การสื่อสารแบบหนึ่งอนุกรมสองทางสำหรับการติดต่อระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์

นายภาสกร ทองสันตติ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2559

FullDOSC : A Software-Defined Inter-Processor Communication

Mr. Pasakorn Tongsan

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2016

หัวข้อวิทยานิพนธ์	การสื่อสารแบบหนึ่งอนุกรมสองทางสำหรับการติดต่อระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์
โดย	นายภาสกร ทองสันตติ์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(รองศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา)

.....กรรมการ

(รองศาสตราจารย์ ดร.กุลธิดา โรจน์วิบูลย์ชัย)

.....กรรมการภายนอกมหาวิทยาลัย

(ดร.พงศ์ธวัช ชีพพิมลชัย)

ภาสกร ทองสันตติ์ : การสื่อสารแบบหนึ่งอนุกรมสองทางสำหรับการติดต่อระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์ (FullDOSC : A Software-Defined Inter-Processor Communication) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร.เกริก ภิรมย์โสภา, 77 หน้า.

ในงานวิจัยนี้ผู้วิจัยได้นำเสนอการติดต่อระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์ (SDIPC) ซึ่งเป็นการประยุกต์ใช้ข่ายงานที่กำหนดได้ด้วยซอฟต์แวร์ (SDN) เพื่อใช้สื่อสารข้อมูลในระดับล่างหรือระดับพื้นฐานระหว่างหน่วยประมวลผล และเพื่อที่จะนำหลักการของ SDIPC ให้สามารถใช้งานได้จริง ผู้วิจัยได้นำเสนอการสื่อสารแบบหนึ่งอนุกรมสองทาง (FullDOSC) ซึ่งเป็นแพลตฟอร์มที่ใช้เพียงหนึ่งส่วนต่อประสานแบบอนุกรม การสื่อสารแบบหนึ่งอนุกรมสองทางนี้จะช่วยลดปัญหาในข้อจำกัดทางด้านฮาร์ดแวร์ของระบบโดยเฉพาะระบบฝังตัว (Embedded System) นอกจากนั้นผู้วิจัยได้ทำการออกแบบภาษาเฉพาะงาน (DSL) เรียกว่าภาษา FullDOSC Routing Language (FRL) เพื่อที่จะทำให้การสื่อสารสามารถกำหนดเพื่อเปลี่ยนแปลงได้ ในช่วงเวลาทำงาน และผู้วิจัยก็ได้ทำการดัดแปลงภาษา FRL เรียกว่า modified FRL (mFRL) เพื่อรองรับความต้องการมากขึ้น ผู้วิจัยได้พิสูจน์ความสามารถของภาษา และทำการนำใช้ใน FPGA โดยใช้ภาษา Verilog HDL เพื่อเป็นการแสดงการทำงานในการจำลองและนำไปใช้งานฮาร์ดแวร์จริง และแสดงถึงความยืดหยุ่นและความสามารถในการนำกลับมาใช้ใหม่สำหรับระบบฝังตัว

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2559

5770559021 : MAJOR COMPUTER ENGINEERING

KEYWORDS: FIELD PROGRAMMABLE GATE ARRAY (FPGA) / INTER-PROCESSOR COMMUNICATION (IPC) / SOFTWARE-DEFINED NETWORK (SDN) / DOMAIN SPECIFIC LANGUAGE / EMBEDDED SYSTEM / SOFTWARE-DEFINED INTER-PROCESSOR COMMUNICATION (SDIPC) / FULL-DUPLEX ONE SERIAL COMMUNICATION (FULLDOSC)

PASAKORN TONGSAN: FullDOSC : A Software-Defined Inter-Processor Communication. ADVISOR: ASST. PROF. KRERK PIROMSOPA, Ph.D., 77 pp.

We presented Software-Defined Inter-Processor Communication (SDIPC), an application of Software-Define Network (SDN) to low-level communication of processors. To implement the concept of SDIPC, we presented Full-Duplex One Serial Communication (FullDOSC) as a platform that uses only one serial interface of processors which aims to alleviate hardware constraints. This, couple with our proposed DSL, FullDOSC Language (FRL), allows the communication channel to be dynamically reprogrammed at runtime. Then we also proposed and modified version of FRL (mFRL) to support more requirements. We validate our prototype using FPGA and coding on Verilog HDL for simulation and implementation on physical hardware. The prototype shows that SDPIC provides flexible and reusable communication for embedded systems.

