่งกลุ่มข้อมูลผ่าน NNC ตามโพรโทคอล AX.25 สถานีต้นทางไม่จำเป็นต้องรู้ว่ากลุ่มข้อมูลที่ส่งผ่านไปยังปลายทางจะต้องผ่านเส้นทางใดบ้าง NNC จะเป็นตัวจัดการเส้นทางเอง เมื่อ NNC ตัวแรกได้รับกลุ่มข้อมูลแล้วก็จะส่งสัญญาณตอบรับกลับไปให้ TNC แล้วส่งกลุ่มข้อมูลให้ NNC ตัวต่อไปพร้อมทั้งรอสัญญาณตอบรับกลับมา จนกระทั่งกลุ่มข้อมูลถูกส่งไปถึงสถานีปลายทาง ถ้าเกิดความผิดพลาดของสัญญาณกลุ่มข้อมูลขึ้นที่โนดใด โนดนั้นก็จะส่งกลุ่มข้อมูลออกไปใหม่ และรอสัญญาณตอบรับอีกครั้ง การทำงานในลักษณะเช่นนี้ทำให้รู้ว่าเกิดปัญหาที่ส่วนใดของเส้นทาง

2.3 **โพรโทคอลเข้าถึงวิทยุกลุ่มข้อมูล** (Packet Radio Access Protocol)

ในเครือข่ายวิทยุกลุ่มข้อมูล สิ่งที่สำคัญที่สุดและถือได้ว่าเป็นตัวกำหนดสมรรถภาพของระบบก็คือ โพรโทคอลเข้าถึงหรือการเข้าใช้ช่องสัญญาณ ซึ่งมีความหมายถึงการที่จะทำให้สถานีสื่อสารข้อมูลหลายๆ สถานีสามารถใช้ช่องสัญญาณใดช่องสัญญาณหนึ่งร่วมกันได้ อีกทั้งยังเป็นการใช้ทรัพยากรความถี่อย่างมีประสิทธิภาพอีกด้วย

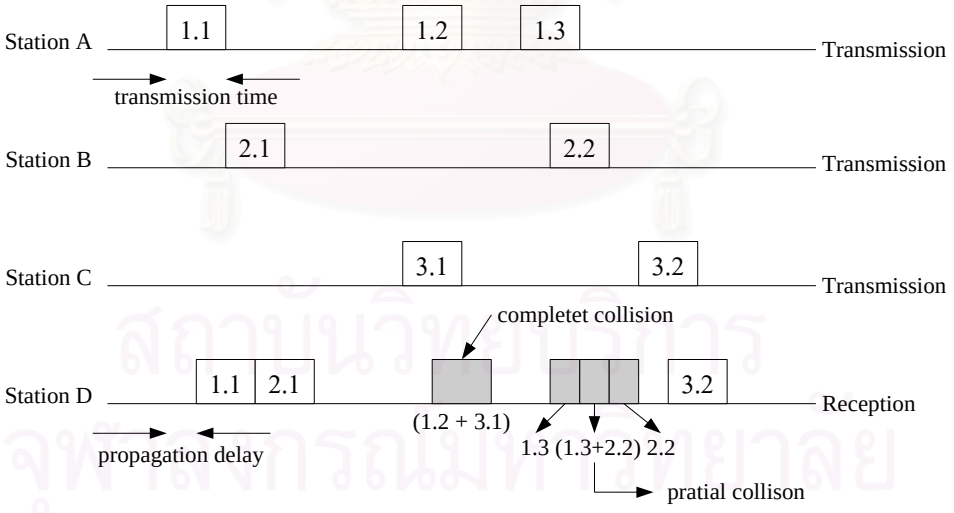
เครือข่ายวิทยุกลุ่มข้อมูลที่ใช้โพรโทคอล AX.25 นิยมใช้โพรโทคอลเข้าถึงอยู่ 3 แบบคือ

- ระบบ ALOHA
- ระบบ Slotted ALOHA
- ระบบ CSMA

2.3.1 **ระบบ** ALOHA

เป็นเทคนิคการเข้าใช้ช่องสัญญาณแบบสุ่ม ระบบนี้ได้ถูกพัฒนาขึ้นที่มหาวิทยาลัยฮาวาย หลักการของระบบนี้คือ สถานีสื่อสารข้อมูลที่ต้องการส่งข้อมูลจะทำการส่งกลุ่มข้อมูลทันทีโดยไม่คำนึงถึงสถานีสื่อสารข้อมูลอื่นๆ ที่ใช้ช่องสัญญาณเดียวกัน หลังจากนั้นสถานีที่ส่งข้อมูลจะทำการรอสัญญาณตอบรับช่วงเวลาหนึ่งถ้าหากไม่มีสัญญาณตอบรับจากสถานีคู่สื่อสาร จะถือว่ากลุ่มข้อมูลที่ส่งไปนั้นชนกับกลุ่มข้อมูลจากสถานีสื่อสารอื่นๆ และจะทำการรอเป็นเวลาค่าสุ่ม ก่อนที่จะทำการส่งกลุ่มข้อมูลใหม่

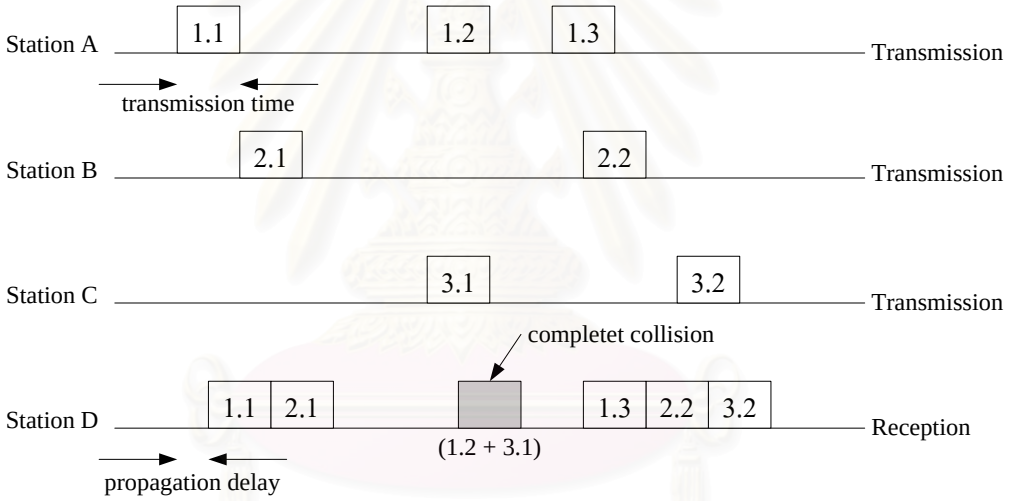
เหตุผลที่สถานีสื่อสารข้อมูลต้องรอด้วยเวลาค่าสุ่มนั้น เนื่องจากต้องการหลีกเลี่ยงไม่ให้สถานีสื่อสารข้อมูล 2 สถานีหรือมากกว่า ที่ส่งกลุ่มข้อมูลมาชนกันไม่ให้ชนกันซ้ำอีกครั้ง นอกจากนี้กระบวนการตอบรับ ก็จะมีรูปแบบขึ้นอยู่กับระบบรับส่งสัญญาณที่ใช้งาน รูปที่ 2.4 แสดงพฤติกรรมของระบบ ALOHA ในเครือข่ายวิทยุกลุ่มข้อมูล



รูปที่ 2.4 พฤติกรรมของระบบ ALOHA ในเครือข่ายวิทยุกลุ่มข้อมูล

2.3.2 ระบบ Slotted ALOHA

เป็นระบบที่ทำการพัฒนาจากระบบ ALOHA โดยพยายามลดช่วงเวลาที่ช่องสัญญาณจะเสียไปหากข้อมูลเกิดการชนกันขึ้น เนื่องจากในระบบ ALOHA นั้นไม่มีการเข้าจังหวะในการสื่อสารกันระหว่างสถานีสื่อสารข้อมูล กล่าวคือสถานีสื่อสารข้อมูลจะทำการส่งกลุ่มข้อมูลเวลาใดก็ได้ ข้อมูลจึงเกิดการชนกันได้ง่าย (การชนกันเกิดขึ้นเมื่อส่วนใดๆ ของกลุ่มข้อมูลหนึ่งเกิดทับกันกับอีกกลุ่มข้อมูลหนึ่ง) ซึ่งจะเกิดการสูญเปล่าของช่องสัญญาณเท่ากับเวลาของการส่งกลุ่มข้อมูลยาว 2 กลุ่มข้อมูล (ในกรณีที่ระบบมีการรับส่งกลุ่มข้อมูลที่มีความยาวคงที่) Slotted ALOHA เป็นระบบที่แก้ไขจุดอ่อนของระบบ ALOHA ในส่วนนี้โดย Slotted ALOHA จะใช้ความยาวของกลุ่มข้อมูลที่มีค่าคงที่ และมีการรับส่งกลุ่มข้อมูลแบบเข้าจังหวะกัน (Synchronize) ระหว่างสถานีสื่อสารข้อมูลทุกสถานีเพื่อกำหนดการเริ่มส่งข้อมูล ซึ่งจะทำให้เวลาที่ช่องสัญญาณจะเสียไปลดลงเหลือเพียงความยาว 1 กลุ่มข้อมูลเท่านั้น รูปที่ 2.5 แสดงพฤติกรรมของระบบ Slotted ALOHA ในเครือข่ายวิทยุกลุ่มข้อมูล



รูปที่ 2.5 พฤติกรรมของระบบ Slotted ALOHA ในเครือข่ายวิทยุกลุ่มข้อมูล

2.3.3 ระบบ CSMA (Carrier sense multiple access)

ในระบบ ALOHA และ Slotted ALOHA มีหลักการที่เหมือนกันคือ การให้แต่ละสถานีในระบบใช้การแย่งชิงช่องสัญญาณกัน แต่ระบบ CSMA เป็นระบบที่ใช้เทคนิคที่ช่วยลดระดับการแย่งชิงช่องสัญญาณลง วิธีการลดระดับการแย่งชิงช่องสัญญาณระหว่างสถานีสื่อสารข้อมูล ทำโดยพยายามหลีกเลี่ยงการรับส่งกลุ่มข้อมูลในขณะสถานีสื่อสารข้อมูลใดๆ กำลังใช้งานช่องสัญญาณอยู่ โดยยึดหลักว่าสถานีสื่อสารข้อมูลที่จะทำการส่งข้อมูลจะตรวจสอบสถานะของช่องสัญญาณก่อนว่าถูกใช้งานอยู่

หรือไม่ หลักการนี้จะทำงานอย่างมีประสิทธิภาพก็ต่อเมื่อช่องสัญญาณมีการหน่วงเวลารับส่งสัญญาณ (Propagation delay) ต่ำ

ระบบ CSMA นี้ยังได้มีการพัฒนารูปแบบของการทำงานออกเป็น 3 โมด ได้แก่ 1–persistent, Non-persistent และ p-persistent

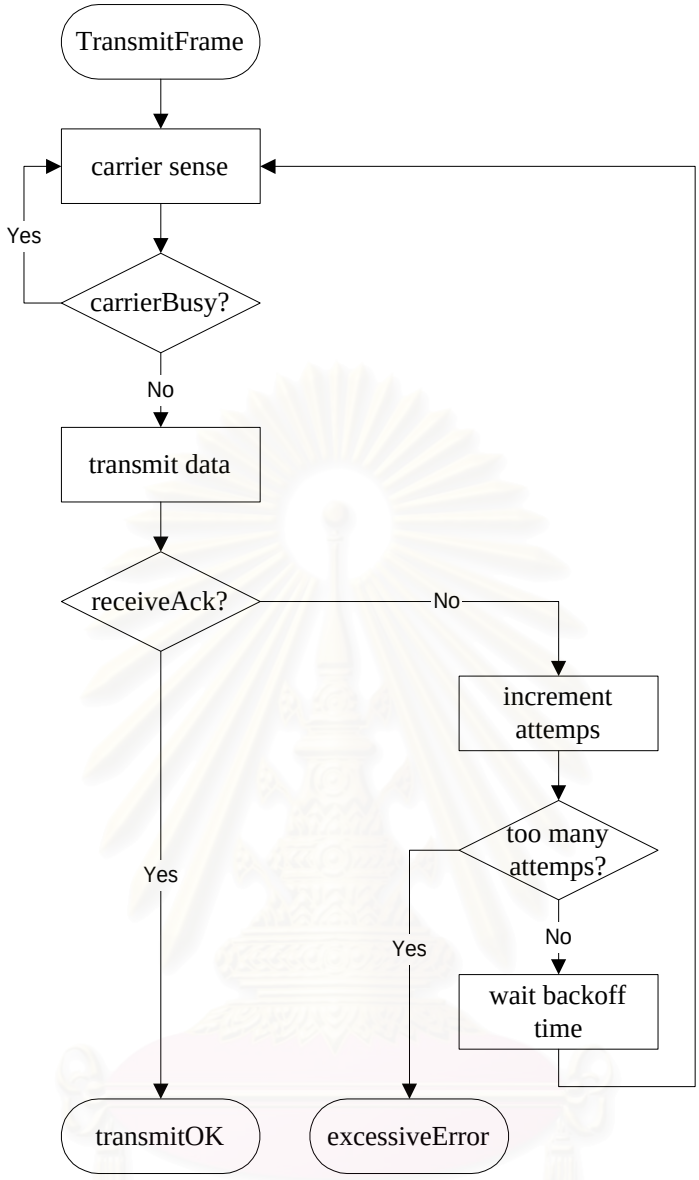
ในโมด 1–persistent CSMA สถานีสื่อสารข้อมูลจะตรวจสอบช่องสัญญาณก่อนส่ง สถานีสื่อสารข้อมูลจะตรวจสอบไปจนกระทั่งช่องสัญญาณว่างก็จะส่งกลุ่มข้อมูลในทันที ในกรณีที่มีสถานีสื่อสารข้อมูลมากกว่า 1 สถานีรอช่องสัญญาณว่างและส่งกลุ่มข้อมูลในช่องสัญญาณพร้อมกันกลุ่มข้อมูลจะเกิดการชนกันขึ้น สถานีสื่อสารข้อมูลที่ส่งกลุ่มข้อมูลนั้นจะรอด้วยค่าเวลาสุ่มค่าหนึ่ง (โดยเวลาที่แต่ละสถานีรอนั้นจะไม่เท่ากัน) ก่อนที่จะส่งข้อมูลนั้นใหม่อีกครั้ง

ในโมด Non-persistent CSMA สถานีสื่อสารข้อมูลต้องการส่งกลุ่มข้อมูลจะตรวจสอบช่องสัญญาณ ถ้าช่องสัญญาณว่างก็จะส่งกลุ่มข้อมูลทันที ถ้าช่องสัญญาณไม่ว่างสถานีสื่อสารข้อมูลนั้นหยุดการตรวจสอบไปด้วยเวลาสุ่มค่าหนึ่งแล้วจึงเริ่มตรวจสอบใหม่ และจะกระทำเช่นนี้ไปเรื่อยๆ จนกว่าช่องสัญญาณว่างแล้วจึงส่งกลุ่มข้อมูล ด้วยวิธีนี้ประสิทธิภาพของการใช้ช่องสัญญาณสูงกว่า 1–persistent CSMA หากแต่ว่าจะมีค่าหน่วงเวลาในการส่งสัญญาณสูงกว่า

ในโมด p-persistent CSMA ซึ่งโดยทั่วไปจะใช้กับระบบที่มีการใช้หลายช่องความถี่โดยมีหลักการคือ สถานีสื่อสารข้อมูลจะทำการตรวจช่องสัญญาณ หากไม่ว่างเครื่องจะรอตรวจสอบช่องสัญญาณช่องถัดไป แต่ถ้าหากช่องสัญญาณว่าง สถานีสื่อสารข้อมูลจะทำการส่งข้อมูลด้วยความน่าจะเป็น p (หมายความว่าสถานีจะมีความน่าจะเป็นที่จะส่งข้อมูลในช่องสัญญาณถัดไปเท่ากับ 1-p) ถ้าหากช่องสัญญาณถัดไปว่าง สถานีสื่อสารข้อมูลก็จะมีความน่าจะเป็นในการส่งกลุ่มข้อมูลเท่ากับ p เป็นเช่นนี้ไปเรื่อยๆ แต่หากตรวจสอบพบว่าช่องสัญญาณไม่ว่าง เครื่องจะหยุดรอด้วยเวลาค่าสุ่มค่าหนึ่งแล้วจึงเริ่มตรวจสอบช่องสัญญาณใหม่

ระบบวิทยุสื่อสารแบบสองทางนั้นส่วนมากทำงานในลักษณะครึ่งดูเพล็กซ์ (Half-duplex) ดังนั้นจึงไม่สามารถตรวจสอบสัญญาณระหว่างส่งข้อมูลได้ และไม่สามารถใช้การเข้าถึงแบบ CSMA/CD (Carrier sense multiple access with collision detection) ได้ และระบบวิทยุที่ใช้การผสมสัญญาณแบบ FM เมื่อกลุ่มข้อมูล 2 ชุดเกิดการชนกันจะสามารถดึงข้อมูลกลับได้อย่างถูกต้อง การดึงกลับนั้นจะดึงจากแหล่งที่มีกำลังสูงสุดเรียกวิธีนี้ว่า Capture effect [9]

คุณสมบัติที่น่าสนใจของ Capture effect ก็คือ สถานีที่อยู่ใกล้กว่าจะส่งข้อมูลได้มากกว่าสถานีที่อยู่ไกลออกไป เนื่องจากสถานีที่อยู่ใกล้กว่าจะมีขนาดสัญญาณแรงกว่า ดังนั้นเมื่อมีการชนกันของกลุ่มข้อมูล สถานีที่มีกำลังสูงจะชนะเสมอ และกลุ่มข้อมูลก็จะไม่เสียหายด้วย นอกจากนี้ Capture effect ยังขึ้นกับคุณสมบัติของเครื่องรับส่งวิทยุแต่ละยี่ห้อด้วย



รูปที่ 2.6 พฤติกรรมการเข้าใช้ช่องสัญญาณแบบ CSMA

ไม่ว่าโพรโทคอลเข้าถึงจะเป็นวิธีใดก็ตาม การวัดประสิทธิภาพของเครือข่ายจะวัดจากค่าที่เรียกว่า Throughput ซึ่งเป็นค่าที่แสดงว่าการส่งข้อมูลมีโอกาสเท่าใดที่กลุ่มข้อมูลที่ส่งออกไปจะไม่ชนกัน เมื่อเทียบกับจำนวนกลุ่มข้อมูลที่ส่งออกมาทั้งหมดภายใต้สภาวะเดียวกัน [10] โดยสรุปแล้ว Throughput แต่ละระบบมีค่าดังนี้

- ระบบ ALOHA ให้ค่า Throughput สูงสุดที่ 18%
- ระบบ Slotted ALOHA ให้ค่า Throughput สูงสุดที่ 36%
- ระบบ CSMA ให้ค่า Throughput สูงสุดที่ประมาณ 50% (ในกรณี CSMA ในโมด 1–persisten)

2.4 แบบจำลองการเชื่อมโยงระบบเปิด (Open system interconnection: OSI)

แบบจำลองการเชื่อมโยงระบบเปิด (Open system interconnection: OSI) เป็นชุดมาตรฐานในการสื่อสารที่ทำให้สภาวะแวดล้อมเหมาะสมในการเคลื่อนย้ายข้อมูลระหว่างชุดทำงาน 2 ชุด [19] แบบจำลอง OSI แบ่งออกได้เป็น 7 ชั้นด้วยกัน กล่าวโดยสรุปได้ดังนี้

2.4.1 ชั้นกายภาพ (Physical layer)

การทำงานในชั้นกายภาพจะเกี่ยวกับการส่งข้อมูลระดับบิตข้อมูลผ่านช่องสัญญาณสื่อสาร วิธีการออกแบบจะต้องทำให้แน่ใจได้ว่าข้อมูล '1' ที่ส่งออกไปนั้น ที่ปลายทางสามารถรับข้อมูล '1' ได้ถูกต้อง คำถามที่เกี่ยวข้องก็คือ จะต้องใช้แรงดันไฟฟ้าเท่าใดแทนข้อมูล '1' และ '0' และแต่ละบิตจะต้องห่างกันเท่าใด การส่งข้อมูลสามารถทำแบบ 2 ทิศทางได้หรือไม่ ก่อนจะเริ่มต้นส่งข้อมูลจะต้องทำให้เกิดการติดต่อกันได้อย่างไร และเมื่อส่งเสร็จแล้วจะยกเลิกการติดต่ออย่างไร ลึกลงไปอีกก็คือ คอนเน็กเตอร์ (Connector) ของเน็ตเวิร์กจะใช้กี่ขา และแต่ละขาทำหน้าที่อะไร จะเห็นได้ว่าประเด็นที่ต้องสนใจจะเกี่ยวข้องกับทั้งทางไฟฟ้า รูปร่างลักษณะภายนอก และก็ทั้งวิธีการเชื่อมต่อทางกายภาพ รวมทั้งตัวกลางที่ใช้ส่งข้อมูลจริงๆ

2.4.2 ชั้นเชื่อมโยงข้อมูล (Data link layer)

จุดประสงค์หลักชั้นเชื่อมโยงข้อมูลก็คือ พยายามทำให้การรับส่งข้อมูลดิบไม่มีความผิดพลาดเกิดขึ้น ทำให้ชั้นที่สูงขึ้นไปนำข้อมูลไปใช้งานได้ถูกต้อง วิธีการก็คือทางฝ่ายส่งจะแตกข้อมูลออกเป็นกลุ่มๆ เรียกว่าเฟรมข้อมูล (Data frame) ส่งเฟรมข้อมูลออกไปทีละชุด และรอรับการตอบรับ (Acknowledge freme) ซึ่งตอบกลับมาโดยผู้รับ โดยปกติแล้วชั้นกายภาพจะไม่สนใจว่าข้อมูลเป็นอย่างไร เป็นหน้าที่ของชั้นเชื่อมโยงข้อมูล ที่จะต้องสร้างและตรวจรับขอบเขตของเฟรมข้อมูล ซึ่งสามารถทำได้โดยการเติมบิตข้อมูลในตำแหน่งเริ่มต้นและสิ้นสุดของเฟรม แต่จุดที่ต้องการระวังก็คือ ข้อมูลที่เพิ่มเข้าไปจะต้องไม่มีผลหรือไม่ซ้ำกับข้อมูลจริงๆ

สัญญาณรบกวนจากภายนอกก็เป็นปัญหาหนึ่งที่อาจจะทำให้เฟรมขาดหายไปได้ ในกรณีนี้โปรแกรมที่ควบคุมชั้นเชื่อมโยงข้อมูลที่เครื่องต้นทางจะส่งข้อมูลซ้ำมาใหม่ อย่างไรก็ตามการส่งเฟรมเดียวกันออกมาหลายๆ ครั้งก็อาจจะทำให้เกิดเฟรมซ้ำกันได้ วิธีการป้องกันก็คือ เฟรมที่ซ้ำกันจะส่งออกไปก็ต่อเมื่อมีการตอบรับส่งมาบอกว่าข้อมูลหายหรือถูกทำลายไปก่อน นอกจากนี้ภายในชั้นนี้จะมีการอินเตอร์เฟซที่รองรับการทำงานร่วมกับชั้นเครือข่ายได้หลายแบบ

ข้อที่น่าสนใจอีกอย่างหนึ่งก็คือ ทำอย่างไรจึงจะทำให้ทางฝั่งที่ส่งข้อมูลซึ่งส่งได้เร็วกว่าทางฝั่งรับสามารถทำงานได้อย่างไม่มีปัญหา วิธีการที่น่าสนใจก็คือ การระบายปริมาณการสื่อสารให้สม่ำเสมอโดยการพักข้อมูลในหน่วยความจำข้อมูลไว้ชั่วคราว แล้วค่อยส่งต่อออกไป

2.4.3 **ชั้นเครือข่าย** (Network layer)

ภายในชั้นเครือข่ายจะเกี่ยวข้องกับการควบคุมการทำงานของซับเน็ต ประเด็นที่สำคัญก็คือการพิจารณาว่าแพ็กเกตจะถูกส่งจากต้นทางไปยังปลายทางได้อย่างไร การหาเส้นทางอาจจะวางอยู่บนตารางที่คงที่ และเชื่อมโดยตรงเข้ากับเครือข่าย และมีการเปลี่ยนตารางน้อยมาก การกำหนดเส้นทางจะกำหนดตอนเริ่มต้นติดต่อเลยก็ได้ หรืออาจจะใช้วิธีที่สามารถเปลี่ยนแปลงได้ตลอดเวลา ซึ่งเส้นทางทางที่แพ็กเกตเดินทางไปจะถูกกำหนดแบบแพ็กเกตต่อแพ็กเกต

ถ้ามีจำนวนแพ็กเกตมากเกินไปภายในซับเนตจะทำให้เกิดปัญหาคอขวด คือจำนวนแพ็กเกตเข้ามามากแต่ทางออกน้อย ภายในชั้นเครือข่ายนี้จะต้องจัดการกับปัญหาเหล่านี้ด้วย

การจัดการด้านปริมาณข้อมูลที่รับส่งก็เป็นเรื่องหนึ่งที่ต้องจัดการภายในชั้นเครือข่ายนี้ อย่างน้อยที่สุดซอฟต์แวร์ที่ใช้งานภายในชั้นนี้จะต้องนับจำนวนแพ็กเกตหรือจำนวนตัวอักษรที่ส่งโดยลูกค้าเพื่อจะได้นำไปคำนวณยอดการใช้สำหรับคิดค่าบริการได้ถูกต้อง แต่ถ้าหากมีการส่งแพ็กเกตข้ามระหว่างประเทศ การคิดค่าบริการก็จะต้องคิดในอัตราที่ต่างกันของแต่ละฝั่ง ซึ่งซับซ้อนมากขึ้นไปอีก

เมื่อแพ็กเกตเดินทางจากเครือข่ายหนึ่งไปยังอีกเครือข่ายหนึ่งจะทำให้เกิดปัญหาต่างๆ ขึ้น เช่น แอดเดรสของเครือข่ายที่ 2 อาจจะแตกต่างจากเครือข่ายที่ 1 ทำให้เครือข่ายที่ 2 ไม่สามารถรับแพ็กเกตนี้ได้เลย หรือโพรโทคอลที่ใช้อาจจะแตกต่างกัน การทำงานในชั้นเครือข่ายนี้จะต้องจัดการกับปัญหาเหล่านี้ได้ เพื่อให้เครือข่ายทั้งหลายต่อถึงกันได้เสมือนเป็นเครือข่ายเดียวกัน

ส่วนเครือข่ายแบบกระจาย (Broadcast network) วิธีการหาเส้นทางของแพ็กเกตจะง่ายมาก ทำให้การทำงานในชั้นเครือข่ายมีขนาดเล็ก หรืออาจจะไม่มีเลยก็ได้

2.4.4 **ชั้นทรานสพอร์ต** (Transport layer)

หลักการทำงานของชั้นทรานสพอร์ตก็คือ คอยติดต่อกับชั้นเซสชัน แยกข้อมูลให้มีขนาดพอเหมาะ และส่งต่อให้ชั้นเครือข่าย พร้อมทั้งคอยตรวจสอบว่าข้อมูลได้ถูกส่งไปถึงปลายทางอย่างเรียบร้อยหรือไม่ ทั้งหมดนี้จะต้องทำให้ได้อย่างมีประสิทธิภาพ เพื่อเป็นการแยกให้ชั้นเซสชันเป็นอิสระออกจากการเปลี่ยนแปลงทางด้านฮาร์ดแวร์

ภายใต้ภาวะปกติชั้น ทรานสพอร์ตจะสร้างการติดต่อกับเครือข่ายสำหรับแต่ละคู่ของชั้นทรานส-พอร์ตอีกฝั่งหนึ่ง แต่ถ้ามีการขอเพิ่มประสิทธิภาพการส่งข้อมูล ชั้นทรานสพอร์ตก็จะสร้างการติดต่อกับเครือข่ายขึ้นมาหลายๆ ชุดก็ได้ และแบ่งข้อมูลส่งออกไปตามการติดต่อต่างๆ หรืออีกแง่หนึ่งถ้าหากการสร้างการติดต่อกับเครือข่ายมีราคาแพง ชั้นทรานส์พอร์ตก็จะใช้วิธีการมัลติเพล็กซ์ช่องทางการติดต่อของชั้นทรานสพอร์ตเข้าไปในช่องทางการติดต่อเดียว อย่างไรก็ตามชั้นทรานสพอร์ตจะต้องทำงานไปพร้อมกันกับชั้นเซสชัน

นอกจากนี้ชั้นทรานสพอร์ตยังต้องคอยดูว่าจะต้องให้บริการชนิดใดแก่ชั้นเซสชัน ความต้องการของผู้ใช้บริการเครือข่ายที่นิยมใช้กันแพร่หลายก็เช่น การสื่อสารแบบจุดถึงจุดโดยไม่มีความผิดพลาด (Error free point-to-pint) ซึ่งจะส่งข้อมูลตามลำดับที่เข้ามา ชั้นทรานสพอร์ตชนิดอื่นๆ ที่เป็นไปได้ เช่น การส่งข้อมูลที่เป็นอิสระต่อกันแต่ไม่มีการเรียงลำดับการรับและการส่ง, แบบกระจายข้อมูลสู่ปลายทางหลายจุด ปกติแล้วชนิดของการให้บริการจะถูกกำหนดตอนที่สร้างช่องทางการติดต่อ (Connection establishment)

2.4.5 **ชั้นเซสชัน** (Session layer)

ชั้นเซสชันจะยอมให้ผู้ใช้งานจัดตั้งเซสชันระหว่างเครื่อง กระบวนการเซสชันจะยอมให้มีการส่งข้อมูลเช่นเดียวกันกับชั้นทรานสพอร์ต และยังเพิ่มการให้บริการที่ก้าวหน้ามากกว่าอีก เช่น ยอมให้ผู้ใช้เข้าไปใช้งานที่เครื่องอื่น หรือทำการถ่ายไฟล์ข้อมูลระหว่างเครื่องได้

บริการอีกอย่างของชั้นเซสชันก็คือ การจัดการการสนทนา ชั้นเซสชันจะยอมให้มีการส่งผ่านข้อมูลได้ทั้งสองทางในเวลาเดียวกัน ถ้าเป็นการส่งในทิศทางเดียว ชั้นเซสชันจะช่วยในจัดการให้ว่าถึงฝ่ายใดจะผู้ส่งและผู้รับ

มีบริการอื่นที่เกี่ยวข้องอีก เช่น การจัดการโทเค็น (Token) ในโพรโทคอลบางแบบมีความจำเป็นที่ทั้งสองฝั่งจะทำการสิ่งใดเหมือนกันในเวลาเดียวกันไม่ได้ จะต้องมีวิธีควบคุมกระบวนการเหล่านี้ ชั้นเซสชันจะใช้โทเค็นเป็นตัวแลกเปลี่ยนระหว่างเครื่องทั้งสอง เพื่อให้มีเพียงเครื่องเดียวที่ถือโทเค็นอยู่เท่านั้นที่จะสามารถทำงานได้ก่อน

นอกจากนี้ยังมีบริการอีกอย่างคือ การซิงโครไนซ์ (synchronization) หรือการทำให้ทั้งสองระบบทำงานสัมพันธ์กัน ลองพิจารณาปัญหาของการถ่ายไฟล์ข้อมูลระหว่าง 2 เครื่องที่ใช้เวลาร่วม 2 ชั่วโมง บนเครือข่ายที่มีค่าเฉลี่ยของการเสียหายประมาณ 1 ชั่วโมง หลังจากการถ่ายข้อมูลถูกหยุดกลางทาง กระบวนการถ่ายข้อมูลทั้งหมดจะต้องเริ่มต้นใหม่อีกครั้ง และอาจทำให้เกิดความเสียหายในเครือข่ายได้ ชั้นเซสชันจะจัดการปัญหานี้โดยการใส่ชุดตรวจสอบเข้าในข้อมูล ทำให้หลังจากเกิดความเสียหายขึ้นในเครือข่ายสามารถกลับมาถ่ายข้อมูลใหม่ต่อจากจุดเดิมได้ทันที ไม่ต้องไปเริ่มต้นใหม่

2.4.6 **ชั้นพรีเซนเทชัน** (Presentation layer)

บนชั้นพรีเซนเทชันจะมีฟังก์ชันการทำงานสำหรับแก้ปัญหาในบางระดับ โดยที่ผู้ใช้ไม่ต้องเข้าไปแก้ปัญหาเอง ซึ่งจะต่างกับการทำงานในชั้นล่างๆ ที่สนใจเพียงการทำให้บิตของข้อมูลรับส่งได้อย่างถูกต้องเท่านั้น แต่ชั้นพรีเซนเทชันจะสนใจในวากยสัมพันธ์ (Syntax) และความเปลี่ยนในความหมายของคำ (Semantic) ของข้อมูลที่ส่งผ่านด้วย

ตัวอย่างที่เห็นได้ชัดเจนก็คือ การเข้ารหัสข้อมูล โดยปกติโปรแกรมของผู้ใช้งานจะไม่แลกเปลี่ยนข้อมูลแต่เพียงข้อมูลดิบเท่านั้น จะมีข้อมูลอื่นๆ ด้วย เช่น ชื่อ, วันที่, จำนวนเงิน และใบเรียกเก็บเงิน

ข้อมูลเหล่านี้จะถูกแทนด้วยข้อมูลแบบตัวอักษรและตัวเลข ซึ่งคอมพิวเตอร์แต่ละเครื่องที่มีการแทนรหัสที่แตกต่างกันสามารถสื่อสารกันได้ รหัสที่ใช้งานรวมทั้งโครงสร้างของข้อมูลจะต้องถูกกำหนดให้มีรูปแบบที่ตรงกัน การกำหนดและการเปลี่ยนไปมาจากรหัสที่แทนอยู่ภายในเครื่องไปเป็นรหัสที่มีการใช้เป็นมาตรฐานในเครือข่ายเป็นหน้าที่ของชั้นพรีเซนเทชันนั่นเอง

นอกจากนี้ชั้นพรีเซนเทชันยังทำหน้าที่อื่นๆ อีก เช่น การลดขนาดของข้อมูล เพื่อเป็นการลดจำนวนบิตข้อมูลที่ต้องส่ง และยังอาจจะทำการเข้ารหัสเพื่อป้องกันไม่ให้มีการขโมยข้อมูลที่เป็นความลับได้

2.4.7 **ชั้นแอปพลิเคชัน** (Application layer)

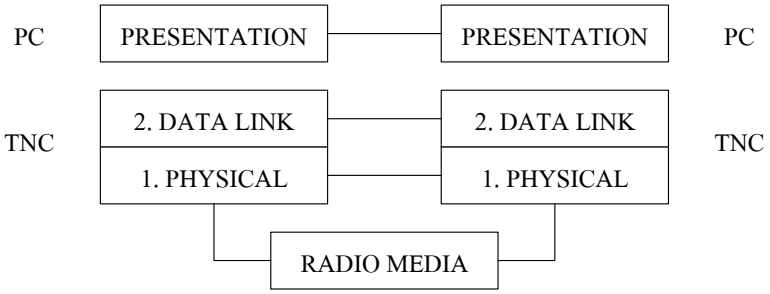
ภายในชั้นแอปพลิเคชัน จะประกอบด้วยโพรโทคอลชนิดต่างๆ มากมายที่มักจะเป็นที่ต้องการลองพิจารณาเทอร์มินัลชนิดต่างๆ ที่มีอยู่ในโลกนี้ แต่ละเครื่องจะมีลักษณะไม่เหมือนกัน มีลักษณะจอภาพขนาดไม่เท่ากัน มีการควบคุมการแสดงผลไม่เหมือนกัน เป็นต้น

วิธีทางหนึ่งที่จะแก้ปัญหานี้ก็คือ กำหนดเทอร์มินัลเสมือนสำหรับเครือข่าย (Network virtual terminal) ขึ้นมา ซึ่งเป็นตัวกลางที่โปรแกรมอื่นๆ จะติดต่อด้วย สำหรับเทอร์มินัลแต่ละแบบจะต้องมีโปรแกรมที่เครื่องเทอร์มินัลเพื่อแปลงรหัสของเทอร์มินัลเสมือน ไปเป็นการควบคุมเทอร์มินัลจริงๆ เช่น เมื่อโปรแกรมแก้ไขข้อความเลื่อนเคอร์เซอร์บนเทอร์มินัลเสมือนไปยังมุมบนด้านซ้าย โปรแกรมที่เทอร์-มินัลนี้จะต้องรับรู้และสามารถทำให้เคอร์เซอร์เลื่อนไปที่ตำแหน่งที่ถูกต้องจริงๆ ลักษณะของโปรแกรมเช่นนี้จะเป็นทำงานในชั้นแอปพลิเคชัน

การประยุกต์แบบอื่นๆ เช่น การถ่ายไฟล์ข้อมูลแต่ละระบบไฟล์จะมีวิธีการตั้งชื่อที่ต่างกัน มีวิธีการแทนตัวอักษรในบรรทัดแต่ละบรรทัดที่แตกต่างกัน ฉะนั้นการถ่ายไฟล์ข้อมูลระหว่างระบบที่ต่างกันจะเกี่ยวข้องกับความไม่เหมือนกันเหล่านี้ งานเหล่านี้เป็นหน้าที่โดยตรงของชั้นแอปพลิเคชัน เช่นเดียวกับในไปรษณีย์อิเล็กทรอนิกส์, การส่งงานระยะไกล, และอื่นๆ อีกมากมาย

2.5 **สถาปัตยกรรมเครือข่ายวิทยุกลุ่มข้อมูล**

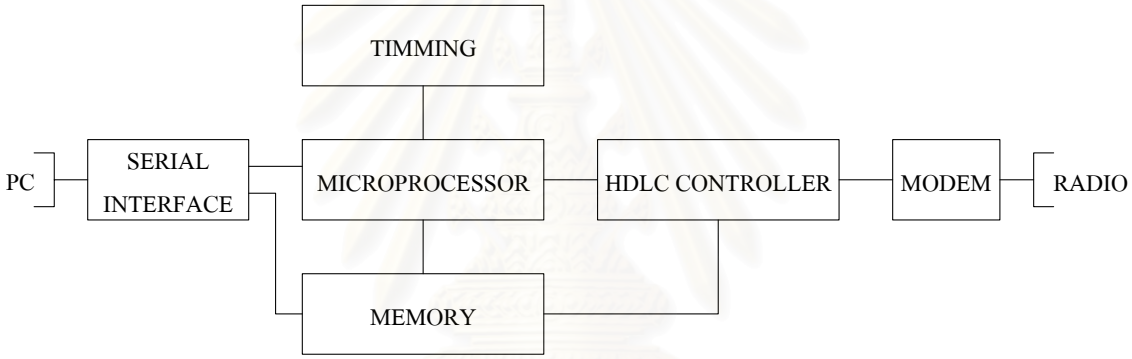
จากที่กล่าวมาแล้วในตอนต้นสรุปได้ว่า พื้นฐานของระบบเครือข่ายวิทยุกลุ่มข้อมูลประกอบด้วยตัวควบคุมเทอร์มินัลโนด (TNC) ทำหน้าที่ดูแลการทำงานในชั้นกายภาพ (ส่วนโมเด็ม) และชั้นเชื่อมโยงข้อมูล (ส่วนสร้างกลุ่มข้อมูล) ตามแบบจำลอง OSI ส่วนคอมพิวเตอร์ที่ทำหน้าที่เป็นเทอร์มินัลนั้นมีการทำงานจัดอยู่ในชั้นพรีเซนเทชันตามแบบจำลอง OSI เนื่องจากทำหน้าที่ในการแปลงรหัสเลขฐาน 2 ไปเป็นรหัสแอสกี (ASCII) เพื่อแสดงผลข้อมูล และยังทำหน้าที่ดูแลคำสั่งและการตอบรับระหว่างคอมพิวเตอร์และ TNC ปัจจุบันเครือข่ายวิทยุกลุ่มข้อมูลที่ใช้โพรโทคอลแบบ AX.25 ยังไม่ได้มีการกำหนดมาตรฐานในชั้นเครือข่าย



รูปที่ 2.7 สถาปัตยกรรมเครื่อง TNC ในเครือข่ายวิทยุกลุ่มข้อมูล

2.5.1 ตัวควบคุมเทอร์มินัลโนด (TNC)

ตัวควบคุมเทอร์มินัลโนดเป็นอุปกรณ์ฮาร์ดแวร์ที่สำคัญในเครือข่ายวิทยุกลุ่มข้อมูล ทำหน้าที่ดูแลการทำงานบนเครือข่ายวิทยุกลุ่มข้อมูลในชั้นกายภาพและชั้นเชื่อมโยงข้อมูล มีโครงสร้างการทำงานดังรูปที่ 2.8



รูปที่ 2.8 โครงสร้างการทำงานภายในตัวควบคุมเทอร์มินัลโนด

จากรูปที่ 2.8 อธิบายการทำงานได้ดังนี้ ในขั้นตอนการส่งข้อมูลนั้น ข้อมูลจากเทอร์มินัล (คอมพิวเตอร์) จะถูกส่งผ่านพอร์ตอนุกรม RS-232 ไปยังตัวควบคุมเทอร์มินัลโนด เมื่อตัวควบคุมเทอร์-มินัลโนดได้รับข้อมูลจากคอมพิวเตอร์แล้วก็จะจัดการกับข้อมูลนั้น โดยส่งข้อมูลไปยังตัวควบคุม HDLC (High-level data link controller) ซึ่งเป็นส่วนสร้างกลุ่มข้อมูลให้เป็นเฟรม โดยมีส่วนกำเนิดฐานเวลา (Timing) ควบคุมจังหวะในการรับส่งข้อมูล และหน่วยความจำ (Memory) ทำหน้าที่เก็บข้อมูลชั่วคราว จากนั้นเฟรมข้อมูลที่ได้จากตัวควบคุม HDLC จะถูกส่งผ่านไปยังส่วนโมเด็มเพื่อมอดูเลตสัญญาณแล้วส่งให้เครื่องรับส่งวิทยุเพื่อแผ่กระจายเป็นคลื่นวิทยุออกไป

ส่วนขั้นตอนการรับข้อมูล ข้อมูลที่เครื่องรับส่งวิทยุรับเข้ามาจะส่งผ่านไปยังส่วนโมเด็ม ซึ่งจะทำการดีมอดูเลตสัญญาณแล้วส่งต่อให้กับตัวควบคุม HDLC ถอดกลุ่มข้อมูลที่ได้รับให้เป็นข้อมูลที่ถูกต้องแล้วส่งต่อไปยังเทอร์มินัลผ่านทางพอร์ต RS-232 ต่อไป เป็นอันเสร็จสิ้นการทำงานของตัวควบคุมเทอร์มินัลโนด

ตัวอย่างคุณสมบัติทางเทคนิคของตัวควบคุมเทอร์มินัลโนดในวงการวิทยุสมัครเล่น ซึ่งนักวิทยุสมัครเล่นใช้เป็นอุปกรณ์รับส่งข้อมูลระหว่างสถานีสื่อสาร ที่มีผู้ผลิตและจำหน่ายนิยมกันอย่างแพร่หลายมีดังนี้

โพรโทคอล	:	AX.25 level 2 เวอร์ชัน 2.2
โมเด็ม	:	Bell 202
พอร์ตสื่อสาร	:	RS-232
อัตรารับส่งข้อมูล	:	300, 1200, 2400, 4800, 9600, 19,200 bps
สายสัญญาณด้าน RS-232	:	DB 25, DB9
สายสัญญาณด้านโมเด็ม	:	3.5 มม. สำหรับอินพุต (ลำโพง)
		2.5 มม. สำหรับเอาต์พุต (ไมโครโฟน/PTT)
วิธีการมอดูเลต	:	FSK
ระดับสัญญาณอินพุต	:	100 mVp-p - 2 Vp-p
ระดับสัญญาณเอาต์พุต	:	0 - 450 mVp-p

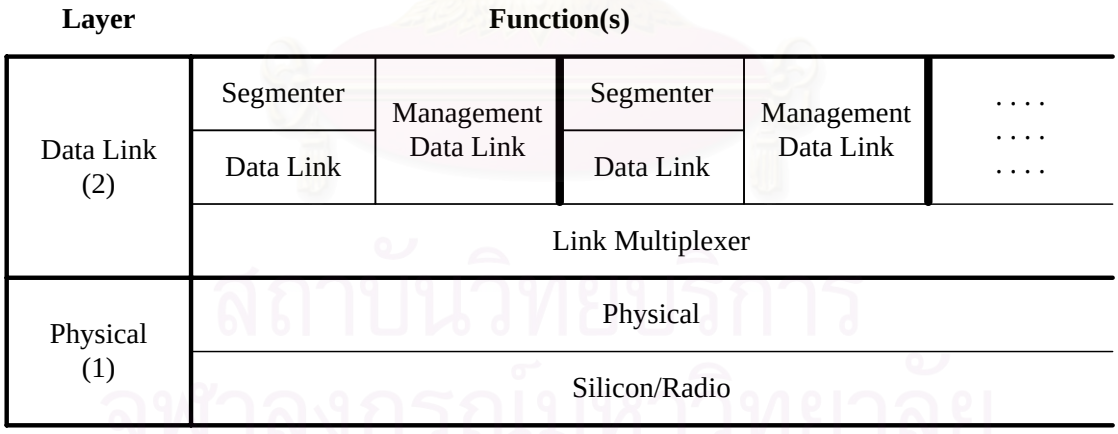
2.6 **โพรโทคอล** AX.25 [3,4]

ในการติดต่อสื่อสารเพื่อรับส่งข้อมูลระหว่างของสถานีปลายทาง 2 จุดใดๆ ให้มีความถูกต้องเชื่อถือได้ จำเป็นต้องมีการกำหนดโพรโทคอล ที่สามารถรับส่งข้อมูลได้บนเส้นทางของระบบสื่อสาร AX.25 เป็นโพรโทคอลมาตรฐานที่มีโครงสร้างสอดคล้องกับแบบจำลอง OSI โดยให้บริการใน 2 ชั้นแรก คือ ชั้นกายภาพ และชั้นเชื่อมโยงข้อมูล เพื่อรับประกันว่าข้อมูลที่ส่งจากอุปกรณ์ต้นทาง จะได้รับโดยอุปกรณ์ปลายทางอย่างถูกต้องไม่มีข้อผิดพลาด มีความเชื่อถือได้

โพรโทคอลส่วนมากในชั้นเชื่อมโยงข้อมูลจะสมมติว่าในระบบเครือข่ายจะมีสถานีหลักหรือสถานีแม่ข่าย ที่นิยมเรียกกันว่า DCE (Data circuit terminating equipment) ซึ่งเชื่อมต่อกับสถานีลูกข่ายที่อาจเพียง 1 สถานีหรือหลายสถานี ซึ่งนิยมเรียกกันว่า DTE (Data terminating equipment) การทำงานตามหลักการข้างต้นเรียกกันว่า หลักการทำงานแบบไม่ได้ดุล (Unbalanced) ซึ่งไม่ที่จะนำมาใช้ในเครือข่ายวิทยุกลุ่มข้อมูล ในโพรโทคอล AX.25 จะจัดให้ทั้ง 2 สถานีปลายทางมีสถานะอยู่ในระดับเดียวกัน ไม่มีความแตกต่างระหว่างอุปกรณ์ปลายทางทั้ง 2 แห่ง โดยยึดหลักการทำงานแบบได้ดุล (Balanced) และเรียกอุปกรณ์ปลายทางประเภทนี้ว่า DXE ซึ่งจะพบได้ในเครือข่ายวิทยุกลุ่มข้อมูลสมัครเล่น

จากข้อกำหนดของโพรโทคอล AX.25 สามารถนำมาใช้เป็นโพรโทคอลสำหรับการเข้าถึงช่องสัญญาณที่เชื่อมโยงในเครือข่ายข้อมูลของวิทยุสมัครเล่นได้เป็นอย่างดี ทั้งแบบครึ่งดูเพล็กซ์และเต็มดูเพล็กซ์ โดยจะใช้งานได้ดีเมื่อใช้งานแบบติดต่อกันโดยตรงระหว่างสถานีวิทยุกลุ่มข้อมูลสมัครเล่น 2 สถานี โดยที่ AX.25 สมมติว่าทั้งสองสถานีมีการเชื่อมต่อแบบได้ดุล กล่าวคือสถานีใด ๆ ก็ตามสามารถที่จะเป็นฝ่ายที่ติดต่อไปยังสถานีอื่นได้เสมอ ไม่มีลำดับชั้นแต่อย่างใด และเรียกอุปกรณ์ที่ทำหน้าที่เชื่อมต่อในแต่ละสถานีนี้ว่า ตัวควบคุมเทอร์มินัลโนด (TNC) หรือ DXE นอกจากนี้โพรโทคอลนี้ยังอนุญาตให้สถานีวิทยุกลุ่มข้อมูลใดๆ สามารถเชื่อมโยงข้อมูลกับสถานีอื่นได้มากกว่า 1 สถานี ถ้าอุปกรณ์ TNC มีความสามารถเพียงพอ และยังไม่มีข้อห้ามสำหรับการเชื่อมโยงเข้ากับตัวเอง (Self-connections) การเชื่อมโยงแบบนี้จะเกิดขึ้นเมื่อฟิลด์ที่อยู่เป็นค่าเดียวกันทั้งที่อยู่ต้นทางและที่อยู่ปลายทาง

AX.25 เป็นโพรโทคอลแบบ OSI-oriented, Bit-oriented, Synchronous และ Full duplex โดยโพรโทคอลในชั้นเชื่อมต่อข้อมูลอ้างอิงมาจากมาตรฐาน ISO ตามมาตรฐาน IS 3309, 4335 และ 7809 ในส่วนของตัวควบคุมการจัดเฟรมข้อมูล (HDLC) และสอดคล้องกับมาตรฐาน CCITT X.25 ตามข้อแนะนำ Q.920 และ Q.921 (LAP-B) ในเรื่องความสามารถในการใช้ช่องสัญญาณในชั้นกายภาพร่วมกันระหว่างชั้นเชื่อมโยงข้อมูลหลายชุด โดยใช้ค่าตำแหน่งเป็นตัวบ่งชี้ชั้นเชื่อมโยงข้อมูล โพรโทคอล AX.25 มีข้อแตกต่างออกไปตรงที่มีการเพิ่มขนาดของฟิลด์ที่อยู่ และเพิ่มเฟรมแบบไม่มีลำดับ (Unnumbered information) เข้ามา



รูปที่ 2.9 แบบจำลองโครงสร้างการทำงานของ AX.25 (มีการเชื่อมต่อข้อมูลตั้งแต่ 1 ชุดขึ้นไป)

2.6.1 โครงสร้างเฟรม

ในชั้นเชื่อมโยงข้อมูลจะแบ่งข้อมูลที่ต้องการส่งออกเป็นกลุ่มย่อยๆ เรียกว่า เฟรม (Frame) AX.25 ได้แบ่งเฟรมออกเป็น 3 ชนิด ได้แก่

- เฟรมข่าวสาร (Information frame หรือ I frame)
- เฟรมตรวจตรา (Supervisory frame หรือ S frame)
- เฟรมไม่มีหมายเลขลำดับ (Unnumbered frame หรือ U frame)

ภายในเฟรมแบ่งเป็นกลุ่มข้อมูลย่อย ๆ เรียกว่า ฟิลด์ (Field) ซึ่งแสดงโครงสร้างของเฟรมได้ดังรูปที่ 2.10 และรูปที่ 2.11 โดยบิตเริ่มต้นอยู่ทางซ้ายสุดของเฟรม

Flag	Address	Control	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110

รูปที่ 2.10 โครงสร้างเฟรมตรวจตรา (S) และเฟรมไม่มีหมายเลขลำดับ (U)

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

รูปที่ 2.11 โครงสร้างเฟรมข่าวสาร (I)

ทุกฟิลด์จะต้องมีจำนวนบิตข้อมูลเป็น 8 เท่าของจำนวนเต็ม และทุกฟิลด์ยกเว้นฟิลด์ FCS เริ่มส่งบิตที่ 0 ก่อน โดยฟิลด์ FCS เริ่มส่งบิตที่ 15 ก่อน หน้าที่การทำงานในแต่ละฟิลด์มีดังนี้

ฟิลด์แฟล็ก (Flag field)

มีขนาด 1 ไบต์สำหรับบอกจุดเริ่มต้นและสิ้นสุดของเฟรม โดยเฟรม 2 เฟรมอาจมีแฟล็กร่วมกันได้ คือ แฟล็กปิดท้ายของเฟรมแรกและเป็นแฟล็กเริ่มต้นของเฟรมถัดไป ค่าของแฟล็ก คือ “01111110” (7E hex)

ฟิลด์ที่อยู่ (Address field)

ระบุทั้งที่อยู่ของผู้ส่ง (Source address) และที่อยู่ของผู้รับ (Destination address) รวมทั้งอาจมีที่อยู่ของสถานีทวนสัญญาณ (Digipeater) ด้วย มีขนาด 14 ถึง 28 ไบต์ ซึ่งจะอธิบายรายละเอียดในภายหลัง

Destination Address						SSID	Source Address						SSID	1st Digitpeater						SSID	
A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	

รูปที่ 2.12 โครงสร้างของฟิลด์ที่อยู่

ฟิลด์ควบคุม (Control field)

บอกว่าเป็นเฟรมชนิดใด รวมทั้งควบคุมลักษณะการเชื่อมต่อในชั้นเชื่อมโยงข้อมูล มีขนาด 1 ถึง 2 ไบต์ ซึ่งจะอธิบายเพิ่มเติมในภายหลัง

Type	7	6	5	4	3	2	1	0
I Frame	N(R)			P	N(S)			0
S Frame	N(R)			P/F	S	S	0	1
U Frame	M	M	M	P/F	S	S	1	1

รูปที่ 2.13 ตัวอย่างโครงสร้างฟิลด์ควบคุมขนาด 1 ไบต์

ฟิลด์ PID (Protocol identifier field)

จะปรากฏเฉพาะในเฟรมข่าวสาร คือ เฟรม I และเฟรม UI เท่านั้น เป็นส่วนที่บอกว่าโพรโทคอลที่ติดต่ออยู่ในชั้นที่ 3 เป็นโพรโทคอลชนิดใด ทุกรูปแบบของ yy11yyyy และ yy00yyyy นอกเหนือจากตัวอย่างในรูปที่ 2.13 จะถูกสงวนไว้สำหรับอนาคตที่จะสร้างโพรโทคอลชั้นที่ 3 ข้อตกลงสำหรับรูปแบบฟิลด์ PID นี้เป็นที่เข้าใจกันในวงการวิทยุสมัครเล่น ผู้ที่จะสร้างโพรโทคอลในชั้นที่ 3 นี้จะต้องติดต่อกับ ARRL Ad Hoc committee เพื่อที่จะแนะนำการจัดรูปแบบข้อมูลสำหรับการสื่อสารในระบบดิจิทัล

รหัสฐาน 16	รหัสฐาน 2	ความหมาย
**	yy01yyyy	มีการสร้างโพรโทคอล AX.25 ในชั้นที่ 3 (สงวนไว้)
**	yy10yyyy	มีการสร้างโพรโทคอล AX.25 ในชั้นที่ 3 (สงวนไว้)
01	00000001	ISO8208/CCITT X.25 PLP
06	00000110	Compressed TCP/IP packet
07	00000111	Uncompressed TCP/IP packet
08	00001000	Segmentation flagment
C3	11000011	TEXNET datagram protocol
C4	11000100	Link quality protocol
CA	11001010	Appletalk
CB	11001011	Appletalk ARP

รูปที่ 2.14 ตัวอย่างรหัส PID และความหมาย

รหัสฐาน 16	รหัสฐาน 2	ความหมาย
CC	11001100	ARPA internet protocol
CD	11001101	ARPA address resolution
CE	11001110	FlexNet
CF	11001111	NET/ROM
F0	11110000	ไม่มีโพรโทคอลในชั้นที่ 3
FF	11111111	อักขระ Escape บอกว่ามีข้อมูลเพิ่มเติมเกี่ยวกับโพรโทคอลในชั้นที่ 3

รูปที่ 2.14 ตัวอย่างรหัส PID และความหมาย (ต่อ)

ฟิลด์ข่าวสาร (Information Field)

บรรจุข้อมูลที่ผู้ส่งต้องการส่ง ฟิลด์ข่าวสารนี้จะมีได้เฉพาะในเฟรม 5 ชนิดต่อไปนี้เท่านั้น คือ เฟรม I, UI, XID, TEST และ FRMR โดยทั่วไปฟิลด์ข่าวสารจะมีขนาดไม่เกิน 256 ไบต์โดยยังไม่รวมบิต '0' ที่แทรกเข้ามาเพื่อป้องกันการสับสนกับฟิลด์แฟล็กที่ปรากฏ

ตัวตรวจสอบเฟรม (Frame check sequence: FCS)

มีขนาด 16 บิต ใช้สำหรับตรวจสอบความถูกต้องของข้อมูลที่ได้รับ โดยคำนวณตามมาตรฐาน ISO 3309 (HDLC) ใช้ตัวคำนวณ CRC-CCITT คือ $X^{16} + X^{15} + X^2 + 1$

การแทรกบิต (Bit stuffing)

เพื่อรับประกันว่าไม่มีข้อมูลส่วนใดในเฟรมไปซ้ำกับค่าแฟล็ก ทุก ๆ 5 บิตของบิต '1' ที่ติดกัน (ยกเว้นฟิลด์แฟล็ก) จะถูกแทรกด้วยบิต '0' หลังบิต '1' ตัวที่ 5 โดยเมื่อผู้รับได้รับบิต '1' ติดกัน 5 บิต บิต '0' ที่ตามหลังบิต '1' ตัวที่ 5 จะถูกตัดออกไป

ลำดับการส่งของบิต (Order of bit transmission)

แต่ละฟิลด์จะส่งบิต LSB ก่อนยกเว้นฟิลด์ FCS จะส่งบิต MSB ก่อน

เฟรมที่ไม่สมบูรณ์ (Invalid frame)

เฟรมจะไม่มีความสมบูรณ์ถ้ามีข้อมูลน้อยกว่า 136 บิต เมื่อรวมแฟล็กเปิดและปิดเฟรมด้วย, ไม่มีแฟล็กเปิดหรือแฟล็กปิด หรือมีจำนวนบิตข้อมูลไม่เป็น 8 เท่าของจำนวน-เต็ม โดยไม่รวมบิต '0' ที่แทรกเข้ามา

การยกเลิกเฟรม (Frame abort)

เฟรมจะถูกยกเลิกถ้ามีบิต ‘1’ ติดต่อกันอย่างน้อย 15 บิตโดยที่ไม่มีบิต ‘0’ แทรก

ช่วงเวลาระหว่างเฟรม (Inter-frame time fill)

เมื่อ DXE ต้องการให้การส่งข้อมูลยังคงทำงานอยู่ขณะที่ไม่การส่งข้อมูลที่แท้จริง ให้ส่งค่าแฟล็กไปอย่างต่อเนื่องจนกระทั่งมีการส่งข้อมูลที่แท้จริงเกิดขึ้น

2.6.2 การเข้ารหัสฟีลด์ที่อยู่

ฟีลด์ที่อยู่ของเฟรมทุกๆ เฟรมจะประกอบด้วยส่วนสำคัญ 2 ส่วน ส่วนแรกคือฟีลด์ที่อยู่ของสถานีต้นทาง และส่วนที่สองคือฟีลด์ที่อยู่ของสถานีปลายทาง ฟีลด์ที่อยู่ทั้งสองฟีลด์นี้จะใช้สัญญาณเรียกขาน (Callsign) เป็นตัวกำหนดรหัสที่อยู่ ซึ่งแทนด้วยรหัสแอสกีของตัวอักษรพิมพ์ใหญ่และตัวเลข และ SSID (Secondary station identifier) ถ้ามีการใช้สถานีทวนสัญญาณ จะมีสัญญาณเรียกขานของสถานีทวนสัญญาณนั้นรวมอยู่ในฟีลด์ที่อยู่ด้วย

ฟีลด์ที่อยู่ของ HDLC สามารถขยายขนาดได้ (เมื่อมีสถานีทวนสัญญาณ) โดยการกำหนดค่าบิต LSB ของแต่ละไบต์เป็นบิตขยาย (Extension bit) เมื่อบิตขยายนี้มีค่าเป็น ‘0’ แสดงว่าไบต์ถัดไปยังเป็นส่วนของฟีลด์ที่อยู่ ถ้าเป็น ‘1’ แสดงว่าเป็นไบต์สุดท้ายของฟีลด์ที่อยู่ของเฟรม HDLC จากหลักการดังกล่าวทำให้สัญญาณเรียกขานของสถานีวิทยุต้องเลื่อนไปทางซ้าย 1 บิต

การเข้ารหัสฟีลด์ที่อยู่แบบไม่มีสถานีทวนสัญญาณ

ถ้าไม่มีการใช้สถานีทวนสัญญาณ จะเข้ารหัสฟีลด์ที่อยู่ได้ดังรูปที่ 2.15 ได้แก่ A1-A14 ขนาด 14 ไบต์ ประกอบด้วยฟีลด์ที่อยู่ย่อย 2 ฟีลด์ คือฟีลด์ที่อยู่ย่อยของสถานีปลายทางขนาด 7 ไบต์เป็นส่วนแรกในตำแหน่ง A1-A7 เหตุที่ต้องไว้เป็นส่วนแรกเพื่อให้เวลากับสถานีรับข้อมูลในการตรวจสอบฟีลด์ที่อยู่ย่อยของสถานีปลายทาง อีกส่วนคือฟีลด์ที่อยู่ย่อยของสถานีต้นทางขนาด 7 ไบต์ในตำแหน่ง A8-A14 จะถูกส่งต่อจากฟีลด์ที่อยู่ย่อยของสถานีปลายทาง การเข้ารหัสทั้งฟีลด์ที่อยู่ต้นทางและฟีลด์ที่อยู่ปลายทางใช้หลักการเดียวกัน และบิตสุดท้ายเป็นบิตขยายเช่นเดียวกัน

Destination Address						**SSID**	**Source Address**						**SSID**
A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14

รูปที่ 2.15 การเข้ารหัสฟีลด์ที่อยู่แบบไม่มีสถานีทวนสัญญาณ

ในฟีลด์แต่ละฟีลด์ ที่อยู่ย่อยจะปิดท้ายด้วย SSID ฟีลด์ SSID นี้จะทำให้ผู้ใช้ที่มีสัญญาณเรียกขานเดียวกันแต่มีมากกว่า 1 สถานี สามารถรับข่าวสารที่ส่งมาได้เช่นเดียวกัน ประโยชน์ในส่วนนี้จะใช้เมื่อต้องการเพิ่มสถานีทวนสัญญาณเข้ามาในระบบ บิต C และบิต H จะถูกนำมาใช้ ส่วนบิต R2 จะสงวนไว้ใช้ในอนาคต รูปที่ 2.16 เป็นเฟรม AX.25 ทั่วไปในกรณีไม่มีสถานีทวนสัญญาณ

Octet	ASCII	Bin Data	Hex Data
Flag		01111110	7E
A1	K	10010110	96
A2	8	01110000	70
A3	M	10011010	9A
A4	M	10011010	9A
A5	0	10011110	9E
A6	space	01000000	40
A7	SSID	11100000	E0
A8	W	10101110	AE
A9	B	10000100	84
A10	4	01101000	68
A11	J	10010100	94
A12	F	10001100	8C
A13	I	10010010	92
A14	SSID	01100001	61
Control	I	00111110	3E
PID	None	11110000	F0
FCS	Part 1	XXXXXXXX	XX
FCS	Part 2	XXXXXXXX	XX
Flag		01111110	7E

รูปที่ 2.16 เฟรม AX.25 แบบไม่มีสถานีทวนสัญญาณ

เฟรมที่แสดงเป็นเฟรมข่าวสาร (I) ที่ไม่มีการส่งข้อมูลผ่านสถานีทวนสัญญาณโดยส่งจากสถานี WB4JFI (SSID=0) ไปยังสถานี K8MM0, ไม่มีโพรโทคอลในชั้นที่ 3, บิต P/F = ‘1’, มีหมายเลขลำดับ

ของการรับ (Receive sequence number) N(R) เป็น 1 และมีหมายเลขลำดับของการส่ง (Send sequence number) N(S) เป็น 7

การเข้ารหัสฟิลด์ที่อยู่ย่อยของสถานีปลายทาง

รูปที่ 2.17 เป็นการเข้ารหัสสัญญาณเรียกขานในฟิลด์ที่อยู่ย่อยของสถานีปลายทางในตำแหน่ง A1 ถึง A7

Octet	ASCII	Bin Data	Hex Data
A1	K	10010110	96
A2	8	01110000	70
A3	M	10011010	9A
A4	M	10011010	9A
A5	0	10011110	9E
A6	space	01000000	40
A7	SSID	11100000	E0
A7	SSID	CRRSSID0	-

รูปที่ 2.17 การเข้ารหัสฟิลด์ที่อยู่ย่อยของสถานีปลายทาง

จากรูปที่ 2.17 อธิบายได้ดังนี้

- ไบต์บนสุด (A1) เป็นไบต์แรกที่จะส่งออกไป โดยจะส่งจากบิตที่ 0 ก่อนและส่งบิตที่ 7 เป็นบิตสุดท้าย
- รหัสของสัญญาณเรียกขานมาตรฐานแทนด้วยรหัสแอสกีของตัวอักษรพิมพ์ใหญ่และตัวเลข ขนาด 7 บิต โดยรหัสแอสกี 7 บิตนี้จะเรียงบิต MSB อยู่ซ้ายสุด และเหลือบิตสุดท้าย 1 บิต เป็นบิตขยาย ถ้าสัญญาณเรียกขานมีความยาวน้อยกว่า 6 อักษร ส่วนที่เหลือจะป้อนเป็นรหัสแอสกีของช่องว่าง (Space) ระหว่างตัวอักษรสัญญาณเรียกขานตัวสุดท้ายและ SSID
- บิตที่ 0 ของไบต์แต่ละไบต์เป็นบิตขยายรหัสที่อยู่ของ HDLC เมื่อเป็น ‘0’ แสดงว่าไบต์ถัดไปยังเป็นฟิลด์ที่อยู่ แต่ถ้าเป็น ‘1’ แสดงว่าสิ้นสุดฟิลด์ที่อยู่แล้ว
- บิต C เป็นบิตคำสั่ง/ตอบสนองของเฟรม AX.25
- บิต R เป็นบิตที่สงวนไว้ อาจนำไปใช้ตามวัตถุประสงค์ที่ต้องการในเครือข่ายเฉพาะ เมื่อไม่ใช้บิตนี้จะมีค่าเป็น ‘1’

- SSID = “0000” ถูกสงวนไว้สำหรับสถานีแบบ AX.25 สถานีแรก ซึ่งเป็นมาตรฐานของการเชื่อมต่อโดยปกติสำหรับสถานีแรก

การเข้ารหัสฟีลด์ที่อยู่ย่อยของสถานีต้นทาง

รูปที่ 2.18 เป็นการเข้ารหัสสัญญาณเรียกขานในฟีลด์ที่อยู่ย่อยของสถานีต้นทางในตำแหน่ง A8 ถึง A14 ซึ่งมีหลักการเดียวกันกับการเข้ารหัสในฟีลด์ที่อยู่ย่อยของสถานีปลายทาง สังเกตว่าบิตที่ 0 ของไบต์ SSID เป็น ‘1’ แสดงว่าสิ้นสุดฟีลด์ที่อยู่ คือไม่มีสถานีทวนสัญญาณในการส่งข้อมูลสำหรับเฟรมนี้

Octet	ASCII	Bin Data	Hex Data
A8	W	10101110	AE
A9	B	10000100	84
A10	4	01101000	68
A11	J	10010100	94
A12	F	10001100	8C
A13	I	10010010	92
A14	SSID	01100001	61
A14	SSID	CRRSSID0	-

รูปที่ 2.18 การเข้ารหัสฟีลด์ที่อยู่ย่อยของสถานีต้นทาง

การเข้ารหัสฟีลด์ที่อยู่ย่อยของสถานีทวนสัญญาณ

ถ้ามีการส่งเฟรมข้อมูลผ่านสถานีทวนสัญญาณ จะต้องเพิ่มฟีลด์ที่อยู่ย่อยของสถานีทวนสัญญาณต่อท้ายฟีลด์ที่อยู่เดิม โดยอนุญาตให้ใช้สถานีทวนสัญญาณได้มากกว่า 1 สถานีในการแบ่งกันใช้ช่องสัญญาณ ถ้ามีฟีลด์ที่อยู่ย่อยส่วนนี้เพิ่มเติมเข้ามา ไบต์สุดท้ายของฟีลด์ที่อยู่ย่อยของสถานีต้นทาง (A14) จะต้องมีบิต 0 เป็น ‘0’ เพื่อบอกว่ามีข้อมูลของฟีลด์ที่อยู่เพิ่มเติมอีก การเข้ารหัสฟีลด์ที่อยู่ย่อยส่วนนี้ใช้หลักการเช่นเดียวกับฟีลด์ที่อยู่ย่อยต้นทางและปลายทาง ยกเว้นบิต MSB ของไบต์สุดท้าย ซึ่งเรียกว่าบิต H จะเป็นบิตที่แสดงว่าเฟรมนั้นเป็นเฟรมที่ถูกทวนสัญญาณหรือไม่

เมื่อต้องการส่งเฟรมข้อมูลผ่านสถานีทวนสัญญาณ บิต H ในฟีลด์ที่อยู่ย่อยของสถานีทวนสัญญาณจะถูกตั้งค่าให้เป็น ‘0’ บนเฟรมที่ส่งไปให้สถานีทวนสัญญาณนั้น เมื่อสถานีทวนสัญญาณนั้นได้รับเฟรมข้อมูลดังกล่าว และตรวจสอบพบว่าบิต H ในฟีลด์ที่อยู่ย่อยของสถานีทวนสัญญาณที่ตรงกับที่อยู่ของสถานีนั้นมีค่าเป็น ‘0’ ก็จะเปลี่ยนบิต H นั้นเป็น ‘1’ แล้วจึงส่งเฟรมนั้นต่อไป โดยที่ข้อมูลส่วนอื่นๆ ยังเหมือนเดิม (ยกเว้นฟีลด์ FCS ที่ต้องคำนวณใหม่)

Octet	ASCII	Bin Data	Hex Data
A15	N	10011100	9C
A16	7	01101110	6E
A17	L	10011000	98
A18	E	10001010	8A
A19	M	10011010	9A
A20	space	01000000	40
A21	SSID	11100011	E3
A21	SSID	HRRSSID1	

รูปที่ 2.19 การเข้ารหัสฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณ

Octet	ASCII	Bin Data	Hex Data
Flag		01111110	7E
A1	K	10010110	96
A2	8	01110000	70
A3	M	10011010	9A
A4	M	10011010	9A
A5	0	10011110	9E
A6	Space	01000000	40
A7	SSID	11100000	E0
A8	W	10101110	AE
A9	B	10000100	84
A10	4	01101000	68
A11	J	10010100	94
A12	F	10001100	8C
A13	I	10010010	92
A14	SSID	01100000	60

รูปที่ 2.20 เฟรม AX.25 แบบมีสถานีทวนสัญญาณ

Octet	ASCII	Bin Data	Hex Data
A15	N	10011100	9C
A16	7	01101110	6E
A17	L	10011000	98
A18	E	10001010	8A
A19	M	10011010	9A
A20	space	01000000	40
A21	SSID	11100011	E3
Control	I	00111100	3C
PID	none	11110000	F0
FCS	part 1	XXXXXXXX	XX
FCS	part 2	XXXXXXXX	XX
Flag		01111110	7E

รูปที่ 2.20 เฟรม AX.25 แบบมีสถานีทวนสัญญาณ (ต่อ)

เฟรมตัวอย่างในรูปที่ 2.20 นี้แก้ไขจากเฟรมในรูปที่ 2.16 โดยเพิ่มฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณ (N7LEM, SSID = 1) บิต H มีค่าให้เป็น '1' แสดงถึงว่าเฟรมนี้ถูกส่งมาจากสถานีทวนสัญญาณนั่นเอง

การทำงานแบบมีสถานีทวนสัญญาณหลายสถานี

โพรโทคอล AX.25 สามารถรองรับสถานีทวนสัญญาณได้มากกว่า 1 สถานี ซึ่งโดยทั่วไปสำหรับการทำงานในชั้นที่ 2 จะมีสถานีทวนสัญญาณได้ไม่เกิน 2 สถานี แต่ก็สามารถมีมากกว่านี้ได้ด้วยวิธีการขยายฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณ คือเมื่อมีสถานีทวนสัญญาณมากกว่า 1 สถานี ฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณสถานีแรกจะต่อท้ายฟิลด์ที่อยู่ย่อยของสถานีต้นทาง ส่วนฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณสถานีถัดไปจะต่อท้ายฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณก่อนหน้านี้ เมื่อสถานีทวนสัญญาณได้รับเฟรมข้อมูลจะส่งต่อเฟรมนั้นออกไปก็ต่อเมื่อ

- ที่อยู่ของสถานีนั้นตรงกับที่อยู่ในฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณ
- บิต H ในฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณนั้นเป็น '0'
- บิต H ในฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณก่อนหน้านี้เป็น '1' ทั้งหมด

หากข้อมูลในฟิลด์ที่อยู่สอดคล้องกับข้อกำหนดข้างต้น สถานีทวนสัญญาณจะกำหนดบิต H เป็น ‘1’ เฉพาะในฟิลด์ที่อยู่ย่อยที่ตรงกับที่อยู่ของสถานีนั้น และจะเป็นเช่นนี้สำหรับสถานีทวนสัญญาณถัดไป จนเฟรมข้อมูลส่งถึงสถานีปลายทาง สถานปลายทางสามารถตรวจสอบเส้นทางการส่งเฟรมมาได้โดยตรวจสอบฟิลด์ที่อยู่ของเฟรม

จำนวนของสถานีทวนสัญญาณเป็นค่าที่แปรเปลี่ยนได้ โดยฟิลด์ที่อยู่ย่อยของสถานีทวนสัญญาณที่ไม่ใช่สถานีสุดท้ายจะมีบิตขยายของแต่ละไบต์เป็น ‘0’ ยกเว้นสถานีสุดท้ายจะถูกกำหนดให้มีค่าเป็น ‘1’ เป็นการแสดงว่าสิ้นสุดฟิลด์ที่อยู่แล้ว

การทำงานแบบมีสถานีทวนสัญญาณหลายสถานีสำหรับการส่งข้อมูลระยะไกลนี้ เป็นที่คาดกันว่าใช้ในลักษณะนี้ไปชั่วคราวก่อนจนกว่าโพรโทคอลในระดับที่ 3 จะถูกนำมาใช้ และเมื่อถึงเวลานั้นวิธีการทำงานที่กล่าวมาในเรื่องของสถานีทวนสัญญาณหลายสถานีก็จะถูกยกเลิกไป

2.6.3 การเข้ารหัสฟิลด์ควบคุม

ฟิลด์ควบคุมเป็นส่วนที่ใช้แสดงชนิดของเฟรม, คำสั่ง และการตอบสนอง จากสถานีหนึ่งไปยังอีกสถานีหนึ่ง เพื่อควบคุมการเชื่อมโยงข้อมูลระหว่าง 2 สถานีในเครือข่าย โดยฟิลด์ควบคุมใน AX.25 ใช้หลักการแบบเดียวกับฟิลด์ควบคุมของ CCITT X.25 ในส่วนการทำงานแบบได้ดุล ฟิลด์ควบคุมมี 2 แบบ คือ ฟิลด์ควบคุมขนาด 2 ไบต์ และฟิลด์ควบคุมขนาด 1 ไบต์ ดังรูปที่ 2.21 และ 2.22

Control-Field Type	Control-Field Bits															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I Frame	N(R)							P	N(S)							0
S Frame	N(R)							P/F	0	0	0	0	S	S	0	1

รูปที่ 2.21 รูปแบบของฟิลด์ควบคุมขนาด 2 ไบต์

Control-Field Type	Control-Field Bits							
	7	6	5	4	3	2	1	0
I Frame	N(R)			P	N(S)			0
S Frame	N(R)			P/F	S	S	0	1
U Frame	M	M	M	P/F	S	S	1	1

รูปที่ 2.22 รูปแบบของฟิลด์ควบคุมขนาด 1 ไบต์

ในที่นี้จะขอกล่าวถึงเฉพาะฟิลด์ควบคุมขนาด 1 ไบต์เท่านั้น จากรูปที่ 2.22 อธิบายได้ดังนี้

- บิต 0 เป็นบิตแรก และบิต 7 เป็นบิตสุดท้ายในการส่งของฟิลด์ควบคุม
- N(S) คือ หมายเลขลำดับของการส่ง (บิต 1 เป็น LSB)
- N(R) คือ หมายเลขลำดับของการรับ (บิต 5 เป็น LSB)
- บิต S เป็นบิตที่ทำหน้าที่ตรวจตราการเข้ารหัสบิต S
- บิต M เป็นบิตในเฟรม U ทำหน้าที่เป็นบิตแก้ไข (Modifier bit)
- บิต P/F เป็นโพลบิต (Poll) หรือบิตสุดท้าย (Final) เป็นบิตที่แสดงความแตกต่างระหว่างคำสั่งและการตอบสนอง ดังนั้นความแตกต่างระหว่างบิต P และบิต F ถูกสร้างโดยกฎของการหาที่อยู่ (Addressing rules)

รูปแบบของเฟรมข่าวสาร

ในเฟรมข่าวสาร บิต 0 จะถูกกำหนดค่าให้เป็น '0', N(S) คือหมายเลขลำดับของการส่งของผู้ส่ง (หมายเลขลำดับของการส่งของเฟรมที่กำลังจะส่ง) และ N(R) คือหมายเลขสำหรับรับของผู้ส่ง (หมายเลขสำหรับรับของเฟรมที่คาดว่าจะได้รับในเฟรมต่อไป)

รูปแบบของเฟรมตรวจตรา

เฟรมตรวจตราจะให้บิต 0 ของฟิลด์ควบคุมเป็น '1' และบิต 1 ของฟิลด์ควบคุมเป็น '0' เฟรมตรวจตราจะใช้เป็นเฟรมสำหรับควบคุมการสื่อสารในระบบเครือข่าย เช่น การตอบรับ การแสดงความจำนงในการส่งเฟรม I ใหม่ และเนื่องจากไม่มีส่วนของข่าวสาร ดังนั้นตัวแปรหมายเลขลำดับการส่งและการรับจะไม่มีการเพิ่ม ณ ที่สถานีรับส่ง

รูปแบบของเฟรมไม่มีหมายเลขลำดับ

เฟรมไม่มีหมายเลขลำดับจะมีบิต 0 และบิต 1 ของฟิลด์ควบคุมมีค่าเป็น '1' เป็นเฟรมที่ช่วยในการควบคุมการเชื่อมโยงในระบบสื่อสารโดยทำงานในส่วนที่เฟรมตรวจตราไม่ได้ทำ โดยมันจะทำหน้าที่ในการสร้างหรือยกเลิกการเชื่อมต่อข้อมูล เฟรมไม่มีหมายเลขลำดับบางครั้งส่งข่าวสารพร้อมทั้งฟิลด์ PID ด้วยบิต P/F

หมายเลขลำดับ (Sequence numbers)

เฟรมข่าวสารทุกเฟรมของ AX.25 ถูกกำหนดเป็นโมดูโล 8 (Modulo 8) คือมีหมายเลขลำดับจาก 0 ถึง 7 สามารถส่งเฟรม I ได้ถ้าจำนวนสัญญาณที่ยังไม่ตอบรับมีค่าน้อยกว่า 8

ตัวแปรสถานะการส่ง (Send state variable: V(S))

ตัวแปรสถานะการส่งเป็นตัวแปรที่อยู่ใน DXE และจะไม่ส่งตัวแปรชุดนี้ออกไป ตัวแปรนี้จะเก็บค่าหมายเลขลำดับที่จะส่งเฟรมข่าวสารเฟมถัดไป โดยตัวแปรชนิดนี้จะเปลี่ยนค่าไปทุกๆ ครั้งที่มีการส่งเฟรมข่าวสาร

หมายเลขลำดับของการส่ง (Send sequence number: N(S))

หมายเลขลำดับของการส่งจะพบในส่วนควบคุมของเฟรมข่าวสารทุกเฟรม มันจะเก็บค่าหมายเลขลำดับของเฟรมข่าวสารที่กำลังจะส่ง ก่อนการส่งเฟรมข่าวสารค่า N(S) จะเปลี่ยนไปเท่ากับค่า V(S)

ตัวแปรสถานะการรับ (Receive State Variable: V(R))

หมายเลขลำดับของการรับ จะอยู่ในทั้งเฟรม I และเฟรม S โดยก่อนที่จะส่งเฟรม I และเฟรม S จะตั้งค่า N(R) ในเฟรมให้เท่ากับ V(R) ซึ่งเป็นการตอบรับความถูกต้องของข้อมูลที่ได้รับทุกเฟรมจนถึงเฟรมที่ N(R)-1

หน้าที่การทำงานของบิตโพล (Poll)/**บิตสุดท้าย** (Final)

บิต P/F นี้จะใช้ในเฟรมทุกชนิด การใช้ในลักษณะเป็นคำสั่ง (Poll) เพื่อขอการตอบกลับแบบทันที การตอบกลับนี้โดยการกำหนดบิตการตอบสนอง (Final)

ฟิลด์ควบคุมของคำสั่งเฟรมข่าวสาร

หน้าที่ของคำสั่งในเฟรมข่าวสาร เพื่อที่จะส่งผ่านข้อมูลเป็นหมายเลขลำดับพร้อมทั้งส่งข่าวสาร การเข้ารหัสฟิลด์ควบคุมในเฟรมข่าวสารแสดงในรูปที่ 2.23 เฟรมเหล่านี้จะมีหมายเลขลำดับโดย N(S) เพื่อที่จะส่งผ่านรหัสควบคุมเหล่านี้ผ่านบนชั้นเชื่อมต่อข้อมูลของการสื่อสารข้อมูล

7	6	5	4	3	2	1	0
N(R)			P	N(S)			0

รูปที่ 2.23 ฟิลด์ควบคุมของเฟรมข่าวสาร (เฟรม I)

ฟิลด์ควบคุมของเฟรมตรวจตรา

การเข้ารหัสฟิลด์ควบคุมของเฟรมตรวจตราแสดงดังรูปที่ 2.24

Control Field Bits		7	6	5	4	3	2	1	0
Receive Ready	RR	N(R)			P/F	0	0	0	1
Receive Not Ready	RNR	N(R)			P/F	0	1	0	1
Reject	REJ	N(R)			P/F	1	0	0	1

รูปที่ 2.24 ฟิลด์ควบคุมของเฟรมตรวจตรา (เฟรม S)

คำสั่งและคำตอบแบบ "พร้อมรับ" (Receive ready: RR) แสดงเพื่อวัตถุประสงค์ดังนี้

- เพื่อชี้ให้เห็นว่าผู้ส่ง RR พร้อมที่จะรับเฟรมข่าวสารได้ในเวลานี้
- เพื่อตอบรับการรับเฟรมข่าวสารโดยบอกให้คู่สื่อสารรู้ว่ารับเฟรมข่าวสารจนถึง N(R)-1 โดยไม่มีความผิดพลาดเกิดขึ้น
- เพื่อยกเลิกภาวะไม่ว่าง (Busy condition) โดยก่อนนี้ได้ส่ง RNR ไป

สามารถเรียกดูสถานะของ DXE ของคู่สื่อสาร โดยการส่ง RR พร้อมบิต P ที่เป็น '1' ไปด้วย

คำสั่งและคำตอบแบบ "ยังไม่พร้อมรับ" (Receive not ready: RNR) เป็นตัวแจ้งให้คู่สถานีทราบว่าสถานีที่ส่งเฟรม RNR ออกไปไม่สามารถรับเฟรมข่าวสารได้เป็นการชั่วคราว คือสถานีที่ส่งเฟรม RNR อยู่ในสถานะไม่ว่าง โดยจะมีคำตอบรับในเฟรมจนถึง N(R)-1 ส่วนเฟรม N(R) และสูงขึ้นไปอีกจะไม่มีคำตอบรับ

สถานะของ RNR จะถูกยกเลิกไปเมื่อมีการส่งสัญญาณ UA, RR, REJ หรือ SABM ออกไป

สามารถเรียกดูสถานะของ DXE ที่เป็นคู่สื่อสาร โดยส่งเฟรมคำสั่ง RNR ที่บิต P เท่ากับ '1'

คำสั่งและคำตอบแบบ "ยกเลิก" (Reject: REJ) ใช้เรียกข้อมูลให้คู่สื่อสารส่งข่าวสารในเฟรมข่าวสารก่อนหน้านี้มาให้ใหม่ (โดยเริ่มจาก N(R), ทุกๆ เฟรมที่ถูกส่งมาแล้วตั้งแต่ลำดับหมายเลข N(R)-1 หรือน้อยกว่าถูกตอบรับกลับหมดแล้ว) โดยระบุเฟรมข่าวสารที่ต้องการให้ส่งมาใหม่ในเฟรม REJ ด้วย ทำให้ข่าวสารในเฟรมข่าวสารที่จะทำการส่งต่อไปต้องรอจนกว่าจะส่งเฟรมที่ขอให้ส่งมาใหม่ได้ส่งออกไปแล้ว

สถานะ REJ จะถูกยกเลิกก็ต่อเมื่อรับข่าวสารที่ได้ขอไปครบแล้ว

สามารถเรียกดูสถานะของ DXE ของคู่สื่อสารด้วยการส่งเฟรมคำสั่ง REJ ที่บิต P เท่ากับ '1'

ฟิลด์ควบคุมของเฟรมไม่มีลำดับหมายเลข

ฟิลด์ควบคุมของเฟรม U สามารถที่จะเป็นได้ทั้งคำสั่งและคำตอบ รูปที่ 2.25 แสดงเฟรม U ที่สร้างขึ้นในโพรโทคอล AX.25

Control Field		Type	7	6	5	4	3	2	1	0
Set Asynchronous Balance Mode	SABME	Cmd	1	1	1	P	1	1	1	1
Set Asynchronous Balance Mode	SABM	Cmd	0	0	1	P	1	1	1	1
Disconnect	DISC	Cmd	0	1	0	P	0	0	1	1

รูปที่ 2.25 ฟิลด์ควบคุมของเฟรมไม่มีหมายเลขลำดับ (เฟรม U)

Control Field		Type	7	6	5	4	3	2	1	0
Disconnect Mode	DM	Res	0	0	0	F	1	1	1	1
Unnumbered Acknowledge	UA	Res	0	1	1	F	0	0	1	1
Frame Reject	FRMR	Res	1	0	0	F	0	1	1	1
Unnumbered Information	UI	Either	0	0	0	P/F	0	0	1	1
Exchange Identification	XID	Either	1	0	1	P/F	1	1	1	1
Test	TEST	Either	1	1	1	P/F	0	0	1	1

รูปที่ 2.25 ฟิลด์ควบคุมของเฟรมไม่มีหมายเลขลำดับ (เฟรม U) (ต่อ)

คำสั่งกำหนดสถานีคู่สื่อสารให้อยู่ในโมดได้ดุลอะซิงโครนัส (Set asynchronous balanced mode: SABM) คำสั่ง SABM นี้ใช้กำหนดให้สถานีคู่สื่อสารทั้งสองอยู่ในโมดได้ดุลแบบอะซิงโครนัส คือสถานีสื่อสารทั้งสองข้างมีสถานะเท่าเทียมกัน

ไม่อนุญาตให้ใส่ฟิลด์ข่าวสารไว้ในเฟรมที่มีคำสั่งแบบ SABM ในขณะที่เฟรมคำสั่ง SABM ถูกส่งออกไปนั้น ถ้ามีเฟรมข่าวสารที่ยังไม่ได้รับการตอบรับกลับค้างอยู่เฟรมข่าวสารนั้นก็จะคงค้างอยู่อย่างนั้น และจะไม่ได้รับการตอบรับเช่นเดิม

การให้การตอบรับคำสั่ง SABM นั้น ทำโดยการส่งเฟรมคำตอบ UA ในทันทีที่ได้รับเฟรมคำสั่ง SABM แต่ถ้าคู่สื่อสารอยู่ในสถานะไม่พร้อมก็จะส่งเฟรมคำตอบ DM

ส่วนคำสั่ง *SABME (Set asynchronous balanced mode extended)* เป็นเฟรมคำสั่งที่ใช้กำหนดสถานีคู่สื่อสารในโมดสมดุลอะซิงโครนัสที่ใช้ฟิลด์ควบคุมขนาด 2 ไบต์

คำสั่งหยุดการส่งข้อมูล (Disconnect command: DISC) คำสั่ง DISC เป็นเฟรมคำสั่งที่ส่งออกไปเพื่อหยุดการส่งข้อมูล และไม่อนุญาตให้มีฟิลด์ข่าวสารอยู่ในเฟรมนี้ด้วย

ก่อนที่ทำการหยุดการส่งข้อมูลตามเฟรมคำสั่ง DISC ที่ได้รับ DXE ที่ได้รับเฟรมแบบ DISC จะยืนยันการรับเฟรมโดยส่งเฟรมคำตอบแบบ UA ไปในทันทีที่ได้รับเฟรม DISC DXE ที่ส่งเฟรม DISC ไปจะเปลี่ยนสถานะไปเป็นหยุดส่งข้อมูลเมื่อมันได้รับเฟรมแบบ UA

เช่นเดียวกับคำสั่ง SABM ในขณะที่เฟรมคำสั่ง DISC ถูกส่งออกไปนั้น เฟรมข่าวสารที่ยังไม่ได้รับการตอบรับกลับมาค้างอยู่ เฟรมข่าวสารก็ยังคงค้างอยู่อย่างนั้น

การตอบสนองต่อคำสั่งแบบไม่มีหมายเลข (Unnumbered acknowledge response: UA) เฟรมตอบสนอง UA นี้เป็นเฟรมตอบสนองการรับรู้จากการรับเฟรม SABM หรือเฟรมคำสั่ง DISC โดยฝ่ายที่รับเฟรมจะไม่ทำคำสั่งที่ส่งมาจนกว่าจะส่งเฟรมตอบรับ UA ก่อน และไม่อนุญาตให้บรรจุฟิลด์ข่าวสารในเฟรม UA นี้

การตอบสนองโมดยกเลิก (Disconnected mode response: DM) เฟรม DM จะถูกส่งออกไปเมื่อ DXE ได้รับเฟรมอื่นๆ นอกเหนือจากเฟรม SABA เฟรม UI เมื่ออยู่ในโมดแบบยกเลิกเฟรม DM นี้จะถูกส่งออกไปเป็นคำสั่งร้องขอการ Set Mode, หรือเพื่อบอกว่ามันไม่สามารถที่จะเชื่อมต่อได้ในเวลานี้ การตอบสนองแบบ DM นี้จะไม่อนุญาตให้ส่งฟิลด์ข่าวสารมาในเฟรมด้วย

เมื่อใดก็ตามที่สถานีใดๆ ได้รับเฟรมแบบ SABM และตัวมันยังไม่สามารถที่จะทำการเชื่อมต่อได้ มันจะส่งเฟรม DM ออกไป เพื่อเป็นการบอกว่ายังไม่อนุญาตให้สถานีต้นทางเชื่อมต่อได้

เมื่อ DXE อยู่ในโหลดยกเลิก DXE จะตอบสนองการรับเฟรมทุกๆ แบบด้วย เฟรมแบบ DM ซึ่งบิต P/F จะเซ็ตเป็นค่า '1' ยกเว้นการรับเฟรม SABM หรือเฟรม UI

เฟรมข่าวสารแบบไม่มีหมายเลข (Unnumbered information frame: UI) เฟรมข่าวสารแบบไม่มีหมายเลข มีการบรรจุฟิลด์ PID และฟิลด์ข่าวสารและส่งผ่านข้อมูลเหล่านี้ไปตามเส้นทางการสื่อสารภายนอกการควบคุมแบบปกติ ฟิลด์ข่าวสารนี้จะส่งผ่านไป-กลับตามเส้นทางการเชื่อมโยงการควบคุมการส่งผ่านข่าวสาร เฟรมชนิดนี้ไม่ต้องการการตอบรับ เพราะฉะนั้นถ้ามีเฟรมใดเฟรมหนึ่งขาดหายไป ก็ไม่สามารถที่จะตรวจสอบได้ การรับเฟรม UI ที่มีบิต P เป็น '1' จะได้รับผลตอบสนองเป็นเฟรม DM เมื่ออยู่ในสถานะยกเลิกหรือเฟรม RR ถ้าอยู่ในสถานะการส่งผ่านข่าวสาร

การรายงานความผิดพลาดจากเส้นทางการสื่อสาร และการแก้ไขข้อมูล

มีความผิดพลาดหลายลักษณะในชั้นเชื่อมโยงข้อมูลที่สามารถทำข้อมูลให้ถูกต้องเหมือนเดิมได้โดยไม่ต้องยกเลิกการเชื่อมต่อในระบบสื่อสาร สถานะการผิดพลาดที่เกิดขึ้นอาจเกิดจากความบกพร่องของการทำงานภายในสถานีของ DXE หรือการผิดพลาดของระบบสื่อสาร

เงื่อนไขการไม่ว่างของ DXE (DXE busy condition)

เมื่อ DXE ไม่สามารถรับเฟรมข่าวสารได้ชั่วคราว เช่น บัฟเฟอร์ข้อมูลเต็ม DXE จะส่งเฟรม RNR เพื่อเป็นการบอก DXE อีกตัวว่าไม่สามารถที่จะรับเฟรม I ได้ในขณะนั้น การยกเลิกสภาวะเงื่อนไขแบบนี้ทำได้โดยส่งเฟรม UA, RR, REJ หรือเฟรมคำสั่ง SABM

ความผิดพลาดของหมายเลขลำดับของการส่ง (Send sequence number error)

ถ้าหมายเลขของการส่ง N(S) ไม่เท่ากับค่า V(R) ของสถานีรับ แสดงว่าหมายเลขของการส่งผิดพลาด สถานีรับก็จะตัดข้อมูลเฟรม I ทิ้งไป สถานีรับข่าวสารจะไม่ตอบรับจนกว่า N(S) จะเท่ากับ V(R)

ฟิลด์ควบคุมของเฟรมข่าวสารที่ผิดพลาดก็ยังคงมีการตรวจสอบบิต P/F อยู่ จนกระทั่งมีการส่งเฟรม I ใหม่ ที่ถูกต้องก็จะมีการปรับเปลี่ยนค่าบิต P และ N(R)

การเรียกข้อมูลซ้ำ (Reject Recovery)

เฟรม REJ ใช้ในการขอให้ส่งเฟรม I มาใหม่ เนื่องจากได้รับเฟรมที่มีค่า N(S) ผิดพลาดโดยยอมให้ส่งเฟรม REJ ได้เพียงครั้งเดียวก่อนที่ได้รับเฟรมในการตอบรับเฟรม REJ ที่ส่งไปในครั้งแรก เมื่อฝ่ายผู้

ที่ได้รับเฟรม REJ ได้ส่งเฟรมข่าวสารตามที่ผู้ส่ง REJ ร้องขอไปเรียบร้อยแล้ว สถานะของการเรียกข้อมูลซ้ำก็จะถูกยกเลิก DXE ที่ได้รับเฟรม REJ จะส่งเฟรมข่าวสารที่ค้างการตอบรับออกไปทั้งหมด โดยเริ่มจากเฟรมที่มีหมายเลขตามที่ระบุด้วยค่า N(R) ของเฟรม REJ เมื่อส่งเฟรมเสร็จแล้วก็จะยกเลิกสถานะการเรียกข้อมูลซ้ำ

การแก้ไขข้อมูลที่ผิดพลาดตามระยะเวลาที่กำหนด (Time-out Error Recovery)

การแก้ไขข้อมูลตามระยะเวลาของตัวตั้งเวลา T1 (T1 Timer Recovery) ถ้า DXE ปลายทางไม่ได้รับเฟรมข้อมูลเนื่องจากความผิดพลาดระหว่างการส่งหรือรับได้แต่ทิ้งข่าวสารนั้นไป ทำให้ไม่สามารถตรวจสอบได้ว่าเกิดความผิดพลาดของหมายเลขลำดับในการส่ง DXE ปลายทางนั้นจะยังไม่ส่งเฟรม REJ ไปยัง DXE ที่ต้นทาง DXE ต้นทางที่ส่งข่าวสารจะตั้งเวลาของตัวตั้งเวลา T1 เพื่อรอการตอบรับ ถ้าครบเวลาแล้วยังไม่มีการตอบรับ จะส่งเฟรมข่าวสารไปใหม่ การยกเลิกสภาวะนี้ทำได้ เมื่อได้รับการตอบรับ (Acknowledgement) จากสถานีคู่สื่อสารหรือเมื่อรีเซตการเชื่อมต่อของระบบ

การแก้ไขข้อมูลตามระยะเวลาของตัวตั้งเวลา T3 (Timer T3 Recovery) ตัวตั้งเวลา T3 ใช้งานเพื่อเป็นการประกันว่าการเชื่อมโยงยังคงอยู่ในระหว่างที่มีการส่งผ่านข้อมูลน้อยๆ เมื่อใดก็ตามที่ T1 ยังไม่ได้ใช้งาน (ไม่มีเฟรม I ที่ค้างการตอบรับ) โดยจะตั้งเวลาของตัวตั้งเวลา T3 เป็นระยะเวลาช่วงหนึ่งที่ใช้ในการหยั่ง DXE อื่นๆ ของเส้นทางสื่อสาร เมื่อถึงระยะเวลาตามที่ตั้งไว้ ก็จะทำการส่งเฟรม RR หรือ RNR โดยเซตบิต P จากนั้นกระบวนการรอการตอบรับก็เริ่มดำเนินการ

เฟรมที่มี FCS ผิดพลาด

เมื่อรับเฟรมที่มี FCS ผิดพลาด เฟรมนี้จะถูกทิ้งไป

เงื่อนไขของการยกเลิกเฟรม

เงื่อนไขการยกเลิกเฟรม ที่กล่าวผ่านมาเมื่อเกิดการยกเลิกเฟรม จะไม่มีการรับเฟรมข่าวสารที่ส่งตามมาภายหลัง (ยกเว้นการตรวจสอบบิต P/F) จนกว่าความผิดพลาดนั้นได้รับการแก้ไขเรียบร้อยแล้ว ความผิดพลาดต่างๆ จะถูกรายงานไปยัง DXE ของคู่สื่อสารด้วยเฟรม FRMR

2.7 การวิจัยและพัฒนาตัวควบคุมเทอร์มินัลโนดที่ผ่านมา

ระบบเครือข่ายวิทยุกลุ่มข้อมูลได้ถูกคิดค้นและพัฒนาขึ้นมาตามลำดับเป็นระยะเวลาไม่น้อยกว่า 30 ปี นับตั้งแต่ ARPANET ได้คิดค้นเทคโนโลยีกลุ่มข้อมูล (Data packet) ขึ้นมาในปี ค.ศ. 1969 ในปีถัดมา มหาวิทยาลัยฮาวายได้ริเริ่มเครือข่ายวิทยุกลุ่มข้อมูล ALOHANET ซึ่งถือเป็นเครือข่ายวิทยุกลุ่มข้อมูลเครือข่ายแรก และเริ่มมีการใช้งานเครือข่ายวิทยุกลุ่มข้อมูลเกิดขึ้นในประเทศต่างๆ มากขึ้นตามลำดับ สำหรับประเทศไทยได้เริ่มทดลองใช้งานเครือข่ายวิทยุกลุ่มข้อมูลเป็นครั้งแรกในปี ค.ศ. 1995 (พ.ศ. 2538)

ปี ค.ศ.	เหตุการณ์
1969	คิดค้นเทคโนโลยีกลุ่มข้อมูล (Data packet) โดย ARPANET
1970	เครือข่ายวิทยุกลุ่มข้อมูล ALOHANET ที่ มหาวิทยาลัยฮาวาย
1978	เครือข่ายวิทยุกลุ่มข้อมูลสมัครเล่น Montreal Packet Net (MP-Net) ที่เมืองมอนทรีออล ประเทศแคนาดา
1980	เริ่มทดลองเครือข่ายวิทยุกลุ่มข้อมูลในสหรัฐอเมริกา
1995	เริ่มทดลองเครือข่ายวิทยุกลุ่มข้อมูลในประเทศไทย
1998	จัดตั้งระบบเครื่อข่ายวิทยุกลุ่มข้อมูล Packet Radio Network of Thailand (PRNT)

ตารางที่ 2.1 ประวัติเครือข่ายวิทยุกลุ่มข้อมูล [14]