Department: Computer Engineering Student's Signature .....
Field of Study: Computer Engineering Advisor's Signature .....
Academic Year: 2016

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงลงด้วยดี มีองค์ประกอบมาจากส่วนสำคัญที่ขาดไม่ได้ คือ ความรู้ทางวิชาการที่คณาจารย์ ในภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยทุกท่านที่ได้ประสิทธิ์ประสาทให้แก่ข้าพเจ้า อีกทั้งประกอบกับคำแนะนำต่างๆที่เป็นประโยชน์ ต่อการดำเนินการวิจัย โดยเฉพาะอย่างยิ่ง การดูแล และ ให้ความช่วยเหลืออย่างสม่ำเสมอ จาก ผศ. ดร.เกริก ภิรมย์โสภา อาจารย์ที่ปรึกษาวิทยานิพนธ์ของข้าพเจ้า ซึ่งเป็นแบบอย่างทั้งในด้านการศึกษา และยังเป็นแบบอย่างสำคัญในด้านการดำเนินชีวิต ข้าพเจ้าจึงขอกราบแสดงความขอบพระคุณในความเมตตากรุณามา ณ ที่นี้

ขอขอบพระคุณคณาจารย์ และ คณะกรรมการสอบวิทยานิพนธ์ทุกท่าน ได้แก่ ผศ. ดร. ณัฐวุฒิ หนูไพโรจน์ (ประธานกรรมการสอบวิทยานิพนธ์), รศ. ดร. กุลธิดา โรจน์วิบูลย์ชัย (กรรมการสอบสอบวิทยานิพนธ์) และ ดร. พงศ์ธวัช ชีพพิมลชัย (กรรมการจากภายนอกมหาวิทยาลัย) ที่ได้ชี้แนะแนวคิดในการพัฒนาการวิจัย แนวทางการปรับปรุงแก้ไข เพิ่มเติมในส่วนที่บกพร่อง และ รวมถึงชี้แนะวิธีการนำเสนอ เพื่อให้งานวิจัยสำเร็จลุล่วงครบถ้วนตามเป้าหมาย

ขอขอบคุณทุนอัจฉริยะคืนรังจากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยซึ่งได้สนับสนุนค่าเล่าเรียน ในภาคการศึกษาที่ลงทะเบียนตามจำนวนที่จ่ายจริง เป็นเวลา 4 ภาคการศึกษา นับตั้งแต่เริ่มเข้าศึกษา

สุดท้ายนี้ขอกล่าวคำขอบพระคุณ รายนามบุคคลดังต่อไปนี้ ซึ่งมีคุโณปการสำคัญยิ่งต่องานวิจัยฉบับนี้ อันประกอบด้วย นายสมิทธ์ ธรรมบำรุง ,นายกรกฤต สีมาคุปต์ และ อ. ดร. พิชญะ สิทธิอมร ที่ได้เอื้อเฟื้อทั้งแรงกาย กำลังสติปัญญา ความรู้ ประสบการณ์ และ คำแนะนำในการดำเนินงาน และการแก้ไขปัญหาต่างๆตลอดมา งานวิจัยนี้คงสำเร็จลงได้ยาก หากปราศจากความเอื้อเฟื้อจากทุกท่าน จึงขอเอ่ยนามเพื่อแสดงความขอบคุณสำหรับความกรุณาไว้ ณ ที่นี้

สารบัญ

หน้า

หน้า

หน้า

1. ที่มาและความสำคัญ

ในปัจจุบันมีการใช้งานระบบคอมพิวเตอร์ที่มีหลายหน่วยประมวลผล (multi-processor computer system) เพื่อที่จะได้รับประโยชน์เหนือระบบที่หน่วยประมวลผลเดียวในหลายด้าน

การทำงานในระบบแบบทันกาล (real-time operation) ระบบทันกาลคือระบบที่มีงานตามเวลาที่กำหนด และเนื่องจากการทำงานของระบบหนึ่งๆมักจะมีการทำงานหลายๆงานร่วมกันในเวลาเดียวกัน ซึ่งถ้าเป็นระบบที่มีหน่วยประมวลผลเดี่ยวมักจะใช้การสลับการทำงานแต่ละงานของหน่วยประมวลผล วิธีนี้จะทำให้งานแต่ละงานไม่สามารถทำงานได้พร้อมกันจริงได้ ส่งผลให้เวลาของผลการทำงานมีการคลาดเคลื่อน การใช้ระบบที่มีหลายหน่วยประมวลผลจะสามารถแก้ปัญหาในส่วนนี้ได้ เนื่องจากสามารถแจกงานหลายๆ งานให้แก่หน่วยประมวลผลแต่ละหน่วยให้ทำงานพร้อมๆกันได้ และยังสามารถจัดสรรงานให้เหมาะกับชนิดของหน่วยประมวลผลเพื่อเพิ่มประสิทธิภาพในการทำงานอีกด้วย