ปี ค.ศ.	เหตุการณ์
1980	เครื่องต้นแบบ TNC โดย the Vancouver Amateur Digital Communication Group (VADCG) ใช้ CPU 8085
1983	เครื่อง TNC 1 โดย Tucson Amateur Packet Radio Corporation (TAPR) ใช้ CPU 6809
1983	เครื่อง TNC รุ่น PK-1 โดย GLB Electronics ใช้ CPU Z80
1985	เครื่อง TNC 2 โดย TAPR ใช้ CPU Z80
1985	ซอฟต์แวร์ TNC ที่ใช้กับโมเด็มธรรมดา โดย Baycom

ตารางที่ 2.2 การพัฒนาตัวควบคุมเทอร์มินัลโนด (TNC) ในต่างประเทศ [14]

ในส่วนของตัวควบคุมเทอร์มินัลโนด ซึ่งถือเป็นอุปกรณ์สำคัญที่ใช้เชื่อมอุปกรณ์เทอร์มินัล (คอมพิวเตอร์) เข้ากับเครือข่ายวิทยุกลุ่มข้อมูลนั้น จากตารางที่ 2.2 จะเห็นได้ว่าในอดีตมักจะพัฒนาตัวควบคุมเทอร์มินัลโนดโดยใช้ไมโครคอนโทรลเลอร์ หรือพัฒนาโดยใช้ซอฟต์แวร์ ในปัจจุบันผู้ผลิตบางรายเริ่มมีการนำ FPGA (Field programmable gate array) มาใช้พัฒนาตัวควบคุมเทอร์มินัลโนด เช่น PacComm และ IV3NWV [5] เป็นต้น

สำหรับประเทศไทยได้มีผู้สนใจระบบเครือข่ายวิทยุกลุ่มข้อมูลสำหรับนักวิทยุสมัครเล่นอยู่ไม่มากนัก ตัวอย่างเช่น การพัฒนาอุปกรณ์โมเด็มวิทยุโดยใช้ซอฟต์แวร์ทำงานเป็นตัวควบคุมเทอร์มินัลโนด (Soft TNC) [15,16] หรือการออกแบบตัวควบคุมเทอร์มินัลโนดในเครือข่ายวิทยุกลุ่มข้อมูล ออกแบบ

โดยใช้ไมโครคอนโทรลเลอร์และวงจรรวมมาตรฐาน [17] ซึ่งมีอัตราเร็วในการส่งข้อมูลค่อนข้างต่ำ และใช้อุปกรณ์ค่อนข้างมาก

ปัจจุบันได้มีการพัฒนาวงจรรวมดิจิทัลแบบโปรแกรมได้ที่มีประสิทธิภาพสูง จึงมีแนวคิดที่จะนำวงจรรวมนี้มาใช้ในการออกแบบตัวควบคุมเทอร์มินัลโนด แทนการใช้อุปกรณ์ไมโครคอนโทรลเลอร์และวงจรรวมมาตรฐาน โดยเลือกใช้ FPGA เป็นอุปกรณ์หลักในการออกแบบตัวควบคุมเทอร์มินัลโนด เพื่อรวมฟังก์ชันการทำงานต่างๆ ไว้ในวงจรรวมเพียงตัวเดียว ช่วยลดจำนวนอุปกรณ์ และเพิ่มประสิทธิภาพการทำงานของระบบให้ดีขึ้น

บทที่ 3

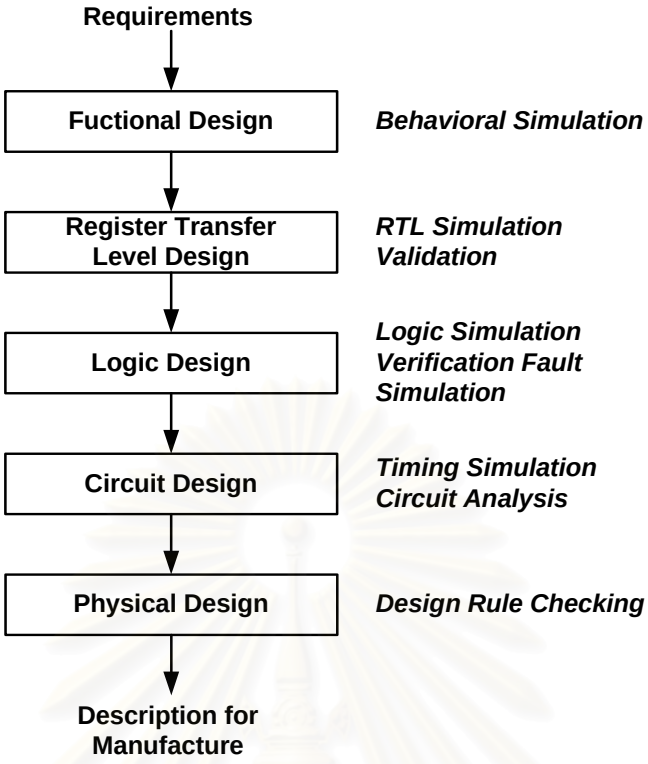
ภาษา VHDL และอุปกรณ์ FPGA

3.1 บทนำ

ปัจจุบันในการออกแบบระบบดิจิทัลที่เรารู้จักกันจะเป็นการออกแบบโดยใช้การวาดวงจร (Capture schematic) โดยใช้โปรแกรมช่วยในการวาด (Schematic entry tools) ซึ่งผู้ออกแบบจะต้องมีทักษะสูงในการออกแบบ และต้องใช้เวลามากในการออกแบบระบบ จำลองการทำงาน (Simulation) และตลอดจนถึงการแก้ไขข้อผิดพลาดของระบบให้ถูกต้อง (Debugging) ในการออกแบบจะต้องอ้างอิงเทคโนโลยีที่ใช้ออกแบบระบบดิจิทัล ถ้าต้องการเปลี่ยนเทคโนโลยีของระบบที่ออกแบบค่อนข้างทำได้ยากและใช้เวลามาก และเมื่อต้องการออกแบบระบบดิจิทัลที่มีความซับซ้อนมาก ยิ่งทำได้ยากหรืออาจทำไม่ได้ แต่ในปัจจุบันการออกแบบระบบดิจิทัลมีกระบวนการออกแบบรูปแบบใหม่ที่มีประสิทธิภาพสูง รวดเร็ว และไม่ยึดติดกับเทคโนโลยีที่ใช้ออกแบบ อาศัยกระบวนการออกแบบในลักษณะจากบนลงล่าง หรือ Top-down design โดยใช้ภาษาบรรยายการทำงานของฮาร์ดแวร์ (Hardware Description Language: HDL) ในการออกแบบ จำลองการทำงานของวงจรที่ได้ และสังเคราะห์วงจร (Synthesis) ในรูปแบบของเทคโนโลยีที่ต้องการ โดยสามารถทดสอบวงจรที่ออกแบบได้บนอุปกรณ์จำพวกชิป FPGA (Field Programmable Gate Arrays) หรือ ASIC (Application Specific Integrated Circuits) ซึ่งช่วยให้ออกแบบได้ง่ายและมีความสะดวกรวดเร็วมากยิ่งขึ้น

3.2 กระบวนการออกแบบในลักษณะ Top-down design

กระบวนการออกแบบในลักษณะจากบนลงล่าง (Top-down design) เป็นกระบวนการที่ได้รับความนิยมในการออกแบบระบบดิจิทัลในปัจจุบัน มีขั้นตอนต่างๆ ดังรูปที่ 3.1 คือเริ่มต้นจากการวางข้อกำหนดคุณลักษณะของระบบที่ต้องการ ไปจนถึงการทำให้เกิดผลทางกายภาพ เช่นเป็นชิปที่ใช้งานได้จริง



รูปที่ 3.1 ขั้นตอนการออกแบบในลักษณะจากบนลงล่าง

เพื่อความเข้าใจกระบวนการออกแบบในลักษณะนี้ จะขอยกตัวอย่างการออกแบบวงจรรวมเฉพาะงาน (ASIC) สำหรับประมวลผลภาพดิจิทัล เพื่อใช้อธิบายกระบวนการดังกล่าว

ในขั้นตอนแรกคือ การวางข้อกำหนดคุณลักษณะของระบบ (Requirements) ทั้งด้านสมรรถนะและด้านกายภาพ เช่นจำนวนภาพที่ระบบสามารถประมวลได้ในหนึ่งวินาที, การดำเนินการต่อรูปภาพ, ข้อกำหนดเรื่องส่วนต่อประสาน, ราคาต้นทุน, ขนาด, และการสูญเสียกำลังเป็นความร้อน จากข้อกำหนดเหล่านี้สามารถออกแบบเชิงหน้าที่ระดับสูง (High-level functional design) ในเบื้องต้นได้ ซึ่งต้องอาศัยการจำลองการทำงาน เพื่อดูว่าระบบที่ออกแบบไว้ทำงานสอดคล้องกับข้อกำหนดต่างๆ หรือไม่ และแก้ไขให้สอดคล้องในที่สุด

เมื่อเสร็จสิ้นการออกแบบเชิงหน้าที่หรือฟังก์ชันแล้ว ขั้นต่อไปคือการใส่รายละเอียดลงไปในระดับหน่วยบันทึกหรือรีจิสเตอร์ (Register) หน่วยความจำ (Memory) หน่วยคำนวณและตรรกะ (ALU) และตัวควบคุมสถานะ (State machine) โดยเรียกการออกแบบในระดับนี้ว่า Register Transfer Level หรือ RTL หลังการออกแบบในระดับ RTL ก็จะเป็นการออกแบบในระดับตรรกะ (Logic design) เป็นการเพิ่มเติมรายละเอียดให้กับหน่วยต่างๆ ที่เป็นส่วนประกอบของงานออกแบบในระดับ RTL เช่นการกำหนดในรายละเอียดว่ารีจิสเตอร์ตัวหนึ่งประกอบด้วยฟลิปฟลอปและลอจิกเกตแบบใดและจำนวน

เท่าใด เป็นต้น การออกแบบทั้งในระดับ RTL และระดับลอจิก ต้องมีการจำลองการทำงานควบคู่ไปด้วยเช่นกัน เหมือนการออกแบบในระดับฟังก์ชัน

ในบางระบบนอกจากการจำลองการทำงานเชิงตรรกะ (Logic simulation) แล้ว อาจมีการจำลองการทำงานเพื่อหาข้อบกพร่อง (Fault simulation) ของวงจรหรือระบบด้วย ซึ่งสามารถจำลองผลกระทบที่เกิดจากข้อบกพร่องในกระบวนการผลิต รวมถึงความผิดพลาดที่เกิดจากการเหนี่ยวนำของสิ่งแวดล้อม ตัวอย่างเช่น กรณีที่มีผู้นำชิปประมวลผลภาพไปใช้บนดาวเทียม รังสีต่างๆ ในอวกาศอาจเหนี่ยวนำให้วงจรบนชิปเปลี่ยนสถานะไปโดยไม่ต้องการ ทำให้เกิดความผิดพลาดในการทำงานขึ้นได้ ถ้าอัตราความผิดพลาดบิต (bit error rate) สูงเกินไป อาจปรับแก้การออกแบบการทำงานให้ทนทานต่อความผิดพลาดด้วยเทคนิคต่างๆ ได้

ขั้นตอนสุดท้าย คือการออกแบบในระดับวงจร (Circuit design) และการออกแบบในระดับกายภาพ (Physical design) การออกแบบในระดับวงจร คือการแปลงลอจิกเกตและฟลิปฟลอปต่างๆ ให้กลายเป็นทรานซิสเตอร์ ในขั้นตอนนี้จะมีการวิเคราะห์วงจรและการจำลองเชิงเวลา (Timing simulation) ควบคู่กันไปด้วย เพื่อตรวจสอบว่าวงจรและระบบที่ออกแบบไว้ยังทำงานสอดคล้องกับข้อกำหนดต่างๆ ที่ตั้งไว้ในตอนแรกหรือไม่

ส่วนการออกแบบในระดับกายภาพ ซึ่งเป็นการออกแบบระดับล่างสุด คือการออกแบบผังภูมิ (Layout) ของวงจรรวมที่ได้จากการออกแบบในระดับวงจร โดยการแปลงทรานซิสเตอร์ต่างๆ ให้เป็นรูปเหลี่ยมและลายเส้นที่มีสีสันแทนวัตถุชั้นต่างๆ ตามกระบวนการผลิตชิปวงจรรวม ภายหลังการออกแบบจะมีการตรวจสอบความถูกต้องตามกฎการออกแบบ การหาค่าพารามิเตอร์ต่างๆ และการจำลองในระดับวงจร เช่น Timing simulation อีกครั้ง โดยนำค่าพารามิเตอร์ที่ได้มาประกอบการคำนวณ ในขั้นนี้สามารถประเมินคุณสมบัติทางกายภาพที่ถูกต้องได้ เช่น พื้นที่ชิป และกำลังที่สูญเสียเป็นความร้อน เป็นต้น

การแบ่งระดับชั้นของการออกแบบนี้จะเรียกแต่ละระดับว่า ระดับของการกำหนดสาระสำคัญ (Level of abstraction) การกำหนดสาระสำคัญในระดับสูง จะประกอบด้วยส่วนประกอบที่ซับซ้อน และมีหน้าที่การทำงานต่างกันไป เช่น วงจรบวก (Adder) และหน่วยความจำ (Memory) โดยจะมีส่วนประกอบไม่มาก ตรงข้ามกับการกำหนดสาระสำคัญในระดับล่าง ที่จะประกอบด้วยส่วนประกอบจำนวนมากที่ไม่ซับซ้อน เช่น ลอจิกเกต และทรานซิสเตอร์ ซึ่งโดยทั่วไปกระบวนการออกแบบจะเป็นไปในลักษณะจากบนลงล่าง ดังที่ได้ยกตัวอย่างไปแล้ว

หากมีการค้นพบความผิดพลาดที่ระดับล่าง การแก้ไขในระดับนี้จะเป็นไปได้ยากและเสียความใช้จ่ายมากกว่ากรณีที่ตรวจพบข้อผิดพลาดแต่เนิ่นๆ ในระดับสูง เพราะในขั้นตอนนี้มีรายละเอียดที่ต้องแก้ไขมาก ทำให้เสียเวลาและโอกาสในการนำผลิตภัณฑ์ออกสู่ตลาด นำมาซึ่งการสูญเสียรายได้ที่คาดหวัง นี่คือแรงจูงใจที่ผลักดันให้เกิดการพัฒนาภาษาชั้นสูงต่างๆ เพื่อใช้บรรยายการทำงานในระดับ

ฮาร์ดแวร์ ช่วยอำนวยความสะดวกให้นักออกแบบสามารถจำลองการทำงานที่ระดับต่างๆ ได้หลายระดับ ทำให้ความผิดพลาดต่างๆ ถูกค้นพบและแก้ไขได้แต่เนิ่นๆ ภาษา VHDL ก็เป็นภาษาหนึ่งที่มีความสามารถเช่นนี้

3.3 VHDL คืออะไร

VHDL ย่อมาจาก VHSIC Hardware Description Language เป็นภาษาบรรยายการทำงานของฮาร์ดแวร์ประเภทหนึ่ง โดยภาษานี้เกิดขึ้นจากโครงการที่มีชื่อ VHSIC (Very High Speed Integrated Circuits) ที่พัฒนาขึ้นโดยกระทรวงกลาโหมของสหรัฐอเมริกา (Department of Defense: DoD) ในช่วงปี ค.ศ. 1980 โดยเป้าหมายของโครงการนี้คือ เพื่อพัฒนาขีดความสามารถในการออกแบบวงจรรวมให้สูงขึ้น และสามารถทำได้ง่ายมากยิ่งขึ้น โดยมีเป้าหมายหลัก 2 ประการคือ

- ต้องการภาษาที่สามารถรองรับการออกแบบวงจรที่มีความซับซ้อนได้
- ต้องการภาษาที่เป็นมาตรฐาน หรือเป็นภาษากลางที่ทำให้สามารถเผยแพร่ผลงานการออกแบบกันภายใน กลุ่มนักออกแบบด้วยกันได้

จนกระทั่งในปี ค.ศ 1986 ภาษา VHDL ได้รับการปรับปรุงแก้ไขภาษา เพื่อให้สามารถกำหนดเป็นมาตรฐานของ IEEE (Institute of Electrical and Electronics Engineers) โดยสามารถประกาศเป็นมาตรฐานได้ในเดือนธันวาคมปี ค.ศ.1987 อยู่ในหมวด IEEE 1076–1987 หรือเรียกสั้นๆ ว่า VHDL'87 หลังจากนั้นก็ได้มีการพัฒนาปรับปรุงอย่างต่อเนื่องโดยได้มีประกาศแก้ไขอีกครั้งในปี ค.ศ 1993 ซึ่งเรียกว่า IEEE 1076–1993 หรือเรียกสั้นๆ ว่า VHDL'93 โดยได้มีการเพิ่มเติมไวยากรณ์เสริมเพื่อให้ผู้ใช้สามารถใช้งานได้สะดวกมากยิ่งขึ้น สำหรับขีดความสามารถในการออกแบบโดยใช้ภาษานั้นสามารถออกแบบได้เฉพาะวงจรที่มีลักษณะเป็นระบบดิจิทัลเท่านั้น ในปัจจุบันกำลังมีการวิจัยและพัฒนาภาษา VHDL ให้มีความสามารถออกแบบวงจรแอนาล็อกเพิ่มเติมจากพื้นฐานเดิม โดยมีชื่อเรียกใหม่ว่า VHDL-AMS (Analog Mixed Signal) ซึ่งในที่นี้จะไม่กล่าวถึง

3.4 ประโยชน์ของภาษา VHDL

ประโยชน์ของภาษา VHDL มีดังนี้

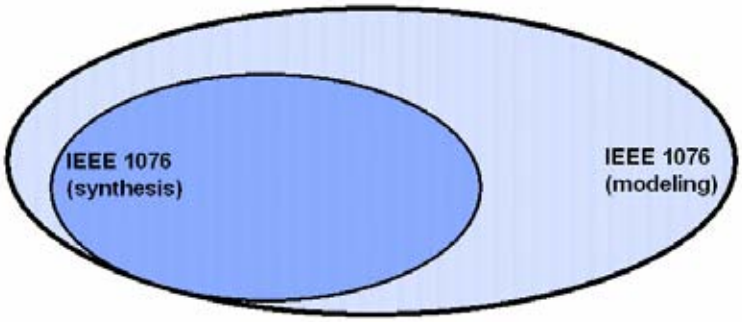
- เป็นภาษามาตรฐานสากล โดยรับรองจากสถาบัน IEEE ทำให้มีโปรแกรมและเครื่องมือต่างๆ และบริษัทที่สนับสนุนการทำงานมากมาย นอกจากนี้วงจรที่ออกแบบโดยภาษา VHDL จะใช้งานได้นาน เนื่องจากมีความเข้ากันได้ของภาษาที่ใช้ออกแบบวงจรเวอร์ชันใหม่ๆ

- ได้รับการสนับสนุนจากรัฐบาล เนื่องจากภาษา VHDL ได้รับการพัฒนาโดยกระทรวงกลาโหมของสหรัฐอเมริกา ดังนั้นการออกแบบวงจรต่างๆ ย่อมได้รับการสนับสนุนจากรัฐบาลสหรัฐอเมริกา
- เป็นภาษาที่ใช้งานจริงในอุตสาหกรรม เนื่องจากภาษา VHDL เป็นภาษาที่เป็นมาตรฐานจากสถาบัน IEEE จึงมีอุตสาหกรรมจำนวนมากที่นำภาษานี้ไปใช้ออกแบบ
- เป็นภาษาที่สามารถใช้ได้หลายระบบ การออกแบบโดยใช้ภาษา VHDL สามารถนำไปจำลองการทำงานหรือสังเคราะห์ด้วยซอฟต์แวร์ตัวใดก็ได้ที่รองรับภาษา VHDL จึงทำให้การออกแบบด้วยภาษา VHDL เป็นการออกแบบที่ไม่ยึดติดกับซอฟต์แวร์ที่ใช้ในการออกแบบ
- เป็นภาษาที่สามารถใช้จำลองรูปแบบการทำงานของวงจร ผู้ออกแบบวงจรสามารถออกแบบวงจรโดยใช้ภาษา VHDL ได้หลายระดับ ตั้งแต่ระดับมาโครบล็อก (Macro block) จนถึงระดับทรานซิสเตอร์ และสามารถออกแบบวงจรที่มีความซับซ้อนสูง และมีขนาดใหญ่มากได้
- เป็นภาษาที่สามารถนำกลับมาใช้ใหม่ได้ วงจรที่ออกแบบโดยภาษา VHDL สามารถนำกลับมาใช้ใหม่ได้ง่าย เนื่องจากสามารถเปลี่ยนแปลงแก้ไขวงจรได้ง่าย คล้ายคลึงกับโปรแกรมของซอฟต์แวร์
- เป็นภาษาที่สามารถนำไปใช้เป็นเอกสารประกอบได้ เป็นภาษาในรูปแบบบรรยายพฤติ-กรรม ทำให้สามารถอธิบายการทำงานของวงจรภายในรหัสต้นทางภาษาที่ออกแบบได้ทันที

3.5 การใช้งานภาษา VHDL

การใช้งานภาษา VHDL อาจจำแนกออกเป็น 5 ประเภทดังนี้

- Document Language: ใช้บรรยายรายละเอียดการทำงานของวงจรที่ออกแบบ
- Design Language: ใช้สำหรับออกแบบวงจรที่มีความซับซ้อนสูง
- Verification Language: ใช้ตรวจสอบความถูกต้องของวงจรที่ออกแบบ
- Test Language: ใช้สำหรับทดสอบทดสอบการทำงานของวงจรที่ออกแบบ
- Synthesis Language: ใช้สำหรับสังเคราะห์วงจรจริง แต่รูปแบบภาษาดังกล่าวนี้ไม่ได้ครอบคลุมทุกรูปแบบทั้งหมดของการเขียนในภาษา VHDL มีเพียงบางรูปแบบเท่านั้นที่สามารถนำไปสังเคราะห์เป็นวงจรได้ ซึ่งเรียกว่ารูปแบบการเขียนภาษา VHDL สำหรับการสังเคราะห์ (VHDL for synthesis) ดังรูปที่ 1.1 จะเห็นได้ว่ารูปแบบที่สามารถเขียนแล้วใช้ในการสังเคราะห์ได้นั้นเป็นเพียงส่วนหนึ่งของรูปแบบการเขียนบรรยายพฤติกรรมทั้งหมด



รูปที่ 3.2 ขอบเขตของรูปแบบการเขียนภาษา VHDL สำหรับการสังเคราะห์วงจร

3.6 พื้นฐานของภาษา VHDL

ภาษา VHDL เป็นภาษาโครงสร้างที่ใช้ออกแบบวงจรดิจิทัลได้ตั้งแต่ระบบดิจิทัลที่ซับซ้อน ไปจนถึงระดับลอจิกเกต เป็นภาษาที่ง่ายต่อการเรียนรู้และเข้าใจ และที่สำคัญคือ มีความเป็น Concurrent ซึ่งเป็นหัวใจสำคัญของการเขียนออกแบบอุปกรณ์อิเล็กทรอนิกส์

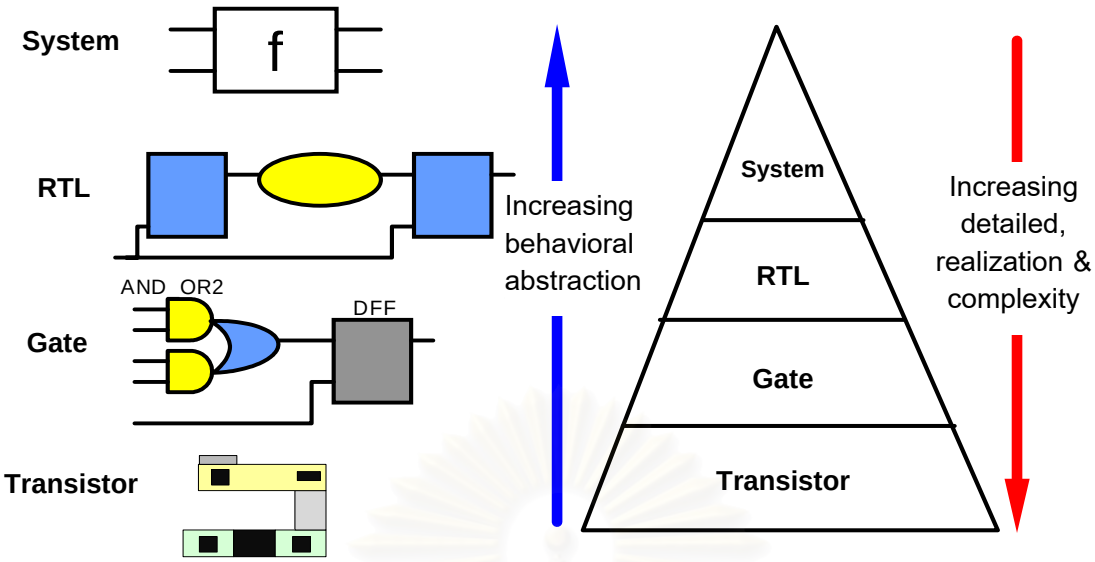
ภาษา VHDL สามารถบรรยายวงจรดิจิทัลได้หลายระดับ ตั้งแต่ระดับพฤติกรรมการทำงานของวงจร (Behavioural level) จนถึงระดับลอจิกเกต (Gate level) ซึ่งในแต่ละระดับก็จะมีรายละเอียดที่แตกต่างกันไป ตัวอย่างเช่น ถ้าต้องการออกแบบวงจรคูณ 2 อินพุต ด้วยใช้ภาษา VHDL สามารถออกแบบได้หลากหลายระดับ เช่น

บรรยายพฤติกรรมหรือฟังก์ชันโดยใช้ตัวดำเนินการคูณ ในภาษา VHDL คล้ายกับเขียนโปรแกรมซอฟต์แวร์ เช่น a <= b * c;

หรือ ออกแบบวงจรคูณในระดับของลอจิกเกต

หรือ ออกแบบวงจรคูณในระดับล่างสุด นั่นคือระดับทรานซิสเตอร์ (Layout level)

จากตัวอย่างที่กล่าวมานั้นจะเห็นได้ว่า วงจรที่มีฟังก์ชันการทำงานเหมือนกัน สามารถออกแบบได้ในหลายระดับที่แตกต่างกัน ตั้งแต่ระดับสูงสุดไปจนระดับล่างสุด ขึ้นอยู่กับผู้ออกแบบจะออกแบบในระดับใด ตามความเหมาะสมและทักษะของผู้ออกแบบเป็นหลัก



รูปที่ 3.3 ระดับของการออกแบบวงจรดิจิทัลในภาษา VHDL

จากรูป 3.3 พิจารณาจากรูปมีรายละเอียดดังนี้

- System level เป็นระดับบนสุดของการเขียน VHDL โดยจะบรรยายพฤติกรรมการทำงานของระบบ หรือแบบอัลกอริทึมของระบบ โดยคำนึงถึงเรื่องข้อมูลของเวลาเป็นหลัก
- RTL (Register Transfer Level) เป็นการบรรยายลงลึกในแต่ส่วนของวงจรในระดับโครงสร้าง โดยในระดับนี้อาจเห็นรายละเอียดลึกลงไปในวงจร เช่น เป็นวงจรแบบอะซิงโครนัสหรือซิงโครนัส, ตัวควบคุมสถานะ, รีจิสเตอร์ต่างๆ ตลอดจนการไหลเข้าออกของข้อมูลในบล็อกแต่ละบล็อกภายในวงจร
- Gate level เป็นการบรรยายระดับลึกกว่าในระดับ RTL โดยจะมีรายละเอียดวงจรระดับลอจิกเกต และฟลิปฟลอปต่างๆ ที่เชื่อมต่อกันเป็นวงจร
- Transistor level เป็นระดับล่างสุดของการออกแบบวงจรหรือระบบดิจิทัล บางครั้งอาจเรียกระดับนี้ว่า Layout level ในระดับนี้ภาษา VHDL ไม่สามารถใช้เขียนได้ เนื่องจากในระดับนี้จะมีข้อมูลเกี่ยวกับค่าเก็บประจุและค่าความต้านทานของทรานซิสเตอร์เข้ามาเกี่ยวข้อง ซึ่งต้องอาศัยโปรแกรมช่วยในการสังเคราะห์วงจรในระดับทรานซิสเตอร์ (Layout synthesis) หรือโดยทั่วไปมักเรียกว่า โปรแกรมช่วยในการวางและเชื่อมต่อทรานซิสเตอร์ (Place and Route)

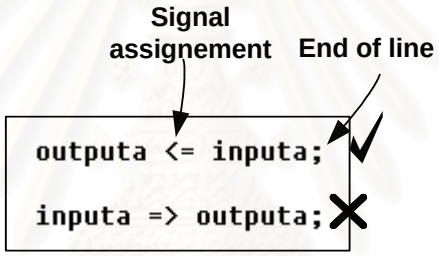
จะเห็นได้ว่าในแต่ละระดับจะแตกต่างกันในเรื่องของรายละเอียดภายในวงจรในมุมมองของระดับนั้นๆ ขึ้นอยู่ว่าผู้ออกแบบจะออกแบบโดยมองระบบดิจิทัลที่จะออกแบบในระดับใด ตามความเหมาะสม

3.7 การเขียนภาษา VHDL เบื้องต้น

ปกติภาษา VHDL เป็นภาษาที่มีลักษณะ Case insensitive หมายถึงไม่มีความแตกต่างกันในการเขียนอักษรพิมพ์เล็กหรือพิมพ์ใหญ่ กล่าวคือไม่ว่าจะเขียนด้วยอักษรพิมพ์เล็กหรือพิมพ์ใหญ่ก็ตามจะมีความหมายเหมือนกัน

```
EntiTY = ENTITY = entity = eNtitY
```

หลักการเขียนภาษา VHDL เบื้องต้นในการออกแบบวงจรดิจิทัลนั้น เริ่มจากการเขียนโค้ดในแต่ละบรรทัดจะต้องจบด้วยเครื่องหมายอัฒภาค (;) การกำหนดค่าให้วัตถุ (Object) เช่น การกำหนดค่าสัญญาณ (Signal) จะใช้เครื่องหมาย <= ส่วนการกำหนดค่าตัวแปร (Variable) จะใช้เครื่องหมาย := โดยทางขวามือของเครื่องหมายจะเป็นค่าอินพุต (Input) ส่วนทางซ้ายมือจะเป็นค่าเอาต์พุต (Output)



การประกาศสัญญาณหลายค่าให้ใช้เครื่องหมายจุลภาค (,) คั่นระหว่างสัญญาณแต่ละสัญญาณ

```
SIGNAL clk, reset, sel : BIT;
SIGNAL databus, address : BIT_VECTOR(7 DOWNTO 0);
```

ในบางครั้งอาจมีการกำหนดค่าเริ่มต้น (Initial value) ให้กับสัญญาณ เพื่อใช้ในการจำลองการทำงาน

```
SIGNAL Sample_signal : BIT := '1';
```

การเขียนคำอธิบาย (Comment) ในภาษา VHDL จะใช้เครื่องหมาย (--) เขียนไว้ที่หน้าบรรทัดหรือข้อความที่ไม่ต้องการให้คอมไพเลอร์ (Compiler) สนใจ หรือเป็นคำอธิบายที่เขียนขึ้นเพื่อทำให้อ่าน

โค้ดได้เข้าใจง่ายขึ้น และง่ายแก่การแก้ไขปรับปรุงในภายหลัง โดยการเขียนเครื่องหมายดังกล่าว จะใช้ได้บรรทัดต่อบรรทัดเท่านั้น ไม่สามารถทำเป็นกลุ่มของบรรทัดได้

```
-- Comment line
SIGNAL Clk : BIT;           -- 10 MHz clock signal
SIGNAL Reset : BIT;         -- Global reset signal
```

3.7.1 Concurrency

ก่อนที่จะเริ่มศึกษารูปแบบการเขียนภาษา VHDL ในการออกแบบวงจรฮาร์ดแวร์ในระบบดิจิทัล จะต้องมาทำความเข้าใจ คำว่า Concurrency เสียก่อน ภาษา VHDL นั้นเป็นภาษาที่มีคุณสมบัติ Concurrency หมายถึง ไม่มีลำดับความสำคัญหรือลำดับก่อนหลังของโค้ดในแต่ละบรรทัด คำสั่งทุกบรรทัดจะทำงานพร้อมกัน ซึ่งภาษาอื่นๆ จะไม่มีคุณสมบัตินี้ จึงไม่สามารถนำไปใช้ออกแบบพฤติกรรมการทำงานวงจรฮาร์ดแวร์จริงได้ เพื่อความเข้าใจยิ่งขึ้นขอให้พิจารณาจากตัวอย่างที่ 3.1 และ 3.2

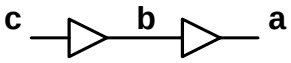
ตัวอย่าง 3.1

```
ARCHITECTURE example OF my_design IS
BEGIN
        a <= b; -- Line no. 1
        b <= c; -- Line no. 2

END example;
```

ตัวอย่าง 3.2

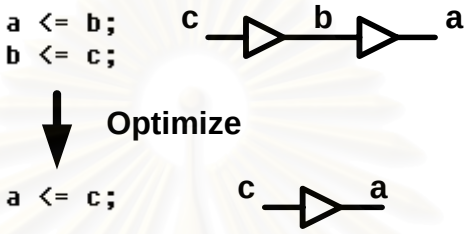
```
ARCHITECTURE example OF my_design IS
BEGIN
        b <= c; -- Line no. 2
        a <= b; -- Line no. 1

END example;
```

จากตัวอย่าง 3.1 และ 3.2 เนื่องจากภาษา VHDL ไม่คำนึงถึงลำดับความสำคัญหรือลำดับก่อนหลังในการเขียนโค้ดในแต่ละบรรทัด ไม่ว่าจะเขียนบรรทัดที่ 1 หรือ 2 ก่อนก็ตาม จะมีความหมายเหมือนกัน นั่นคือ ถ้านำโค้ดที่เขียนในตัวอย่าง 3.1 หรือ 3.2 ไปสังเคราะห์ จะได้วงจรดังรูปที่ 3.4 เหมือนกัน



รูปที่ 3.4 วงจรที่สังเคราะห์จากโค้ดในตัวอย่าง 3.1 และ 3.2

แต่โค้ดที่เขียนขึ้นสามารถลดรูปได้เหลือเพียง a <= c; เท่านั้น ดังรูปที่ 3.5

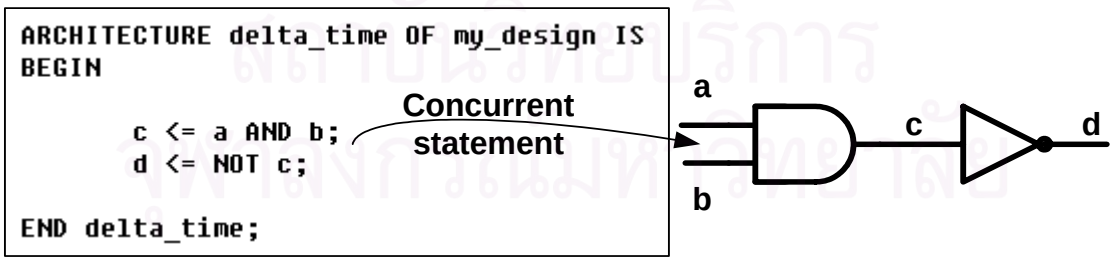


รูปที่ 3.5 การลดรูปวงจรจากโค้ดที่เขียนขึ้น

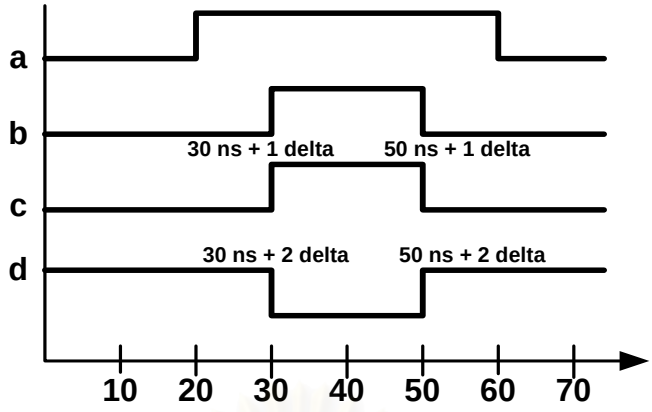
3.7.2 Delta time และ Propagation delay

การที่การจำลองการทำงานของโค้ดที่เขียนด้วยภาษา VHDL สามารถทำงานแบบ Concurrent ได้ โดยไม่มีการเพิ่มเวลาจำลองการทำงานนั้น กลไกสำคัญที่ช่วยให้การทำงานปราศจากข้อแม้ทางเวลานี้เรียกว่า Delta time (σ-time) พิจารณาจากตัวอย่างต่อไปนี้

ตัวอย่าง 3.3



จากตัวอย่างเมื่อนำไปสังเคราะห์เป็นผังวงจรจะได้ผลลัพธ์ดังรูปข้างต้น เมื่อนำไปจำลองการทำงานโดยป้อนค่าสัญญาณ a และ b ตามรูปที่ 3.6



รูปที่ 3.6 ผลการจำลองการทำงานที่แสดงถึง Delta time

ผลลัพธ์ที่ได้คือ สัญญาณ c ที่เกิดจากสัญญาณ a AND b จะมีค่าลอจิกเป็น '1' ที่ 30 นาโน-วินาที (บวกอีก 1 Delta time ในระบบพัฒนา VHDL) และค่าผลลัพธ์สัญญาณ d ที่เกิดจากค่าของ NOT c จะมีค่าลอจิกเป็น '0' ที่ 30 นาโนวินาที (บวกอีก 2 Delta time) แต่เวลาที่ผู้ออกแบบสังเกตได้จะเห็นว่าผลลัพธ์ของสัญญาณ c และ d เปลี่ยนแปลงพร้อมกัน ในบางกรณีที่เขียนรหัสภาษาไม่ดี จะทำให้ Delta time เกิดค่าสะสมมากขึ้น หรือมีค่าไม่รู้จบ ทำให้โปรแกรมจำลองการทำงานเกิดการแฮงก์ขึ้นได้ เราควรหลีกเลี่ยงการเขียนรหัสภาษาในลักษณะนี้

```
a <= NOT a;
```

ตัวอย่างเช่น สัญญาณ a จะเปลี่ยนแปลงค่าโดยมีการกลับค่าลอจิกหลังจาก Delta time ซึ่งในกรณีค่าสุดท้ายของสัญญาณ a จะมีค่าไม่รู้จบ การแก้ปัญหาในลักษณะนี้สามารถทำได้ไม่ยาก โดยการกำหนดค่าหน่วงเวลา (Propagation delay) ให้กับสัญญาณ ดังตัวอย่างต่อไป

```
a <= NOT a AFTER 10 ns;
```

3.7.3 Objects

วัตถุ (Object) ในภาษา VHDL จะใช้บ่งบอกรูปแบบขององค์ประกอบนั้นๆ ซึ่งเปรียบได้กับภาชนะที่มีไว้สำหรับบรรจุค่าต่างๆ มีอยู่ 3 ประเภทคือ

- **ค่าคงตัว** (Constant) เป็นวัตถุที่เมื่อกำหนดค่าเริ่มต้นให้แล้วจะคงค่านั้นไว้ ไม่สามารถดัดแปลงแก้ไขได้ สามารถประกาศใช้ได้ในส่วนที่เป็นส่วนประกาศต่างๆ ของแบบจำลอง

- **สัญญาณ** (Signal) เป็นวัตถุที่สามารถกำหนดค่าที่สัมพันธ์กับเวลานั้น หมายความว่า สัญญาณสามารถรับค่าได้เพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง โดยสัญญาณจะรับค่าได้จากตัวขับสัญญาณหรือไดรเวอร์ (Driver) ตัวขับสัญญาณนี้อาจจะเก็บค่าในอนาคตสำหรับสัญญาณไว้ด้วย สัญญาณสามารถประกาศใช้ได้ในส่วนที่เป็นเนื้อที่ของ Concurrent body เท่านั้น จึงสามารถนำสัญญาณไปใช้ได้ทุกส่วนในโครงสร้างของแบบจำลอง เรียกว่าเป็น Global object
- **ตัวแปร** (Variable) เป็นวัตถุที่สามารถกำหนดค่าใดๆ ให้ได้ และสามารถเปลี่ยนแปลงค่าได้ตลอดเวลาการจำลองการทำงาน แต่จะเก็บค่าเพียงค่าเดี่ยวเท่านั้นในขณะเวลาหนึ่ง เนื่องจากตัวแปรสามารถประกาศใช้ได้ในส่วนของ Sequential body เท่านั้น อันได้แก่ ส่วนประกาศของ Process, Function หรือ Procedure จึงสามารถนำตัวแปรไปใช้ได้เฉพาะในขอบเขตที่ถูกประกาศใช้เท่านั้น เรียกว่าเป็น Local object

Object declaration

การที่จะใช้งานวัตถุต่างๆ ตามที่กล่าวมาแล้วในการเขียนรูปแบบด้วยภาษา VHDL นั้น จะต้องมีการประกาศใช้ก่อนเสมอจึงจะสามารถนำไปใช้งานได้ การประกาศใช้วัตถุมีรูปแบบดังนี้

```
object_class identifier : TYPE := initial_value;
```

- ประเภท (object_class) ได้แก่ Constant, Signal และ Variable
- ชื่อ (identifier) เป็นไปตามกฎของภาษา VHDL
- ชนิด (TYPE) เป็นการกำหนดชนิดของวัตถุที่ประกาศ และสามารถกำหนดค่าเริ่มต้นของวัตถุ (initial_value) ได้ โดยเป็นส่วนเสริม คือไม่ต้องกำหนดค่าเริ่มต้นก็ได้

Predefined type

หมายถึง ชนิดของวัตถุที่กำหนดไว้ใน Package ชื่อ Standard และกำหนดโดย IEEE ว่าจะต้องมีในระบบที่ใช้พัฒนาด้วยภาษา VHDL ได้แก่

- Boolean คือกลุ่มของค่าการตัดสินใจ False และ True
- Bit คือกลุ่มของค่าลอจิก ‘0’ และ ‘1’

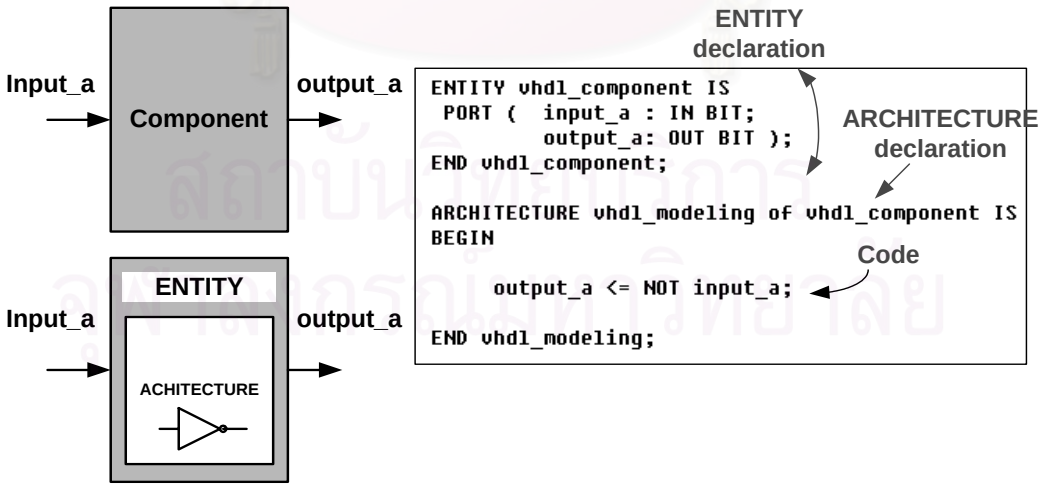
- **Integer** คือกลุ่มของค่าจำนวนเต็ม มีค่าตั้งแต่ –214748347 ถึง 214748347
- **Real** คือกลุ่มของค่าจำนวนจริง มีค่าตั้งแต่ –1.0E38 ถึง 1.05E38
- **Character** คือกลุ่มของค่าอักษร ‘A’-‘Z’, ‘a’-‘z’ อักษรหรือเครื่องหมายพิเศษ และอักษรควบคุม
- **Time** คือหน่วยเวลาที่มีค่าพื้นฐานเป็นวินาที
- **Severity level** คือกลุ่มของ Note, Warning, Error, Failure

3.7.4 VHDL Component

การใช้ภาษา VHDL ในการออกแบบวงจรดิจิทัลไม่ว่าจะเป็นวงจรพื้นฐานหรือวงจรที่ซับซ้อน ทุกวงจรที่ออกแบบจะต้องเขียนในรูปแบบของ VHDL Component ซึ่งเป็นพื้นฐานของการออกแบบ โดยหน่วยการออกแบบ (Design unit) ในภาษา VHDL จะแบ่งออกเป็นประเภทต่างๆ ดังนี้

- Entity
- Architecture
- Package
- Configuration

หน่วยการออกแบบพื้นฐานที่ต้องมีเสมอในการออกแบบวงจรก็คือ Entity และ Architecture โดยเรียกหน่วยการออกแบบทั้งสองนี้ว่า Component



รูปที่ 3.7 รูปแบบของ VHDL component

Entity

เป็นหน่วยการออกแบบที่ใช้สำหรับติดต่อระหว่างอุปกรณ์ภายนอกกับวงจรที่จะเขียนขึ้น รวมทั้งการส่งผ่านค่าพารามิเตอร์บางอย่างระหว่างวงจรกับอุปกรณ์ภายนอก โดยมีรูปแบบการเขียนดังนี้

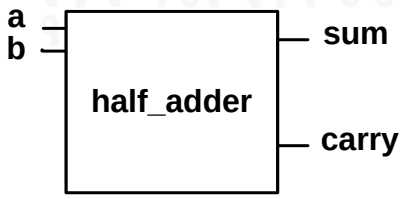
```
ENTITY entity_name IS
    GENERIC (generic_list);
    PORT (port_list);
END entity_name;
```

หมายเหตุ: ถ้าเป็นรูปแบบการเขียน VHDL'93 ในบรรทัด END จะมีคำว่า ENTITY ต่อท้ายคำสั่ง END ด้วย

ประเภทของพอร์ตที่สามารถประกาศใช้ใน Entity มี 4 ประเภทดังนี้

- In (Input) เป็นพอร์ตทิศทางเดียว ที่นำค่าสัญญาณจากอุปกรณ์ภายนอกเข้ามาใช้ภายในวงจร สามารถนำมาป้อนให้กับสัญญาณอื่นหรืออ่านค่าได้ แต่ไม่สามารถถูกเขียนจากภายในวงจรได้
- Out (Output) เป็นพอร์ตทิศทางเดียว ที่นำค่าสัญญาณจากวงจรส่งออกไปยังอุปกรณ์ภายนอก สามารถเขียนจากภายในวงจรได้ แต่ไม่สามารถอ่านจากภายในวงจรได้
- InOut (Bidirectional) เป็นพอร์ต 2 ทิศทาง ที่สามารถส่งถูกเขียนและอ่านได้จากภายในวงจร
- Buffer (Output w/ internal feedback) เป็นพอร์ตเอาต์พุตประเภทหนึ่ง ที่สามารถอ่านค่ากลับเข้ามาภายในวงจรได้

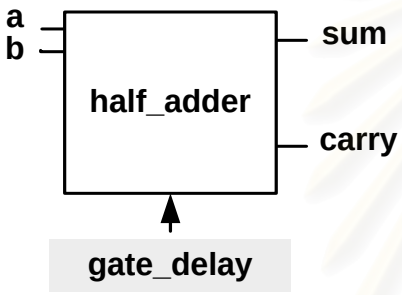
ตัวอย่าง 3.4 การประกาศส่วนของ Entity ของวงจร Half-adder ในรูปแบบ VHDL'87




```
ENTITY half_adder IS
   PORT (  a :      IN  BIT;
           b :      IN  BIT;
           sum :    OUT BIT;
           carry :  OUT BIT
           );
END half_adder;    -- VHDL'87
```

ในตัวอย่างที่ 3.4 เป็นการเขียน Entity ที่บรรยายฮาร์ดแวร์วงจร half_adder ซึ่งในส่วนของหัวของ Entity มีการกำหนดพอร์ตเชื่อมต่อ 4 พอร์ตภายใต้คำสั่ง PORT โดยมีอินพุตพอร์ต 2 พอร์ตได้แก่ a และ b ซึ่งถูกกำหนดทิศทางของพอร์ตแบบ IN และมีพอร์ตเอาต์พุต 2 พอร์ต ได้แก่ sum และ carry ซึ่งถูกกำหนดทิศทางของพอร์ตแบบ OUT โดยทั้งแบบอินพุตและเอาต์พุตพอร์ตเป็นประเภท BIT (ชนิดข้อมูลมาตรฐาน มีค่าสถานะข้อมูลคือ '0' และ '1' เท่านั้น) และจบด้วยคำสั่ง END half_adder ซึ่งเป็นรูปแบบการเขียนภาษา VHDL'87 ถ้าเป็นรูปแบบการเขียน VHDL'93 จะต้องจบด้วยคำสั่ง END ENTITY half_adder

ตัวอย่าง 3.5 การเขียน Entity ของวงจร Half-adder แบบมีการใช้คำสั่ง GENERIC




```
ENTITY half_adder IS
   GENERIC( gate_delay : TIME := 2 ns);
   PORT (  a :      IN  BIT;
           b :      IN  BIT;
           sum :    OUT BIT;
           carry :  OUT BIT
           );
END ENTITY half_adder;     -- VHDL'93
```

ผู้ออกแบบสามารถ กำหนดค่าพารามิเตอร์ที่เป็นข้อมูลอื่นๆ เพิ่มเติมในส่วนของ Entity ได้ เช่น ข้อมูลเกี่ยวกับความเร็วในการทำงานของวงจร เช่น ค่าหน่วงเวลา โดยใช้คำสั่ง GENERIC ดังแสดงในตัวอย่างที่ 3.5 ซึ่งค่าหน่วงเวลาของวงจร ถูกกำหนดเป็นตัวแปร gate_delay เป็นตัวแปรชนิดเวลา มีค่า 2 นาโนวินาที

แต่ในบางกรณี Entity ที่เขียนขึ้นอาจไม่มีพอร์ตที่ใช้เชื่อมต่อกับอุปกรณ์ภายนอกได้เช่นกัน ซึ่งส่วนใหญ่จะพบ Entity รูปแบบดังกล่าวในการเขียนโค้ดสำหรับทดสอบการทำงานของวงจร (Test bench)


```
ENTITY testbench IS
END testbench;
```

Architecture

เป็นหน่วยการออกแบบส่วนที่ใช้เขียนบรรยายพฤติกรรมการทำงานของวงจรที่ต้องการออกแบบ โดยมีความสัมพันธ์กับสิ่งที่กำหนดใน Entity มีรูปแบบการเขียนดังนี้

```
ARCHITECTURE arch_name OF entity_name IS
  Signal_declaratons
  Constant_declarations
  Type_declarations
  Subtype_declarations
  Subprogram_declarations
  Component_declarations

BEGIN
     -- <concurrent_statements>
     PROCESS_Statements
     Concurent_Signal_assignments
     Component_instantiation_statements
     Generate_Statements

END arch_name;
```

หมายเหตุ: ถ้าเป็นรูปแบบการเขียน VHDL'93 ในบรรทัด END จะมีคำว่า ARCHITECTURE ต่อท้ายคำสั่ง END ด้วย

หน่วยการออกแบบส่วนของ Architecture นั้น จะเริ่มต้นด้วยคำว่า ARCHITECTURE และตามด้วยชื่อของหน่วยการออกแบบ (arch_name) ที่ต้องการกำหนดไป และสิ่งที่แสดงให้เห็นว่า Architecture นั้นใช้บรรยาย Entity ใด (OF entity_name) และตามด้วย IS จากนั้นจะเป็นส่วนของการประกาศวัตถุต่างๆ ที่เรียกใช้ภายใน Architecture เช่น สัญญาณ, ค่าคงที่, ชนิดของข้อมูลที่กำหนด, โปรแกรมย่อย และอุปกรณ์ และตามด้วยคำสั่ง BEGIN กับ END ระหว่างคำสั่ง BEGIN กับ END จะเป็นคำสั่งประเภท Concurrent statement ทั้งหมด เช่น คำสั่ง PROCESS, Concurrent signal assignment, Component instantiation statements และ Generate statement เป็นต้น

ในหน่วยการออกแบบส่วนของ Architecture สามารถเขียนรูปแบบของแบบจำลอง (Modeling styles) ได้หลายรูปแบบ เช่น

- แบบอธิบายเชิงพฤติกรรม (Behavioral style) (Algorithm)
- แบบอธิบายเชิงข้อมูล (Data flow style) (RTL)
- แบบอธิบายเชิงโครงสร้าง (Structural style) (Netlist)
- แบบจำลองผสม (Mixed model style)

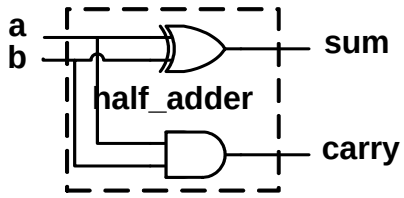
ตัวอย่าง 3.6 การเขียน Architecture ในรูปแบบ Behavioral style ของวงจร Half-adder

```
-- Behavioral descriptions
ARCHITECTURE algorithms OF half_adder IS

BEGIN
      PROCESS(a,b)
        VARIABLE x,y : BIT;
      BEGIN
        x := a;
        y := b;
        sum   <= a XOR b AFTER gate_delay;
        carry <= a AND b AFTER gate_delay;
      END PROCESS;

END algorithms;
```

ตัวอย่าง 3.7 การเขียน Architecture ในรูปแบบ Data flow style ของวงจร Half-adder

```
-- Dataflow descriptions
ARCHITECTURE rtl OF half_adder IS

BEGIN

      sum   <= a XOR b AFTER gate_delay;
      carry <= a AND b AFTER gate_delay;

END rtl;
```

จากตัวอย่าง ผลลัพธ์ของการดำเนินการระหว่าง a XOR b ที่ได้จะส่งไปที่เอาต์พุต sum โดยมีการหน่วงเวลา 2 นาโนวินาที (gate_delay) พร้อมกับค่าผลลัพธ์ของการดำเนินการระหว่าง a AND b ที่ได้จะส่งไปที่เอาต์พุต carry โดยมีการหน่วงเวลา 2 นาโนวินาทีเช่นกัน จะเห็นได้ว่าผลลัพธ์ของทั้ง 2 บรรทัดจะออกพร้อมกัน เพราะทั้ง 2 บรรทัดเป็นคำสั่งแบบ Concurrent ซึ่งจะไม่สนใจลำดับก่อนหลังของบรรทัดการเขียนคำสั่ง

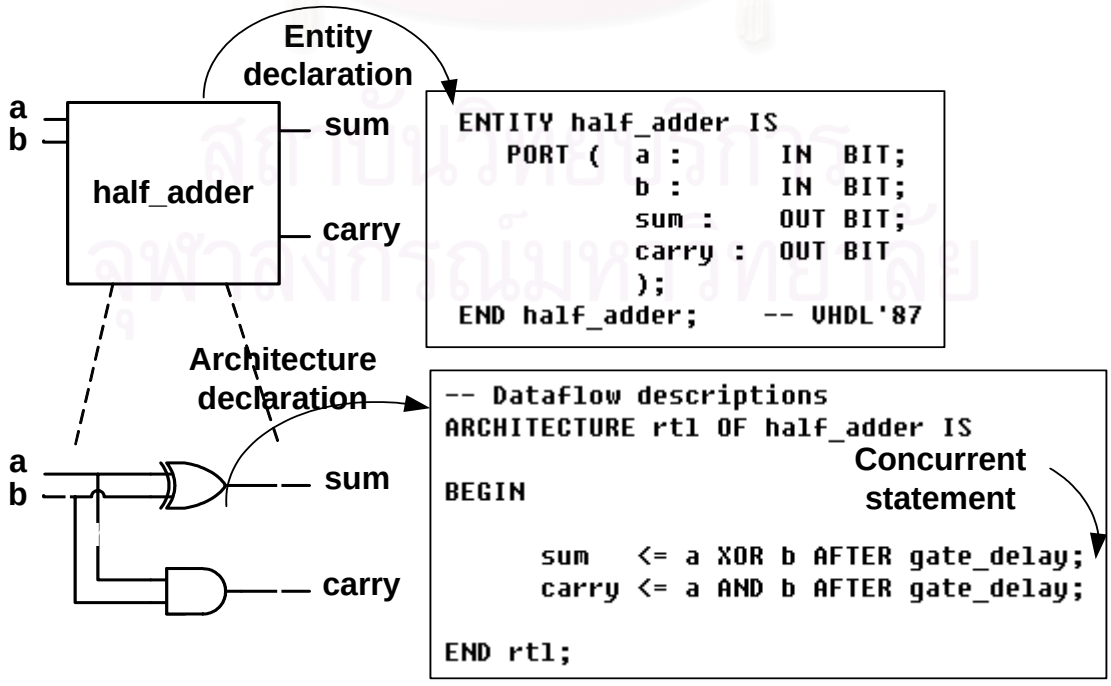


รูปที่ 3.8 รายละเอียดวงจร Half-adder ขนาด 1 บิต

ตัวอย่าง 3.8 การเขียน Architecture ในรูปแบบ Structural style ของวงจร Half-adder

```
-- Structural descriptions
ARCHITECTURE netlist OF half_adder IS

    -- XOR-gate 2 inputs component
    COMPONENT xor2
        GENERIC (gate_delay : TIME );
        PORT (x,y : IN  BIT;
              z   : OUT BIT );
    END COMPONENT;

    -- AND-gate 2 inputs component
    COMPONENT and2
        GENERIC (gate_delay : TIME );
        PORT (m,n : IN  BIT;
              o   : OUT BIT );
    END COMPONENT;

BEGIN

    u0 : xor2
                GENERIC MAP (gate_delay => 2 ns)
                PORT MAP (x => a,
                          y => b,
                          z => sum );
    u1 : and2
                GENERIC MAP (gate_delay => 2 ns)
                PORT MAP (m => a,
                          n => b,
                          o => carry );

END netlist;
```

ตัวอย่าง 3.9 การเขียนบรรยายวงจร Half-adder ในส่วนของ Entity และ Architecture



Package

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อยที่เป็นประโยชน์ในการเขียนรูปแบบบรรยายวงจรดิจิทัล สามารถเก็บไว้ในส่วนที่เรียกว่า Package โดยข้อมูลใน Package สามารถถูกเรียกใช้ได้โดยหน่วยการออกแบบ Entity, Architecture หรือ Package อื่นๆ นอกจากนั้นสิ่งที่นิยมทำกันมากคือ รูปแบบมาตรฐานต่างๆ เช่น Standard components ต่างๆ จะถูกเก็บไว้ใน Package ที่ทุกคนเรียกใช้งานได้ โดยสิ่งที่สามารถสร้างไว้ใน Package ได้แก่

- Subprogram
- Types
- Constants
- Signals
- Aliases
- Attributes
- Component
- Disconnection specification

การเรียกใช้งาน Package จะใช้คำสั่ง USE โดยมีรูปแบบดังนี้

```
USE library_name.package_name.item;
```

การเขียน Package จะแบ่งออกเป็น 2 ส่วนคือ

- **Package declaration** เป็นส่วนที่สำคัญที่สุดของ Package เพราะจะเป็นส่วนที่กำหนดชื่อ (Identifier) ของสิ่งที่ประกาศอยู่ภายใน Package สำหรับนำไปใช้ภายนอก Package ถ้าสิ่งใดถูกประกาศในส่วนของ Package body แต่ไม่ถูกประกาศใน Package declaration ก็จะไม่สามารถนำไปใช้งานได้จากภายนอก ซึ่งเปรียบเทียบได้กับการประกาศพอร์ตใน Entity ที่มีหน้าติดต่ออุปกรณ์ภายนอก ดังนั้นโดยทั่วไปแล้ว Package สามารถเขียนขึ้นได้โดยไม่จำเป็นต้องมีส่วนของ Package body และยังสามารถถูกนำไปใช้ได้จากภายนอก เช่น ใช้สำหรับประกาศชนิดของข้อมูล (Type) หรือสัญญาณ (Signal) ในทางกลับกันกับ Package body ที่ไม่จำเป็นต้องมี Package declaration แต่ Package นั้นจะไม่สามารถถูกนำไปใช้จากภายนอกได้ การเขียน Package declaration มีรูปแบบดังนี้

```
PACKAGE package_name IS
    Constant_declarations
    Type_declarations
    Signal_declarations
    Component_declarations
END package_name;
```

หมายเหตุ: ถ้าเป็นรูปแบบการเขียน VHDL'93 ในบรรทัด END จะมีคำว่า PACKAGE ต่อท้ายคำสั่ง END ด้วย

- **Package body** เป็นโครงสร้างที่ประกอบด้วยคำสั่งต่างๆ ในรูปของคำสั่งลำดับ (Sequential statement) ที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อย (Subprogram) ที่ได้ประกาศไว้ใน Package declaration แล้ว มีรูปแบบการเขียนดังนี้

```
PACKAGE BODY package_name IS
    Constant_declarations
    Type_declarations
    Subprogram_body
END package_name;
```

หมายเหตุ: ถ้าเป็นรูปแบบการเขียน VHDL'93 ในบรรทัด END จะมีคำว่า PACKAGE ต่อท้ายคำสั่ง END ด้วย

ตัวอย่าง 3.10 การเขียน Package declairation และ Package body

```
-- Package declaration
PACKAGE ref_pack IS
    PROCEDURE parity
        (SIGNAL x : IN BIT_VECTOR;
         SIGNAL y : OUT BIT );
END ref_pack;

-- Package body
PACKAGE BODY ref_pack IS

    PROCEDURE parity
        (SIGNAL x : IN BIT_VECTOR;
         SIGNAL y : OUT BIT );
    BEGIN
        -- procedure code
    END parity;
END ref_pack;
```

Configuration

การเขียน Entity ในภาษา VHDL นั้นสามารถมีได้หลาย Architecture และภายใน Architecture ยังมีอุปกรณ์ที่เรียกว่า Component ที่อ้างถึง Entity อื่นๆ อีก ดังนั้นจึงเกิดคำถามขึ้นว่า ในการจำลองการทำงานแต่ละครั้ง โปรแกรมจะนำเอา Architecture ใดไปจำลองการทำงาน คำตอบก็คือจะต้องเขียนส่วน Configuration ในการกำหนดว่าจะนำเอา Architecture ใดไปใช้งาน มีรูปแบบดังนี้

```
CONFIGURATION config_name OF entity_name IS
    FOR architecture_name
        FOR instance_label : component_name
            USE ENTITY
            library_name.entity_name(arh_name);
                FOR arch_name
                ...
                lower-level_configuration
                specifications
                ...
                END FOR;
        END FOR;
    END FOR;
END config_name;
```

หมายเหตุ: ถ้าเป็นรูปแบบการเขียน VHDL'93 ในบรรทัด END จะมีคำว่า CONFIGURATION ต่อท้ายคำสั่ง END ด้วย

ตัวอย่าง 3.11 การเขียนหน่วยการออกแบบ Configuration ของวงจร Full-adder

```
CONFIGURATION config_full_adder OF full_adder IS
    FOR structural
        FOR u1 : half_adder
            USE ENTITY
            work.half_adder(rtl);
        END FOR;
        FOR u2 : half_adder
            USE ENTITY
            work.half_adder(rtl);
        END FOR;
        FOR u3 : or_gate
            USE ENTITY
            work.or_gate(rtl);
        END FOR;
    END FOR;
END config_full_adder;
```

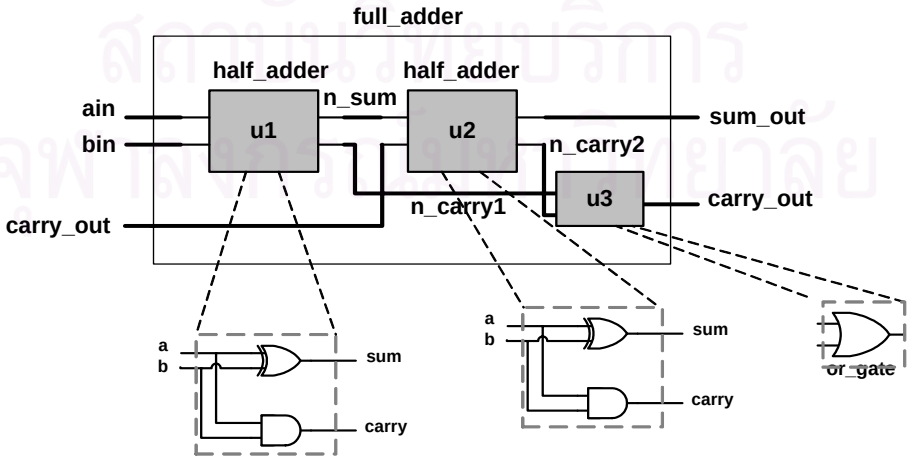
Configuration ส่วนใหญ่จะไม่สามารถนำไปใช้กับโปรแกรมสังเคราะห์วงจร จะใช้เฉพาะขั้นตอนการจำลองการทำงานเท่านั้น เพราะฉะนั้นในขั้นตอนการสังเคราะห์วงจร 1 Entity จะต้องมีเพียง 1 Architecture เท่านั้น ในกรณีที่ออกแบบให้ 1 Entity มีเพียง 1 Architecture ก็ไม่จำเป็นต้องเขียน Configuration บอกโปรแกรมจำลองการทำงาน เรียกว่า Default configuration หรือถ้าบางครั้งออกแบบให้ 1 Entity มีหลาย Architecture ก็อาจไม่จำเป็นต้องเขียน Configuration เพื่อบอกโปรแกรมจำลองการทำงานได้เช่นกัน โดยโปรแกรมจำลองการทำงานจะเลือกเอา Architecture ที่ถูกคอมไพล์ครั้งล่าสุดไปใช้ในการจำลองการทำงาน

3.7.5 Hierarchical model

ในการออกแบบวงจรดิจิทัล บางครั้งจำเป็นต้องมีการแบ่งวงจรเป็นบล็อกย่อยๆ ตามหน้าที่การทำงาน เนื่องระบบที่ต้องการออกแบบมีความซับซ้อนสูง ไม่สามารถที่จะออกแบบได้เพียงบล็อกเดียว ทำให้จำเป็นต้องแบ่งวงจรออกเป็นบล็อกย่อยๆ ซึ่งในภาษา VHDL ก็มีความสามารถในการออกแบบในลักษณะที่เป็นโครงสร้างแบบลำดับชั้น (Hierarchy) ได้ โดยบรรยายวงจรตั้งแต่ระดับบนสุด (Top model) ที่มีแต่ความสัมพันธ์ของการเชื่อมต่อในแต่ละบล็อกย่อย โดยยังไม่มีรายละเอียดของวงจรในแต่ละบล็อก และในระดับล่างลงไปถึงจะมีการอธิบายรายละเอียดของวงจร

ตัวอย่าง 3.12 การออกแบบวงจร Full-adder ขนาด 1 บิตแบบลำดับชั้น

ตัวอย่างนี้แสดงให้เห็นถึงรูปแบบลำดับชั้นในภาษา VHDL ที่กล่าวมาข้างต้น มีโครงสร้างวงจรดังรูปที่ 3.9 ประกอบด้วยบล็อกย่อยทั้งหมด 3 บล็อกมาเชื่อมต่อกัน โดยมีบล็อก u1, u2 เป็นวงจร Half-adder ที่ได้ออกแบบไว้แล้วในเบื้องต้น (ดูจากตัวอย่าง 3.9) และบล็อก u3 เป็นวงจรออร์เกต (Or-gate)



รูปที่ 3.9 โครงสร้างของวงจร Full-adder ที่มีลักษณะโครงสร้างเป็นลำดับชั้น

ตัวอย่างการเขียนภาษา VHDL บรรยายวงจร Full-adder ที่มีโครงสร้างแบบลำดับชั้น

```
ENTITY full_adder IS
    PORT ( ain, bin, cin :     IN  BIT;
           sum_out,carry_out : OUT BIT
         );
END full_adder;

ARCHITECTURE structural OF full_adder IS

-- Component declaration
COMPONENT half_adder
    GENERIC (gate_delay : TIME );
    PORT ( a :      IN  BIT;
           b :      IN  BIT;
           sum :    OUT BIT;
           carry :  OUT BIT
           );
END COMPONENT;

COMPONENT or_gate
    PORT ( in1  :  IN  BIT;
           in2  :  IN  BIT;
           outq :  OUT BIT
           );
END COMPONENT;

-- Signal declaration
SIGNAL n_sum, n_carry1, n_carry2 : BIT;
```

```
BEGIN

    -- Component instantiation
    u1 : half_adder
                 GENERIC MAP ( gate_delay => 2 ns )
                 PORT MAP    ( a     =>  ain,
                               b     =>  bin,
                               sum   =>  n_sum,
                               carry =>  n_carry1 );
    u2 : half_adder
                 GENERIC MAP ( gate_delay => 2 ns )
                 PORT MAP    ( a     =>  n_sum,
                               b     =>  cin,
                               sum   =>  sum_out,
                               carry =>  n_carry2 );
    u3 : or_gate
                 PORT MAP ( in1   => n_carry1,
                            in2   => n_carry2,
                            outq  => carry_out );

END structural;
```

จากรูปแบบการเขียนภาษา VHDL ข้างต้นมีคำสั่งใหม่คือ คำสั่ง COMPONENT เป็นคำสั่งใช้ในการประกาศโมดูลย่อยที่ต้องการนำมาเชื่อมต่อในวงจรหลัก และคำสั่ง PORT MAP เป็นคำสั่งที่ทำหน้าที่เชื่อมต่อพอร์ตต่างๆ ของโมดูลย่อยกับสัญญาณต่างๆ ภายในวงจรหลัก รายละเอียดต่างๆ จะกล่าวในหัวข้อต่อไป

3.7.6 Component declairation

เป็นการประกาศโมดูลย่อยที่ต้องการนำมาเชื่อมต่อกันภายในวงจรส่วนบนสุด จะเป็นการเขียนในลักษณะที่มีโครงสร้างเป็นลำดับชั้น โดยการประกาศโมดูลย่อยนั้นจะต้องทำการออกแบบโมดูลย่อยดังกล่าวไว้เรียบร้อยแล้ว เพียงแต่เรียกโมดูลดังกล่าวมาใช้งาน การออกแบบโมดูลย่อยก็คล้ายกับการออกแบบวงจรโดยทั่วไป คือสร้างส่วนของ Entity และ Architecture ขึ้นมาก่อน แล้วเก็บไว้ในไลบรารี (Library) จากนั้นถ้าวงจรที่ออกแบบขึ้นมาใหม่ต้องการเรียกใช้วงจรที่เคยออกแบบไว้แล้ว ก็สามารถทำได้โดยประกาศวงจรที่ได้ออกแบบไว้แล้วนั้นในรูปของอุปกรณ์ (Component declaration) โดยมีรูปแบบการเขียนดังนี้

```
COMPONENT component_name
    GENERIC (generic_list);
    PORT (port_declaration);
END COMPONENT;
```

ตำแหน่งของการประกาศอุปกรณ์นั้น สามารถประกาศได้ในบริเวณที่อยู่ระหว่างคำสั่ง ARCHITECTURE กับ BEGIN ดังตัวอย่างที่ 3.12 ถ้าเปรียบเทียบกับการออกแบบวงจรบน PCB (Print circuit board) แล้ว การประกาศอุปกรณ์ในภาษา VHDL จะเปรียบเสมือนเป็นซ็อกเก็ต (Socket) ว่างๆ สำหรับใส่ตัวไอซี โดยซ็อกเก็ตนี้มีหน้าที่เพียงเชื่อมต่อสัญญาณเท่านั้น ไม่มีหน้าที่สร้างฟังก์ชันการทำงาน การประกาศอุปกรณ์ก็เช่นกัน จะไม่มีการแสดงฟังก์ชันการทำงานของอุปกรณ์ แต่มีหน้าที่เพียงแสดงพอร์ตของอุปกรณ์เท่านั้น

3.7.7 Component instantiation

เป็นการเชื่อมต่อสัญญาณต่างๆ ของอุปกรณ์ที่ประกาศไว้ในส่วนของ Component declaration เข้ากับพอร์ตหรือสัญญาณภายในของวงจรระดับบน โดยมีรูปแบบดังนี้

```
instance_label : component_name
     GENERIC MAP (generic_association_list)
     PORT MAP (
                  port_name => sig_name,
                  port_name => sig_name
                  ...
                );
```

ตำแหน่งของการเขียน Component instantiation เป็นรูปแบบหนึ่งของ Concurrent Statement สามารถเขียนภายใต้ BEGIN ของ ARCHICTURE ได้ ดังตัวอย่างที่ 3.12

คำสั่ง GENERIC MAP เป็นคำสั่งเสริม คือถ้าอุปกรณ์นั้นมีการใช้คำสั่ง GENERIC อยู่ เมื่อนำอุปกรณ์นั้นมาใช้งาน เวลาเขียน Component instantiation จะต้องมีคำสั่ง GENERIC MAP ด้วยเสมอ แต่ถ้าอุปกรณ์นั้นไม่มีการใช้คำสั่ง GENERIC เวลาเขียน Component instantiation ก็ไม่ต้องมีส่วนของ GENERIC MAP ด้วยเช่นกัน ดังตัวอย่างที่ 3.12

ในการเชื่อมต่อสัญญาณต่างๆ เข้ากับพอร์ตของอุปกรณ์ สามารถทำได้ 2 รูปแบบคือ เชื่อมต่อสัญญาณแบบ Port association list คือพิจารณาจากตำแหน่งของพอร์ต และตำแหน่งสัญญาณ พอร์ตและสัญญาณที่อยู่ตำแหน่งตรงกันจะหมายความว่าต่อถึงกันอยู่ ไม่ต้องมีการอ้างชื่อพอร์ตของอุปกรณ์อีก เพียงแต่วางตำแหน่งให้ตรงกันก็พอ ดังตัวอย่างต่อไปนี้

```
u3 : or_gate    PORT MAP (n_carry1, n_carry2, carry_out);
```

จากตัวอย่างมีความหมายดังนี้ (ดูตัวอย่างที่ 3.12 ประกอบ) สัญญาณ n_carry1 ถูกเชื่อมต่อกับพอร์ต in1 ของโมดูล or_gate, สัญญาณ n_carry2 ถูกเชื่อมต่อกับพอร์ต in2 ของโมดูล or_gate และพอร์ต carry_out ของโมดูล full_adder ถูกเชื่อมต่อกับพอร์ต outq ของโมดูล or_gate ตามลำดับ

การใช้คำสั่ง PORT MAP อีกรูปแบบหนึ่งคือ แบบ Name association list เป็นการเชื่อมต่อสัญญาณที่ไม่พิจารณาจากตำแหน่ง แต่จะอ้างอิงด้วยชื่อในการเชื่อมต่อแทน ดังตัวอย่างต่อไปนี้

```
u3 : or_gate
                PORT MAP (
                              in2   => n_carry2,
                              in1   => n_carry1,
                              outq  => carry_out );
```

ในตัวอย่างนี้สามารถเขียนสลับตำแหน่งกันได้ เพราะการเขียนคำสั่ง PORT MAP ในลักษณะนี้จะอ้างอิงจากชื่อ โดยไม่สนใจตำแหน่งหรือลำดับการเขียนแต่อย่างใด

3.7.8 Library

Library เป็นส่วนที่ใช้เก็บข้อมูลต่างๆ ของหน่วยการออกแบบที่ได้ทำการวิเคราะห์ตรวจสอบความถูกต้องตามกฎเกณฑ์การเขียน และความถูกต้องของฟังก์ชันการทำงานแล้ว (Compiled and simulated) เพื่อให้พร้อมที่จะนำไปใช้ได้กับการออกแบบอื่นๆ โดยการอ้างถึงชื่อของ Library จะอ้างจากข้อมูลที่เก็บไว้ใน Library นั้น โดยแบ่งเป็น

- **Primary units** ได้แก่ Entity declarations, Package declarations และ Configuration specifications
- **Secondary units** ได้แก่ Architecture bodies และ Package bodies

ส่วน Secondary units เป็นส่วนที่ต้องอ้างถึง Primary units ดังนั้นจะต้องคอมไพล์ในส่วน Primary units ก่อนเสมอ เช่น คอมไพล์ Package declarations ก่อนคอมไพล์ Package bodies หรือคอมไพล์ Entity ก่อนคอมไพล์ Architecture โดยปกติจะมี Library อยู่ 2 ประเภทคือ

- **Working library** หมายถึง ไดเรกทอรี (Directory) ที่กำลังใช้งานอยู่ ปกติจะถูกกำหนดให้เป็น Work เสมอ
- **Resource library** หมายถึง Library เพิ่มเติม โดยสามารถตั้งชื่ออะไรก็ได้ ซึ่งในภาษา VHDL ได้สงวนชื่อของไลบรารีไว้ 2 ชื่อด้วยกันคือ Std และ Work

การเรียกใช้งาน Library มีรูปแบบดังนี้

```
LIBRARY library_name, another_library_name;
```

ตัวอย่างต่อไปนี้เป็นการเรียกใช้ Library ชื่อ IEEE ในส่วนของ Package ที่ชื่อว่า std_logic_1164 โดยคำสั่ง .ALL ที่ต่อท้ายชื่อ Package หมายความว่าเลือกใช้ทุกอย่างที่ประกาศอยู่ภายใน Package นั้น