หน่วยประมวลผลจำเป็นต้องมีการติดต่อสื่อสารกับระบบภายนอกผ่านทางช่องทางต่างๆ ซึ่งการติดต่อเหล่านี้หน่วยประมวลผลจำเป็นต้องใช้อุปกรณ์รอบข้าง (Peripheral) ที่อยู่ภายในชิป ถึงแม้จะเป็นหน่วยประมวลผลแบบไมโครคอนโทรลเลอร์ (microcontroller) ที่ออกแบบมาเน้นในส่วนของอุปกรณ์รอบข้างมากกว่าในเรื่องสมรรถนะของการประมวลผล แต่เนื่องจากพื้นที่ที่จำกัดจึงทำให้จำนวนของอุปกรณ์รอบข้างนี้มีจำนวนที่จำกัดด้วย ดังนั้น เมื่อต้องการใช้การติดต่อสื่อสารกันภายนอกระบบจึงจำเป็นต้องมีส่วนต่อขยาย และวิธีที่จะติดต่อส่วนขยายเหล่านี้ได้อย่างมีประสิทธิภาพคือการเพิ่มจำนวนหน่วยประมวลผลในระบบ

ในกรณีที่มีการจัดการระบบที่ดีพอ เช่น การใช้ ระบบปฏิบัติการแบบทันกาล (Real-Time Operating System : RTOS) หรือการจัดสรรการแบ่งงานในเวลาที่กำหนด เมื่อทำการเพิ่มจำนวนของหน่วยประมวลผลแล้วจะทำให้สามารถเพิ่มสมรรถนะ (Performance) ของระบบได้ เนื่องจากความสามารถในการประมวลผลรวมของระบบเพิ่มขึ้น

จากตัวอย่างประโยชน์ที่ได้ยกขึ้นมาทั้งสามด้านแล้วจะเห็นได้ว่าระบบที่มีหลายหน่วยประมวลผลนั้นมีประโยชน์เพิ่มขึ้นจากระบบที่มีหน่วยประมวลผลเดียวหรือมีจำนวนหน่วยประมวลผลที่น้อยกว่า แต่นอกจากการจัดสรรแบ่งงานงานแล้ว การพัฒนาทอพอโลยีเพื่อปรับปรุงประสิทธิภาพการติดต่อสื่อสารกันระหว่างหน่วยประมวลผล เป็นอีกหนึ่งวิธีที่จะเพิ่มประสิทธิภาพของระบบได้ และ

เป็นแนวทางที่ได้เลือกนำมาใช้ในวิทยานิพนธ์นี้เนื่องจากสามารถประยุกต์ใช้ในฮาร์ดแวร์ที่ใช้โดยทั่วไปในปัจจุบันรวมถึงไมโครคอนโทรลเลอร์ที่มีทรัพยากรที่มีจำกัด

2. ทฤษฎีที่เกี่ยวข้อง

วิทยานิพนธ์นี้ การสื่อสารแบบหนึ่งอนุกรมสองทางสำหรับการติดต่อระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์, เป็นการนำเสนอรูปของข่ายงานสำหรับติดต่อสื่อสารของอุปกรณ์ที่มีทรัพยากรที่จำกัดโดยมีการจัดการอยู่ในระดับล่าง และทำการสร้างซอฟต์แวร์สำหรับจำลองเพื่อประเมินค่าการทำงานดังกล่าว จึงมีทฤษฎีที่เกี่ยวข้องดังนี้

2.1. ส่วนต่อประสานมาตรฐาน (Standard Interface)

ส่วนต่อประสานของหน่วยประมวลผลคือช่องทางการสื่อสารที่จะทำให้หน่วยประมวลผลที่อยู่ต่างชิปกันสามารถส่งข้อมูลหากันได้ ซึ่งได้กำหนดให้มีมาตรฐานกลางเพื่อให้ผู้พัฒนาระบบจากหน่วยงานที่ต่างกันสามารถพัฒนาระบบที่ติดต่อถึงกันได้ และในกรณีที่เป็นมาตรฐานที่เป็นที่นิยมผู้ผลิตอุปกรณ์ต่างๆและหน่วยประมวลผลก็มักจะพัฒนาอุปกรณ์ของตนให้รองรับการทำงานตามมาตรฐานของส่วนต่อประสานนั้นให้มีประสิทธิภาพที่ดียิ่งขึ้น ซึ่งหน่วยมาตรฐานดังกล่าวได้แก่