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL; -- (std_logic types & related functions)
USE ieee.std_logic_arth.ALL; -- (arithmetic functions)
USE ieee.std_logic_signed.ALL;   -- (signed arithmetic functions)
USE ieee.std_logic_unsigned.ALL; -- (unsigned arithmetic functions)
```

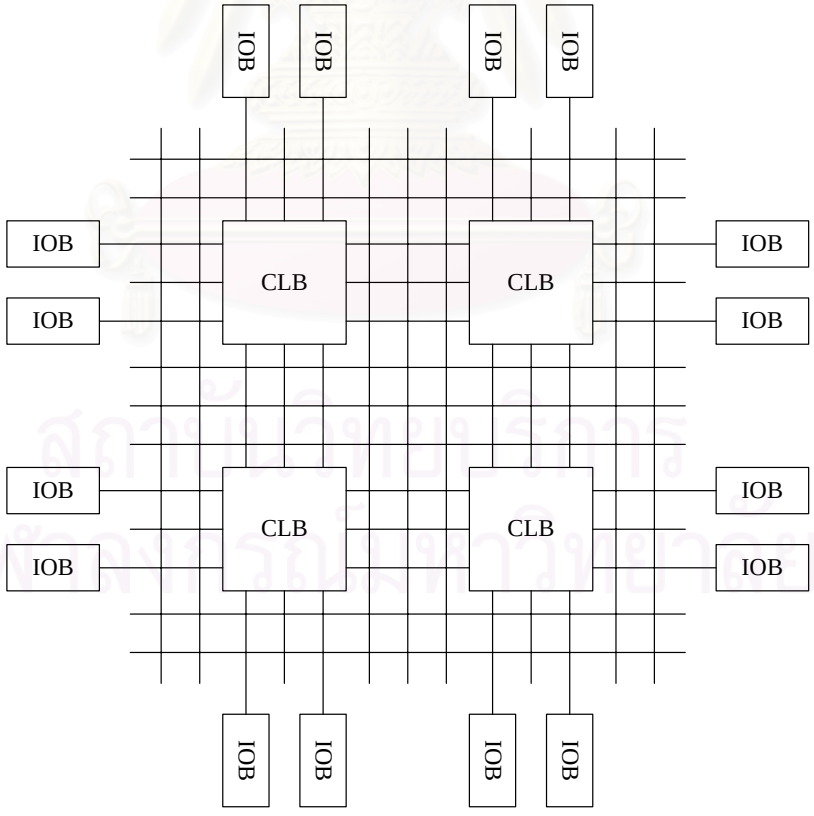
3.8 การพัฒนาวงจรดิจิทัลด้วย FPGA

ความก้าวหน้าในอุตสาหกรรมอิเล็กทรอนิกส์ทำให้เกิดการพัฒนาความสามารถอุปกรณ์ต่างๆ มากมาย ทั้งในด้านการลดค่าใช้จ่าย, การสิ้นเปลืองพลังงานและขนาด ขณะเดียวกันก็เพิ่มประสิทธิภาพการทำงานและความน่าเชื่อถือได้ของวงจรรวมมากขึ้น ทำให้การออกแบบวงจรที่มีขนาดใหญ่และการทำงานซับซ้อนมีความเป็นไปได้และสะดวกยิ่งขึ้น วงจรรวมที่นำมาใช้ในการออกแบบเพื่อการใช้งานเฉพาะอย่างนี้ เรียกว่า ASIC (Application specific integrated circuit)

อุปกรณ์วงจรรวมดิจิทัลประเภท ASIC มี 2 กลุ่ม [18] คือ

- Field programmable เช่น PLD และ FPGA
- Mask programmable เช่น Gata array และ Standard cell

โดยอุปกรณ์ทั้ง 2 กลุ่มมีข้อแตกต่างที่สำคัญ คือ ผู้ออกแบบสามารถโปรแกรมอุปกรณ์ประเภท Field programmable ได้เองโดยไม่ต้องนำไปโรงงานเพื่อผลิต และในปัจจุบันมีการพัฒนาเครื่องมือและอุปกรณ์เพื่อช่วยในการออกแบบและจัดสร้างวงจรออกมาอย่างต่อเนื่อง รวมทั้งอุปกรณ์ Field programmable เหล่านี้สามารถหาซื้อได้ง่าย ทำให้การสร้างวงจรอิเล็กทรอนิกส์หันมาใช้อุปกรณ์ประเภทนี้เป็นอุปกรณ์ประกอบใน วงจรแทนการใช้อุปกรณ์ย่อยๆ (Discrete component) มากขึ้น



รูปที่ 3.10 โครงสร้างโดยทั่วไปของ FPGA

ภายในอุปกรณ์ FPGA จะประกอบด้วยส่วนประกอบหลัก 3 ส่วน ได้แก่

- กลุ่มวงจรลอจิกที่กำหนดการทำงานได้ (Configurable logic block: CLB) โครงสร้างภายในกลุ่มวงจรลอจิกประกอบด้วยวงจรเกตและฟลิปฟลอปที่สามารถกำหนดให้ทำงานตามที่ต้องการได้
- กลุ่มอินพุตเอาต์พุต (Input output block: IOB) ใช้เป็นจุดเชื่อมต่อกับอุปกรณ์ภายนอก
- เส้นทางเชื่อมต่อวงจรภายใน (Interconnect) ซึ่งเชื่อมต่อระหว่างกลุ่มวงจรลอจิกต่างๆ เข้าด้วยกัน และเชื่อมต่อกับกลุ่มอินพุตเอาต์พุต

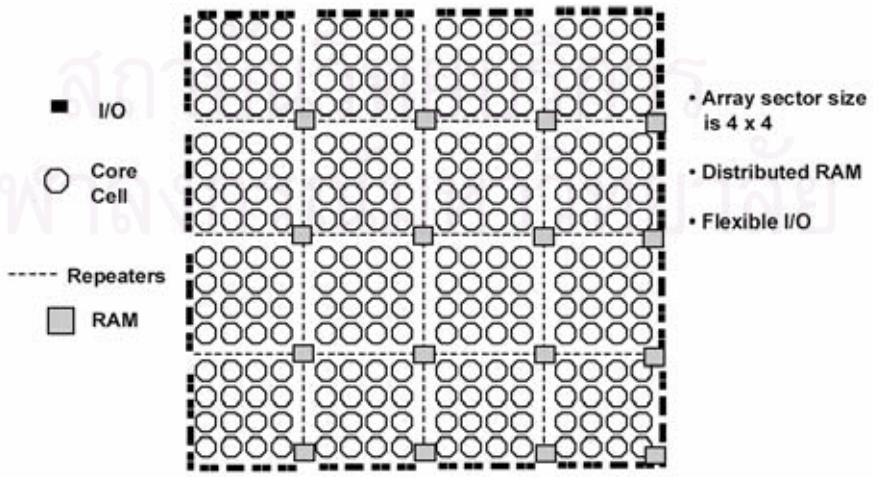
นอกจากนี้ FPGA บางรุ่นยังมีส่วนประกอบอื่นๆ อีก เช่น หน่วยความจำ (SRAM) เป็นต้น

การทำงานของ FPGA ขึ้นอยู่กับการกำหนดการทำงานของกลุ่มวงจรลอจิก และกลุ่มอินพุตเอาต์พุต และกำหนดเส้นทางเชื่อมต่อระหว่างกลุ่มต่างๆ

ในการวิจัยนี้เลือกใช้ FPGA รุ่น AT40K ของบริษัท ATMEL จึงขอกล่าวถึงรายละเอียดของโครงสร้างภายใน FPGA ของเบอร์นี้เพื่อเป็นตัวอย่างดังต่อไปนี้

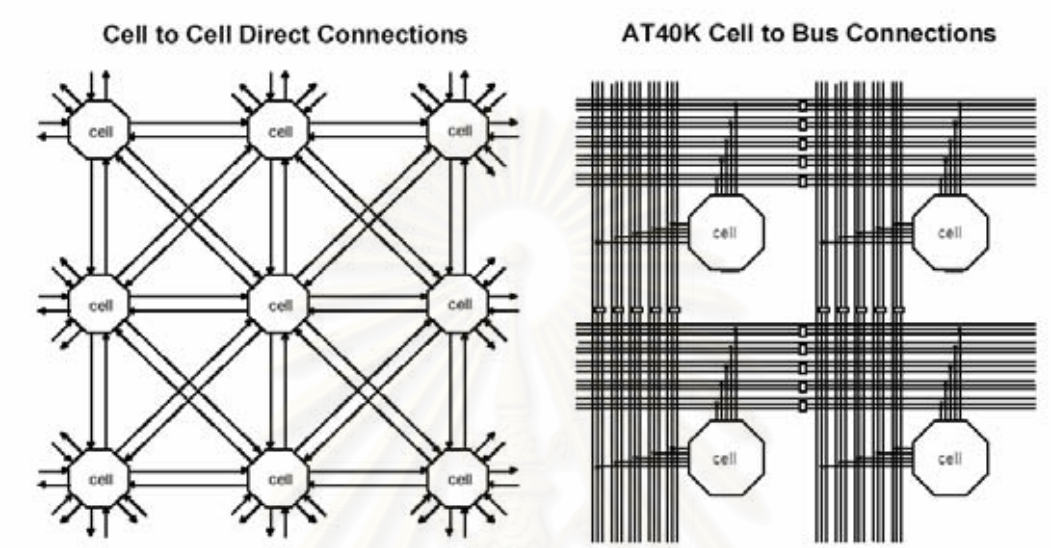
โครงสร้างภายในของชิป AT40K ประกอบด้วยกลุ่มของเซลล์ (Cell) ขนาด 4×4 เซลล์ แต่ละกลุ่มของเซลล์นี้จะมีหน่วยความจำ RAM แบบ 2 พอร์ต (Dual-ported RAM) ขนาด 32×4 บิต ทุกช่วง 4 เซลล์จะมีรีพีตเตอร์ (Repeater) ทำหน้าที่เชื่อมต่อบัสสัญญาณต่างๆ เข้าด้วยกัน [12]

การวิจัยครั้งนี้เลือกใช้ FPGA เบอร์ AT40K10LV-3AJC ประกอบด้วยเซลล์จำนวน 24×24 = 576 เซลล์ มีความจุเกตประมาณ 10,000-20,000 เกต มีรีจิสเตอร์หรือฟลิปฟลอปจำนวน 576 ตัว มี SRAM ขนาด 32×4×36 = 4,608 บิต และมีขาอินพุตเอาต์พุตให้ใช้งานจำนวน 62 ขา เหตุผลที่เลือกใช้ FPGA เบอร์นี้เนื่องจากมีความจุเกตมากพอสมควร, มีหน่วยความจำ RAM ภายใน และส่วนประกอบอื่นๆ ที่เพียงพอต่อการพัฒนางานวิจัยในครั้งนี้

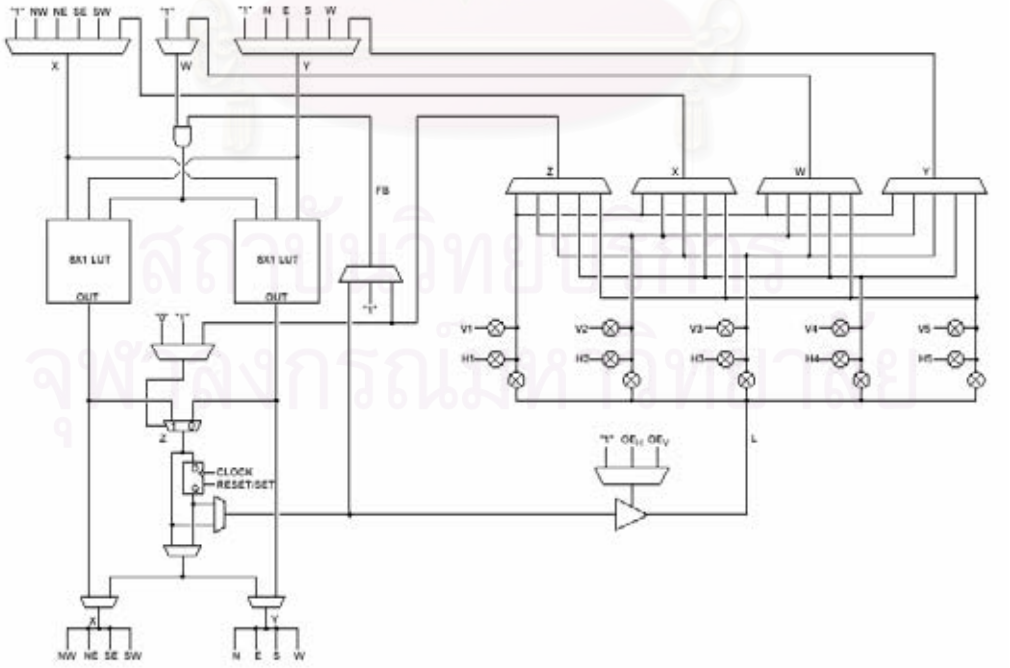


รูปที่ 3.11 โครงสร้างของ FPGA เบอร์ AT40K ของ ATMEL

การเชื่อมต่อสัญญาณระหว่างเซลล์มี 2 แบบ คือ การเชื่อมต่อสัญญาณโดยตรง (Direct connections) เป็นการเชื่อมต่อสัญญาณโดยตรงจากเซลล์หนึ่งไปยังเซลล์ที่อยู่ติดกันทั้ง 8 ทิศทาง อีกแบบหนึ่งคือการเชื่อมต่อสัญญาณแบบบัส (Bus connections) โดยในแต่ละแถวและแต่ละหลักจะมีบัสสัญญาณอยู่ 5 เส้น และทุกๆ 4 แถวและ 4 หลักจะมีรีพีตเตอร์ทำหน้าที่เชื่อมต่อสัญญาณระหว่างบัส



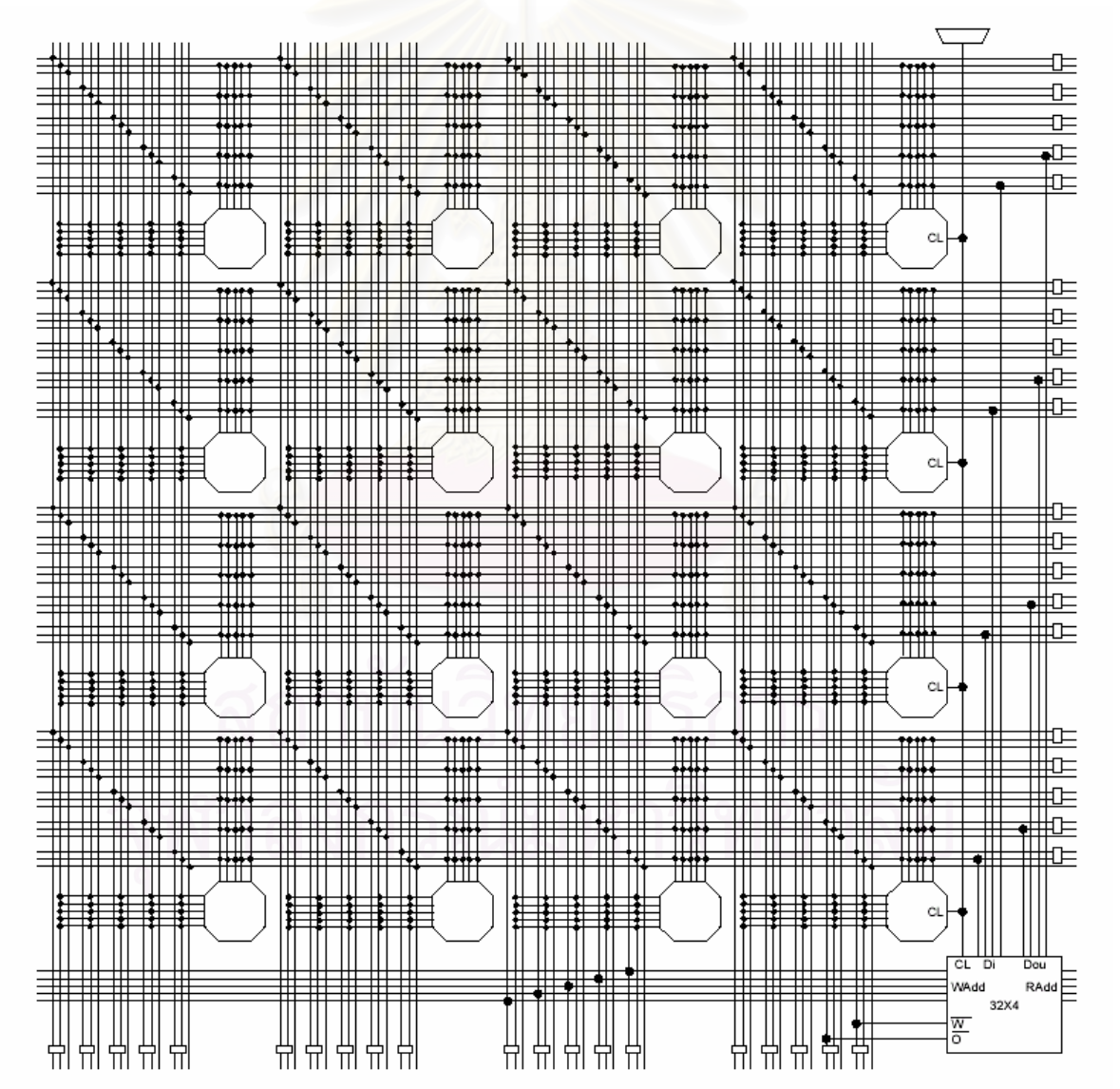
รูปที่ 3.12 การเชื่อมต่อสัญญาณระหว่างเซลล์ภายใน FPGA รุ่น AT40K
มี 2 แบบ คือ แบบ Direct และแบบ Bus



รูปที่ 3.13 โครงสร้างภายในเซลล์ของ FPGA รุ่น AT40K

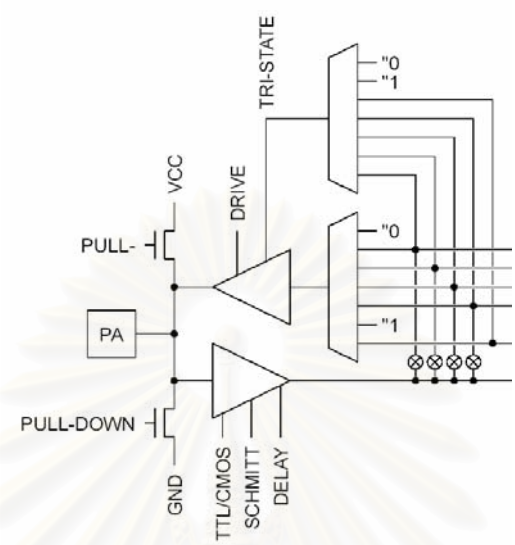
หัวใจสำคัญในการทำงานของ FPGA อยู่ที่ส่วนประกอบภายในเซลล์ โดยแต่ละเซลล์ประกอบด้วย LUT (Look-up table) และฟลิปฟลอป โดยมีมัลติเพล็กเซอร์ทำหน้าที่เลือกสัญญาณที่เข้าและออกเซลล์ รวมทั้งสัญญาณที่ใช้ในการทำงานภายในเซลล์

สิ่งที่พิเศษอย่างหนึ่งของ FPGA รุ่น AT40K นี้ก็คือ มีหน่วยความจำ SRAM แยกออกมาต่างหากจากลอจิกเซลล์ โดยแต่ละกลุ่มเซลล์ขนาด 4X4 เซลล์ จะมี SRAM ขนาด 32X4 บิตอยู่ 1 ตัว การที่มีหน่วยความจำอยู่ใน FPGA มีข้อดีคือ เป็นการประหยัดลอจิกเซลล์เมื่อวงจรต้องการใช้หน่วยความจำ โดยไม่จำเป็นต้องนำลอจิกเซลล์มาใช้สร้างหน่วยความจำ เนื่องจากการสร้างหน่วยความจำด้วยลอจิกเซลล์จะเป็นการสิ้นเปลืองทรัพยากร (ลอจิกเซลล์) มากเมื่อเทียบกับการใช้หน่วยความจำโดยตรง



รูปที่ 3.14 แต่ละกลุ่มของลอจิกเซลล์ จะมีหน่วยความจำ SRAM ขนาด 32X4 บิตประจำอยู่ 1 ตัว

ส่วนสุดท้ายที่จะกล่าวถึงสำหรับโครงสร้างของ FPGA ก็คือ พอร์ตอินพุตเอาต์พุต แต่ละพอร์ตสามารถกำหนดการทำงานได้ว่าจะให้เป็นอินพุตชนิดใด (TTL, CMOS, ชมิตต์ทริกเกอร์) และเอาต์พุตชนิดใด (บัฟเฟอร์, 3-state) และกำหนดได้ว่าจะให้มีการพูลอัปหรือพูลดาวน์ได้อีกด้วย



รูปที่ 3.15 โครงสร้างพอร์ตอินพุต/เอาต์พุตของ FPGA เบอร์ AT40K

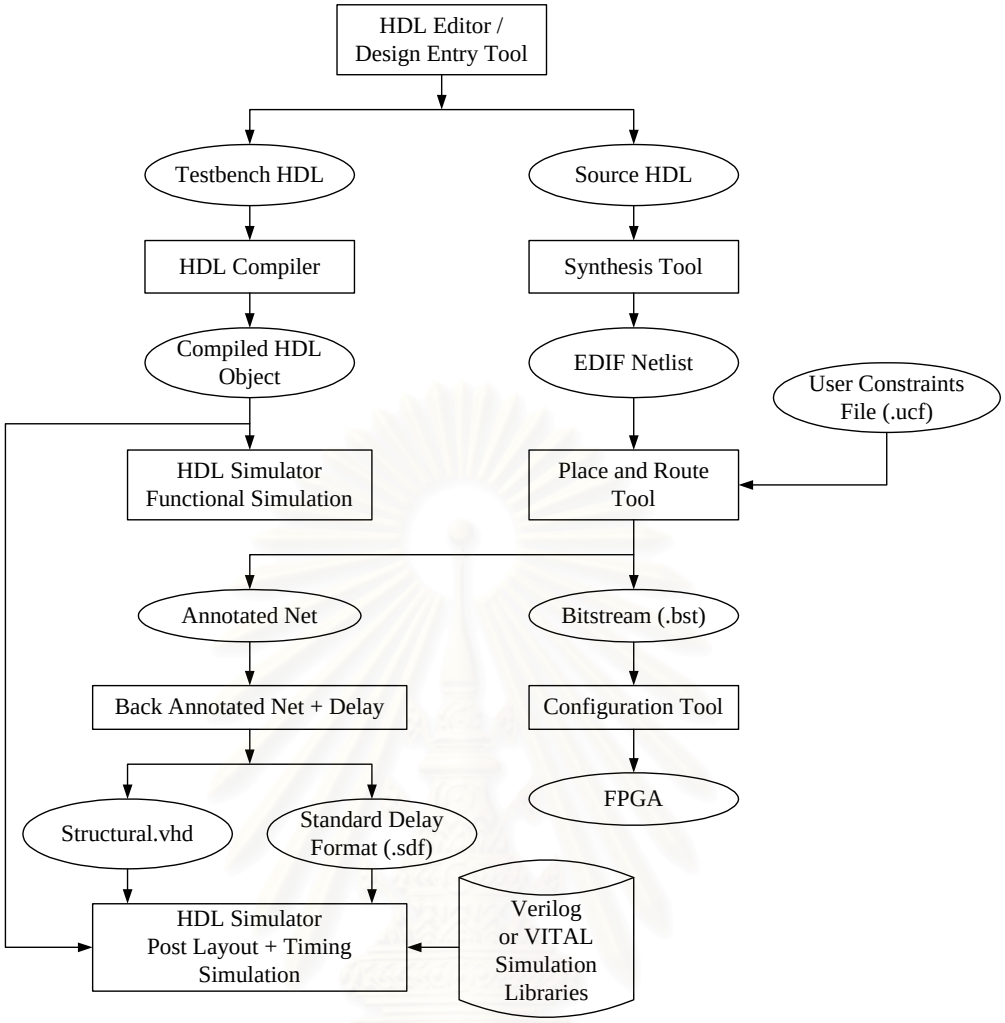
กระบวนการออกแบบวงจรรวมดิจิทัลเพื่อใช้กับอุปกรณ์ FPGA นั้นมีหลายขั้นตอน เริ่มตั้งแต่การออกแบบวงจร, การจำลองการทำงาน, การสังเคราะห์วงจรตามเทคโนโลยีที่เลือกใช้, และการวางและจัดเส้นทางเชื่อมต่อวงจร ตามลำดับ ซึ่งแต่ละขั้นตอนมีรายละเอียดดังนี้

3.8.1 **การออกแบบวงจร** (Design Entry)

สามารถออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ (Hardware description language: HDL) เช่น ภาษา VHDL หรือ ภาษา Verilog เป็นต้น หรือออกแบบโดยใช้แผนภาพวงจร (Schematic) ซึ่งเป็นการอธิบายวงจรเชิงโครงสร้าง (Structural description)

3.8.2 **การจำลองการทำงาน** (Simulation)

เมื่อออกแบบวงจรเรียบร้อยแล้ว ขั้นตอนที่สำคัญขั้นตอนหนึ่งก็คือ การตรวจสอบความถูกต้องของวงจรด้วยวิธีการจำลองการทำงาน โดยในขั้นแรกจะเป็นการตรวจสอบฟังก์ชันการทำงานของวงจร (Functional simulation) เพื่อดูว่าฟังก์ชันการทำงานของวงจรเป็นไปตามที่ออกแบบไว้หรือไม่ เป็นการตรวจสอบทางพฤติกรรมของวงจรเท่านั้น ยังไม่มีข้อจำกัดทางเวลา (Timing) เข้ามาเกี่ยวข้อง ดังนั้นหลังจากทำการสังเคราะห์วงจร และการจัดวางและเชื่อมต่อวงจรตามเทคโนโลยีที่เลือกใช้แล้ว จะจำลองการทำงานของวงจรโดยรวมผลทางเวลา (Timing simulation) อีกครั้ง



รูปที่ 3.16 กระบวนการออกแบบวงจรรวมดิจิทัลบน FPGA

3.8.3 **การสังเคราะห์วงจร** (Synthesis)

ไม่ว่าจะออกแบบโดยวิธีใดก็ตาม เมื่อผ่านการสังเคราะห์วงจรตามเทคโนโลยีที่เลือกใช้แล้วผลที่ได้ก็คือ รายละเอียดการต่อวงจรระดับเกต (Gate-level netlist) ซึ่งมีรูปแบบมาตรฐานที่นิยมใช้ คือ EDIF (Electronic data interchange format) ใช้ในการวางและจัดเส้นทางเชื่อมต่อวงจรต่อไป

3.8.4 **การวางวงจรและจัดเส้นทางเชื่อมต่อวงจร** (Place and Route)

จากรูปแบบรายละเอียดการต่อวงจรระดับเกต (EDIF) ที่ได้ จะนำมาวิเคราะห์และทำการจัดสรรพื้นที่ภายใน FPGA เพื่อวางวงจร (Place) และจัดเส้นทางเชื่อมต่อวงจร (Route) ให้ได้วงจรตามที่กำหนด ซึ่งสุดท้ายจะได้ข้อมูลของข้อกำหนดทางเวลาในรูปแบบของความล่าช้ามาตรฐาน (Standard delay format) เพื่อใช้ในการจำลองการทำงานของวงจรให้ถูกต้องตรงความเป็นจริงมากขึ้น และได้ข้อมูลการจัดวงจรภายในซึ่งอยู่ในรูปขบวนบิตข้อมูล (Bitstream) เพื่อใช้โปรแกรมลงบนอุปกรณ์ต่อไป

3.8.5 **การโปรแกรมอุปกรณ์** FPGA (Configuration)

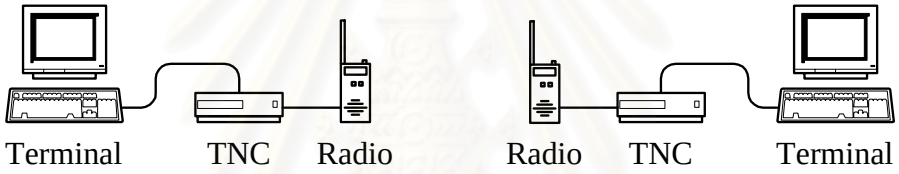
เมื่อผ่านขั้นตอนต่างๆ ทั้งหมดแล้ว ตั้งแต่การออกแบบวงจร การสังเคราะห์วงจร การวางวงจร และจัดเส้นทางเชื่อมต่อวงจร จนได้ข้อมูลการจัดวงจรภายในชิป FPGA แล้ว ก็มาถึงขั้นตอนสุดท้าย นั่นคือ การโปรแกรมหรือดาวน์โหลดข้อมูลที่ได้ลง FPGA มี 2 รูปแบบคือ การดาวน์โหลดข้อมูลจากคอมพิวเตอร์หรือเครื่องโปรแกรม ในกรณีที่อยู่ในช่วงของการพัฒนาวงจรต้นแบบที่ต้องมีการแก้ไขและดาวน์โหลดอยู่บ่อยครั้ง เนื่องจากโครงสร้างของ FPGA เป็นแบบ SRAM จึงสามารถดาวน์โหลดข้อมูลใหม่เพื่อแก้ไขได้ไม่จำกัด แต่ก็มีข้อเสียคือ ไม่สามารถเก็บรักษาข้อมูลไว้ได้เมื่อไม่จ่ายไฟเลี้ยงให้ ดังนั้นจึงต้องมีวิธีการดาวน์โหลดข้อมูลลง FPGA ในอีกรูปแบบหนึ่ง เพื่อให้สามารถนำ FPGA ไปใช้งานได้โดยไม่ต้องพึ่งเครื่องโปรแกรม นั่นคือ การดาวน์โหลดข้อมูลจาก ROM ที่ต่ออยู่ภายนอก FPGA ทั้งแบบขนานและแบบอนุกรม ข้อมูลที่ต้องการดาวน์โหลดลง FPGA จะถูกเก็บไว้ใน ROM นี้ เมื่อเริ่มจ่ายไฟเลี้ยง FPGA จะทำการอ่านข้อมูลจาก ROM ภายนอกเพื่อใช้ในการกำหนดค่าการทำงานภายใน FPGA(Configuration) เมื่อกำหนดค่าเสร็จเรียบร้อย FPGA ก็จะเริ่มทำงานทันที

บทที่ 4

การออกแบบตัวควบคุมเทอร์มินัลโนดโดยใช้ FPGA

4.1 แนวทางการออกแบบ

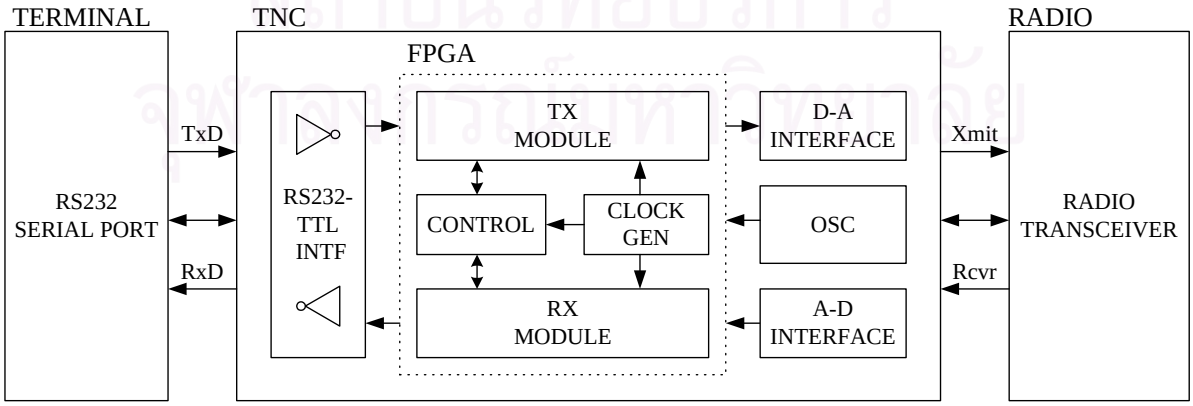
ตัวควบคุมเทอร์มินัลโนด (TNC) ที่ออกแบบจะทำหน้าที่เป็นอุปกรณ์แปลงข้อมูลจากอุปกรณ์เทอร์มินัล (คอมพิวเตอร์) ให้อยู่ในรูปแบบที่สามารถส่งผ่านตัวกลางหรือพาหะ (คลื่นวิทยุ) ได้ การติดต่อกับอุปกรณ์เทอร์มินัลใช้มาตรฐาน RS-232 ในรูปข้อมูลอนุกรมแบบอะซิงโครนัส โดยภาคส่งจะแปลงข้อมูลแล้วส่งไปยังภาคส่งของเครื่องรับส่งวิทยุ เพื่อสร้างคลื่นวิทยุส่งผ่านตัวกลางไปยังภาครับของอุปกรณ์เทอร์มินัลปลายทางต่อไป ในภาครับก็จะรับข้อมูลที่ได้รับจากภาครับของเครื่องรับส่งวิทยุและแปลงข้อมูลกลับแล้วส่งไปยังอุปกรณ์เทอร์มินัล



รูปที่ 4.1 รูปแบบการใช้งานตัวควบคุมเทอร์มินัลโนด

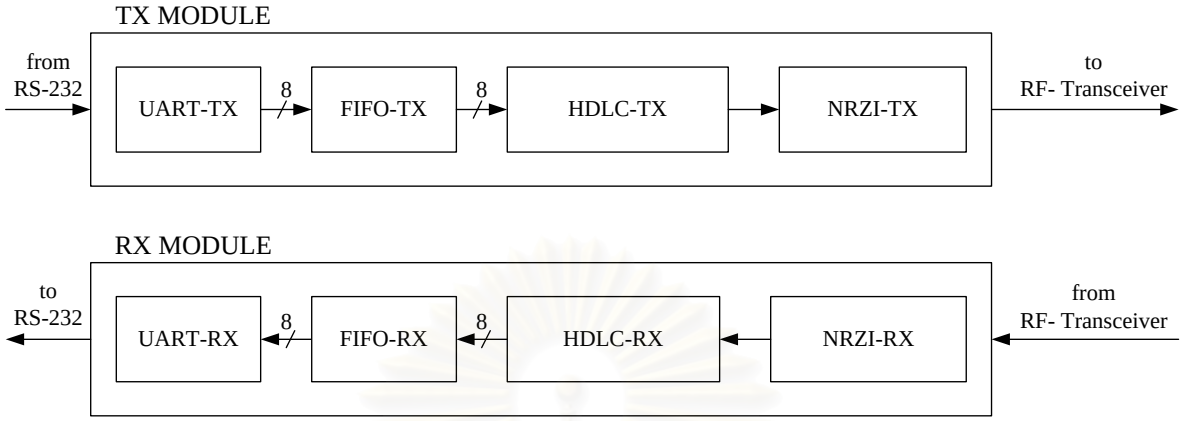
นอกจากนี้ตัวควบคุมเทอร์มินัลโนดยังทำหน้าที่จัดข้อมูลที่ส่งออกเป็นกลุ่มข้อมูลย่อยหรือเฟรมตามโพรโทคอล AX.25 สำหรับเครือข่ายวิทยุกลุ่มข้อมูล

โครงสร้างทางฮาร์ดแวร์ของตัวควบคุมเทอร์มินัลโนดที่ออกแบบจะมีลักษณะดังรูปที่ 4.2



รูปที่ 4.2 โครงสร้างทางฮาร์ดแวร์ของตัวควบคุมเทอร์มินัลโนด

ในส่วนของการทำงานหลักจะออกแบบโดยใช้ FPGA โดยมีส่วนประกอบในการทำงานที่สำคัญ ดังรูปที่ 4.3



รูปที่ 4.3 โครงสร้างการทำงานภายในส่วน FPGA ในส่วนของโมดูลภาคส่ง และโมดูลภาครับ

- UART-TX, UART-RX ทำหน้าที่เปลี่ยนรูปแบบข้อมูลระหว่างข้อมูลแบบขนานกับข้อมูลแบบอนุกรมอะซิงโครนัส
- FIFO-TX, FIFO-RX เป็นหน่วยความจำที่ทำหน้าที่เก็บข้อมูลชั่วคราว
- HDLC-TX, HDLC-RX ทำหน้าที่จัดข้อมูลให้สอดคล้องกับโพรโทคอล AX.25 ทั้งในส่วนของการจัดเฟรม, การใส่แฟล็ก, การเติมบิต '0' และการตรวจสอบความผิดพลาด
- NRZI-TX, NRZI-RX ทำหน้าที่แปลงข้อมูลให้มีรูปแบบที่เหมาะสมสำหรับส่งผ่านระบบรับส่งวิทยุสื่อสาร

4.2 ข้อกำหนดในการออกแบบ

ตัวควบคุมเทอร์มินัลโนดที่ออกแบบในวิทยานิพนธ์นี้ จะอ้างอิงการทำงานตามโพรโทคอล AX.25 โดยเน้นที่การทำงานในเชิงฮาร์ดแวร์เป็นส่วนสำคัญ ได้แก่ การจัดแบ่งเฟรมข้อมูล, การแทรกฟีลด์แฟล็ก, การแทรกบิต/ลบบิต '0', การคำนวน CRC และการแทรกฟีลด์ FCS โดยไม่ครอบคลุมถึงในส่วนของการทำงานในเชิงซอฟต์แวร์ ได้แก่ การเข้ารหัส และการตีความหมายของฟีลด์ที่อยู่, ฟีลด์ควบคุม และฟีลด์ PID รวมถึงการโต้ตอบเฟรม และคำสั่งต่างๆ ตามโพรโทคอล AX.25 โดยการทำงานต่างๆ เหล่านี้คอมพิวเตอร์จะเป็นผู้จัดการ

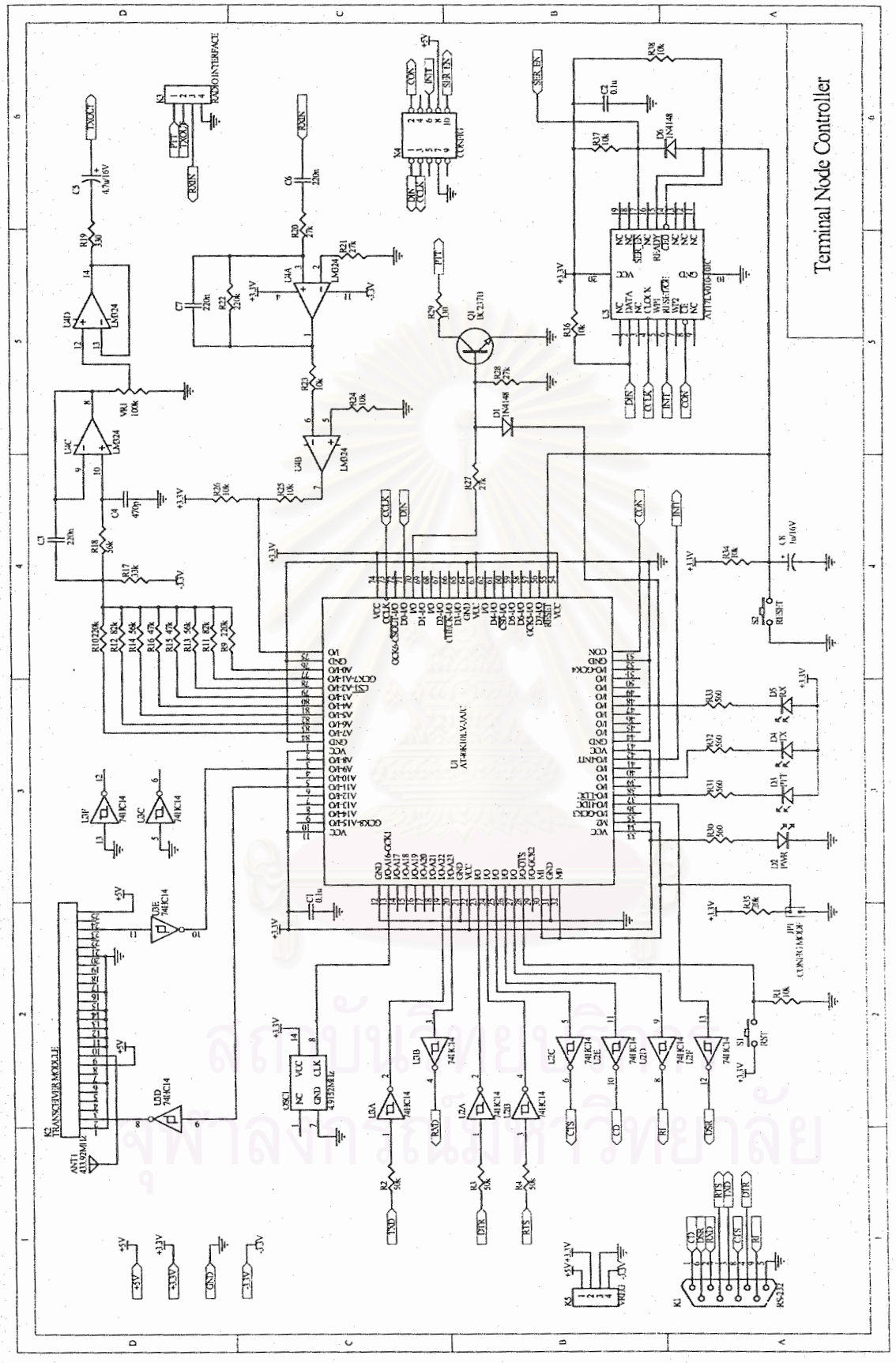
และเนื่องจาก FPGA เบอร์ที่นำมาใช้ คือ AT40K10LV-3AJC นั้นมีความจุเกตไม่มากนัก คือประมาณ 10,000 ถึง 20,000 เกต ดังนั้นเพื่อให้สามารถออกแบบได้โดยไม่ต้องเปลี่ยนเบอร์ FPGA และ

สะดวกในการสร้างเครื่องต้นแบบ จึงกำหนดให้ฟิลด์ที่อยู่มีขนาดเพียง 1 ไบต์ (ในขณะที่ตามโพรโทคอลกำหนดไว้ 14-28 ไบต์) โดยที่รูปแบบของเฟรมข้อมูลยังคงเหมือนเดิม

ในส่วนของการรับส่งข้อมูลทางคลื่นวิทยุ ผู้วิจัยได้ใช้โมดูลรับส่งข้อมูลผ่านคลื่นวิทยุ (RF transceiver) แทนการใช้เครื่องรับส่งวิทยุสมัครเล่น เนื่องจากการใช้เครื่องรับส่งวิทยุสมัครเล่นมีการรบกวนจากผู้ใช้ช่องสัญญาณรายอื่นได้ง่าย และมีอัตราการรับส่งข้อมูลที่ได้ค่อนข้างต่ำ โดยเลือกใช้โมดูลที่มอดูเลตแบบ AM ย่านความถี่ 433.92 เมกะเฮิรตซ์ (UHF) มีอัตราเร็วในการรับส่งข้อมูลสูงสุด 4,800 บิตต่อวินาที ซึ่งมากกว่าอัตราการรับส่งข้อมูลของ HDLC ที่กำหนดไว้ที่ 1,200 บิตต่อวินาที

SYSTEM	การออกแบบส่วนควบคุม TNC	FPGA เบอร์ AT40K10LV-3AJC
	ความถี่ของสัญญาณนาฬิกาที่ใช้	4.9152 เมกะเฮิรตซ์
PC INTERFACE	การอินเตอร์เฟซข้อมูลกับคอมพิวเตอร์	แบบอนุกรมมาตรฐาน RS-232
	อัตราการรับส่งข้อมูลกับคอมพิวเตอร์	19,200 บิตต่อวินาที
BUFFER	รูปแบบหน่วยความจำ	FIFO
	ขนาดหน่วยความจำข้อมูล	64 ไบต์ แบ่งเป็น - ภาคส่ง 32 ไบต์ - ภาครับ 32 ไบต์
HDLC	ข้อกำหนดทั่วไป	- จัดแบ่งเฟรมข้อมูล - แทรกฟิลด์แฟล็ก - แทรก/ลบบิต ‘0’ - คำนวณ/ตรวจสอบ CRC16 - แทรกฟิลด์ FCS - ไม่มีฟิลด์ PID (ถือเป็นข้อมูล)
	ขนาดฟิลด์ที่อยู่	1 ไบต์
	ขนาดฟิลด์ควบคุม	1 ไบต์
	ขนาดฟิลด์ข่าวสาร	0-32 ไบต์
	อัตราการรับส่งข้อมูล HDLC	1,200 บิตต่อวินาที
RF INTERFACE	การเข้ารหัสสัญญาณข้อมูล	NRZI
	โมดูลรับส่ง RF ที่ใช้	RTF-DATA-SAW
	อัตราการรับส่งข้อมูลของโมดูล RF	สูงสุด 4,800 บิตต่อวินาที

ตารางที่ 4.1 ข้อกำหนดในการทำงานของตัวควบคุมเทอร์มินัลโนดที่ออกแบบ



รูปที่ 4.4 วงจรตัวควบคุมเทอร์มินัลโนด (ไม่รวมแหล่งจ่ายไฟ)

4.3 รายละเอียดในการออกแบบ

การออกแบบตัวควบคุมเทอร์มินัลโนดแบ่งออกเป็น 2 ส่วน ส่วนแรกคือ การออกแบบทางด้านฮาร์ดแวร์ ได้แก่ วงจรที่นำมาต่อทำงานร่วมกับ FPGA, วงจรแหล่งจ่ายไฟเลี้ยงสำหรับบอร์ดตัวควบคุมเทอร์มินัลโนด และวงจรของสายดาวน์โหลดข้อมูลสำหรับ FPGA ส่วนที่ 2 คือ การออกแบบส่วนควบคุมหลักของตัวควบคุมเทอร์มินัลโนดในส่วนของ FPGA

4.3.1 การออกแบบทางด้านฮาร์ดแวร์

รูปที่ 4.4 เป็นวงจรของตัวควบคุมเทอร์มินัลโนด ประกอบด้วยวงจร FPGA, วงจรกำเนิดสัญญาณความถี่ 4.9152 เมกะเฮิรตซ์ (OSC1), วงจรอินเตอร์เฟซกับพอร์ต RS-232 (U2, U3A และ U3B), วงจรอินเตอร์เฟซกับโมดูลรับส่งข้อมูลผ่านคลื่นวิทยุ (U3D และ U3E), วงจรอินเตอร์เฟซกับเครื่องรับส่งวิทยุสมัครเล่น (U4 และ Q1), วงจร EEPROM สำหรับการดาวน์โหลดข้อมูลลง FPGA (U5), วงจรสวิตช์ RESET และ RST, จัมเปอร์เลือกโหมดการดาวน์โหลด FPGA (JP1), และ LED แสดงสถานะการทำงาน โดยมีพอร์ตให้ต่อใช้งานผ่านคอนเน็กเตอร์ต่างๆ ซึ่งแต่ละพอร์ตมีหน้าที่ดังนี้

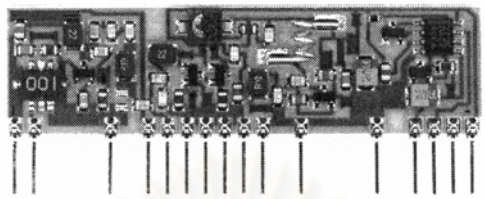
- **พอร์ต K1 (RS-232)** ใช้อินเตอร์เฟซกับคอมพิวเตอร์ แต่ละขามีหน้าที่ดังตารางที่ 4.2

ขาที่	ชื่อขา	ทิศทาง	หน้าที่
1	CD (Carrier Detect)	Out	บอกสถานะของช่องสัญญาณว่ามีผู้ใช้หรือไม่
2	RXD (Receive Data)	Out	ส่งข้อมูล RS-232 ให้กับคอมพิวเตอร์
3	TXD (Transmit Data)	In	รับข้อมูล RS-232 จากคอมพิวเตอร์
4	DTR (Data Terminal Ready)	In	สัญญาณควบคุมการทำงานของ TNC*
5	GND (Ground)	-	กราวด์ของสัญญาณ
6	DSR (Data Set Ready)	Out	บอกสถานะของ TNC ว่าพร้อมทำงานหรือไม่
7	RTS (Request To Send)	In	สัญญาณควบคุมการทำงานของ TNC*
8	CTS (Clear To Send)	Out	บอกสถานะของวงจรรับข้อมูล RS-232
9	RI (Ring Indicator)	Out	แจ้งว่ามีเฟรม (แฟล็กเริ่มต้น) ส่งเข้ามา

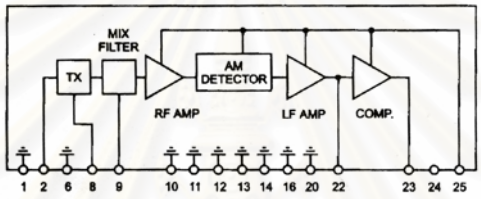
ตารางที่ 4.2 หน้าที่ของขาแต่ละขาในพอร์ต K1

* *หมายเหตุ:* หน้าที่ของขา DTR และ RTS ดูรายละเอียดเพิ่มเติมได้ในส่วนการทำงานของ FPGA

- **พอร์ต K2 (Transceiver Module)** ใช้อินเตอร์เฟซกับโมดูลรับส่งข้อมูลผ่านคลื่นวิทยุ โดยโมดูลที่ใช้เป็นโมดูลรุ่น RTF-DATA-SAW มีลักษณะดังรูปที่ 4.5 และมีวงจรภายในดังรูปที่ 4.6 แต่ละขาของโมดูลนี้มีหน้าที่ดังตารางที่ 4.3



รูปที่ 4.5 โมดูลรับส่งข้อมูลผ่านคลื่นวิทยุ (RF transceiver) รุ่น RTF-DATA-SAW



รูปที่ 4.6 วงจรภายในโมดูลรับส่งข้อมูลผ่านคลื่นวิทยุรุ่น RTF-DATA-SAW

ขาที่	ชื่อขา	ทิศทาง	หน้าที่
1, 6, 10-14, 16, 20	Ground	-	กราวด์ของไฟเลี้ยงและสัญญาณ
2	TX data input	In	ข้อมูลดิจิทัลที่ส่งให้โมดูล
8	TX +5V supply	-	ไฟเลี้ยงวงจรภาคส่ง
9	Antenna	-	ต่อสายอากาศ
22	RX analog out	Out	สัญญาณเอาต์พุตแบบแอนาล็อก
23	RX digital out	Out	สัญญาณเอาต์พุตแบบดิจิทัล
24	Not used	-	ไม่ได้ใช้งาน
25	RX +5V supply	-	ไฟเลี้ยงวงจรภาครับ

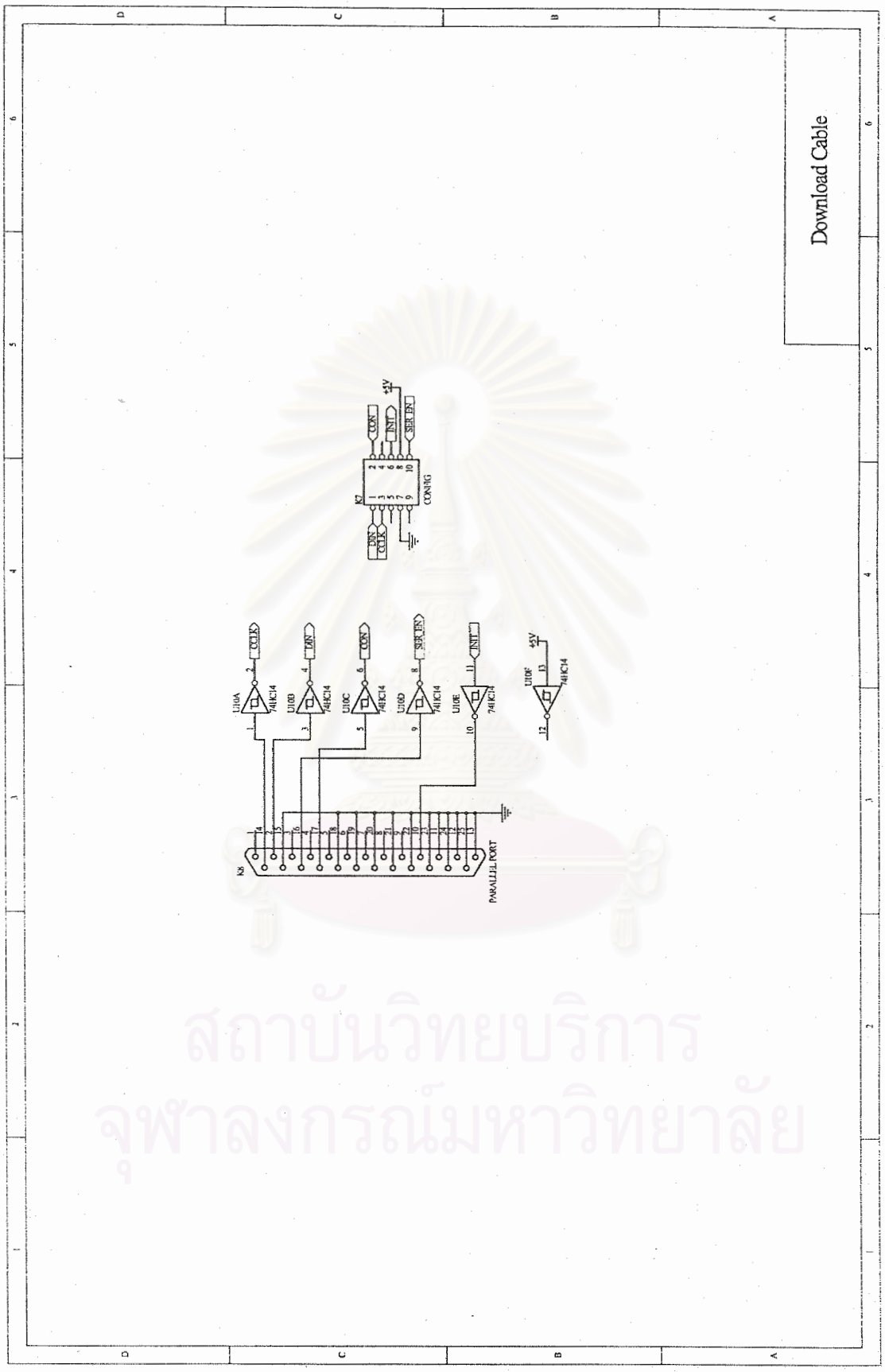
ตารางที่ 4.3 หน้าที่ของขาแต่ละขาของโมดูลรับส่งข้อมูลผ่านคลื่นวิทยุรุ่น RTF-DATA-SAW

- **พอร์ต K3 (Radio Interface)** ใช้อินเตอร์เฟซกับเครื่องรับส่งวิทยุสมัครเล่น แต่ละขามีหน้าที่ดังตารางที่ 4.4

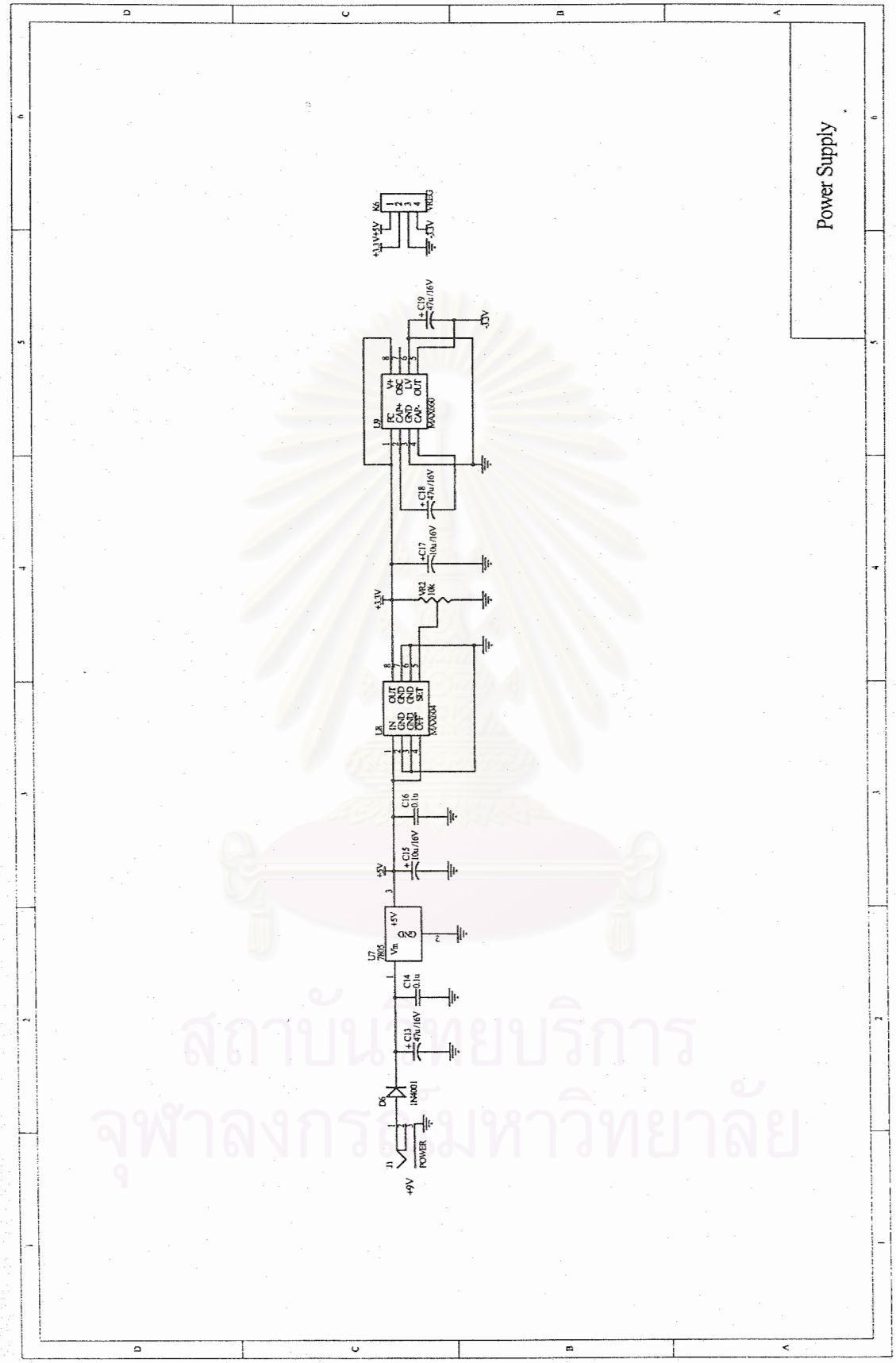
ขาที่	ชื่อขา	ทิศทาง	หน้าที่
1	PTT	Out	ควบคุมการทำงานของเครื่องรับส่งวิทยุว่าจะให้รับหรือส่ง
2	TXOUT	Out	ข้อมูลแบบแอนาล็อกเข้าช่องไมโครโฟนของเครื่องรับส่งวิทยุ
3	RXIN	In	ข้อมูลแบบแอนาล็อกจากช่องลำโพงของเครื่องรับส่งวิทยุ
4	GND	-	กราวด์ของสัญญาณ

ตารางที่ 4.4 หน้าที่ขาของแต่ละขาในพอร์ต K3

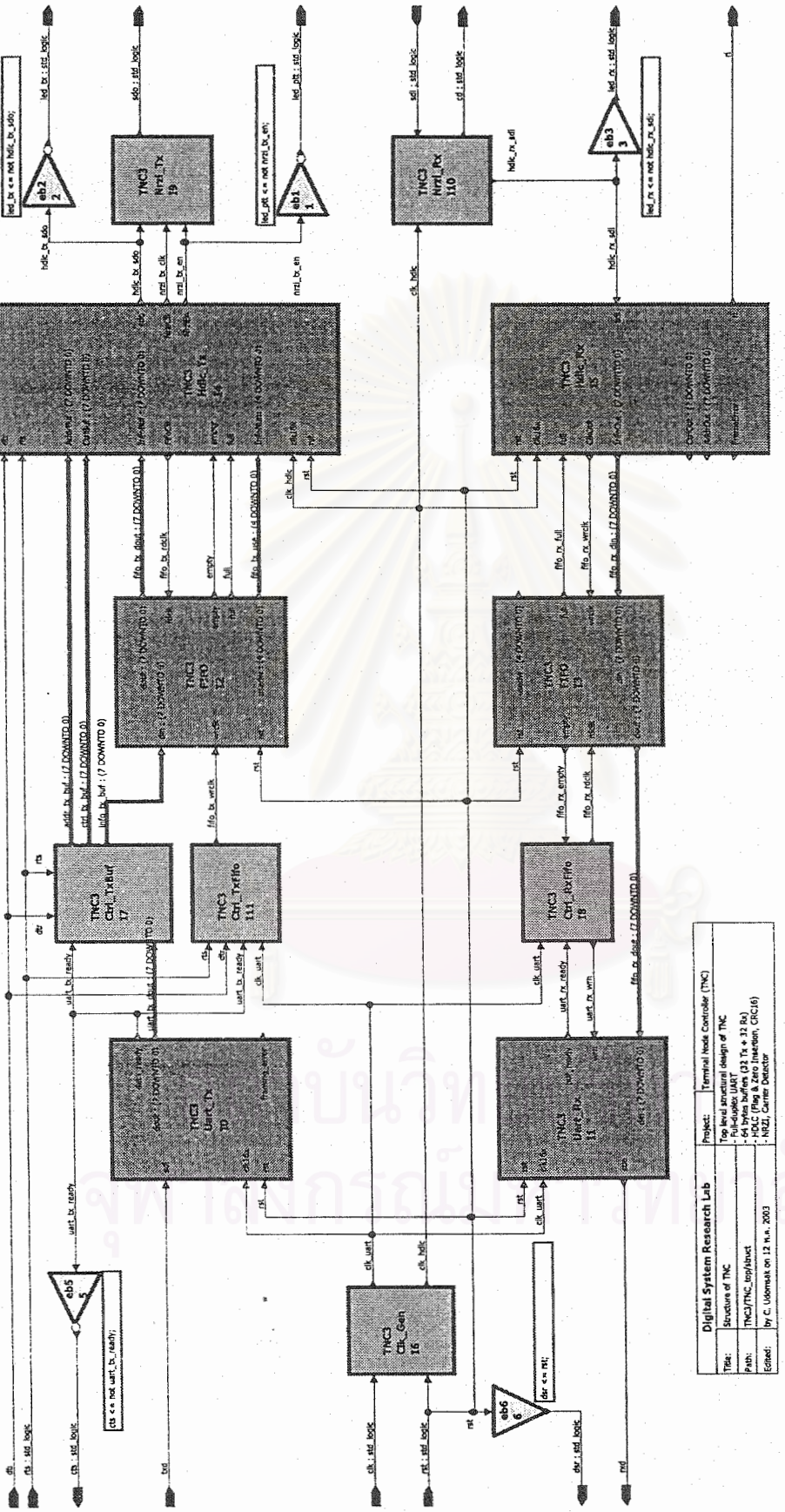
- **พอร์ต K4 (Config)** ใช้สำหรับต่อสายดาวน์โหลดข้อมูล FPGA ซึ่งมีวงจรดังรูปที่ 4.7 โดยรับข้อมูลทางพอร์ตขนานของคอมพิวเตอร์
- **พอร์ต K5 (Vreg)** เนื่องจากวงจรตัวควบคุมเทอร์มินัลโนดที่ออกแบบต้องการไฟเลี้ยงหลายค่า คือ วงจรอินเตอร์เฟซ RS-232 และวงจรอินเตอร์เฟซโมดูลรับส่ง (U2 และ U3) ใช้ไฟเลี้ยง +5 โวลต์, FPGA (U1), EEPROM (U5) และวงจรออสซิลเลเตอร์ (OSC1) ใช้ไฟเลี้ยง +3.3 โวลต์ ส่วนวงจรออปแอมป์ (U4) ที่ใช้สร้างสัญญาณอินเตอร์เฟซกับเครื่องรับส่งวิทยุสมัครเล่นจะใช้ไฟเลี้ยง $\pm$3.3 โวลต์ จึงออกแบบวงจรไฟเลี้ยงแยกไว้ต่างหากดังรูปที่ 4.8 และต่อกับบอร์ดทางพอร์ตนี้



รูปที่ 4.7 วงจรสายดาวน์โหลดข้อมูลสำหรับ FPGA



รูปที่ 4.8 วงจรไฟเลี้ยงสำหรับบอร์ดตัวควบคุมเทอร์มินัลโนด



รูปที่ 4.9 โครงสร้างของวงจรภายใน FPGA

4.3.2 การออกแบบวงจรในส่วน FPGA

โครงสร้างของวงจรภายใน FPGA ที่ออกแบบมีลักษณะดังรูปที่ 4.9 หน้าที่ของอินพุตเอาต์พุตแต่ละขาของวงจรตัวควบคุมเทอร์มินัลโนดในส่วนของ FPGA ที่ออกแบบไว้มีรายละเอียดดังตารางที่ 4.5

ชื่อขา	ตำแหน่งขาบน FPGA	ทิศทาง	หน้าที่
rst	28	In	สัญญาณรีเซต
clk	13	In	สัญญาณนาฬิกาความถี่ 4.9152 MHz
txd	19	In	สัญญาณข้อมูลอินพุตของ UART (จากคอมพิวเตอร์)
rxd	20	Out	สัญญาณข้อมูลเอาต์พุตของ UART (ไปคอมพิวเตอร์)
dtr	23	In	สัญญาณควบคุมการทำงาน (ดูตารางที่ 4.6)
rts	24	In	สัญญาณควบคุมการทำงาน (ดูตารางที่ 4.6)
cts	25	Out	สัญญาณแสดงสถานะของ UART_TX
cd	26	Out	สัญญาณแสดงสถานะของช่องสัญญาณ
ri	27	Out	สัญญาณแสดงการเริ่มต้นเฟรม (แฟล็ก)
dsr	36	Out	สัญญาณแสดงสถานะของ TNC
sdo	6	Out	สัญญาณข้อมูลเอาต์พุต NRZI (ไปภาค RF)
sdi	4	In	สัญญาณข้อมูลอินพุต NRZI (จากภาค RF)
led_ptt	37	Out	สัญญาณแสดงสถานะรับ/ส่งข้อมูล (ภาค RF)
led_tx	39	Out	สัญญาณแสดงสถานะการส่งข้อมูล (ภาค RF)
led_rx	47	Out	สัญญาณแสดงสถานะการรับข้อมูล (ภาค RF)

ตารางที่ 4.5 หน้าที่และตำแหน่งขาบน FPGA ของอินพุตเอาต์พุตต่างๆ

dtr	rts	ความหมาย
‘1’	‘1’	ข้อมูลที่ UART ได้รับเป็นข้อมูลในฟิลด์ที่อยู่
‘1’	‘0’	ข้อมูลที่ UART ได้รับเป็นข้อมูลในฟิลด์ควบคุม
‘0’	‘0’	ข้อมูลที่ UART ได้รับเป็นข้อมูลในฟิลด์ข่าวสาร
‘0’	‘1’	HDLC เริ่มส่งเฟรมของข้อมูลใน FIFO

ตารางที่ 4.6 ความหมายของสัญญาณควบคุม dtr และ rts

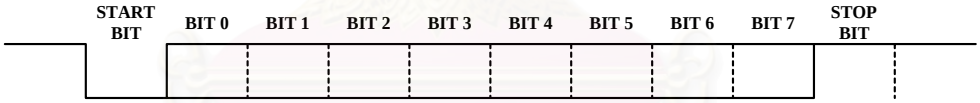
ส่วนประกอบที่สำคัญของวงจรภายใน FPGA ที่ออกแบบสำหรับตัวควบคุมเทอร์มินัลโนด มีอยู่ 4 ส่วนด้วยกัน ได้แก่

1. UART

ทำหน้าที่เปลี่ยนรูปแบบข้อมูลระหว่างข้อมูลแบบขนานที่ใช้งานภายใน FPGA กับข้อมูลแบบอนุกรมอะซิงโครนัสจากพอร์ต RS-232 ซึ่งมีลักษณะดังรูปที่ 4.10 คือ เริ่มต้นด้วยบิตเริ่มต้น (Start bit) ลอจิก '0' จำนวน 1 บิต แล้วตามด้วยบิตข้อมูลขนาด 8 บิตโดยเรียงจากบิต 0 ไปยังบิต 7 ปิดท้ายด้วยบิตหยุด (Stop bit) ลอจิก '1' จำนวน 1 บิต การทำงานของ UART แบ่งเป็น UART_TX ทำหน้าที่แปลงข้อมูลอนุกรมจากพอร์ต RS-232 ให้กลายเป็นข้อมูลแบบขนานขนาด 8 บิต แล้วส่งต่อไปยังส่วนถัดไป นั่นคือ FIFO_TX ส่วน UART_RX ทำหน้าที่รับข้อมูลแบบขนานขนาด 8 บิต จาก FIFO_RX เพื่อแปลงเป็นข้อมูลแบบอนุกรมส่งไปยังพอร์ต RS-232 ของคอมพิวเตอร์ การทำงานของ UART จะมีอัตราเร็วในการรับส่งข้อมูลอยู่ที่ 19,200 บิตต่อวินาที

ในส่วนของ UART_TX นั้น สัญญาณ data_ready จะใช้บอกสถานะการทำงานว่า UART_TX ได้รับข้อมูลเรียบร้อยแล้ว และพร้อมที่จะรับข้อมูลชุดใหม่ ส่วนสัญญาณ framing_error จะใช้บอกสถานะการทำงานว่าเฟรมข้อมูลที่ได้รับไม่ถูกต้อง คือมีบิตหยุดเป็น '0'

ในส่วนของ UART_RX นั้น สัญญาณ buf_ready ใช้บอกสถานะว่าบัฟเฟอร์ใน UART_RX ว่างพร้อมที่จะรับข้อมูลใหม่ โดยการสั่งให้ UART_RX รับข้อมูลใหม่ด้วยสัญญาณ wrn



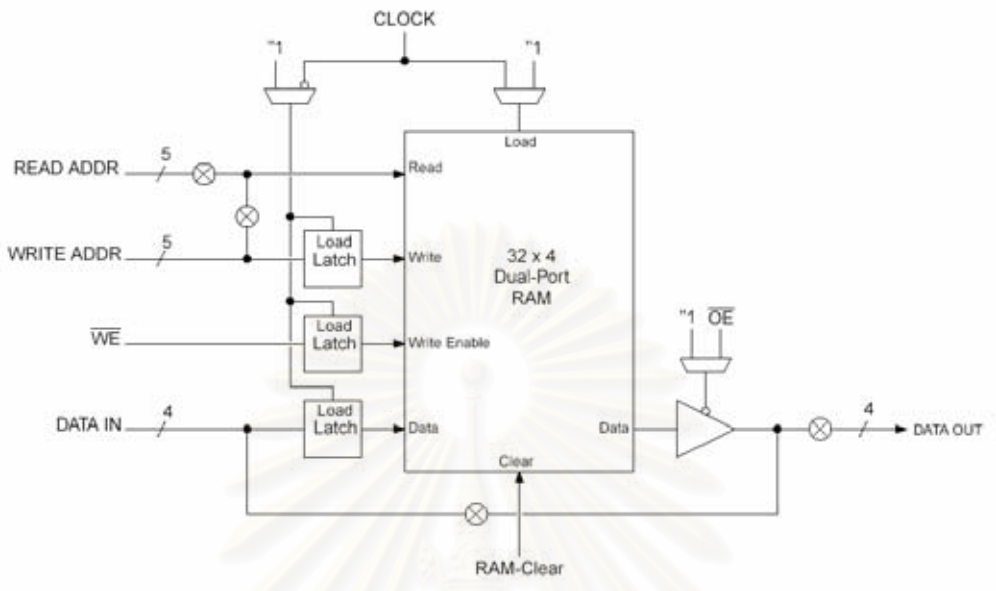
รูปที่ 4.10 รูปแบบข้อมูลของ UART

2. FIFO

เป็นหน่วยความจำที่ทำหน้าที่เก็บข้อมูลชั่วคราว โดยออกแบบให้มีหน่วยความจำขนาด 64 ไบต์ แบ่งเป็นภาคส่ง 32 ไบต์ และภาครับ 32 ไบต์ การออกแบบจะอาศัย RAM ภายใน FPGA ซึ่งมีโครงสร้างการทำงานดังรูปที่ 4.11 คือเป็น RAM แบบ 2 พอร์ตขนาดหน่วยละ 32×4 บิต ซึ่งในที่นี้หน่วยความจำ FIFO 1 ชุดจะประกอบด้วย RAM จำนวน 2 หน่วย ต่อขนานกันเป็นหน่วยความจำขนาด 32×8 บิต

การทำงานของ FIFO จะมีสัญญาณควบคุมการอ่านและเขียนข้อมูลแยกจากกัน คือสัญญาณ wrclk สำหรับเขียนข้อมูล และสัญญาณ rdclk สำหรับอ่านข้อมูล สัญญาณทั้งสองจะทำงานที่ขอบขาขึ้น โดยถ้าเกิดขอบขาขึ้นของสัญญาณ wrclk จะทำการเขียนข้อมูลลง RAM และเพิ่มค่าตัวชี้ตำแหน่งเขียนข้อมูลขึ้น 1 ในขณะที่ถ้าเกิดขอบขาขึ้นของสัญญาณ rdclk จะทำการอ่านข้อมูลจาก RAM และ

เพิ่มค่าตัวชี้ตำแหน่งอ่านข้อมูลขึ้น 1 โดยมีสัญญาณ empty และ full บอกสถานะว่า RAM ว่างและเต็มหรือไม่ ตามลำดับ และสัญญาณ usedw บอกจำนวนข้อมูลที่ถูกเก็บอยู่ใน RAM



รูปที่ 4.11 การทำงานของ RAM ภายใน FPGA

ตัวอย่างที่ 4.1 การเขียน ARCHITECTURE ในภาษา VHDL ที่ใช้แทนการทำงานของ RAM แบบ 2 พอร์ต

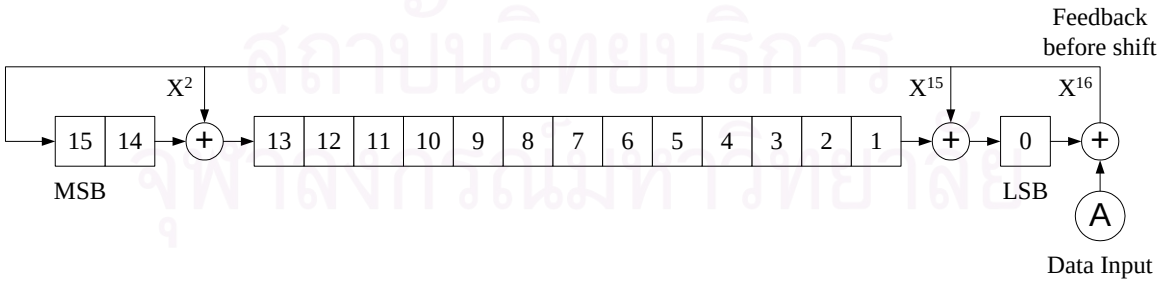
```
ARCHITECTURE behv OF dpram_32X4 IS
        type twoDarray is aray(0 to 31) of std_logic_vector (3 downto 0);
        signal mem      : twoDarray ;
BEGIN
        dwrite: process (WEN,ADDRI,DATAI)
                variable write_address : integer;
        begin
                write_address :=  conv_integer(ADDRI);
                if(WEN = '0') then
                        mem(write_address) <= DATAI;
                end if;
        end process dwrite;
```

```
        dread: process (OEN,ADDRO,mem)
        variable read_address : integer;
        begin
                read_address :=  CONV_INTEGER(ADDRO);
                if(OEN = '0') then
                        DATAO <= mem(read_address) ;
                else
                        DATAO <= "ZZZZ";
                end if;
        end process dread;
END behv;
```

3. HDLC

ทำหน้าที่จัดข้อมูลให้สอดคล้องกับโพรโทคอล AX.25 ทั้งในส่วนของการจัดเฟรม, การใส่แฟล็ก, การเติมและลบบิต ‘0’ และการตรวจสอบความผิดพลาดด้วย CRC16

HDLC_TX จะเริ่มต้นส่งเฟรมข้อมูลเมื่อ FIFO_TX เก็บข้อมูลเต็มหน่วยความจำ 32 ไบต์ (full = ‘1’) หรือสัญญาณ dtr = ‘0’ และ rts = ‘1’ โดยจะเริ่มต้นเฟรมด้วยการส่งฟิลด์แฟล็ก “01111110” แล้วตามด้วยฟิลด์ที่อยู่ (1 ไบต์), ฟิลด์ควบคุม (1 ไบต์), ฟิลด์ข่าวสาร (เท่ากับจำนวนข้อมูลใน FIFO_TX), ฟิลด์ FCS (2 ไบต์) และปิดท้ายด้วยฟิลด์แฟล็กอีกครั้ง ระหว่างฟิลด์แฟล็กเริ่มต้นและฟิลด์แฟล็กปิดท้าย หากมีข้อมูล ‘1’ ติดกัน 5 ตัว จะทำการแทรกบิต ‘0’ คั่นไว้ 1 บิต เพื่อป้องกันไม่ให้ข้อมูลมีค่าซ้ำกับฟิลด์แฟล็ก ข้อมูลที่ได้จาก HDLC_TX จะส่งไปยัง NRZI_TX เพื่อเข้ารหัสสัญญาณแบบ NRZI ต่อไป



รูปที่ 4.12 วงจรคำนวณค่า CRC16 ที่ใช้ใน HDLC มีสมการคือ $X^{16}+X^{15}+X^{2}+1$

ในส่วนของ HDLC_RX เมื่อได้รับข้อมูลที่ถอดรหัสสัญญาณ NRZI จาก NRZI_RX แล้ว ถ้าอยู่ในสถานะ IDLE ก็จะตรวจดูว่าพบฟีลด์แฟล็กหรือไม่ ถ้าพบฟีลด์แฟล็กก็จะเริ่มต้นถอดเฟรมข้อมูลที่ได้รับ โดยแยกข้อมูลของฟีลด์ที่อยู่, ฟีลด์ควบคุม และฟีลด์ข่าวสารออกจากกัน โดยข้อมูลในฟีลด์ข่าวสารจะถูกส่งไปเก็บไว้ที่ FIFO_RX โดยขณะที่ถอดข้อมูลก็จะทำการตรวจและลบบิต '0' ที่แทรกมาพร้อมคำนวณค่า CRC16 ของข้อมูลที่ได้รับ เพื่อตรวจสอบกับค่า CRC16 ในฟีลด์ FCS ว่าตรงกันหรือไม่ ถ้าไม่ตรงกันก็จะแสดงสถานะเฟรมข้อมูลผิดพลาดที่เอาต์พุต FrameError

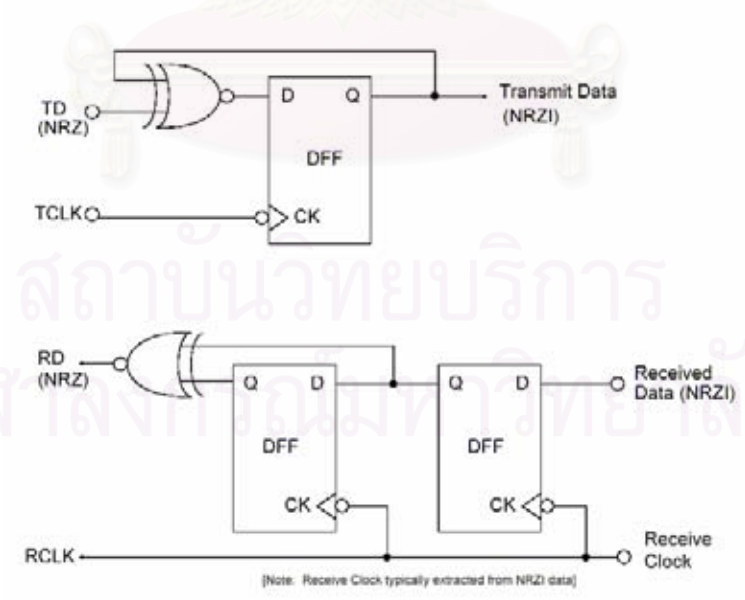
ตัวอย่างที่ 4.2 การเขียน FUNCTION ในภาษา VHDL สำหรับคำนวณ CRC16 ที่มีสมการเป็น $X^{16}+X^{15}+X^{2}+1$

```
FUNCTION nextCRC16_D1 (
        Data:   std_logic;
        CRC:    std_logic_vector(15 downto 0))
RETURN std_logic_vector IS

        variable D:             std_logic_vector(0 downto 0);
        variable C:             std_logic_vector(15 downto 0);
        variable NewCRC:        std_logic_vector(15 downto 0);
BEGIN
        D(0) := Data;
        C := CRC;
        NewCRC(0) := D(0) xor C(15);
        NewCRC(1) := C(0);
        NewCRC(2) := D(0) xor C(1) xor C(15);
        NewCRC(3) := C(2);
        NewCRC(4) := C(3);
        NewCRC(5) := C(4);
        NewCRC(6) := C(5);
        NewCRC(7) := C(6);
        NewCRC(8) := C(7);
        NewCRC(9) := C(8);
```

```
        NewCRC(10) := C(9);
        NewCRC(11) := C(10);
        NewCRC(12) := C(11);
        NewCRC(13) := C(12);
        NewCRC(14) := C(13);
        NewCRC(15) := D(0) xor C(14) xor C(15);
        return NewCRC;
END nextCRC16_D1;
```

4. NRZI

เนื่องข้อมูลที่ผ่าน HDLC มีโอกาสที่จะเป็น '0' ติดกันได้หลายบิตซึ่งไม่เหมาะที่จะส่งออกไป เนื่องจากอาจทำให้การเข้าจังหวะข้อมูลระหว่างภาครับและภาคส่งเกิดความผิดพลาดขึ้นได้ง่าย จึงต้องหาวิธีการมอดูเลตข้อมูลเพื่อไม่ให้ข้อมูลมีค่าซ้ำกันติดกันมากเกินไป วิธีหนึ่งก็คือ การมอดูเลตแบบ NRZI (None return to zero invert) โดยทั้งภาครับและภาคส่งใช้ D-flipflop และ Xnor เพียง 1-2 ตัวเท่านั้น ถ้าข้อมูลอินพุต NRZ (None return to zero) เป็น '1' เอาต์พุต NRZI จะไม่เปลี่ยนแปลง แต่ถ้าอินพุต NRZ เป็น '0' เอาต์พุต NRZI จะกลับลอจิกจากค่าเดิม เป็นการรับประกันว่า เมื่อใช้ร่วมกับ HDLC เอาต์พุต NRZI ที่ได้จะมีค่าซ้ำติดกันได้ไม่เกิน 6 ตัว (เท่ากับจำนวน '1' ของฟิลด์แฟล็ก)



รูปที่ 4.13 วงจรแปลงข้อมูลระหว่าง NRZ และ NRZI

ตัวอย่างที่ 4.3 การเขียน ARCHITECTURE ในภาษา VHDL สำหรับการเข้ารหัสสัญญาณแบบ NRZI

```
ARCHITECTURE spec OF Nrzi_Tx IS
        signal nrzi_sdo : std_logic;
BEGIN
        process (nrzi_tx_en,nrzi_tx_clk,hdlc_tx_sdo)
        begin
                if (nrzi_tx_en = '0') then
                        nrzi_sdo <= '0';
                elsif (nrzi_tx_clk'event and nrzi_tx_clk = '1') then
                        nrzi_sdo <= not(nrzi_sdo xor hdlc_tx_sdo);
                end if;
        end process;
        sdo <= nrzi_sdo;
END spec;
```

ตัวอย่างที่ 4.3 การเขียน ARCHITECTURE ในภาษา VHDL สำหรับการถอดรหัสสัญญาณแบบ NRZI

```
ARCHITECTURE spec OF Nrzi_Rx IS
        signal dff_sdi1 : std_logic;
        signal dff_sdi2 : std_logic;
BEGIN
        d_ff: process (car_det,clk1x)
        begin
                if (car_det = '0') then
                        dff_sdi2 <= '0';
                        dff_sdi1 <= '0';
                elsif (clk1x'event and clk1x = '1') then
                        dff_sdi2 <= dff_sdi1;
                        dff_sdi1 <= sdi;
                end if;
```

```
        end process;
        hdlc_rx_sdi <= not(dff_sdi2 xor dff_sdi1);
END spec;
```

จุดหนึ่งที่ต้องกล่าวถึงสำหรับการทำงานของตัวควบคุมเทอร์มินัลโนดที่ออกแบบนี้ก็คือ การควบคุมการส่งฟิลด์ที่อยู่, ฟิลด์ควบคุม และฟิลด์ข่าวสาร รวมถึงการสั่งให้ตัวควบคุมเทอร์มินัลโนดเริ่มทำการส่งข้อมูลออกไป

การควบคุมที่กล่าวมานี้ทำได้โดยการควบคุมด้วยสัญญาณ dtr และ rts ซึ่งเป็นสัญญาณเอาต์พุตจากพอร์ต RS-232 เมื่อส่งข้อมูลให้กับตัวควบคุมเทอร์มินัลโนดทางพอร์ต RS-232 ในขณะที่สัญญาณ dtr และ rts เป็น ‘0’ ทั้งคู่ จะเป็นการส่งข้อมูลในฟิลด์ข่าวสาร ถ้า dtr และ rts เป็น ‘1’ ทั้งคู่ จะเป็นการส่งข้อมูลในฟิลด์ที่อยู่ ถ้า dtr เป็น ‘1’ และ rts เป็น ‘0’ จะเป็นการส่งข้อมูลให้กับฟิลด์ควบคุม แต่ถ้า dtr เป็น ‘0’ และ rts เป็น ‘1’ จะเป็นการหยุดรับข้อมูลทางพอร์ต RS-232 และสั่งให้ตัวควบคุมเทอร์มินัลโนดเริ่มส่งข้อมูลที่อยู่ในหน่วยความจำข้อมูลชั่วคราว (FIFO)

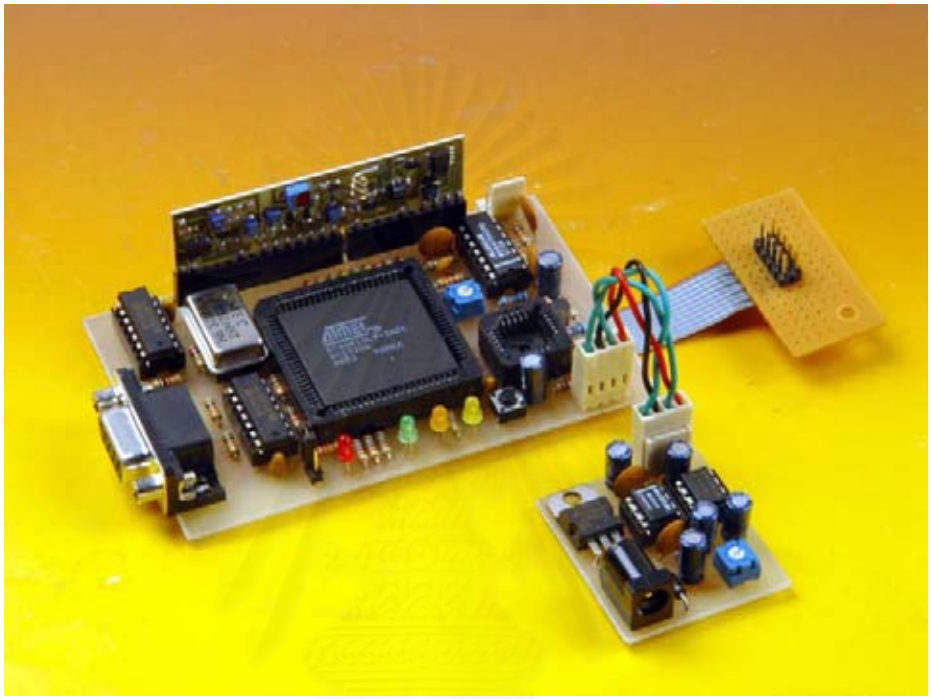
ขั้นตอน	ซอฟต์แวร์ที่ใช้	บริษัทผู้ผลิต
Design Entry	HDL Designer Series 2001.1	Mentor Graphics
Simulation	ModelSim SE 5.5a	Mentor Graphics
Synthesis	LeonardoSpectrum v2001.1a	Mentor Graphics
Place and Route	Atmel Figaro 7.2	Cadence/Atmel
Configuration	Atmel Figaro 7.2	Cadence/Atmel

ตารางที่ 4.7 ซอฟต์แวร์ที่ใช้ออกแบบและพัฒนาวงจรบน FPGA ในวิทยานิพนธ์นี้

บทที่ 5

การทดสอบการทำงาน

5.1 บอร์ดต้นแบบ



รูปที่ 5.1 บอร์ดวงจรต้นแบบของตัวควบคุมเทอร์มินัลโนดที่สร้างขึ้น

เมื่อได้ทำการออกแบบส่วนควบคุมของตัวควบคุมเทอร์มินัลโนดในส่วนที่ใช้ FPGA ด้วยภาษา VHDL ดังรายละเอียดในภาคผนวก ก และจำลองการทำงานบนซอฟต์แวร์ได้ผลเป็นที่เรียบร้อยแล้ว ดังผลในภาคผนวก ข ได้นำการออกแบบที่ได้ไปสังเคราะห์วงจรทั้งหมด แล้วโปรแกรมลงไปในชิป FPGA เบอร์ AT40K10LV-3AJC ที่อยู่บนบอร์ดต้นแบบดังรูปที่ 5.1

5.2 การจำลองการทำงานด้วยซอฟต์แวร์

จากการทดลองด้วยซอฟต์แวร์ โดยทดลองส่งข้อมูลจากพอร์ต RS-232 ทั้ง 3 ชนิด คือข้อมูลฟิลด์ที่อยู่, ข้อมูลฟิลด์ควบคุม และข้อมูลฟิลด์ข่าวสาร โดยการควบคุมด้วยสัญญาณ dtr และ rts ทำการป้อนสัญญาณที่ได้ย้อนกลับมาเพื่อทดสอบการทำงานของวงจรภาครับด้วย เพื่อดูผลว่าตัวควบคุมเทอร์มินัลโนดที่ออกแบบทำงานได้ถูกต้องหรือไม่

ผลการจำลองการทำงานตัวควบคุมเทอร์มินัลโนดที่ออกแบบไว้ด้วยซอฟต์แวร์ ปรากฏว่าให้ผลตรงตามที่ออกแบบไว้ คือสามารถรับข้อมูลจากพอร์ตอนุกรม RS-232 ในอัตราการรับส่งข้อมูลที่ 19,200 บิตต่อวินาที โดยข้อมูลจะถูกเก็บอยู่ในหน่วยความจำข้อมูลชั่วคราว จนกระทั่งหน่วยความจำเต็ม คือมีข้อมูลส่งมาครบ 32 ไบต์ หรือสัญญาณ dtr = '0' และ rts = '1' จะสั่งให้วงจร HDLC ภาคส่งเริ่มทำงาน โดยอ่านข้อมูลข่าวสารจากหน่วยความจำข้อมูล และข้อมูลที่อยู่และรหัสควบคุมจากบัฟเฟอร์ ที่ได้มีการส่งมาให้แล้วก่อนหน้านี้ โดยส่งข้อมูลฟิลด์ต่างๆ ออกไปตามลำดับได้อย่างถูกต้อง ข้อมูลจาก HDLC ในภาคส่งจะถูกส่งต่อไปยังวงจรมอดูเลต NRZI เพื่อเข้ารหัสให้ข้อมูลไม่ซ้ำค่าเดิมเกินกว่า 6 บิต

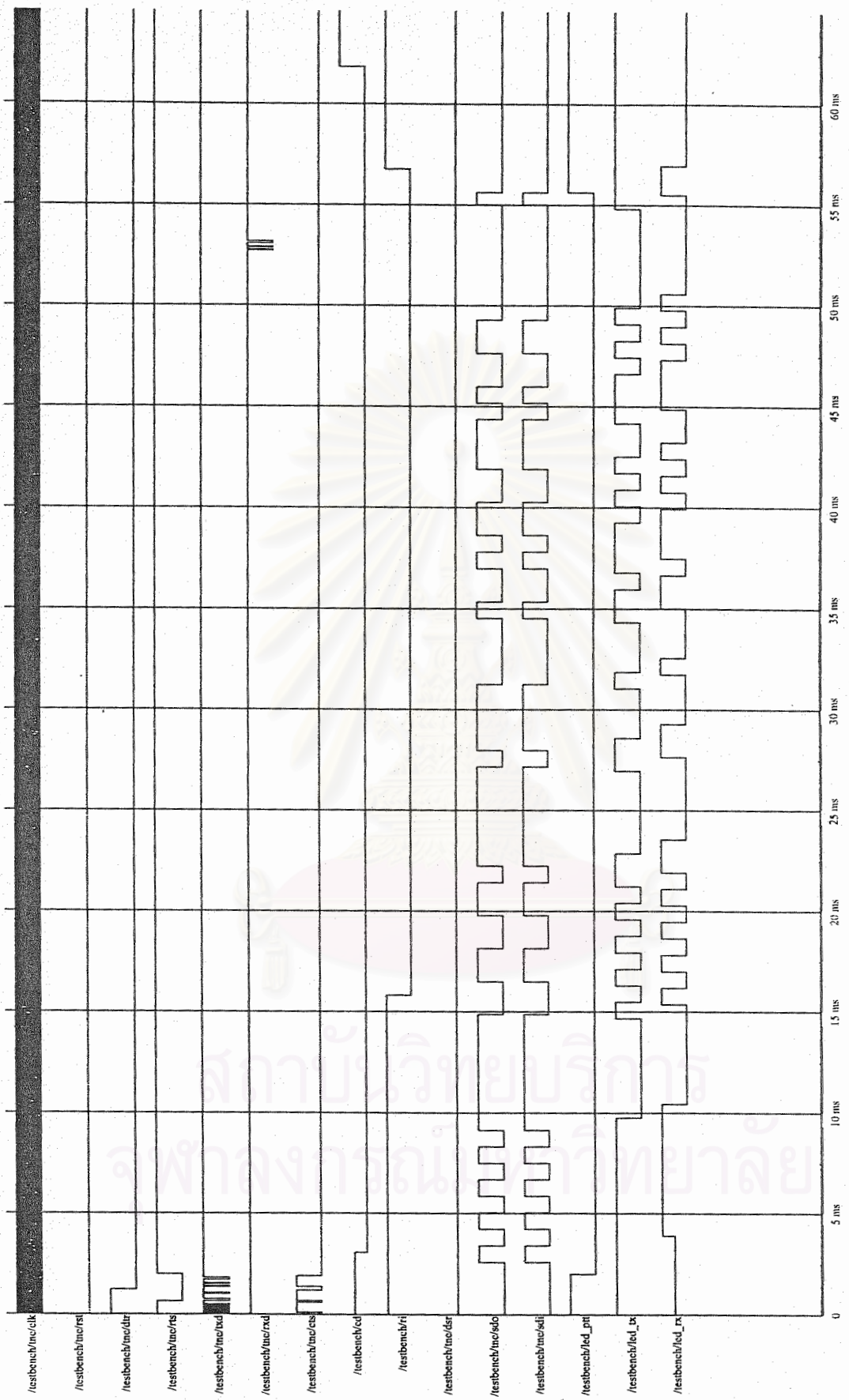
เมื่อต่อสัญญาณที่ส่งออกไปเข้ามาเพื่อทดสอบวงจรภาครับ จากวงจรดีมอดูเลตสัญญาณ NRZI เพื่อแปลงเป็นสัญญาณ NRZ แล้วส่งต่อไปยังวงจร HDLC ภาครับ วงจร HDLC จะทำการตรวจสอบว่าแฟล็กเริ่มต้นเข้ามาหรือยัง เมื่อตรวจสอบพบแฟล็กเริ่มต้น วงจร HDLC จะเริ่มกระบวนการถอดข้อมูลออกจากเฟรม ทั้งข้อมูลที่อยู่, รหัสควบคุม และข้อมูลข่าวสาร พร้อมทั้งตรวจสอบความผิดพลาดของข้อมูลที่ได้รับนี้ด้วย ข้อมูลในฟิลด์ข่าวสารจะถูกส่งเป็นเก็บไว้ในหน่วยความจำข้อมูลภาครับ เพื่อรอให้ UART ภาครับดึงข้อมูล และดำเนินการแปลงข้อมูลเป็นแบบอนุกรมส่งออกไปทางพอร์ต RS-232 ต่อไป

ผลการจำลองการทำงานด้วยซอฟต์แวร์ดูได้จากรูปที่ 5.2-5.4 โดยเป็นการจำลองการส่งค่า "01010101" ให้กับฟิลด์ที่อยู่ (dtr = '1', rts = '1'), ส่งค่า "10111110" ให้กับฟิลด์ควบคุม (dtr = '1', rts = '0') และส่งค่า "00111011" ให้กับฟิลด์ข่าวสาร (dtr = '0', rts = '0') จากนั้นสั่งให้ TNC เริ่มส่งข้อมูลโดยให้ dtr = '0' และ rts = '1' ตามลำดับ

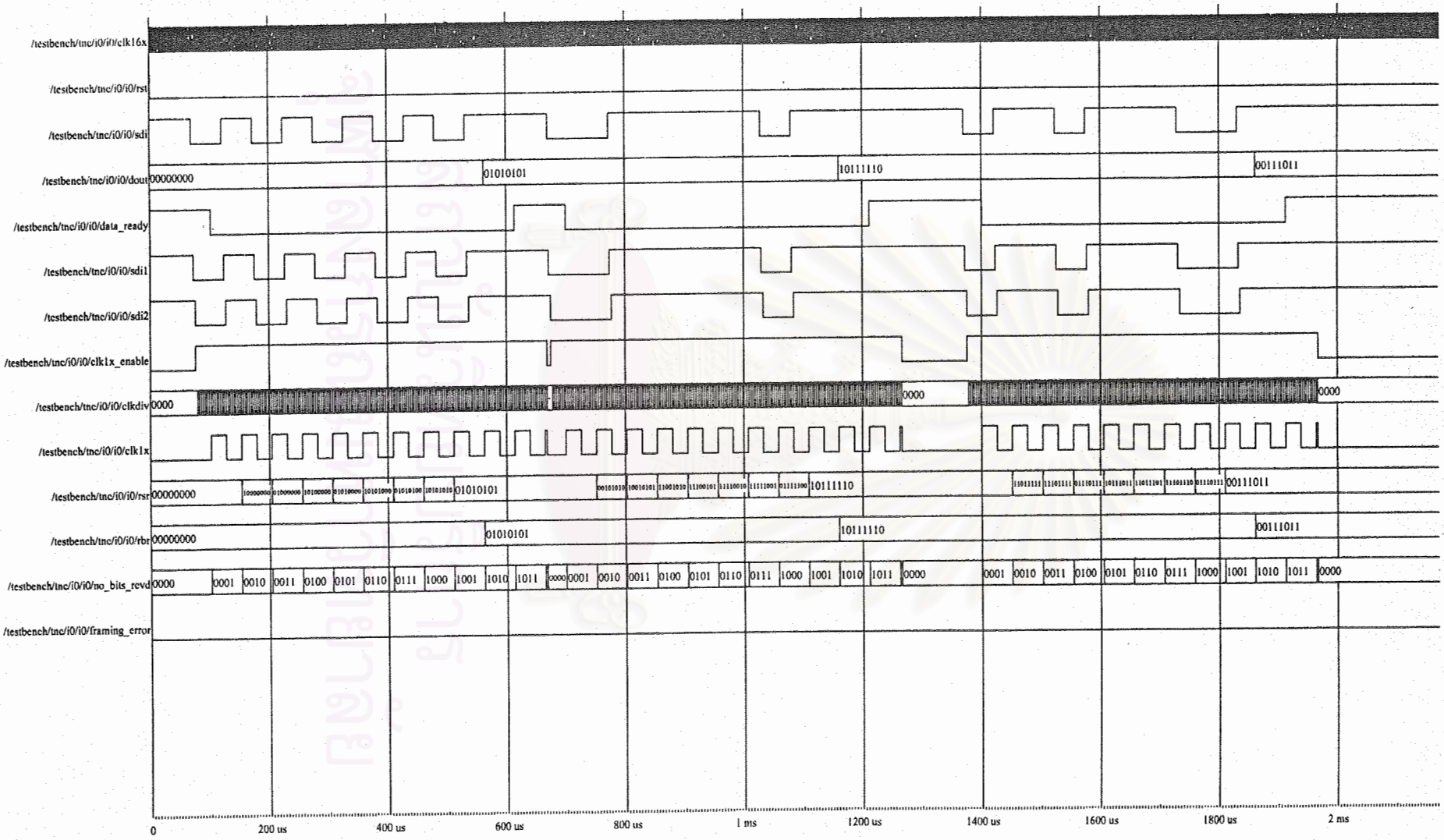
รูปที่ 5.2 เป็นผลการจำลองการทำงานตั้งแต่เริ่มจนจบการทำงาน เริ่มจากส่งสัญญาณอินพุต UART ไปที่ txd สัญญาณจะถูกส่งผ่านส่วนต่างๆ จนกระทั่งสุดท้าย ข้อมูลข่าวสารที่ส่งไป (ในที่นี้คือ "00111011") จะถูกส่งกลับมา โดยออกมาที่ขา rxd จากรูปที่ 5.4 ซึ่งเป็นภาพขยายของรูปที่ 5.2 ในส่วนการทำงานของ UART_RX สังเกตว่าสัญญาณ rxd ที่ส่งออกมาจะมีค่า "00111011" ซึ่งถูกต้อง

รูปที่ 5.3 เป็นภาพขยายผลการทำงานในรูปที่ 5.2 ในส่วนของการทำงานของ UART_TX สังเกตว่าเอาต์พุต dout ที่ได้จะมีค่าเดียวกันกับอินพุตที่ป้อนที่ขา sdi (ซึ่งเป็นขาเดียวกับขา txd ใน Top level) โดยจะให้เอาต์พุต dout ออกมาในช่วงบิตหยุดของเฟรมนั้น ในขณะที่สัญญาณ data_ready จะมีค่าเป็น '0' ในช่วงที่มีการรับสัญญาณข้อมูลที่ขา sdi

รูปที่ 5.4 เป็นภาพขยายผลการทำงานในรูปที่ 5.2 ในส่วนของการทำงานของ UART_RX สังเกตว่าเมื่อมีสัญญาณ wrn เกิดขึ้น UART_RX เริ่มส่งสัญญาณเอาต์พุตออกไปที่ขา sdo (ซึ่งเป็นขาเดียวกับ rxd ใน Top level) โดยนำข้อมูลอินพุต din มาทำการเลื่อนบิตเริ่มจากบิต 0 ไปจนถึงบิต 7 โดยที่มีบิตเริ่มต้นเป็น '0' และมีบิตหยุดเป็น '1' ในขณะที่สัญญาณ buf_ready จะมีค่าเป็น '0' ในช่วงที่มีการส่งสัญญาณข้อมูลออกไปที่ขา sdo

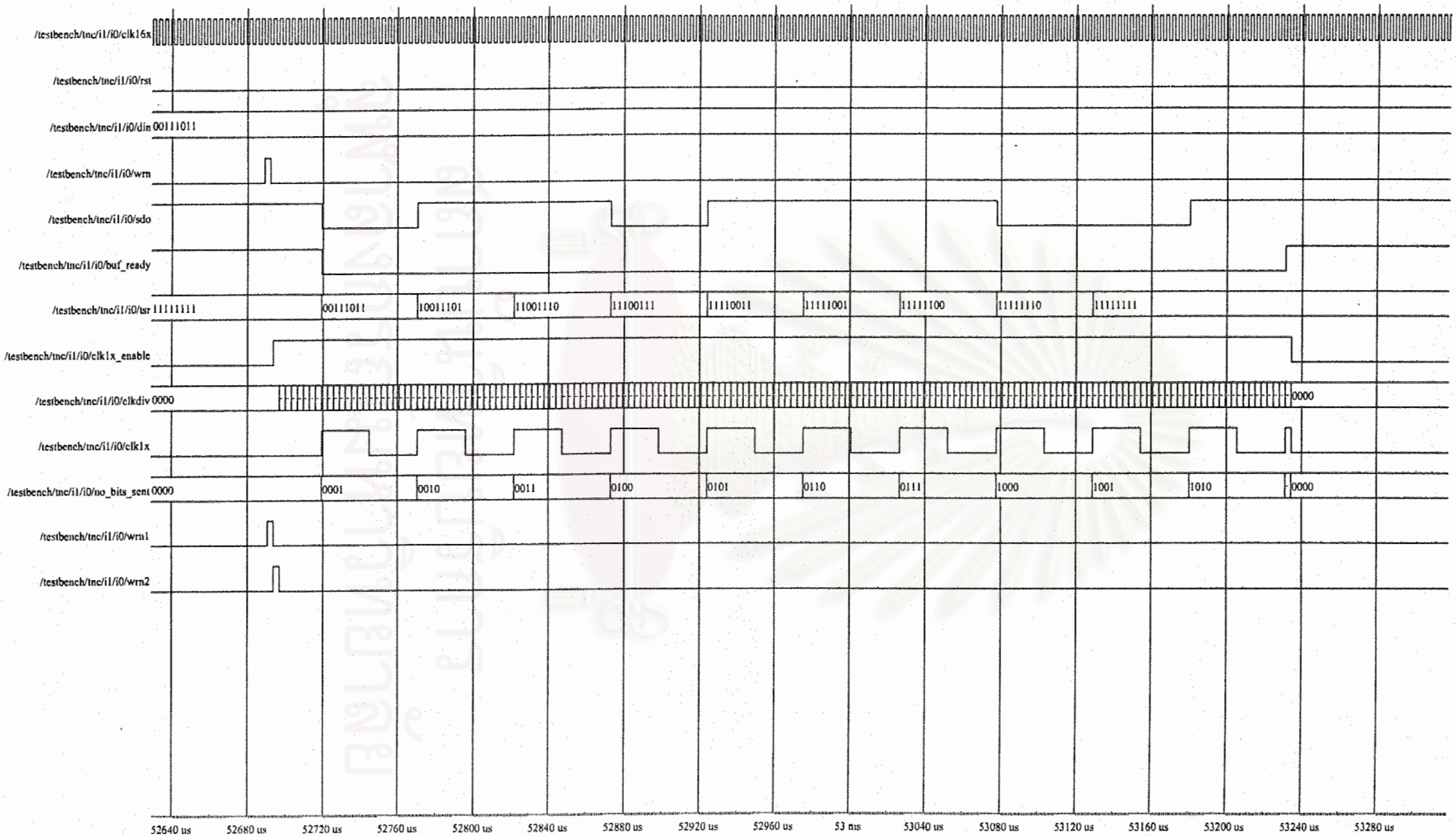


รูปที่ 5.2 ผลการจำลองการทำงานของ TNC ในส่วนบนสุด (Top-level)



Entity:testbench Architecture:behavior Date: Fri May 16 02:58:43 SE Asia Standard Time 2003 Row: 1 Page: 1

รูปที่ 5.3 การเขียนข้อมูลลงในโมดูล UART_TX



รูปที่ 5.4 การจำลองการทำงานของโมดูลภาครับ UART_RX

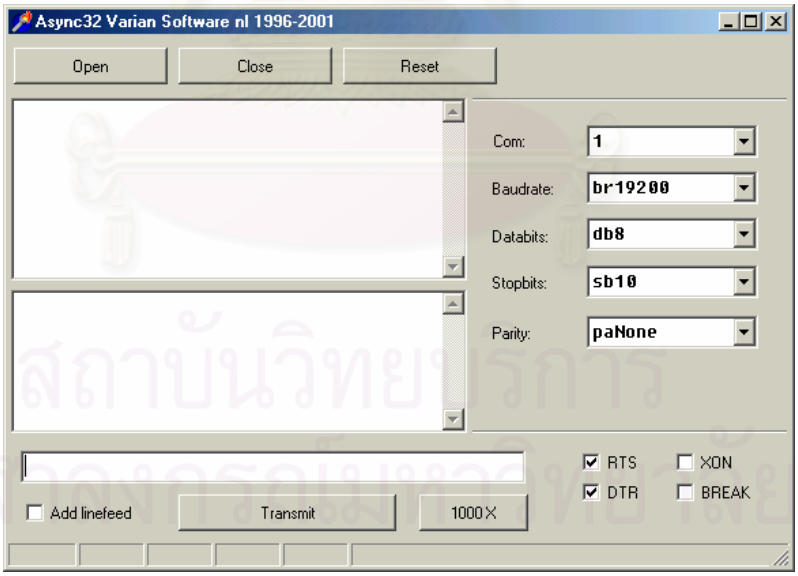
Entity:testbench Architecture:behavior Date: Fri May 16 02:59:24 SE Asia Standard Time 2003 Row: 1 Page: 1

5.3 การทดสอบการทำงานของบอร์ดต้นแบบ

จากการทดสอบสังเคราะห์วงจรที่ได้จากการออกแบบและผ่านการจำลองการทำงานด้วยซอฟต์แวร์แล้ว พบว่าวงจรที่ได้มีขนาดใหญ่เกินกว่าที่จะบรรจุลงใน FPGA เบอร์ที่เลือกใช้ได้ จึงได้พยายามลดฟังก์ชันการทำงานบางอย่างลง เช่น ขนาดฟิลด์ที่อยู่ และขนาดหน่วยความจำข้อมูลชั่วคราว (FIFO) เพื่อให้สามารถทดสอบวงจรที่ออกแบบและสังเคราะห์ได้

ตัวควบคุมเทอร์มินัลโนดที่พัฒนาขึ้นนี้ไม่สามารถทำงานได้ครบทุกฟังก์ชันการทำงานที่ได้ออกแบบไว้ เนื่องจากวงจรที่สังเคราะห์ได้จากการออกแบบมีขนาดใหญ่กว่าความจุเกตของ FPGA เบอร์ที่ใช้ เมื่อพยายามสังเคราะห์วงจรทั้งหมดในคราวเดียว รวมทั้งมีปัญหาในเรื่องผลของค่าความหน่วงเวลาในการทำงานของเกตและบัสสัญญาณภายใน ทำให้วงจรที่สังเคราะห์ได้เมื่อนำไปใช้งานจริงทำงานได้ไม่ถูกต้อง จึงลดขนาดของวงจรลง โดยทดลองได้เพียงการสังเคราะห์บางฟังก์ชันการทำงานลงไปใน FPGA

การทดสอบการทำงานของตัวควบคุมเทอร์มินัลโนดที่สร้างขึ้น จะทดสอบโดยใช้โปรแกรมดังรูปที่ 5.5 ซึ่งเป็นโปรแกรมสื่อสารข้อมูลทางพอร์ต RS-232 ที่มีสามารถกำหนดค่าของสัญญาณ dtr และ rts ได้ตามต้องการ และยังสามารถแสดงค่าสถานะของสัญญาณ cts, dsr, cd และ ri ได้ด้วย จึงเหมาะที่จะนำมาทดสอบกับตัวควบคุมเทอร์มินัลโนดนี้



รูปที่ 5.5 โปรแกรมที่ใช้ทดสอบการทำงานของตัวควบคุมเทอร์มินัลโนดที่สร้างขึ้น

ผลการทดสอบพบว่า ตัวควบคุมเทอร์มินัลโนดสามารถรับรู้ว่ามีข้อมูลจากพอร์ต RS-232 เข้ามา โดยสัญญาณ cts จะเป็น ‘1’ เมื่อมีการส่งข้อมูล RS-232 มาที่ตัวควบคุมเทอร์มินัลโนด และเมื่อส่งข้อมูลครบมาครบ 32 ไบต์ หรือสัญญาณ dtr = ‘0’ และ rts =’1’ ตัวควบคุมจะทำการส่งข้อมูลออกไป

สัญญาณ cd สามารถแสดงสถานะของช่องสัญญาณได้ถูกต้อง คือเมื่อมีสัญญาณ ‘1’ ที่ขา sdi cd จะมีค่าเป็น ‘0’ เพื่อแสดงว่ามีการใช้ช่องสัญญาณอยู่

สัญญาณ rst หรือสัญญาณรีเซตทำงานได้ถูกต้อง คือเมื่อกดปุ่มรีเซต (rst = ‘1’) วงจรตัวควบคุมเทอร์มินัลโนดจะอยู่ในสถานะเริ่มต้น

บทที่ 6

บทสรุป

6.1 สรุปผลการวิจัย

ผู้วิจัยได้ศึกษาโพรโทคอล AX.25 ซึ่งใช้ในเครือข่ายวิทยุกลุ่มข้อมูล, ออกแบบและสร้างตัวควบคุมเทอร์มินัลโนดเพื่อใช้ในเครือข่ายวิทยุกลุ่มข้อมูล ที่สามารถใช้งานร่วมกับคอมพิวเตอร์โดยการเชื่อมต่อทางพอร์ตอนุกรมได้, จำลองการทำงานของตัวควบคุมเทอร์มินัลโนดที่ออกแบบด้วยซอฟต์แวร์ และทดสอบการทำงานของเครื่องต้นแบบตัวควบคุมเทอร์มินัลโนดที่สร้างขึ้น โดยมีเป้าหมายของวิทยานิพนธ์คือ สร้างบอร์ดต้นแบบของตัวควบคุมเทอร์มินัลโนดขึ้นมาจำนวน 3 บอร์ด แต่ละบอร์ดสามารถสื่อสารกันได้ และทำงานได้อย่างถูกต้องตามมาตรฐาน AX.25 โดยเน้นในส่วนของการทำงานทางด้านฮาร์ดแวร์เป็นหลัก ได้แก่ การจัดแบ่งเฟรมข้อมูล, การแทรกบิตและลบบิต '0', การตรวจสอบความผิดพลาดด้วย CRC รวมถึงการสร้างสัญญาณควบคุม และสัญญาณแสดงสถานะต่างๆ ที่ช่วยอำนวยความสะดวกในการใช้งานตัวควบคุมเทอร์มินัลโนดที่สร้างขึ้น โดยอาศัยการออกแบบส่วนทำงานต่างๆ เหล่านี้ด้วย FPGA

ผลการจำลองการทำงานของตัวควบคุมเทอร์มินัลโนดที่ออกแบบไว้ด้วยซอฟต์แวร์ในส่วนของ FPGA ปรากฏว่าให้ผลตรงตามที่ออกแบบไว้ คือ ในส่วนของภาคส่ง สามารถรับข้อมูลจากพอร์ตอนุกรม RS-232 ในอัตราการรับส่งข้อมูลที่ 19,200 บิตต่อวินาที โดยข้อมูลจะถูกเก็บอยู่ในหน่วยความจำข้อมูลชั่วคราว จนกระทั่งหน่วยความจำเต็ม คือมีข้อมูลส่งมาครบ 32 ไบต์ หรือสัญญาณ DTR = '0' และ RTS = '1' จะสั่งให้วงจร HDLC ภาคส่งเริ่มทำงาน โดยอ่านข้อมูลฟิลด์ข่าวสารจากหน่วยความจำข้อมูล และข้อมูลฟิลด์ที่อยู่และฟิลด์ควบคุมจากบัฟเฟอร์ ที่ได้มีการส่งมาให้แล้วก่อนหน้านี้ โดยส่งข้อมูลฟิลด์ต่างๆ ออกไปตามลำดับได้อย่างถูกต้อง ข้อมูลจาก HDLC ในภาคส่งจะถูกส่งต่อไปยังวงจรแปลงสัญญาณแบบ NRZI เพื่อเข้ารหัสให้ข้อมูลไม่ซ้ำค่าเดิมเกินกว่า 6 บิต

ในส่วนของภาครับ สัญญาณข้อมูลจะผ่านวงจรแปลงกลับสัญญาณ NRZI เพื่อแปลงรหัสสัญญาณ NRZI ที่ได้รับกลับไปเป็นสัญญาณ NRZ แล้วส่งต่อไปยังวงจร HDLC ภาครับ วงจร HDLC จะทำการตรวจสอบว่ามีแฟล็กเริ่มต้นเข้ามาหรือไม่ เมื่อตรวจสอบพบแฟล็กเริ่มต้น วงจร HDLC จะเริ่มกระบวนการถอดข้อมูลจากเฟรม ทั้งข้อมูลรหัสที่อยู่, รหัสควบคุม และข้อมูลข่าวสาร พร้อมทั้งตรวจสอบความผิดพลาดของข้อมูลที่ได้รับนี้ด้วย ข้อมูลในฟิลด์ข่าวสารจะถูกส่งเป็นเก็บไว้ในหน่วยความจำข้อมูลภาครับ เพื่อรอให้ UART ภาครับดึงข้อมูล และดำเนินการแปลงข้อมูลเป็นแบบอนุกรมส่งออกไปทางพอร์ต RS-232 ต่อไป

จุดเด่นของตัวควบคุมเทอร์มินัลโนดที่ออกแบบนี้ คือเป็นการออกแบบโดยรวมส่วนทำงานต่างๆ ทั้งหมดมาไว้บนชิป FPGA เพียงตัวเดียว ทำให้ตัวควบคุมเทอร์มินัลโนดที่ได้มีขนาดเล็ก และมีความเร็วในการทำงานสูง นอกจากนี้ยังช่วยลดเวลาในการออกแบบและพัฒนาตัวควบคุมเทอร์มินัลโนดต้นแบบลง และสามารถแก้ไขหรือเพิ่มเติมฟังก์ชันการทำงานของตัวควบคุมเทอร์มินัลโนดได้ง่ายในอนาคต

6.2 ปัญหาและข้อเสนอแนะ

ปัญหาที่พบคือ ตัวควบคุมเทอร์มินัลโนดที่พัฒนาขึ้นนี้ไม่สามารถทำงานได้ถูกต้องครบทุกฟังก์ชันการทำงานที่ได้ออกแบบไว้ เนื่องจากวงจรที่สังเคราะห์ได้จากการออกแบบมีขนาดใหญ่กว่าความจุเกตของ FPGA เบอร์ที่ใช้ เมื่อพยายามสังเคราะห์วงจรทั้งหมดในคราวเดียว แม้ว่าจะสามารถบรรจุลงใน FPGA ที่ใช้อยู่ได้ แต่ก็มีผลของค่าความหน่วงเวลาในการทำงานของเกตและบัสสัญญาณภายใน ทำให้วงจรที่สังเคราะห์ได้เมื่อนำไปใช้งานจริงทำงานได้ไม่ถูกต้อง จึงลดขนาดของวงจรลง โดยทดลองได้เพียงการสังเคราะห์บางฟังก์ชันการทำงานลงไปใน FPGA ทำให้ตัวควบคุมเทอร์มินัลโนดที่ได้ทำงานได้ไม่เต็มประสิทธิภาพ แนวทางการแก้ไข และพัฒนาตัวควบคุมเทอร์มินัลโนดให้ดีขึ้น มีดังนี้

- เพิ่มความจุเกตของ FPGA เพื่อให้สามารถเพิ่มฟังก์ชันการทำงานของตัวควบคุมเทอร์มินัลโนดให้มากขึ้น รวมทั้งเลือกใช้ FPGA ที่มีความเร็วในการทำงานที่สูงขึ้น
- เพิ่มขนาดฟิลด์ที่อยู่ให้เป็นไปตามโพรโทคอล AX.25 คือ 14-28 ไบต์
- เพิ่มขนาดฟิลด์ข่าวสารเป็น 256 ไบต์ โดยมี 2 แนวทางในการพัฒนา แนวทางแรกคือ เพิ่มขนาดหน่วยความจำข้อมูลใช้งานภายใน โดยจะต้องเพิ่มขนาดของหน่วยความจำ, ตัวชี้ตำแหน่ง และตัวบอกขนาดข้อมูล นั่นหมายถึงต้องใช้พื้นที่มากขึ้น แนวทางที่ 2 คือ ออกแบบให้ HDLC_TX ตรวจสอบ FIFO ตลอดเวลาว่ามีข้อมูลอยู่หรือไม่ แม้ขณะที่ทำการส่งข้อมูลอยู่ โดยถ้า FIFO ยังมีข้อมูลอยู่ และข้อมูลในฟิลด์ข่าวสารของเฟรมที่กำลังส่งอยู่นั้นยังไม่ถึง 256 ไบต์ ก็ให้ HDLC_TX ส่งข้อมูลที่เหลืออยู่ต่อไปได้เลย ไม่ต้องรอส่งในเฟรมถัดไป หรือรอให้เต็ม FIFO ก่อน
- เพิ่มความสามารถในการเข้ารหัสและถอดรหัสฟิลด์ที่อยู่และฟิลด์ควบคุม เพื่อลดภาระในการทำงานของคอมพิวเตอร์ลง
- เพิ่มฟังก์ชันการติดต่อกับตัวควบคุมเทอร์มินัลโนดโดยใช้คำสั่งผ่านทางพอร์ต RS-232 เพื่อความสะดวกในการนำตัวควบคุมเทอร์มินัลที่ได้ไปใช้งาน สะดวกในการเขียนซอฟต์แวร์แอปพลิเคชันเพื่อใช้งานตัวควบคุมเทอร์มินัล
- พัฒนาให้ใช้งานร่วมกับเครื่องรับส่งวิทยุสมัครเล่นได้ เพื่อการใช้งานที่สะดวกและหลากหลายมากขึ้น

รายการอ้างอิง

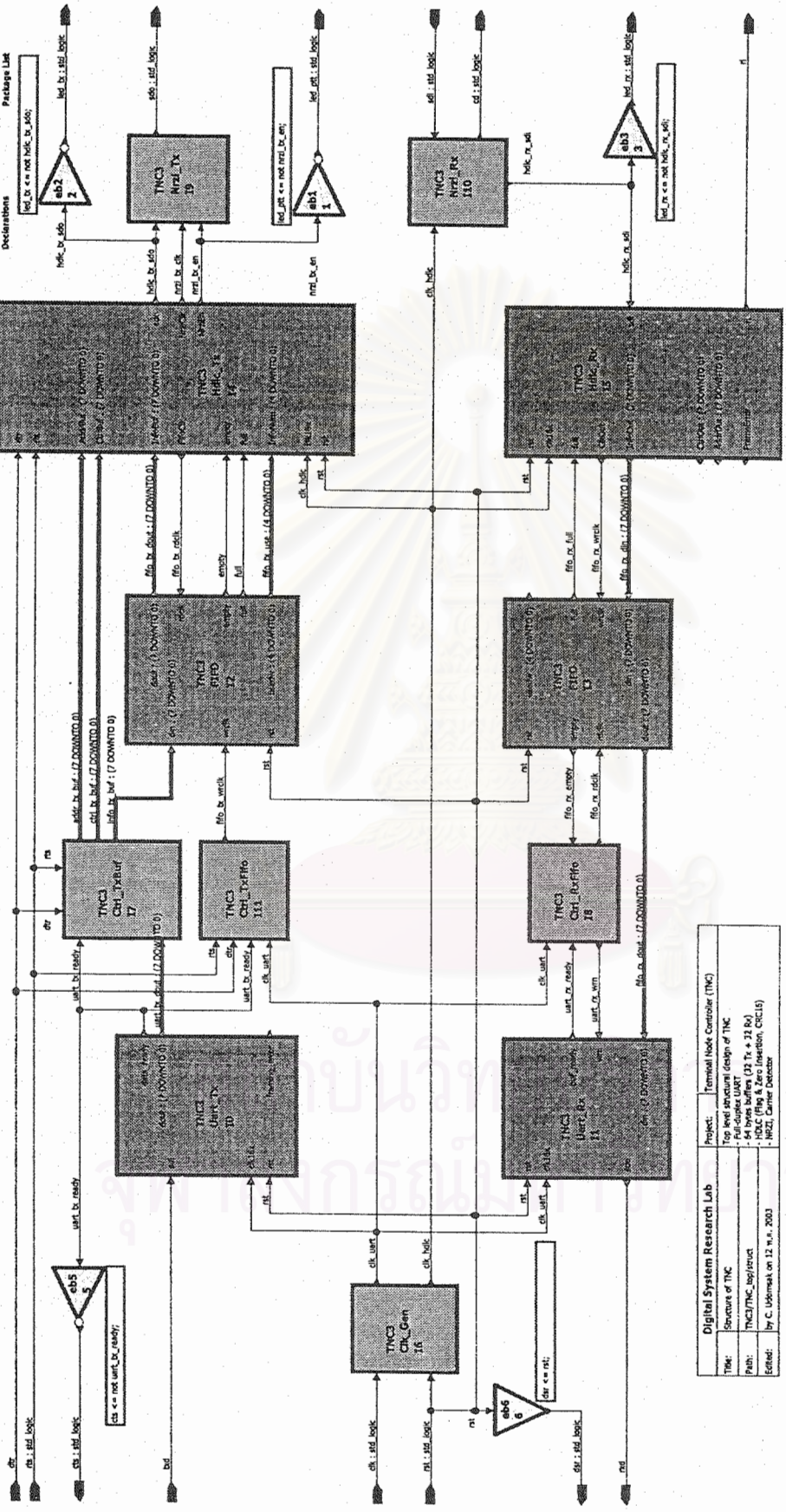
1. Stephen D. Brown ,Zvonko G. Vranesic, Fundamentals of Digital Logic with VHDL Design, McGraw-Hill, 2000.
2. Zainalabedin Navabi, VHDL: Analysis and Modeling of Digital Systems, 2nd Edition, McGraw-Hill, 1998.
3. Terry L. Fox (WB4JFI), AX.25 Amateur Packet-Radio Link-Layer Protocol: Version 2.0, The American Radio Relay League (ARRL), 1984.
4. William A. Beech (NJ7P), Douglas E. Nielsen (N7LEM), Jack Taylor (N7OO), AX.25 Link Access Protocol for Amateur Packet Radio: Version 2.2, Tucson Amateur Packet Radio Corporation (TAPR), 1997.
5. Nico Palermo (IV3NWV), Inside the YAM Modem, www.microlet.com/yam, 1999.
6. Theodore S. Rappaport, Wireless Communications Principles and Practice, Prentice Hall PTR, 1999.
7. William Stallings, Handbook of Computer-Communication Standards Volume 1: The Open Systems Interconnection (OSI) Model and OSI-Related Standards, Macmillan Publishing Company, 1987.
8. William Stallings, Handbook of Computer-Communication Standards Volume 2: Local Network Standards, Howard W. Sams & Company, 1987.
9. Andrew S. Tanenbaum, Computer Network, Prentice-Hall, 1989.
10. Edward N. Singer, Land Mobile Radio Systems, Prentice-Hall, 1994.
11. Intel, Microprocessors and Peripherals Volume 1, Intel Corporation, Santa clara, California, 1985.
12. Atmel, AT40K FPGAs with FreeRAM™, Atmel Corporation, 1998.
13. Atmel, AT40K Series Configuration, Atmel Corporation, 1998.
14. สมลักษณ์ วันโย (HS4DOR), ประวัติความเป็นมาของแพ็คเก็ตเรดิโอ, ชมรมแพ็คเก็ตเรดิโอแห่งประเทศไทย (PGOT), 2543.
15. 100 วัตต์ ฉบับที่ 70, สำนักงานนิตยสาร 100 วัตต์, กุมภาพันธ์ 2544.
16. รวมบทความและโครงงานวิทยุสมัครเล่น เล่มที่ 2, ซีเอ็ดยูเคชั่น, 2539.

17. สมบัติ สิริพัฒนากุล, กฤษดา วิศวธีรานนท์, ตัวควบคุมเทอร์มินัลโนดสำหรับเครือข่ายวิทยุกลุ่มข้อมูล, วิทยานิพนธ์ปริญญาวิศวกรรมศาสตรมหาบัณฑิต จุฬาลงกรณ์มหาวิทยาลัย, 2538.
18. ฝ่ายออกแบบวงจรรวม, FPGA Design Workshop, ศูนย์วิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์ (TMEC).
19. ประสิทธิ์ ทีฆพุติ, โครงข่ายบริการสื่อสารร่วมระบบดิจิตอล, วิศวกรรมสถานแห่งประเทศไทยในพระบรมราชูปถัมภ์, 2535.
20. วาทิต เบญจพลกุล, การสื่อสารข้อมูล, ไฮ-เทค การพิมพ์, 2543.

ภาคผนวก

ภาคผนวก ก

รายละเอียดของภาษา VHDL ที่ใช้ออกแบบตัวควบคุมเทอร์มินัลโนด



โครงสร้างของวงจรภายใน FPGA