- Universal Asynchronous Receiver/Transmitter (UART) : เป็นมาตรฐานที่มีรูปแบบการสื่อสารแบบสองทาง (Full-duplex) ที่ไม่ประสานเวลา (Asynchronous) โดยมีช่องทางสำหรับส่งข้อที่นิยมใช้สัญลักษณ์เป็น Tx และช่องทางสำหรับรับข้อมูลที่นิยมใช้สัญลักษณ์เป็น Rx และนอกจากสองช่องทางนี้แล้ว ยังมีช่องสัญญาณอื่นเพื่ออำนวยความสะดวกให้กับการส่งข้อมูล ได้แก่ Request To Send (RTS) เป็นช่องทางส่งออก (Output) สำหรับการบอกให้อุปกรณ์ภายนอกว่าอุปกรณ์นี้พร้อมสำหรับการรับข้อมูลหรือไม่, Clear To Send (CTS) เป็นช่องทางรับเข้า (Input) สำหรับให้อุปกรณ์ภายนอกบอกว่ากำลังส่งข้อมูลอยู่, Serial clock (SCLK) เป็นช่องทางส่งออกและรับเข้าสำหรับการให้จังหวะการส่งข้อมูล ซึ่ง UART ที่มีช่องทางดังกล่าวเรียกว่า Universal Asynchronous/Synchronous Receiver/Transmitter (USART) ซึ่งจะทำให้สามารถทำงานแบบประสานเวลา (Synchronous)

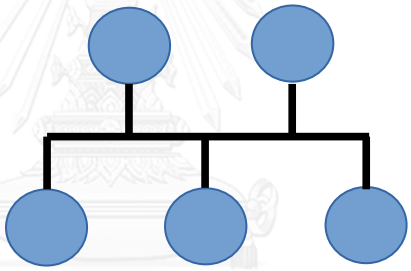
- Serial Peripheral Interface (SPI) : เป็นมาตรฐานที่มีรูปแบบการสื่อสารแบบสองทาง (Full-duplex) ที่ประสานเวลา (Synchronous) เป็นแบบมีอุปกรณ์ผู้ควบคุมและภายใต้การควบคุม (Master-Slave) โดยมีช่องทางสำหรับส่งข้อมูลจากผู้ควบคุมไปยังผู้ภายใต้การควบคุม (MOSI), ช่องทางสำหรับส่งข้อมูลจากผู้ภายใต้ควบคุมไปยังผู้การควบคุม (MISO), ช่องทางสำหรับให้สัญญาณนาฬิกา (SCLK) และ ช่องทางสำหรับเลือกผู้ภายใต้การควบคุมที่จะทำการสื่อสาร (SS)

- Inter-Integrated Circuit (I^2C) : เป็นมาตรฐานที่มีรูปแบบการสื่อสารแบบทางเดียว (Half-duplex) ที่ประสานเวลา (Synchronous) โดยมีช่องทางสำหรับส่งข้อมูล (SDA) และช่องทางสำหรับให้สัญญาณนาฬิกา (SCL)

2.2. ภูมิลักษณะของข่ายงาน (Network Topology) [1]

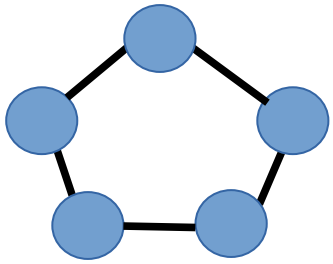
ข่ายงาน (Network) คือการที่มีอุปกรณ์จำนวนมากกว่าหนึ่งหน่วยทำการเชื่อมต่อเพื่อทำการสื่อสารกันผ่านทางช่องทางการสื่อสารใดๆ ภูมิลักษณะของข่ายงาน (Network Topology) คือการอธิบายผังของการเชื่อมต่อในข่ายงานซึ่งสามารถแบ่งตามระดับของมุมมองที่สนใจได้สองประเภทคือ ทอพอโลยีระดับกายภาพ (Physical Topology) และ ทอพอโลยีระดับตรรกะ (Logical Topology) ซึ่งทอพอโลยีระดับกายภาพหมายถึงรูปแบบการเชื่อมต่อของสายสัญญาณระหว่างอุปกรณ์ภายในระบบในโลกจริง ส่วนทอพอโลยีระดับตรรกะหมายรูปแบบเส้นทางการส่งข้อมูลกันระหว่างอุปกรณ์ และนอกจากแบ่งตามระดับของมุมมองที่สนใจยังสามารถแบ่งตามรูปแบบได้เป็นห้าประเภทดังนี้

ทอพอโลยีแบบบัส (Bus Topology)