```
----------------------------------------------------------------------------------------------------------------------------
---                                          TNC Top Level Structure                                                    ---
----------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY TNC_top IS
PORT(     clk       : IN     std_logic;
          dtr       : IN     std_logic;
          rst       : IN     std_logic;
          rts       : IN     std_logic;
          sdi       : IN     std_logic;
          txd       : IN     std_logic;
          cd        : OUT    std_logic;
          cts       : OUT    std_logic;
          dsr       : OUT    std_logic;
          led_ptt   : OUT    std_logic;
          led_rx    : OUT    std_logic;
           led_tx   : OUT    std_logic;
          ri        : OUT    std_logic;
          rxd       : OUT    std_logic;
          sdo       : OUT    std_logic);
END TNC_top ;

ARCHITECTURE struct OF TNC_top IS
          SIGNAL addr_tx_buf          : std_logic_vector(7 DOWNTO 0);
          SIGNAL clk_hdlc             : std_logic;
          SIGNAL clk_uart             : std_logic;
          SIGNAL ctrl_tx_buf          : std_logic_vector(7 DOWNTO 0);
          SIGNAL empty                : std_logic;
          SIGNAL fifo_rx_din          : std_logic_vector(7 DOWNTO 0);
          SIGNAL fifo_rx_dout         : std_logic_vector(7 DOWNTO 0);
          SIGNAL fifo_rx_empty        : std_logic;
          SIGNAL fifo_rx_full         : std_logic;
          SIGNAL fifo_rx_rdclk        : std_logic;
          SIGNAL fifo_rx_wrclk        : std_logic;
          SIGNAL fifo_tx_dout         : std_logic_vector(7 DOWNTO 0);
          SIGNAL fifo_tx_rdclk        : std_logic;
          SIGNAL fifo_tx_use          : std_logic_vector(4 DOWNTO 0);
          SIGNAL fifo_tx_wrclk        : std_logic;
          SIGNAL full                 : std_logic;
          SIGNAL hdlc_rx_sdi          : std_logic;
          SIGNAL hdlc_tx_sdo          : std_logic;
          SIGNAL info_tx_buf          : std_logic_vector(7 DOWNTO 0);
          SIGNAL nrzi_tx_clk          : std_logic;
          SIGNAL nrzi_tx_en           : std_logic;
          SIGNAL uart_rx_ready        : std_logic;
          SIGNAL uart_rx_wrn          : std_logic;
          SIGNAL uart_tx_dout         : std_logic_vector(7 DOWNTO 0);
          SIGNAL uart_tx_ready        : std_logic;

          COMPONENT Clk_Gen
          PORT (    clk       : IN     std_logic ;
                    rst       : IN     std_logic ;
                    clk_hdlc  : OUT    std_logic ;
```

```
            clk_uart    : OUT     std_logic );
END COMPONENT;

COMPONENT Ctrl_RxFifo
PORT (      clk_uart              : IN      std_logic ;
            fifo_rx_empty         : IN      std_logic ;
            uart_rx_ready         : IN      std_logic ;
            fifo_rx_rdclk         : OUT     std_logic ;
            uart_rx_wrn           : OUT     std_logic );
END COMPONENT;

COMPONENT Ctrl_TxBuf
PORT (      dtr                   : IN      std_logic ;
            rts                   : IN      std_logic ;
            uart_tx_dout          : IN      std_logic_vector (7 DOWNTO 0);
            uart_tx_ready         : IN      std_logic ;
            addr_tx_buf           : OUT     std_logic_vector (7 DOWNTO 0);
            ctrl_tx_buf           : OUT     std_logic_vector (7 DOWNTO 0);
            info_tx_buf           : OUT     std_logic_vector (7 DOWNTO 0) );
END COMPONENT;

COMPONENT Ctrl_TxFifo
PORT (      clk_uart              : IN      std_logic ;
            dtr                   : IN      std_logic ;
            rts                   : IN      std_logic ;
            uart_tx_ready         : IN      std_logic ;
            fifo_tx_wrclk         : OUT     std_logic );
END COMPONENT;

COMPONENT FIFO
PORT (      din         : IN      std_logic_vector (7 DOWNTO 0);
            rdclk       : IN      std_logic ;
            rst         : IN      std_logic ;
            wrclk       : IN      std_logic ;
            dout        : OUT     std_logic_vector (7 DOWNTO 0);
            empty       : OUT     std_logic ;
            full        : OUT     std_logic ;
            usedw       : OUT     std_logic_vector (4 DOWNTO 0));
END COMPONENT;

COMPONENT Hdlc_Rx
PORT (      clk16x                : IN      std_logic ;
            full                  : IN      std_logic ;
            rst                   : IN      std_logic ;
            di                    : IN      std_logic ;
            AddrOut               : OUT     std_logic_vector (7 DOWNTO 0);
            ClkOut                : OUT     std_logic ;
            CtrlOut               : OUT     std_logic_vector (7 DOWNTO 0);
            FrameError            : OUT     std_logic ;
            InfoOut               : OUT     std_logic_vector (7 DOWNTO 0);
            ri                    : OUT     std_logic );
END COMPONENT;

COMPONENT Hdlc_Tx
PORT (      AddrBuf               : IN      std_logic_vector (7 DOWNTO 0);
            CtrlBuf               : IN      std_logic_vector (7 DOWNTO 0);
            InfoBuf               : IN      std_logic_vector (7 DOWNTO 0);
```

```
                InfoNum         : IN    std_logic_vector (4 DOWNTO 0);
                clk16x          : IN    std_logic ;
                dtr             : IN    std_logic ;
                empty           : IN    std_logic ;
                full            : IN    std_logic ;
                rst             : IN    std_logic ;
                rts             : IN    std_logic ;
                FifoClk         : OUT   std_logic ;
                NrziClk         : OUT   std_logic ;
                NrziEn          : OUT   std_logic ;
                sdo             : OUT   std_logic );
        END COMPONENT;

        COMPONENT Nrzi_Rx
        PORT (  clk_hdlc        : IN    std_logic ;
                sdi             : IN    std_logic ;
                cd              : OUT   std_logic ;
                hdlc_rx_sdi     : OUT   std_logic);
        END COMPONENT;


        COMPONENT Nrzi_Tx
        PORT (  hdlc_tx_sdo     : IN    std_logic ;
                nrzi_tx_clk     : IN    std_logic ;
                nrzi_tx_en      : IN    std_logic ;
                sdo             : OUT   std_logic);
        END COMPONENT;

        COMPONENT Uart_Rx
        PORT (  clk16x          : IN    std_logic ;
                din             : IN    std_logic_vector (7 DOWNTO 0);
                rst             : IN    std_logic ;
                wrn             : IN    std_logic ;
                buf_ready       : OUT   std_logic ;
                sdo             : OUT   std_logic );
        END COMPONENT;

        COMPONENT Uart_Tx
        PORT (  clk16x          : IN    std_logic ;
                rst             : IN    std_logic ;
                sdi             : IN    std_logic ;
                data_ready      : OUT   std_logic ;
                dout            : OUT   std_logic_vector (7 DOWNTO 0);
                framing_error   : OUT   std_logic );
        END COMPONENT;

BEGIN
        led_ptt     <= not nrzi_tx_en;
        led_tx      <= not hdlc_tx_sdo;
        led_rx      <= not hdlc_rx_sdi;
        cts         <= not uart_tx_ready;
        dsr         <= rst;

        I6 : Clk_Gen  PORT MAP (
                clk         => clk,
                rst         => rst,
                clk_hdlc    => clk_hdlc,
```

```
        clk_uart    => clk_uart);

I8 : Ctrl_RxFifo PORT MAP (
        clk_uart            => clk_uart,
        fifo_rx_empty       => fifo_rx_empty,
        uart_rx_ready       => uart_rx_ready,
        fifo_rx_rdclk       => fifo_rx_rdclk,
        uart_rx_wrn         => uart_rx_wrn);

I7 : Ctrl_TxBuf PORT MAP (
        dtr                 => dtr,
        rts                 => rts,
        uart_tx_dout        => uart_tx_dout,
        uart_tx_ready       => uart_tx_ready,
        addr_tx_buf         => addr_tx_buf,
        ctrl_tx_buf         => ctrl_tx_buf,
        info_tx_buf         => info_tx_buf );

I11 : Ctrl_TxFifo PORT MAP (
        clk_uart            => clk_uart,
        dtr                 => dtr,
        rts                 => rts,
        uart_tx_ready       => uart_tx_ready,
        fifo_tx_wrclk       => fifo_tx_wrclk );

I2 : FIFO PORT MAP (
        din         => info_tx_buf,
        rdclk       => fifo_tx_rdclk,
        rst         => rst,
        wrclk       => fifo_tx_wrclk,
        dout        => fifo_tx_dout,
        empty       => empty,
        full        => full,
        usedw       => fifo_tx_use );

I3 : FIFO PORT MAP (
        din         => fifo_rx_din,
        rdclk       => fifo_rx_rdclk,
        rst         => rst,
        wrclk       => fifo_rx_wrclk,
        dout        => fifo_rx_dout,
        empty       => fifo_rx_empty,
        full        => fifo_rx_full,
        usedw       => OPEN );

I5 : Hdlc_Rx  PORT MAP (
        clk16x      => clk_hdlc,
        full        => fifo_rx_full,
        rst         => rst,
        sdi         => hdlc_rx_sdi,
        AddrOut     => OPEN,
        ClkOut      => fifo_rx_wrclk,
        CtrlOut     => OPEN,
        FrameError => OPEN,
        InfoOut     => fifo_rx_din,
        ri          => ri );
```

```
I4 : Hdlc_Tx  PORT MAP (
        AddrBuf    => addr_tx_buf,
        CtrlBuf    => ctrl_tx_buf,
        InfoBuf    => fifo_tx_dout,
        InfoNum    => fifo_tx_use,
        clk16x     => clk_hdlc,
        dtr        => dtr,
        empty      => empty,
        full       => full,
        rst        => rst,
        rts        => rts,
        FifoClk    => fifo_tx_rdclk,
        NrziClk    => nrzi_tx_clk,
        NrziEn     => nrzi_tx_en,
        sdo        => hdlc_tx_sdo );

I10 : Nrzi_Rx  PORT MAP (
        clk_hdlc          => clk_hdlc,
        sdi               => sdi,
        cd                => cd,
        hdlc_rx_sdi       => hdlc_rx_sdi );

I9 : Nrzi_Tx  PORT MAP (
        hdlc_tx_sdo       => hdlc_tx_sdo,
        nrzi_tx_clk       => nrzi_tx_clk,
        nrzi_tx_en        => nrzi_tx_en,
        sdo               => sdo );

I1 : Uart_Rx  PORT MAP (
        clk16x     => clk_uart,
        din        => fifo_rx_dout,
        rst        => rst,
        wrn        => uart_rx_wrn,
        buf_ready  => uart_rx_ready,
        sdo        => rxd  );

I0 : Uart_Tx  PORT MAP (
        clk16x            => clk_uart,
        rst               => rst,
        sdi               => txd,
        data_ready        => uart_tx_ready,
        dout              => uart_tx_dout,
        framing_error     => OPEN );

END struct;
```

```
------------------------------------------------------------------------------------------------------------------------
---                                        Clock Generator Subcircuit                                                ---
------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Clk_Gen IS
PORT(       rst         : IN        std_logic;
            clk         : IN        std_logic;
            clk_uart    : OUT       std_logic;
            clk_hdlc    : OUT       std_logic);
END Clk_Gen ;

ARCHITECTURE spec OF Clk_Gen IS
            SIGNAL count_reg    : unsigned(7 downto 0);
            SIGNAL conv_reg     : std_logic_vector(7 downto 0);
BEGIN
            PROCESS (rst,clk)
            BEGIN
                        if (rst = '1') then
                                    count_reg <= (others => '0');
                        elsif (clk'event and clk = '1') then
                                    count_reg <= count_reg + 1;
                        end if;
            END process;
            conv_reg <= std_logic_vector(count_reg);
            clk_uart <= conv_reg(3);
            clk_hdlc <= conv_reg(7);
END spec;
```

```
--------------------------------------------------------------------------------------------------------------------------------
---                                          Control TX Buffer Subcircuit                                                    ---
--------------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY Ctrl_TxBuf IS
PORT(       dtr                 : IN        std_logic;
            rts                 : IN        std_logic;
            uart_tx_ready       : IN        std_logic;
            uart_tx_dout        : IN        std_logic_vector(7 downto 0);
            addr_tx_buf         : OUT       std_logic_vector(7 downto 0);
            ctrl_tx_buf         : OUT       std_logic_vector(7 downto 0);
            info_tx_buf         : OUT       std_logic_vector(7 downto 0));
END Ctrl_TxBuf ;

ARCHITECTURE spec OF Ctrl_TxBuf IS
BEGIN
            PROCESS (uart_tx_ready,dtr,rts)
            BEGIN
                        if (uart_tx_ready'event and uart_tx_ready = '1') then
                                    if (dtr = '0' and rts = '0') then
                                                info_tx_buf <= uart_tx_dout;
                                    elsif (dtr = '1' and rts = '0') then
                                                ctrl_tx_buf <= uart_tx_dout;
                                    elsif (dtr = '1' and rts = '1') then
                                                addr_tx_buf <= uart_tx_dout;
                                    end if;
                        end if;
            END process;
END spec;
```

```
------------------------------------------------------------------------------------------------------------------------
---                                        Control TX Fifo Subcircuit                                                ---
------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY Ctrl_TxFifo IS
PORT(       clk_uart            : IN        std_logic;
            dtr                 : IN        std_logic;
            rts                 : IN        std_logic;
            uart_tx_ready       : IN        std_logic;
            fifo_tx_wrclk       : OUT       std_logic );
END Ctrl_TxFifo ;

ARCHITECTURE spec OF Ctrl_TxFifo IS
BEGIN
            PROCESS (clk_uart,dtr,rts)
            BEGIN
                        if (clk_uart'event and clk_uart = '0') then
                                    if (dtr = '0' and rts = '0') then
                                                fifo_tx_wrclk <= uart_tx_ready;
                                    else
                                                fifo_tx_wrclk <= '1';
                                    end if;
                        end if;
            END process;
END spec;
```

```
--------------------------------------------------------------------------------------------------------------------------------
---                                          Control RX Fifo Subcircuit                                                      ---
--------------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY Ctrl_RxFifo IS
PORT(       clk_uart                : IN        std_logic;
            fifo_rx_empty           : IN        std_logic;
            uart_rx_ready           : IN        std_logic;
            fifo_rx_rdclk           : OUT       std_logic;
            uart_rx_wrn             : OUT       std_logic);
END Ctrl_RxFifo ;

ARCHITECTURE spec OF Ctrl_RxFifo IS
            SIGNAL fifo_clk_out    : std_logic;
BEGIN
            process (clk_uart)
            begin
                        if (clk_uart'event and clk_uart = '1') then
                                    if (uart_rx_ready = '1' and fifo_rx_empty = '0') then
                                                fifo_clk_out <= '1';
                                    else
                                                fifo_clk_out <= '0';
                                    end if;
                        end if;
            end process;

            process (clk_uart)
            begin
            if (clk_uart'event and clk_uart = '0') then
                        if (fifo_clk_out = '1') then
                                    uart_rx_wrn <= '1';
                        else
                                    uart_rx_wrn <= '0';
                        end if;
            end if;
            end process;

            fifo_rx_rdclk <= fifo_clk_out;

END spec;
```

```
--------------------------------------------------------------------------------------------------------------------------------
---                                          NRZI Encoder (TX)  Subcircuit                                                  ---
--------------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY Nrzi_Tx IS
PORT(       nrzi_tx_clk             : IN        std_logic;
            nrzi_tx_en              : IN        std_logic;
            hdlc_tx_sdo             : IN        std_logic;
            sdo                     : OUT       std_logic);
END Nrzi_Tx ;

ARCHITECTURE spec OF Nrzi_Tx IS
            SIGNAL nrzi_sdo       : std_logic;
BEGIN
            process (nrzi_tx_en,nrzi_tx_clk,hdlc_tx_sdo)
            begin
                        if (nrzi_tx_en = '0') then
                                    nrzi_sdo <= '0';
                        elsif (nrzi_tx_clk'event and nrzi_tx_clk = '1') then
                                    nrzi_sdo <= not(nrzi_sdo xor hdlc_tx_sdo);
                        end if;
            end process;

            sdo <= nrzi_sdo;

END spec;
```

```
--------------------------------------------------------------------------------------------------------------------------------
---                                          NRZI Decoder (RX)  Subcircuit                                                  ---
--------------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY Nrzi_Rx IS
port(       clk_hdlc            : in std_logic;
            sdi                 : in std_logic;
            hdlc_rx_sdi         : out std_logic;
            cd                  : out std_logic);
END Nrzi_Rx ;

ARCHITECTURE spec OF Nrzi_Rx IS
            signal sdi1         : std_logic;
            signal sdi2         : std_logic;
            signal clk_cnt      : unsigned(3 downto 0);
            signal clk1x        : std_logic;
            signal shift_reg    : std_logic_vector(7 downto 0);
            signal car_det      : std_logic;
            signal dff_sdi1     : std_logic;
            signal dff_sdi2     : std_logic;
BEGIN
            edge_capture: process (clk_hdlc)
            begin
                        if (clk_hdlc'event and clk_hdlc = '1') then
                                    sdi2 <= sdi1 ;
                                    sdi1 <= sdi ;
                        end if ;
            end process ;

            clock_adjust: process (clk_hdlc)
            begin
                        if (clk_hdlc'event and clk_hdlc = '0') then
                                    if ((sdi1 xor sdi2) = '1') then
                                                clk_cnt <= "0000" ;
                                    else
                                                clk_cnt <= clk_cnt + "0001" ;
                                    end if;
                        end if ;
            end process ;

            clk1x <= clk_cnt(3);

            carrier_detect: process (clk1x)
            begin
                        if (clk1x'event and clk1x = '1') then
                        shift_reg <= sdi & shift_Reg(7 downto 1);
                        end if;
            end process;

            car_det <= shift_reg(7) or shift_reg(6) or shift_reg(5) or shift_reg(4) or shift_reg(3) or shift_reg(2) or shift_reg(1) or shift_reg (0);

            d_ff: process (car_det,clk1x)
            begin
```

```
                if (car_det = '0') then
                        dff_sdi2 <= '0';
                        dff_sdi1 <= '0';
                elsif (clk1x'event and clk1x = '1') then
                        dff_sdi2 <= dff_sdi1;
                        dff_sdi1 <= sdi;
                end if;
        end process;

        hdlc_rx_sdi <= not(dff_sdi2 xor dff_sdi1);
        cd <= not car_det;

END spec;
```

```
----------------------------------------------------------------------------------------------------------------------------
---                                          UART TX  Structure Subcircuit                                              ---
----------------------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

ENTITY Uart_Tx IS
PORT (      rst                 : in        std_logic;
            clk16x              : in        std_logic;
            sdi                 : in        std_logic;
            dout                : out       std_logic_vector (7 downto 0);
            data_ready          : out       std_logic;
            framing_error       : out       std_logic);
END Uart_Tx ;

ARCHITECTURE struct OF Uart_Tx IS
            signal sdi1             : std_logic;
            signal sdi2             : std_logic;
            signal clk1x_enable     : std_logic ;
            signal clkdiv           : unsigned(3 downto 0);
            signal rsr              : unsigned(7 downto 0);
            signal rbr              : unsigned(7 downto 0);
            signal no_bits_rcvd     : unsigned (3 downto 0);
            signal clk1x            : std_logic;
BEGIN
            process (rst,clk16x)
            begin
                        if rst = '1' then
                                    sdi1 <= '1' ;
                                    sdi2 <= '1' ;
                        elsif clk16x'event and clk16x = '1' then
                                    sdi2 <= sdi1 ;
                                    sdi1 <= sdi ;
                        end if ;
            end process ;

            process (rst,clk16x)
            begin
                        if rst = '1' then
                                    clk1x_enable <= '0' ;
                        elsif clk16x'event and clk16x = '1' then
                                    if sdi1 = '0' and sdi2 = '1' then
                                                clk1x_enable <= '1' ;
                                    elsif std_logic_vector(no_bits_rcvd) = "1100" then
                                                clk1x_enable <= '0' ;
                                    end if ;
                        end if ;
            end process ;

            process (rst,clk16x,clk1x_enable)
            begin
                        if rst = '1' then
                                    clkdiv <= "0000" ;
                        elsif clk1x_enable = '0' then
                                    clkdiv <= "0000" ;
                        elsif clk16x'event and clk16x = '1' then
```

```
                clkdiv <= clkdiv + "0001" ;
        end if ;
end process ;

clk1x <= clkdiv(3) ;

process (rst,clk1x)
begin
        if rst = '1' then
                rsr <= "00000000" ;
                rbr <= "00000000" ;
                data_ready <= '1' ;
                framing_error <= '0' ;
        elsif clk1x'event and clk1x = '1' then
                if std_logic_vector(no_bits_rcvd) = "0000" then
                        data_ready <= '0' ;
                        framing_error <= '0' ;
                elsif std_logic_vector(no_bits_rcvd) >= "0001" and std_logic_vector(no_bits_rcvd) <= "1000" then
                        rsr(7) <= sdi2 ;
                        rsr(6 downto 0) <= rsr(7 downto 1) ;
                        data_ready <= '0' ;
                        framing_error <= '0' ;
                elsif std_logic_vector(no_bits_rcvd) = "1001" then
                        rbr <= rsr ;
                        data_ready <= '0' ;
                        framing_error <= '0' ;
                elsif std_logic_vector(no_bits_rcvd) = "1010" then
                        data_ready <= '1' ;
                        if sdi2 = '1' then
                                framing_error <= '0' ;
                        else
                                framing_error <= '1' ;
                        end if ;
                end if ;
        end if;
end process ;

process (rst,clk1x,clk1x_enable)
begin
        if rst = '1' then
                no_bits_rcvd <= "0000" ;
        elsif clk1x_enable = '0' then
                no_bits_rcvd <= "0000" ;
        elsif clk1x'event and clk1x = '1' then
                no_bits_rcvd <= no_bits_rcvd + "0001" ;
        end if ;
end process ;

dout <= std_logic_vector(rbr);

END struct;
```

```
---------------------------------------------------------------------------------------------------------------------------------
---                                                UART RX  Structure Subcircuit                                              ---
---------------------------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

ENTITY Uart_Rx IS
PORT (      rst                 : in        std_logic;
            clk16x              : in        std_logic;
            wrn                 : in        std_logic;
            din                 : in        std_logic_vector(7 downto 0);
            buf_ready           : out       std_logic;
            sdo                 : out       std_logic);
END Uart_Rx ;

ARCHITECTURE struct OF Uart_Rx IS
            signal clk1x_enable   : std_logic;
            signal tsr            : std_logic_vector (7 downto 0);
            signal clkdiv         : unsigned (3 downto 0);
            signal clk1x          : std_logic;
            signal no_bits_sent   : unsigned (3 downto 0);
            signal wrn1           : std_logic;
            signal wrn2           : std_logic;
BEGIN
            process (rst,clk16x)
            begin
                        if rst = '1' then
                                    wrn1 <= '1';
                                    wrn2 <= '1';
                        elsif clk16x'event and clk16x = '1' then
                                    wrn2 <= wrn1;
                                    wrn1 <= wrn;
                        end if;
            end process;

            process (rst,clk16x)
            begin
                        if rst = '1' then
                                    clk1x_enable <= '0';
                        elsif clk16x'event and clk16x = '1' then
                                    if wrn1 = '1' and wrn2 = '0' then
                                                clk1x_enable <= '1';
                                    elsif std_logic_vector(no_bits_sent) = "1011" then
                                                clk1x_enable <= '0';
                                    end if;
                        end if;
            end process;

            process (rst,clk16x,clk1x_enable)
            begin
                        if rst = '1' then
                                    clkdiv <=  "0000";
                        elsif clk1x_enable = '0' then
                                    clkdiv <=  "0000";
                        elsif clk16x'event and clk16x = '1' then
                                    clkdiv <= clkdiv + "0001";
```

```
		end if;
	end process;

	clk1x <= clkdiv(3);

	process (rst,clk1x)
	begin
		if rst = '1' then
		buf_ready <= '1';
			sdo <= '1';
			tsr <= "11111111";
		elsif clk1x'event and clk1x = '1' then
			if std_logic_vector(no_bits_sent) = "0000" then
				buf_ready <= '0';
				sdo <= '0';
				tsr <= din;
			elsif std_logic_vector(no_bits_sent) >= "0001" and std_logic_vector(no_bits_sent) <= "1001" then
				buf_ready <= '0';
				sdo <= tsr(0);
				tsr <= '1' & tsr(7 downto 1);
			elsif std_logic_vector(no_bits_sent) = "1010" then
				buf_ready <= '1';
			end if;
		end if;
	end process;

	process (rst,clk1x,clk1x_enable)
	begin
		if rst = '1' then
			no_bits_sent <= "0000";
		elsif clk1x_enable = '0' then
			no_bits_sent <= "0000";
		elsif clk1x'event and clk1x = '1' then
			no_bits_sent <= no_bits_sent + "0001";
		end if;
	end process;

END struct;
```

```
--------------------------------------------------------------------------------------------------------------------------------
---                                          FIFO Structure  Subcircuit                                                      ---
--------------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

ENTITY fifo IS
GENERIC( width      : positive := 8;
         widthu     : positive := 5;
         numwords : positive := 32 );
PORT(    rst        : IN       std_logic  := '0';
         din        : IN       std_logic_vector (width-1 downto 0);
         rdclk      : IN       std_logic;
         wrclk      : IN       std_logic;
         empty      : OUT      std_logic;
         full       : OUT      std_logic;
         dout       : OUT      std_logic_vector (width-1 downto 0);
         usedw      : OUT      std_logic_vector (widthu-1 downto 0));
END fifo ;

ARCHITECTURE struct OF fifo IS
         signal tmp_din, tmp_dout          : std_logic_vector(width-1 downto 0);
         signal read_id, write_id          : std_logic_vector (widthu-1 downto 0);
         signal count_id                   : std_logic_vector (widthu downto 0);
         signal read_en, write_en          : std_logic;
         signal empty_flag                 : std_logic;
         signal full_flag                  : std_logic;

         COMPONENT dpram_32x8
         PORT (   ain       : IN std_logic_vector (widthu-1 downto 0);
                  aout      : IN std_logic_vector (widthu-1 downto 0);
                  din       : IN std_logic_vector (width-1 downto 0);
                  dout      : OUT std_logic_vector(width-1 downto 0);
                  oe        : IN std_logic;
                  we        : IN std_logic);
         END COMPONENT;
BEGIN
         tmp_din <= din;
         read_en <= '0';
         write_en <= '0';

         async_dualport_ram: dpram_32x8
         port map ( ain => write_id,
                  aout => read_id,
                  din => tmp_din,
                  dout => tmp_dout,
                  oe => read_en,
                  we => write_en);

         process (rst,rdclk,wrclk)
         begin
                  if (rst = '1') then
                           read_id <= (others => '0');
                           write_id <= (others => '0');
                  else
```

```
                        ----- IF READ -----
                        if (rdclk'event and rdclk = '1') then
                                if (empty_flag = '0') then
                                        if (read_id = numwords-1) then
                                                read_id <= (others => '0');
                                        else
                                                read_id <= read_id + 1;
                                        end if;
                                end if;
                        end if;       -- if READ
                        ----- IF WRITE -----
                        if (wrclk'event and wrclk = '1') then
                                if (full_flag = '0') then
                                        if (write_id = numwords-1) then
                                                write_id <= (others => '0');
                                        else
                                                write_id <= write_id + 1;
                                        end if;
                                end if;
                        end if;       -- if WRITE
                end if;  -- if rst = '1'
        end process;

        process(rst,write_id,read_id)
        begin
                if (rst = '1') then
                        count_id <= (others => '0');
                else
                        if (write_id < read_id) then
                                count_id <= ('1' & write_id) - ('0' & read_id);
                        else
                                count_id <= ('0' & write_id) - ('0' & read_id);
                        end if;
                end if;
        end process;

        process(rst,count_id)
        begin
                if (rst = '1') then
                        empty_flag <= '1';
                        full_flag <= '0';
                else
                        if (count_id = 0)then
                                empty_flag <= '1';
                                full_flag <= '0';
                        elsif (count_id = numwords-1) then
                                empty_flag <= '0';
                                full_flag <= '1';
                        else
                                empty_flag <= '0';
                                full_flag <= '0';
                        end if;
                end if;
end process;

        dout <= tmp_dout;
        full <= full_flag;
```

```
        empty <= empty_flag;
        usedw <= count_id(widthu-1 downto 0);

END struct;
```

```
---------------------------------------------------------------------------------------------------------------------------
---                                          DPRAM_32X8 Structure  Subcircuit                                          ---
---------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dpram_32X8 IS
PORT(     ain       : IN      std_logic_vector (4 DOWNTO 0) ;
          aout      : IN      std_logic_vector (4 DOWNTO 0) ;
          din       : IN      std_logic_vector (7 DOWNTO 0) ;
          oe        : IN      std_logic  ;
          we        : IN      std_logic  ;
          dout      : OUT     std_logic_vector (7 DOWNTO 0));
END dpram_32X8 ;

ARCHITECTURE struct OF dpram_32X8 IS
          SIGNAL din0          : std_logic_vector(3 DOWNTO 0);
          SIGNAL din1          : std_logic_vector(3 DOWNTO 0);
          SIGNAL dout0         : std_logic_vector(3 DOWNTO 0);
          SIGNAL dout1         : std_logic_vector(3 DOWNTO 0);

          COMPONENT dpram_32X4
          PORT (    ain       : IN      std_logic_vector (4 DOWNTO 0);
                    aout      : IN      std_logic_vector (4 DOWNTO 0);
                    din       : IN      std_logic_vector (3 DOWNTO 0);
                    oe        : IN      std_logic ;
                    we        : IN       std_logic ;
                    dout      : OUT     std_logic_vector (3 DOWNTO 0) );
          END COMPONENT;

BEGIN
          din1 <= din(7 downto 4);
          din0 <= din(3 downto 0);
          dout(7 downto 4) <= dout1;
          dout(3 downto 0) <= dout0;

          I0 : dpram_32X4  PORT MAP (
                    ain  => ain,
                    aout => aout,
                    din  => din0,
                    oe   => oe,
                    we   => we,
                    dout => dout0 );

          I1 : dpram_32X4 PORT MAP (
                    ain  => ain,
                    aout => aout,
                    din  => din1,
                    oe   => oe,
                    we   => we,
                    dout => dout1 );

END struct;
```

```
---------------------------------------------------------------------------------------------------------------------------
---                                        DPRAM_32X4 Structure Subcircuit                                              ---
---------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dpram_32X4 IS
PORT(       ain         : IN        std_logic_vector (4 DOWNTO 0) ;
            aout        : IN        std_logic_vector (4 DOWNTO 0) ;
            din         : IN        std_logic_vector (3 DOWNTO 0) ;
            oe          : IN        std_logic  ;
            we          : IN        std_logic  ;
            dout        : OUT       std_logic_vector (3 DOWNTO 0));
END dpram_32X4 ;

ARCHITECTURE struct OF dpram_32X4 IS
            TYPE twoDarray IS ARRAY(0 to 31) of std_logic_vector (3 downto 0);
            SIGNAL mem              : twoDarray ;
BEGIN

            dwrite: process (we,ain,din)
                        VARIABLE write_address          : INTEGER;
            begin
                        write_address :=  CONV_INTEGER(ain);
                        if (we = '0') then
                                    mem(write_address) <= din;
                        end if;
            end process dwrite;

            dread: process (oe,aout,mem)
                        VARIABLE read_address           : INTEGER;
            begin
                        read_address :=  CONV_INTEGER(aout);
                        if(oe = '0') then
                                    dout <= mem(read_address) ;
                        else
                                    dout <= "ZZZZ";
                        end if;
            end process dread;

END struct;
```

```
----------------------------------------------------------------------------------------------------------------------------
---                                          HDLC TX Structure Subcircuit                                              ---
----------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

LIBRARY work;
USE work.PCK_CRC16_D1.ALL;

ENTITY Hdlc_Tx IS
PORT(       rst         : IN STD_LOGIC;
            full        : IN std_logic;
            empty       : IN std_logic;
            dtr         : IN std_logic;
            rts         : IN std_logic;
            clk16x      : IN std_logic;
            sdo         : out std_logic;
            NrziClk     : out std_logic;
            NrziEn      : out std_logic;
            FifoClk     : out std_logic;
            InfoBuf     : in std_logic_vector(7 downto 0);
            AddrBuf     : in std_logic_vector(7 downto 0);
            CtrlBuf     : in std_logic_vector(7 downto 0);
            InfoNum     : IN std_logic_vector(4 downto 0));
END Hdlc_Tx;

ARCHITECTURE struct OF Hdlc_Tx IS
            type state_type is (IDLE,STAF,ADDR,CTRL,INFO,FCS1,ENDF,NONE);
            signal state            : state_type;
            signal clk1x            : std_logic;
            signal clk_div          : unsigned(3 downto 0);
            signal clk1x_enable     : std_logic;
            signal clk_en           : std_logic;
            signal shift_reg        : std_logic_vector(7 downto 0);
            signal num_reg          : unsigned(4 downto 0);
            signal zchk_reg         : std_logic_vector(5 downto 0);
            signal count_reg        : unsigned(3 downto 0);
            signal FCS_reg          : std_logic_vector(15 downto 0);
BEGIN
            start_detect: process(rst,state,full,empty,dtr,rts)
            begin
                        if rst = '1' then
                                    clk1x_enable <= '0';
                        elsif state = IDLE then
                                    if full = '1' then
                                                clk1x_enable <= '1';
                                    elsif empty = '0' and dtr = '0' and rts = '1' then
                                                clk1x_enable <= '1';
                                    else
                                                clk1x_enable <= '0';
                                    end if;
                        else
                                    clk1x_enable <= '1';
                        end if ;
            end process ;
```

```
iclk_en: process (rst,clk16x,clk1x_enable)
begin
        if rst = '1' then
                clk_div <= "0000" ;
        elsif clk1x_enable = '0' then
                clk_div <= "0000" ;
        elsif clk16x'event and clk16x = '1' then
                clk_div <= clk_div + "0001" ;
        end if ;
end process ;

clk1x <= clk_div(3);

sequence: process(rst,clk1x,clk1x_enable)
begin
        if rst = '1' then
                state <= IDLE;
                sdo <= '0';
                shift_reg <= (OTHERS => '1');
                num_reg <= (OTHERS => '0');
                zchk_reg <= (OTHERS => '0');
                count_reg <= (OTHERS => '0');
                FCS_reg <= (OTHERS => '1');
                clk_en <= '0';
        elsif clk1x_enable = '0' then
                state <= IDLE;
                sdo <= '0';
                shift_reg <= (OTHERS => '1');
                num_reg <= (OTHERS => '0');
                zchk_reg <= (OTHERS => '0');
                count_reg <= (OTHERS => '0');
                FCS_reg <= (OTHERS => '1');
                clk_en <= '0';
        elsif clk1x'event and clk1x = '1' then
                case state is
                when IDLE =>
                                clk_en <= '0';
                                sdo <= '0';
                                zchk_reg <= (OTHERS => '0');
                                FCS_reg <= (OTHERS => '1');
                                if count_reg(2 downto 0) = "111" then
                                        state <= STAF;
                                        shift_reg <= "01111110";
                                        num_reg <= unsigned(InfoNum);
                                else
                                        state <= IDLE;
                                end if;
                                count_reg <= count_reg + "001";
                        when STAF =>
                                clk_en <= '0';
                                sdo <= shift_reg(0);
                                zchk_reg <= (OTHERS => '0');
                                FCS_reg <= (OTHERS => '1');
                                if count_reg(2 downto 0) = "111" then
                                        state <= ADDR;
                                        shift_reg <= AddrBuf;
                                else
```

```
                        state <= STAF;
                        shift_reg(6 downto 0) <= shift_reg(7 downto 1);
                end if;
                count_reg <= count_reg + "001";
        when ADDR =>
                clk_en <= '0';
                if zchk_reg(4 downto 0) = "11111" then
                        sdo <= '0';
                        zchk_reg(5 downto 0) <= zchk_reg(4 downto 0) & '0';
                else
                        sdo <= shift_reg(0);
                        zchk_reg(5 downto 0) <= zchk_reg(4 downto 0) & shift_reg(0);
                        FCS_reg <= nextCRC16_D1(shift_reg(0),FCS_reg);
                        if count_reg(2 downto 0) = "111" then
                                state <= CTRL;
                                shift_reg <= CtrlBuf;
                        else
                                state <= ADDR;
                                shift_reg(6 downto 0) <= shift_reg(7 downto 1);
                        end if;
                        count_reg <= count_reg + "001";
                end if;
        when CTRL =>
                clk_en <= '0';
                if zchk_reg(4 downto 0) = "11111" then
                        sdo <= '0';
                        zchk_reg(5 downto 0) <= zchk_reg(4 downto 0) & '0';
                else
                        sdo <= shift_reg(0);
                        zchk_reg(5 downto 0) <= zchk_reg(4 downto 0) & shift_reg(0);
                        FCS_reg <= nextCRC16_D1(shift_reg(0),FCS_reg);
                        if count_reg(2 downto 0) = "111" then
                                state <= INFO;
                                shift_reg <= InfoBuf;
                        else
                                state <= CTRL;
                                shift_reg(6 downto 0) <= shift_reg(7 downto 1);
                        end if;
                        count_reg <= count_reg + "001";
                end if;
        when INFO =>
                clk_en <= '1';
                if zchk_reg(4 downto 0) = "11111" then
                        sdo <= '0';
                        zchk_reg(5 downto 0) <= zchk_reg(4 downto 0) & '0';
                else
                        sdo <= shift_reg(0);
                        zchk_reg(5 downto 0) <= zchk_reg(4 downto 0) & shift_reg(0);
                        if count_reg(2 downto 0) = "111" then
                                if num_reg = "00001" then
                                        state <= FCS1;
                                        shift_reg <= "01111110";
                                        count_reg <= "0000";
                                else
                                        state <= INFO;
                                        shift_reg <= InfoBuf;
                                        count_reg <= count_reg + "001";
```

```
                                        end if;
                                        num_reg <= num_reg - "00001";
                                else
                                        state <= INFO;
                                        shift_reg(6 downto 0) <= shift_reg(7 downto 1);
                                        count_reg <= count_reg + "001";
                                end if;
                                FCS_reg <= nextCRC16_D1(shift_reg(0),FCS_reg);
                        end if;
                when FCS1 =>
                        clk_en <= '0';
                        shift_reg <= "01111110";
                        if zchk_reg(4 downto 0) = "11111" then
                                sdo <= '0';
                                zchk_reg(5 downto 0) <= zchk_reg(4 downto 0) & '0';
                        else
                                sdo <= FCS_reg(15);
                                zchk_reg(5 downto 0) <= zchk_reg(4 downto 0) & FCS_reg(15);
                                if count_reg = "1111" then
                                        state <= ENDF;
                                else
                                        state <= FCS1;
                                end if;
                                FCS_reg(15 downto 1) <= FCS_reg(14 downto 0);
                                count_reg <= count_reg + "001";
                        end if;
                when ENDF =>
                        clk_en <= '0';
                        sdo <= shift_reg(0);
                        zchk_reg <= (OTHERS => '0');
                        FCS_reg <= (OTHERS => '1');
                        if count_reg = "111" then
                                state <= NONE;
                        else
                                state <= ENDF;
                        end if;
                        shift_reg(6 downto 0) <= shift_reg(7 downto 1);
                        count_reg <= count_reg + "001";
                when NONE =>
                        state <= IDLE;
        end case;
    end if;
end process;

fifo_clk_en: process (clk_en,count_reg(2))
begin
    if clk_en = '0' then
        FifoClk <= '0';
    else
        FifoClk <= count_reg(2);
    end if;
end process;

nrzi_clk_en: process (clk1x_enable,clk_div(2))
begin
    if (clk1x_enable = '0') then
        NrziClk <= '0';
```

```
                elsif clk_div(2)'event and clk_div(2) = '1' then
                        NrziClk <= clk1x;
                end if;
        end process;

        NrziEn <= clk1x_enable;

END struct;
```

```
----------------------------------------------------------------------------------------------------------------------------
---                                          HDLC RX Structure Subcircuit                                                ---
----------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

LIBRARY work;
USE work.PCK_CRC16_D1.ALL;

ENTITY Hdlc_Rx IS
PORT (      rst         : in std_logic;
            clk16x      : in std_logic;
            sdi         : in std_logic;
            full        : in std_logic;
            InfoOut     : out std_logic_vector (7 downto 0);
            AddrOut     : out std_logic_vector(7 downto 0);
            CtrlOut     : out std_logic_vector(7 downto 0);
            ClkOut      : out std_logic;
            ri          : out std_logic;
            FrameError : out std_logic);
END Hdlc_Rx;

ARCHITECTURE struct OF Hdlc_Rx IS
            type state_type is (IDLE, ADDR, CTRL, INFO);
            signal state            : state_type;
            signal sdi1             : std_logic;
            signal sdi2             : std_logic;
            signal clk1x            : std_logic;
            signal clk_div          : unsigned(3 downto 0);
            signal clk_en           : std_logic;
            signal addr_reg         : std_logic_vector(7 downto 0);
            signal ctrl_reg         : std_logic_vector(7 downto 0);
            signal info1_reg        : std_logic_vector(7 downto 0);
            signal info2_reg        : std_logic_vector(7 downto 0);
            signal shift_reg        : std_logic_vector(7 downto 0);
            signal num_reg          : unsigned(1 downto 0);
            signal count_reg        : unsigned(2 downto 0);
            signal zchk_reg         : std_logic_vector(5 downto 0);
            signal FCS_reg          : std_logic_vector(15 downto 0);
            signal crc_reg          : std_logic_vector(15 downto 0);
            signal crc1_reg         : std_logic_vector(15 downto 0);
            signal crc2_reg         : std_logic_vector(15 downto 0);
            signal ri_buf           : std_logic;
            signal frame_err        : std_logic;
BEGIN
            process (rst,clk16x)
            begin
                        if rst = '1' then
                                    sdi1 <= '1' ;
                                    sdi2 <= '1' ;
                        elsif clk16x'event and clk16x = '1' then
                                    sdi2 <= sdi1 ;
                                    sdi1 <= sdi ;
                        end if ;
            end process ;
```

```
process (rst,clk16x)
begin
        if rst = '1' then
                clk_div <= "0000" ;
        elsif clk16x'event and clk16x = '0' then
                if state = IDLE and sdi1 = '1' and sdi2 = '0' then
                        clk_div <= "0000" ;
                else
                        clk_div <= clk_div + "0001" ;
                end if;
        end if ;
end process ;

clk1x <= clk_div(3);

sequence: process(rst,clk1x)
begin
        if rst = '1' then
                state <= IDLE;
                clk_en <= '0';
                shift_reg <= (OTHERS => '1');
                num_reg <= (OTHERS => '0');
                count_reg <= (OTHERS => '0');
                zchk_reg <= (OTHERS => '0');
                FCS_reg <= (OTHERS => '1');
                crc_reg <= (OTHERS => '1');
                crc1_reg <= (OTHERS => '1');
                crc2_reg <= (OTHERS => '1');
                ri_buf <= '1';
                frame_err <= '0';
                addr_reg <= (OTHERS => '0');
                ctrl_reg <= (OTHERS => '0');
                info1_reg <= (OTHERS => '0');
                info2_reg <= (OTHERS => '0');
                InfoOut <= (OTHERS => '0');
                AddrOut <= (OTHERS => '0');
                CtrlOut <= (OTHERS => '0');
        elsif clk1x'event and clk1x = '1' then
                case state is
                        when IDLE =>
                                num_reg <= (OTHERS => '0');
                                clk_en <= '0';
                                if (sdi & shift_reg(7 downto 1)) = "01111110" then
                                        state <= ADDR;
                                        ri_buf <= '0';
                                else
                                        state <= IDLE;
                                        ri_buf <= '1';
                                end if;
                                zchk_reg <= (OTHERS => '0');
                                FCS_reg <= (OTHERS => '1');
                                shift_reg(7 downto 0) <= sdi & shift_reg(7 downto 1);
                                count_reg <= (OTHERS => '0');
                        when ADDR =>
                                num_reg <= (OTHERS => '0');
                                clk_en <= '0';
                                ri_buf <= '0';
```

```
                if (sdi & zchk_reg(5 downto 1)) = "111111" then
                        state <= IDLE;
                        frame_err <= '1';
                elsif (sdi & zchk_reg(5 downto 1)) = "011111" then
                        state <= ADDR;
                        frame_err <= '0';
                else
                        if count_reg = "111" then
                                state <= CTRL;
                                addr_reg <= sdi & shift_reg(7 downto 1);
                        else
                                state <= ADDR;
                        end if;
                        frame_err <= '0';
                        FCS_reg <= nextCRC16_D1(sdi,FCS_reg);
                        shift_reg(7 downto 0) <= sdi & shift_reg(7 downto 1);
                        count_reg <= count_reg + "001";
                end if;
                zchk_reg <= sdi & zchk_reg(5 downto 1);
        when CTRL =>
                num_reg <= (OTHERS => '0');
                clk_en <= '0';
                ri_buf <= '0';
                if (sdi & zchk_reg(5 downto 1)) = "111111" then
                        state <= IDLE;
                        frame_err <= '1';
                elsif (sdi & zchk_reg(5 downto 1)) = "011111" then
                        state <= CTRL;
                        frame_err <= '0';
                else
                        if count_reg = "111" then
                                state <= INFO;
                                ctrl_reg <= sdi & shift_reg(7 downto 1);
                                crc_reg <= nextCRC16_D1(sdi,FCS_reg);
                        else
                                state <= CTRL;
                        end if;
                        frame_err <= '0';
                        FCS_reg <= nextCRC16_D1(sdi,FCS_reg);
                        shift_reg(7 downto 0) <= sdi & shift_reg(7 downto 1);
                        count_reg <= count_reg + "001";
                end if;
                zchk_reg <= sdi & zchk_reg(5 downto 1);
        when INFO =>
                ri_buf <= '0';
                if (sdi & zchk_reg(5 downto 0)) = "1111111" then
                        state <= IDLE;
                        frame_err <= '1';
                        clk_en <= '0';
                elsif (sdi & zchk_reg(5 downto 0)) = "0111110" then
                        state <= INFO;
                        frame_err <= '0';
                else
                        if count_reg = "111" then
                                if (sdi & shift_reg(7 downto 1)) = "01111110" then
                                        clk_en <= '0';
                                        if num_reg(1) = '0' then
```