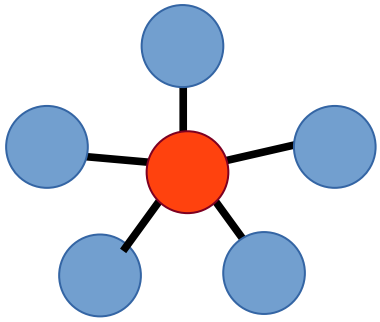
รูปที่ 1: แผนภาพอธิบายโครงสร้างข่ายงานทอพอโลยีแบบบัส

ทอพอโลยีแบบวงแหวน (Ring Topology)



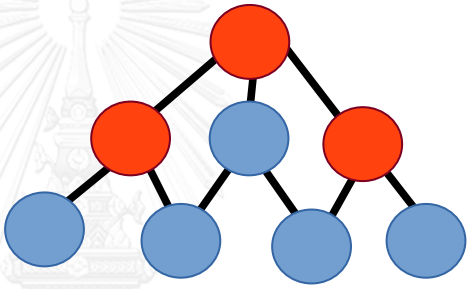
รูปที่ 2: แผนภาพอธิบายโครงสร้างข่ายงานทอพอโลยีแบบวงแหวน

ทอพอโลยีแบบดาว (Star Topology)



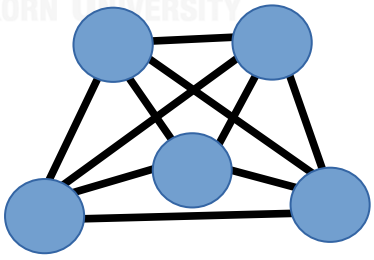
รูปที่ 3: แผนภาพอธิบายโครงสร้างข่ายงานทอพอโลยีแบบดาว

ทอพอโลยีแบบต้นไม้ (Tree Topology)



รูปที่ 4: แผนภาพอธิบายโครงสร้างข่ายงานทอพอโลยีแบบต้นไม้

ทอพอโลยีแบบตาข่าย (Mesh Topology)

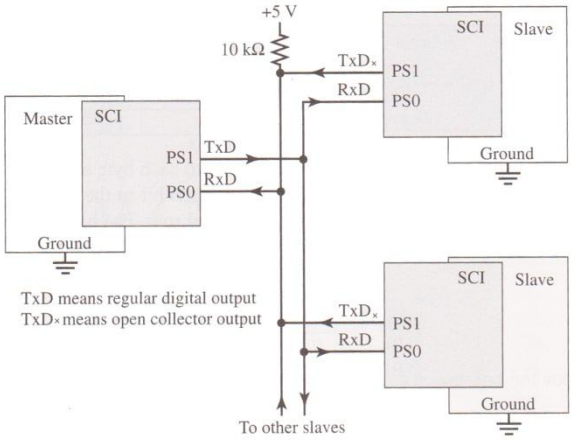


รูปที่ 5: แผนภาพอธิบายโครงสร้างข่ายงานทอพอโลยีแบบตาข่าย

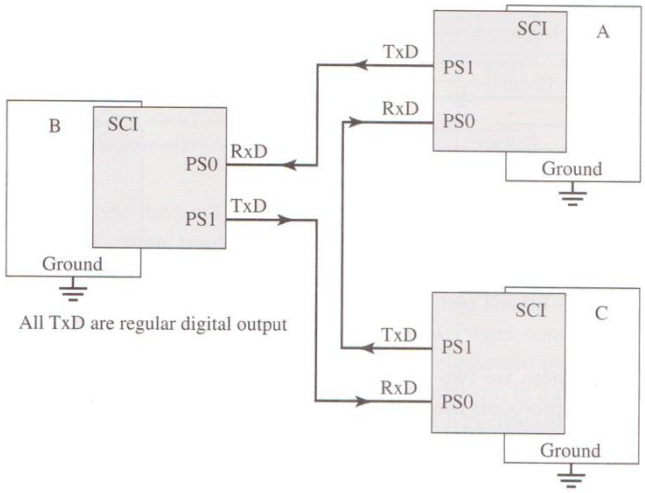
ในวิทยานิพนธ์นี้เป็นการออกแบบสำหรับการติดต่อสื่อสารสำหรับอุปกรณ์ที่มีทรัพยากรที่จำกัด ดังนั้นรูปแบบของทอพอโลยีที่เลือกจึงจำเป็นต้องมีการเชื่อมต่อของทุกอุปกรณ์ที่น้อยที่สุด คือ ใช้เพียงหนึ่งช่องทางการติดต่อแบบสองทาง (Full-Duplex) เท่านั้น ในหนึ่งข่ายงานทอพอโลยีที่

สามารถตอบโจทย์ดังกล่าวได้อย่างชัดเจนคือทอพอโลยีแบบบัส ทอพอโลยีแบบตาข่ายนั้นไม่เหมาสมสำหรับงานนี้อย่างยิ่ง เพราะถึงแม้จะทำให้ระบบสามารถสื่อสารกันได้อย่างมีสมรรถนะ (Performance) ดีที่สุด แต่อุปกรณ์จะต้องใช้ช่องทางการสื่อสารจำนวนมาก ไม่เหมาะสำหรับอุปกรณ์ที่มีทรัพยากรที่จำกัด ทอพอโลยีแบบดาวจำเป็นต้องมีศูนย์กลางการเชื่อมต่อซึ่งอุปกรณ์นี้ต้องสามารถรองรับการเชื่อมต่อของทั้งระบบได้ และมักจะเป็นคอขวดของการสื่อสารกันของระบบจึงไม่เหมาะที่จะนำมาใช้ในงานนี้ ทอพอโลยีแบบต้นไม้เป็นการเชื่อมต่อโดยส่งข้อมูลจากราก (root) ไปยังอุปกรณ์อื่นๆหรือโหนด (Node) อื่นๆ ดังนั้นระบบนี้จำเป็นต้องมีช่องทางสำหรับการติดต่อทั้งกับโหนดแม่และโหนดลูกแต่ละโหนด เพื่อให้สอดคล้องกับความต้องการเรื่องการใช้ทรัพยากร ทอพอโลยีแบบต้นไม้จึงต้องมีลูกเพียงโหนดละหนึ่ง และเนื่องจากเป็นการติดต่อแบบสองทางในช่องทางเดียวกันคือรับและส่งข้อมูล จึงสามารถแบ่งได้เป็นรับจากโหนดแม่ และส่งไปยังโหนดลูก และเพื่อให้สามารถส่อสารกันได้ครบทั้งระบบข่ายงาน โหนดลูกสุดท้ายจึงต้องส่งข้อมูลไปยังโหนดราก ซึ่งเมื่อพิจารณาแล้ว ทอพอโลยีดังกล่าวจึงจะมีลักษณะเป็น ทอพอโลยีแบบวงแหวน ดังนั้นรูปแบบทอพอโลยีที่จะใช้ในวิทยานิพนธ์นี้จึงเป็นแบบบัสและแบบวงแหวน

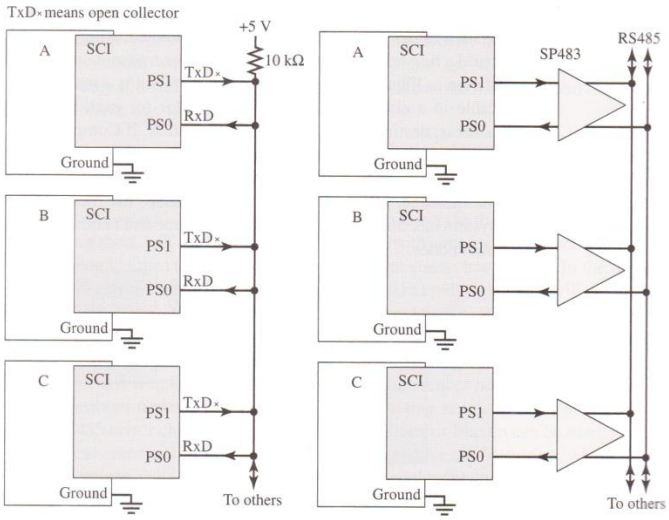
จากที่กล่าวไปนั้น ตรงกับในหนังสือ Embedded Mivrocomputer Systems Real Time Interfacing โดย Jonathan W. Valvano[2] ในหัวข้อ Communication System Based on the SCI Serial Port ซึ่งแบ่งเป็น master-slave , multi-drop และ ring network ดังรูป



รูปที่ 6 แผนภาพแสดงการเชื่อมต่อในรูปแบบ master-slave



รูปที่ 7 แผนภาพแสดงการเชื่อมต่อในรูปแบบ ring network



รูปที่ 8 แผนภาพแสดงการเชื่อมต่อในรูปแบบ multi-drop

2.3. ข่ายงานที่กำหนดได้ด้วยซอฟต์แวร์ (Software-Defined Networking)

โดยส่วนมากข่ายงานคอมพิวเตอร์ (Computer Network) มักจะประกอบไปด้วยอุปกรณ์จำนวนมากที่ทำการเชื่อมต่อกันในข่ายงาน ได้แก่ อุปการณ์ที่ต้องการส่งและรับข้อมูล และอุปกรณ์ที่ทำหน้าที่เป็นตัวจัดการและคุมการสื่อสารเหล่านี้ เช่น สวิตช์ (Switch), เราต์เตอร์ (Router) เป็นต้น ซึ่งหน้าที่ของอุปกรณ์ควบคุมเหล่านี้คือควบคุมการสื่อสารภายในระบบให้เป็นไปตามนโยบายที่ได้กำหนดไว้ และป้องกันการโจมตีที่เกิดจากภายนอกระบบ