```
                                        state <= IDLE;
                                                frame_err <= '1';
                                        else
                                                if (crc2_reg(15) = info2_reg(0) and
                                                crc2_reg(14) = info2_reg(1) and
                                                crc2_reg(13) = info2_reg(2) and
                                                crc2_reg(12) = info2_reg(3) and
                                                crc2_reg(11) = info2_reg(4) and
                                                crc2_reg(10) = info2_reg(5) and
                                                crc2_reg(9) = info2_reg(6) and
                                                crc2_reg(8) = info2_reg(7) and
                                                crc2_reg(7) = info1_reg(0) and
                                                crc2_reg(6) = info1_reg(1) and
                                                crc2_reg(5) = info1_reg(2) and
                                                crc2_reg(4) = info1_reg(3) and
                                                crc2_reg(3) = info1_reg(4) and
                                                crc2_reg(2) = info1_reg(5) and
                                                crc2_reg(1) = info1_reg(6) and
                                                crc2_reg(0) = info1_reg(7)) then
                                                        state <= IDLE;
                                                        frame_err <= '0';
                                                        AddrOut <= addr_reg;
                                                        CtrlOut <= ctrl_reg;
                                                else
                                                        state <= IDLE;
                                                        frame_err <= '1';
                                                end if;
                                        end if;
                                else
                                        state <= INFO;
                                        frame_err <= '0';
                                        if num_reg(1) = '0' then
                                                clk_en <= '0';
                                                num_reg <= num_reg + "01";
                                        else
                                                if full = '0' then
                                                        InfoOut <= info2_reg;
                                                        clk_en <= '1';
                                                else
                                                        clk_en <= '0';
                                                end if;
                                                num_reg <= "10";
                                        end if;
                                end if;
                                info2_reg <= info1_reg;
                                info1_reg <= sdi & shift_reg(7 downto 1);
                                crc2_reg <= crc1_reg;
                                crc1_reg <= crc_reg;
                                crc_reg <= nextCRC16_D1(sdi,FCS_reg);
                        else
                                state <= INFO;
                                frame_err <= '0';
                        end if;
                        FCS_reg <= nextCRC16_D1(sdi,FCS_reg);
                        shift_reg(7 downto 0) <= sdi & shift_reg(7 downto 1);
                        count_reg <= count_reg + "001";
end if;
```

```
                                        zchk_reg <= sdi & zchk_reg(5 downto 1);
                        end case;
                end if;
        end process;

        process(clk_en,count_reg(2))
        begin
                if clk_en = '0' then
                        ClkOut <= '0';
                else
                        ClkOut <= count_reg(2);
                end if;
        end process;

        FrameError <= frame_err;
        ri <= ri_buf;

END struct;
```

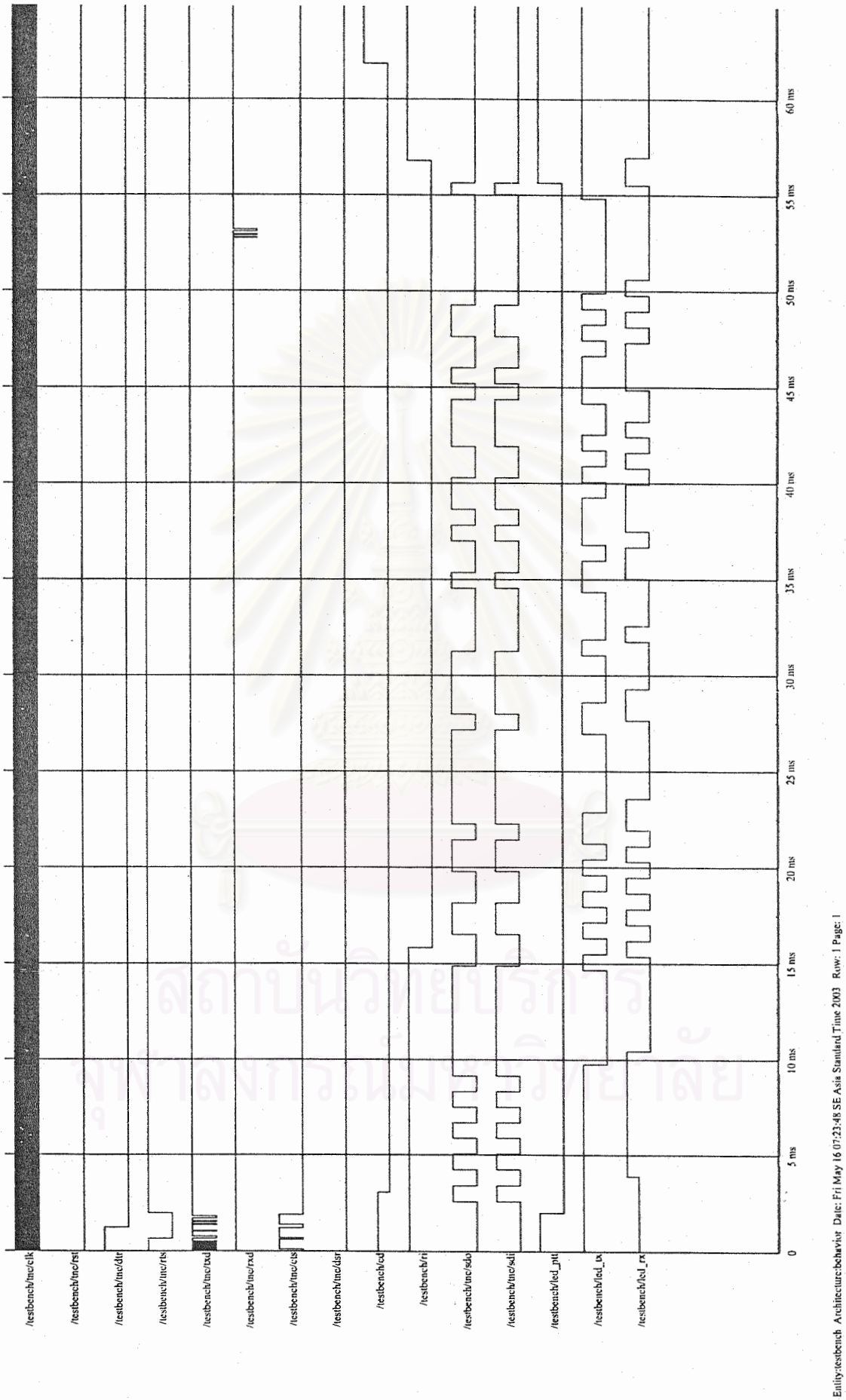
```
--------------------------------------------------------------------------------------------------------------------------------
---                                          CRC16 polynomial: (0 2 15 16)                                          ---
--------------------------------------------------------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

PACKAGE PCK_CRC16_D1 IS
        -- polynomial: (0 2 15 16)
        -- data width: 1
        function nextCRC16_D1 (
                Data    : std_logic;
                CRC     : std_logic_vector(15 downto 0))
        return std_logic_vector;
END PCK_CRC16_D1;

PACKAGE BODY PCK_CRC16_D1 IS
        -- polynomial: (0 2 15 16)
        -- data width: 1
        function nextCRC16_D1 (
                Data    : std_logic;
                CRC     : std_logic_vector(15 downto 0))
        return std_logic_vector is
                variable D              : std_logic_vector(0 downto 0);
                variable C              : std_logic_vector(15 downto 0);
                variable NewCRC         : std_logic_vector(15 downto 0);
        BEGIN
                D(0) := Data;
                C := CRC;

                NewCRC(0) := D(0) xor C(15);
                NewCRC(1) := C(0);
                NewCRC(2) := D(0) xor C(1) xor C(15);
                NewCRC(3) := C(2);
                NewCRC(4) := C(3);
                NewCRC(5) := C(4);
                NewCRC(6) := C(5);
                NewCRC(7) := C(6);
                NewCRC(8) := C(7);
                NewCRC(9) := C(8);
                NewCRC(10) := C(9);
                NewCRC(11) := C(10);
                NewCRC(12) := C(11);
                NewCRC(13) := C(12);
                NewCRC(14) := C(13);
                NewCRC(15) := D(0) xor C(14) xor C(15);

                return NewCRC;
        END nextCRC16_D1;
END PCK_CRC16_D1;
```

ภาคผนวก ข

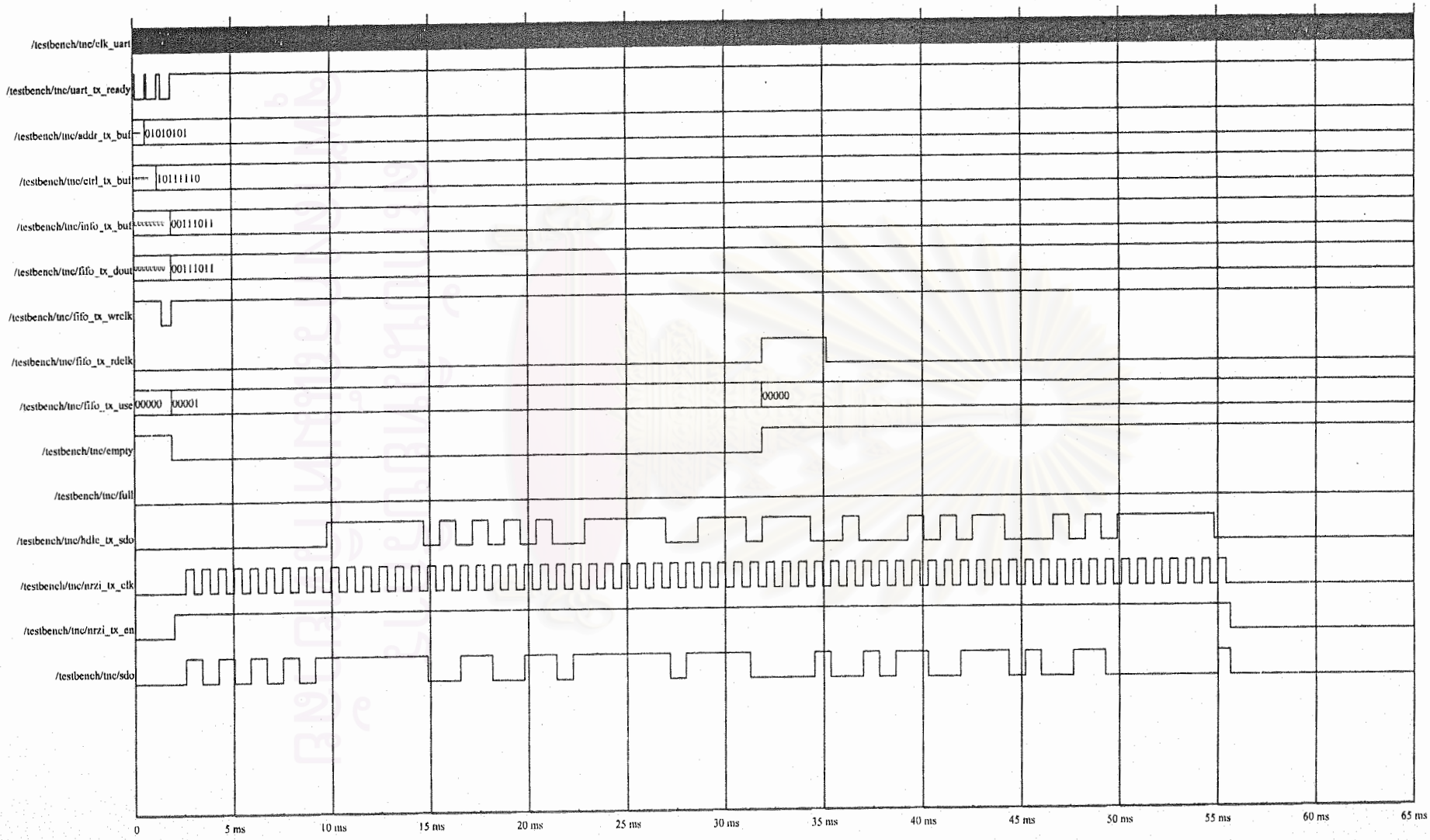
ภาษา VHDL ที่ใช้ทดสอบตัวควบคุมเทอร์มินัลโนดที่ออกแบบ และผลที่ได้

```
--------------------------------------------------------------------------------------------------------------------------------
---                                                Testbench for TNC                                                        ---
--------------------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS
        COMPONENT tnc_top
        PORT(   rst         : IN    std_logic;
                clk         : IN    std_logic;
                dtr         : IN    std_logic;
                rts         : IN    std_logic;
                cts         : OUT   std_logic;
                dsr         : OUT std_logic;
                cd          : OUT std_logic;
                ri          : OUT   std_logic;
                txd         : IN    std_logic;
                sdi         : IN    std_logic;
                sdo         : OUT   std_logic;
                rxd         : OUT   std_logic;
                led_ptt     : OUT   std_logic;
                led_tx      : OUT   std_logic;
                led_rx      : OUT   std_logic);
        END COMPONENT;

        signal rst              : std_logic;
        signal clk              : std_logic;
        signal dtr              : std_logic;
        signal rts              : std_logic;
        signal cts              : std_logic;
        signal txd              : std_logic;
        signal sdi              : std_logic;
        signal rxd              : std_logic;
        signal cd               : std_logic;
        signal ri               : std_logic;
        signal dsr              : std_logic;
        signal led_ptt          : std_logic;
        signal led_tx           : std_logic;
        signal led_rx           : std_logic;
BEGIN
        tnc: tnc_top PORT MAP(
                rst => rst,
                clk => clk,
                dtr => dtr,
                rts => rts,
                cts => cts,
                cd => cd,
                ri => ri,
                dsr => dsr,
                txd => txd,
                sdi => sdi,
                sdo => sdi,
                rxd => rxd,
```

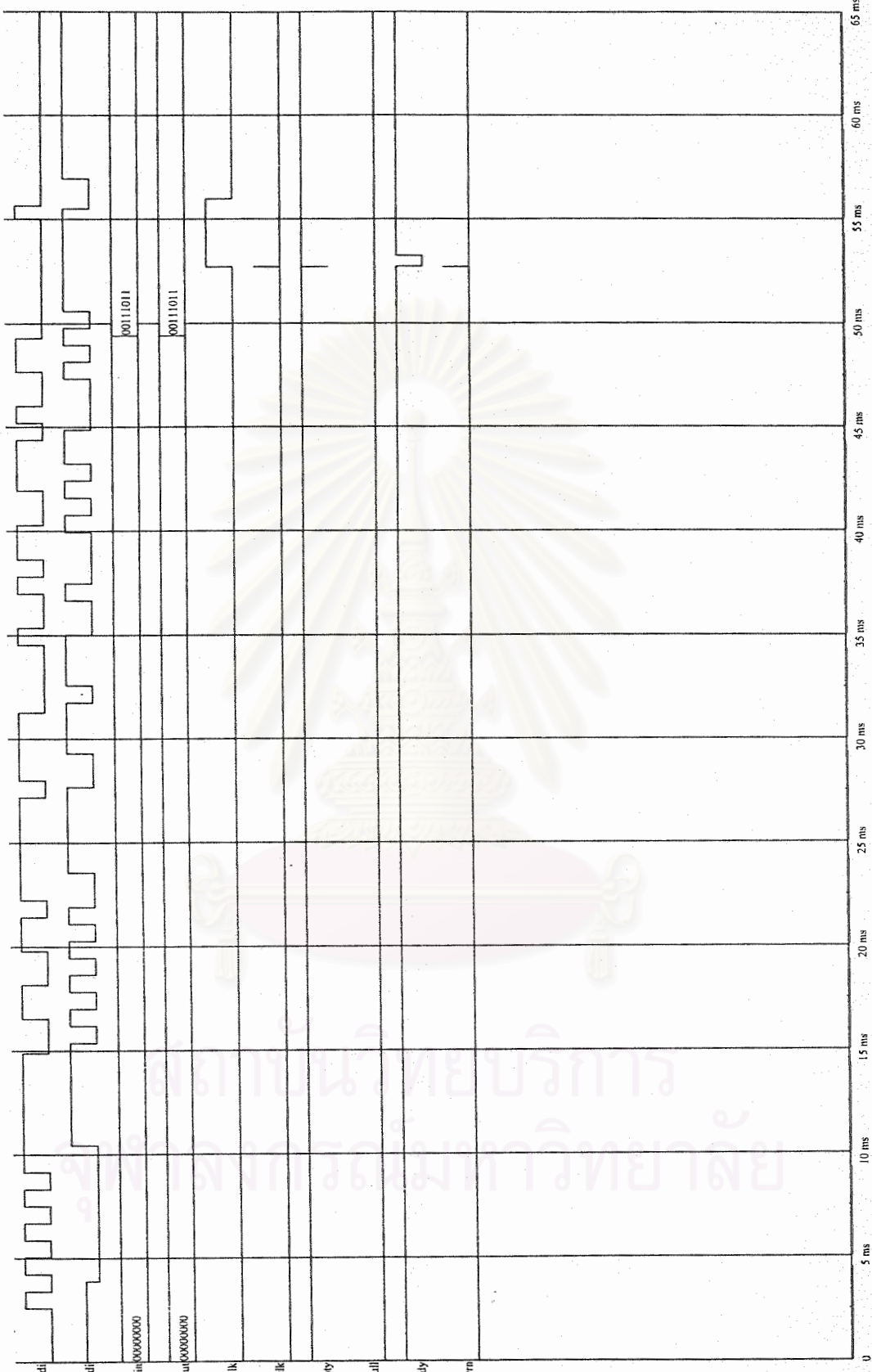
```
                led_ptt => led_ptt,
                led_tx => led_tx,
                led_rx => led_rx );

        clk_gen: process
        begin
                clk <= '0' ;
                wait for 100 ns;
                clk <= '1' ;
                wait for 100 ns;
        end process ;

        tb: process
        begin
                rst <= '0','1' after 50 ns,'0' after 1050 ns;
                dtr <= '1','0' after 1250000 ns;
                rts <= '1','0' after 660000 ns,'1' after 2000000 ns;
                txd <= '1','0' after 70000 ns,'1' after 121200 ns,'0' after 172400 ns,'1' after 223600 ns,'0' after 274800 ns,'1' after
326000 ns,'0' after 377200 ns,'1' after 428400 ns,'0' after 479600 ns,'1' after 530800 ns,
                '0' after 670000 ns,'0' after 721200 ns,'1' after 772400 ns,'1' after 823600 ns,'1' after 874800 ns,'1' after 926000
ns,'1' after 977200 ns,'0' after 1028400 ns,'1' after 1079600 ns,'1' after 1130800 ns,
                '0' after 1370000 ns,'1' after 1421200 ns,'1' after 1472400 ns,'0' after 1523600 ns,'1' after 1574800 ns,'1' after
1626000 ns,'1' after 1677200 ns,'0' after 1728400 ns,'0' after 1779600 ns,'1' after 1830800 ns;
                wait; -- will wait forever
        end process;
END;
```



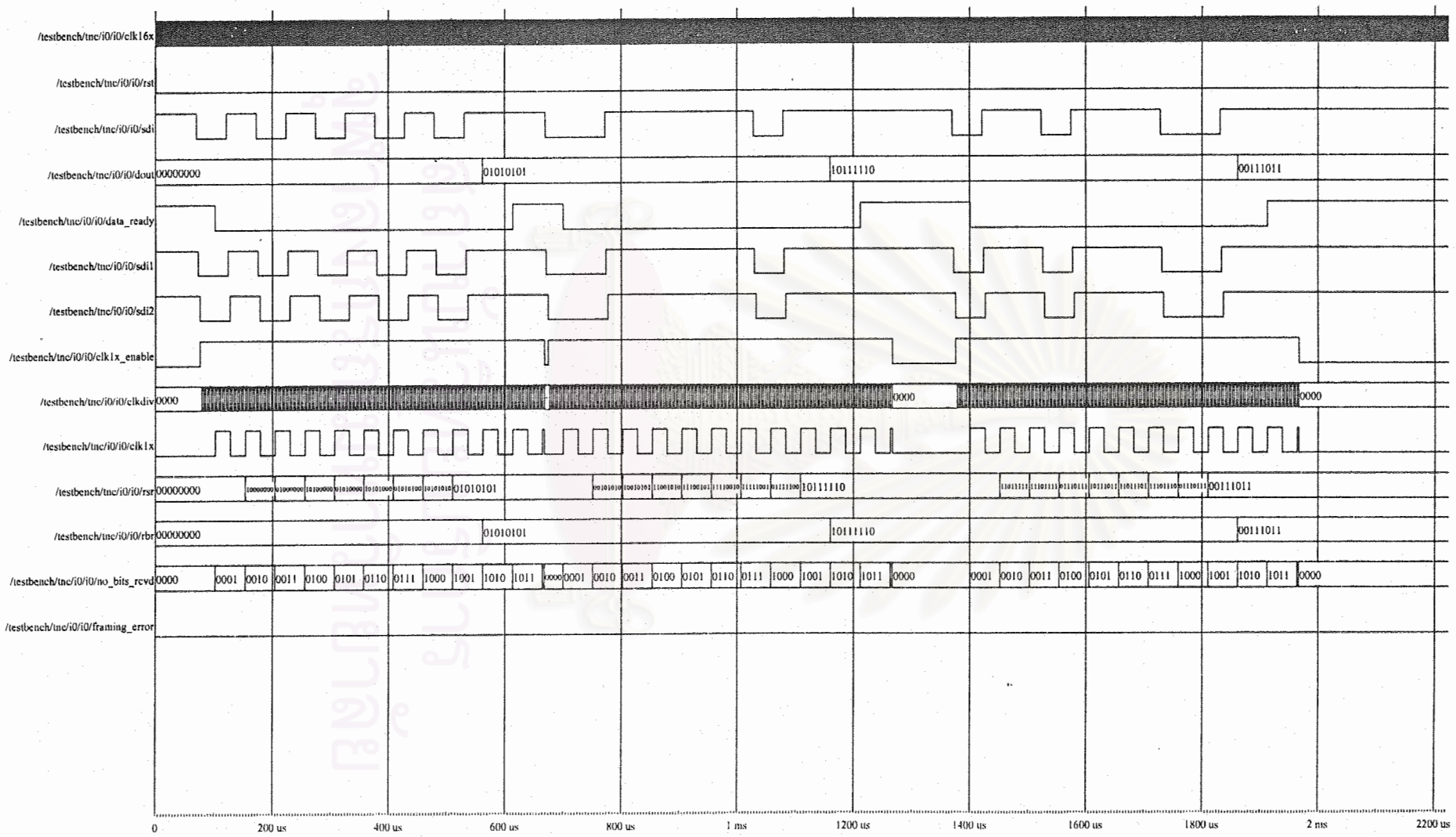
ผลการจำลองการทำงานของ TNC ในส่วนบนสุด (Top-level)



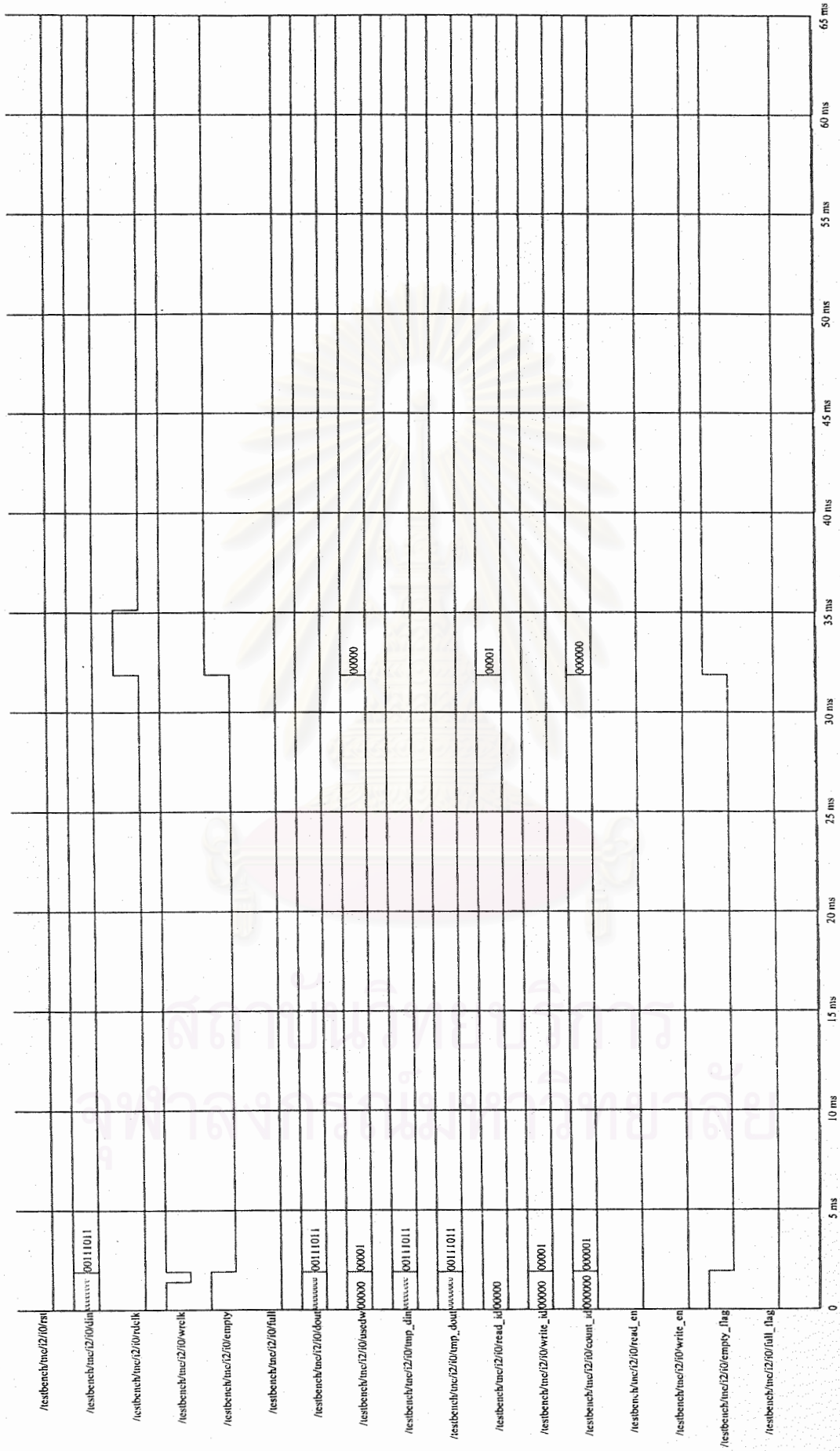
Entity:testbench Architecture:behavior Date: Fri May 16 03:16:21 SE Asia Standard Time 2003 Row: 1 Page: 1



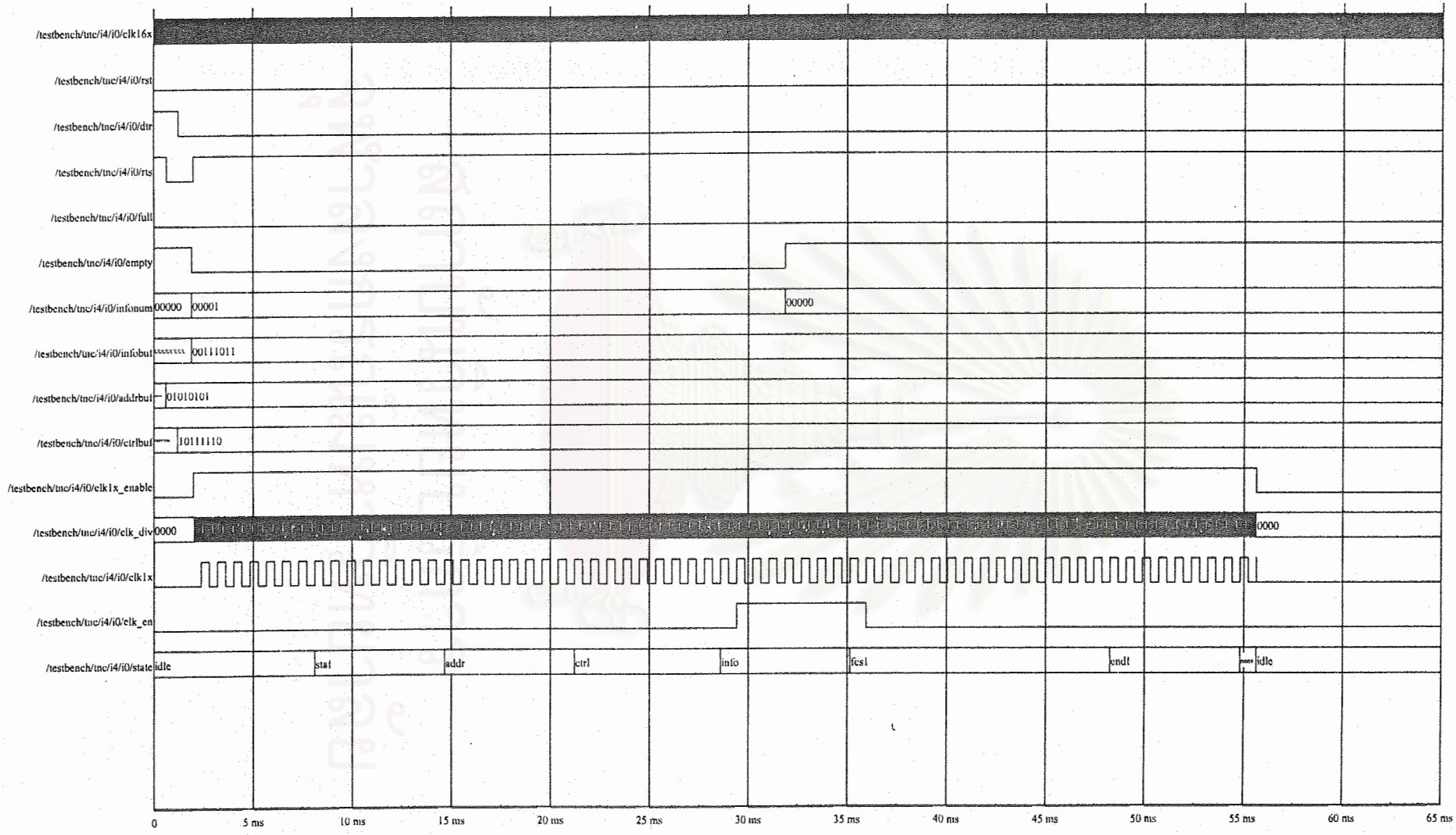
ผลการจำลองการทำงานของสัญญาณภายใน TNC (ต่อ)



Entity:testbench Architecture:behavior Date: Fri May 16 07:24:42 SE Asia Standard Time 2003 Row: 1 Page: 1

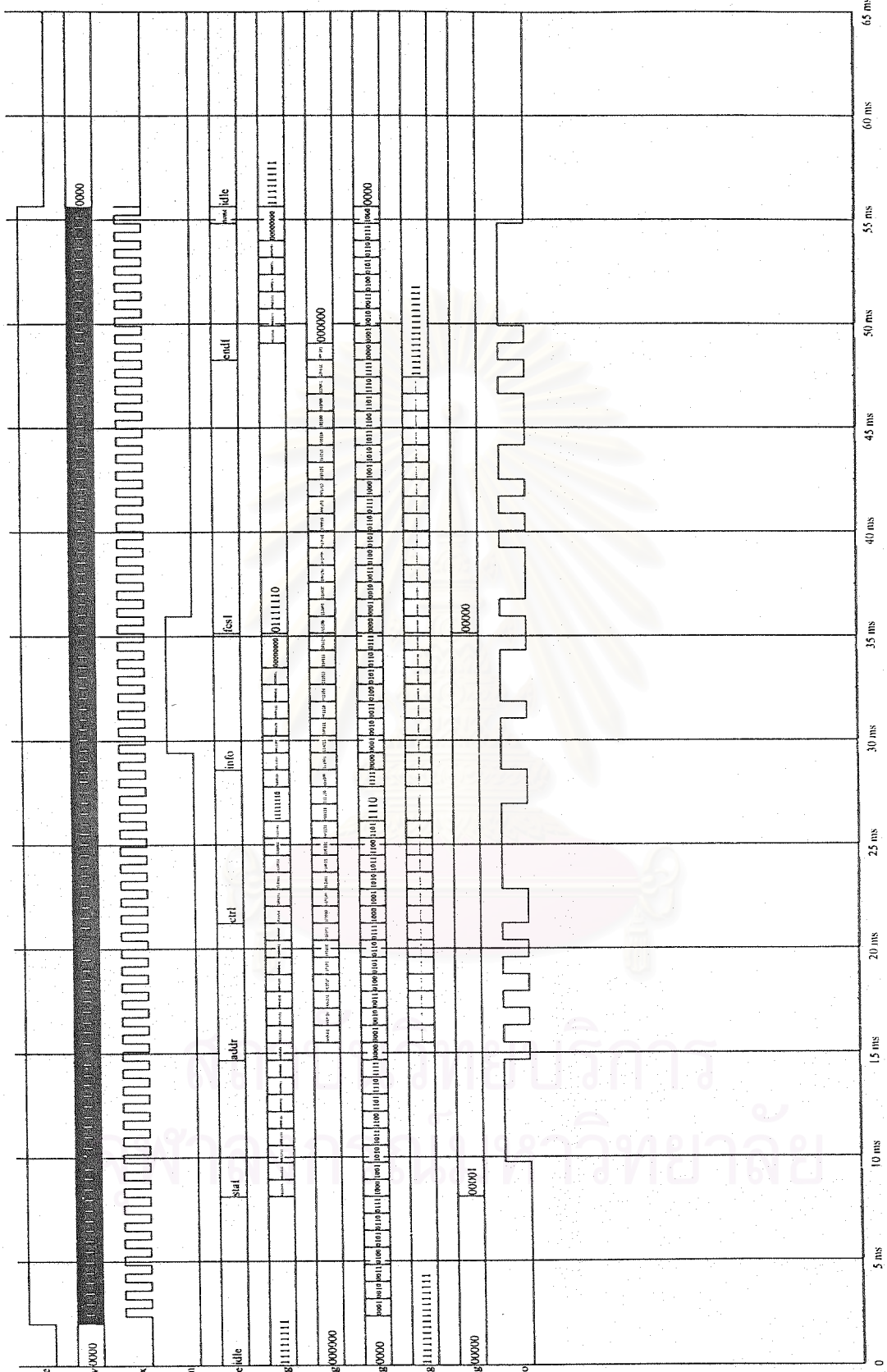


ผลการจำลองการทำงานของสัญญาณภายใน Fifo_Tx (I2)

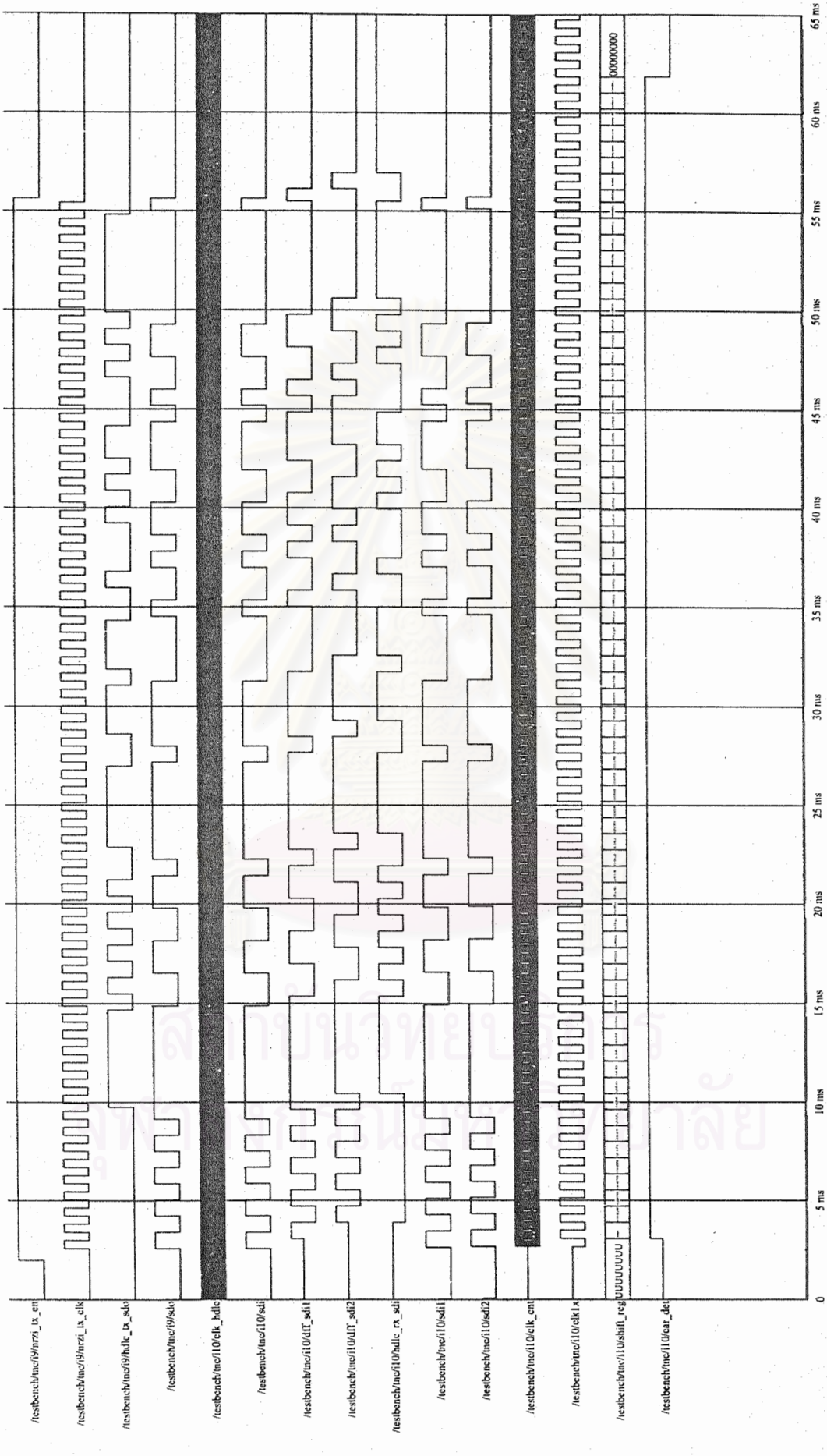


Entity:testbench Architecture:behavior Date: Fri May 16 03:17:44 SE Asia Standard Time 2003 Row: 1 Page: 1

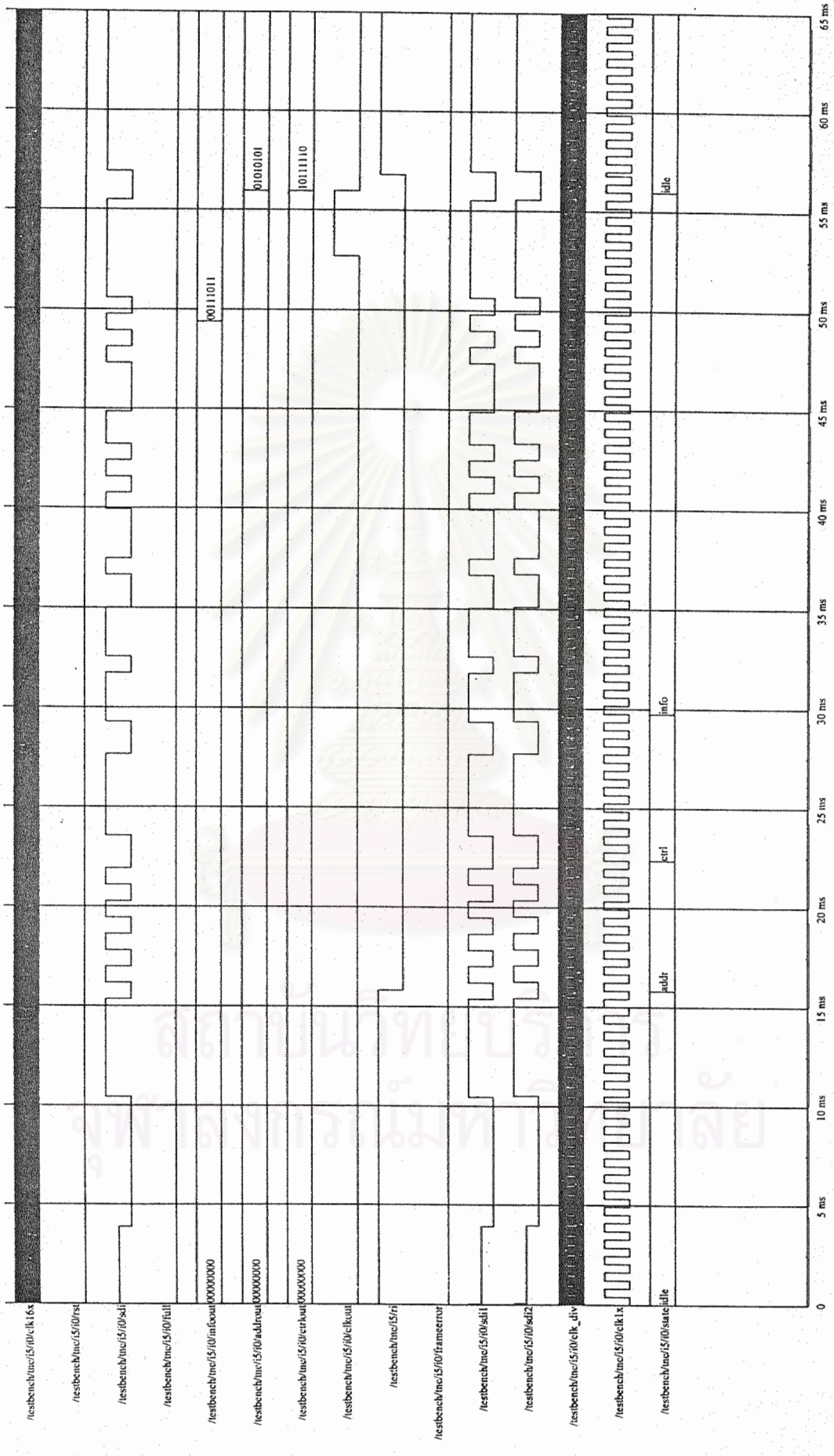
แสดงการทำงานของโมดูล HDLC_TX (14)



ผลการจำลองการทำงานของสัญญาณภายใน Hdlc_Tx (I4) (ต่อ)

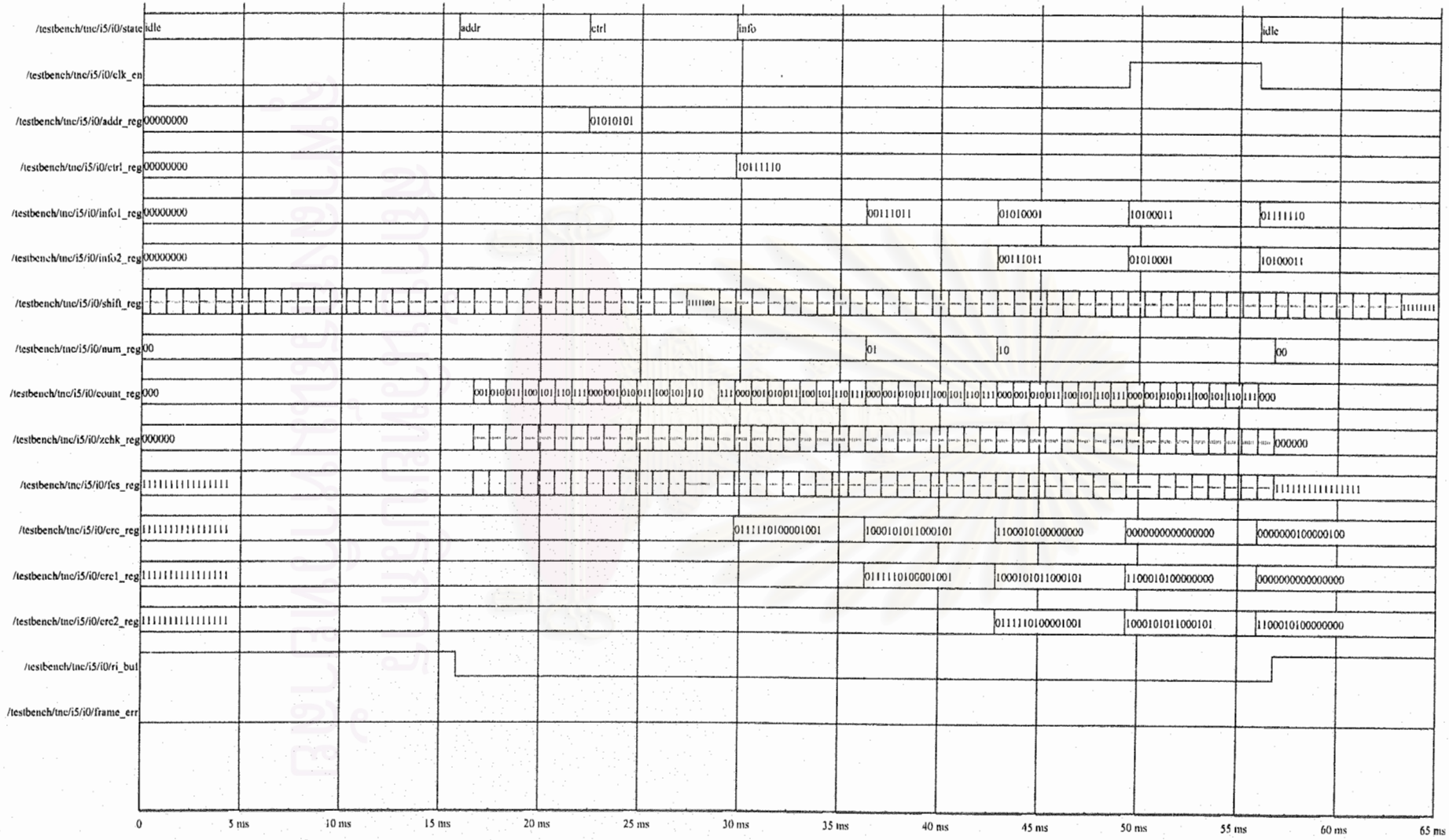


ผลการจำลองการทำงานของสัญญาณภายใน Nrzi_Tx (I9) และ Nrzi_Rx (I10)



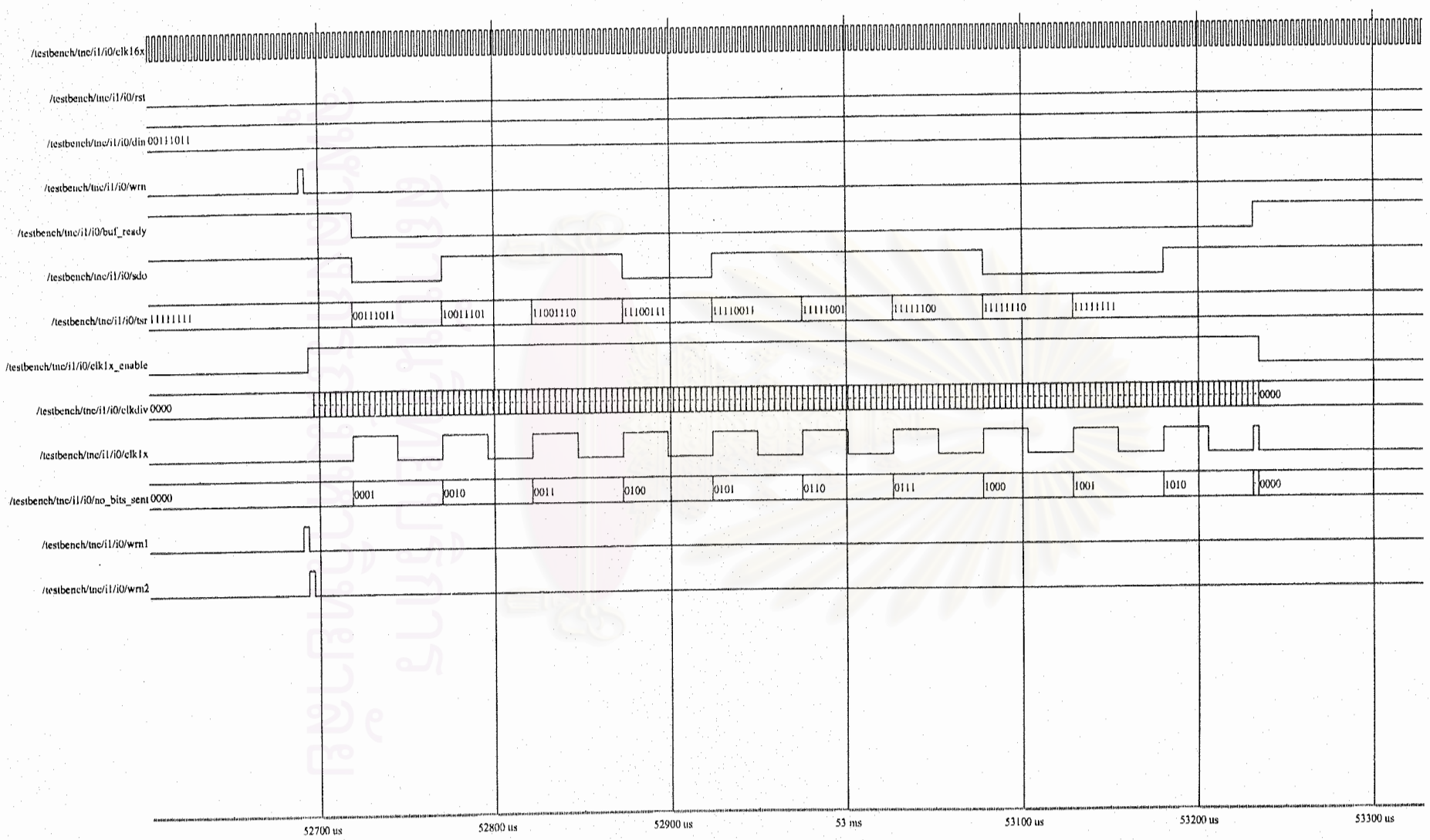
ผลการจำลองการทำงานของสัญญาณภายใน Hdlc_Rx (I5)

ผลการจำลองการทำงานของโมดูล hdlc_RX (15) (ต่อ)



Entity:testbench Architecture:behavior Date: Fri May 16 03:19:48 SE Asia Standard Time 2003 Row: 1 Page: 1

ผลการจำลองการทำงานของสัญญาณภายใน Fifo_Rx (I3)



Entity:testbench Architecture:behavior Date: Fri May 16 07:25:16 SE Asia Standard Time 2003 Row: 1 Page: 1

ประวัติผู้เขียนวิทยานิพนธ์

นายอุดมศักดิ์ ใช้ศรีทอง เกิดวันที่ 1 กันยายน พ.ศ. 2520 ที่จังหวัดนครปฐม สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ. 2542 และศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีเดียวกัน ปัจจุบันทำงานในตำแหน่งผู้ช่วยบรรณาธิการ วารสารเซมิคอนดักเตอร์อิเล็กทรอนิกส์ บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)