ในปัจจุบันมีการใช้งานข่ายงานคอมพิวเตอร์กันอย่างมาก มีรูปแบบ จำนวนของอุปกรณ์ และปริมาณของข้อมูลเพิ่มมากขึ้นซึ่งทำให้การดูแลและซ่อมบำรุงระบบทำได้ยาก รวมถึงการพัฒนาของรูปแบบในการโจมตีจากภายนอก การติดตั้งข่ายงานคอมพิวเตอร์แบบเดิมซึ่งคือการมีระบบโครงสร้างของข่ายงานที่มีลักษณะตายตัวจึงไม่สามารถตอบโจทย์การใช้งานในปัจจุบันได้ นำไปสู่แนวคิดเรื่องระบบข่ายงานที่สามารถปรับเปลี่ยนชุดคำสั่งได้ (Programmable network) โดยเฉพาะอย่างยิ่งข่ายงานที่กำหนดได้ด้วยซอฟต์แวร์ (Software-Defined Networking : SDN) เป็นกระบวนทัศน์ (Paradigm) ใหม่ที่จะมาตอบโจทย์ในปัญหาดังกล่าว โดยมีองค์ประกอบแบ่งออกเป็นสองระนาบคือ ระนาบควบคุม (Control plane) เป็นระนาบที่ทำการควบคุมการทำงานตามเงื่อนไขที่ซอฟต์แวร์ได้ระบุไว้ และระนาบข้อมูล (Data plane) เป็นการส่งข้อมูลอย่างง่าย การแบ่งออกเป็นสองระนาบนี้ส่งผลให้มีการดูแลควบคุมการส่งผ่านข้อมูลของทั้งระบบมาจากจุดเดียวซึ่งง่ายต่อการดูแลและปรับเปลี่ยน ส่วนอุปกรณ์ที่ใช้ในการส่งผ่านข้อมูลก็จะอยู่ในรูปแบบที่ง่ายขึ้นเนื่องจากความซับซ้อนจะถูกโยนไปให้ระนาบควบคุมเป็นผู้จัดการ

3. งานวิจัยที่เกี่ยวข้อง

ในปัจจุบันมีงานวิจัยเกี่ยวการการสื่อสารระหว่างหน่วยประมวลผล (Inter-Processor Communication) อยู่จำนวนมาก ซึ่งสามารถแบ่งออกได้เป็นสามประเภทตามการทำงานและแนวคิดเกี่ยวกับการออกแบบดังนี้

3.1. การติดต่อสื่อสารระหว่างหน่วยประมวลผลหลายตัวที่แยกส่วนกัน

ระบบหลายหน่วยประมวลผลแบบที่หน่วยประมวลผลแยกส่วนกัน (Multiple Discrete Processor System) คือหน่วยประมวลผลแต่ละตัวในระบบอยู่ในชิปที่แตกต่างกัน การสื่อสารระหว่างหน่วยประมวลผลนั้นจึงต้องอาศัยส่วนต่อประสาน (Interface) ต่างๆ เช่น I^2C, SPI หรือ UART ในปี 2011 Liu Zhaoqing ได้เสนองานวิจัย “A Design of Multiprotocol Asynchronous Serial Communication Based on M Module”[3] ซึ่งได้ออกแบบการสื่อสารกันระหว่างหน่วยประมวลผลที่ใช้รูปแบบของช่องทางการติดต่อหลายรูปแบบและไม่ได้รองรับการทำงานที่ยืดหยุ่นโดยใช้การปรับเปลี่ยนทอพอโลยี ในปี 2013 Keqiu Li ได้เสนองานวิจัย “Exchanged Crossed Cube: A Novel Interconnection Network for Parallel Computation”[4] ซึ่งเป็นงานวิจัยที่ได้ออกแบบและวิเคราะห์รูปแบบการติดต่อสื่อสารระหว่างหน่วยประมวลผลที่เน้นเรื่องสมรรถนะในการคำนวณแบบขนาน แต่รูปแบบนี้จะไม่เหมาะสำหรับอุปกรณ์ที่มีทรัพยากรจำกัด

3.2. การติดต่อสื่อสารแบบไร้สายระหว่างหน่วยประมวลผลหลายตัวที่แยกส่วนกัน

การติดต่อแบบไร้สายระหว่างหน่วยประมวลผลมีเงื่อนไขของการพัฒนาที่แตกต่างกันอย่างชัดเจนเมื่อเทียบกับการใช้ช่องทางที่มีสายดังในข้อ 3.1 เนื่องจากมีปัจจัยต่างๆ ทางสภาพแวดล้อมเพิ่มขึ้นและการใช้งานก็จะอยู่ในรูปแบบที่ต่างออกไป ตัวอย่างงานวิจัยที่ยกมาอ้างอิงในงานวิจัยนี้ได้แก่ “A Green Software-Defined Communication Processor for Dynamic Spectrum Access”[5] โดย Ching-Kai Liangและ Kwang-Cheng Chen ซึ่งจะเห็นได้อย่างชัดเจนถึงความแตกต่างของการศึกษาการสื่อสารในแบบไร้สายของหน่วยประมวลผล ในงานนี้จะเน้นไปในเรื่องการใช้พลังงานให้มีประสิทธิภาพ เนื่องจากอุปกรณ์ไร้สายมักจะต้องใช้พลังงานภายในตัวเอง แต่ก็มีงานวิจัยที่ใกล้เคียงกับงานวิจัยนี้แต่เป็นแบบไร้สายซึ่งได้แก่ “Hardware-software co-design for heterogeneous multiprocessor sensor nodes”[6] โดย J. Zhang ในงานนี้จะมีการออกแบบโดยใช้โปรแกรมจำลองชื่อ SUNSHINE และทำการพิสูจน์โดยการสร้างระบบขึ้นมาจริงโดยใช้ FPGA และหน่วยประมวลผลแบบ microcontroller แต่เนื่องจากเป็นการสื่อสารแบบไร้สาย ทำให้ไม่สามารถมีทอพอโลยีแบบกายภาพได้

3.3. การติดต่อสื่อสารระหว่างหน่วยประมวลผลหลายตัวที่อยู่ในชิปเดียวกัน

การติดต่อสื่อสารระหว่างหน่วยประมวลผลหลายตัวที่อยู่ในชิปเดียวกันหรือเรียกว่า Network-on-Chip (NoC) เป็นการสื่อสารของหน่วยประมวลผลที่อยู่ภายในระบบที่อยู่ในชิปเดียวกันหรือเรียกว่า System-on-Chip (SoC) ซึ่งจะทำให้มีข้อจำกัดด้านช่องทางการติดต่อสื่อสารหากันน้อยกว่ารูปแบบอื่น แต่สามารถสร้างความยืดหยุ่นได้มากเช่นในงานวิจัยของ Ando Y. ชื่อ “Automatic synthesis of inter-heterogeneous-processor communication implementation for programmable system-on-chip”[7] ซึ่งได้แสดงให้เห็นถึงการออกแบบที่สามารถปรับเปลี่ยนได้ (Dynamic) ในระบบในชิปเดียวกัน และนอกจากงานวิจัยนี้แล้ว “COSI: A Framework for the Design of Interconnection Networks”[8] เป็นหนึ่งในงานวิจัยที่สามารถเป็นพื้นฐานในการออกแบบการสื่อสารภายในชิปได้ดี

4. วัตถุประสงค์

1. เพื่อกำหนดรูปแบบหรือแบบจำลองสำหรับการอธิบายรูปแบบการสื่อสารที่เหมาะสำหรับการใช้งานในหน่วยประมวลผลที่มีข้อจำกัดทางทรัพยากร เพื่อที่ใช้ในระบบการติดต่อระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์
2. เพื่อพัฒนาระบบต้นแบบที่ปรับเปลี่ยนรูปแบบการสื่อสารได้ตามรูปแบบที่กำหนด

5. ขอบเขตการทำงาน

1. ออกแบบและสร้างแพลตฟอร์มการสื่อสารแบบหนึ่งอนุกรมสองทาง (Full-Duplex One Serial Communication : FullDOSC) สำหรับใช้อธิบายการติดต่อสื่อสารระหว่างหน่วยประมวลผลผ่านช่องทางการติดต่อที่มีลักษณะเป็นอนุกรม (Serial) และเป็นการติดต่อสองทาง (Full-Duplex) เช่น UART เพื่อให้สามารถประยุกต์ใช้กับการติดต่อระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์ได้
2. ออกแบบภาษา ซึ่งใช้สำหรับอธิบายรูปแบบของระบบที่ใช้ในการกำหนดในข้อ 1.

3. พิสูจน์ความสามารถขอภาษาในการอธิบายรูปแบบในข้อ 2. และรูปแบบทั่วไปของการเชื่อมต่อในข้อ 1. อธิบายโดยใช้ภาษาในข้อ 2.
4. ทำการใช้งานระบบในข้อ 1. ในรูปแบบการจำลองการทำงานและใช้งานในอุปกรณ์จริง

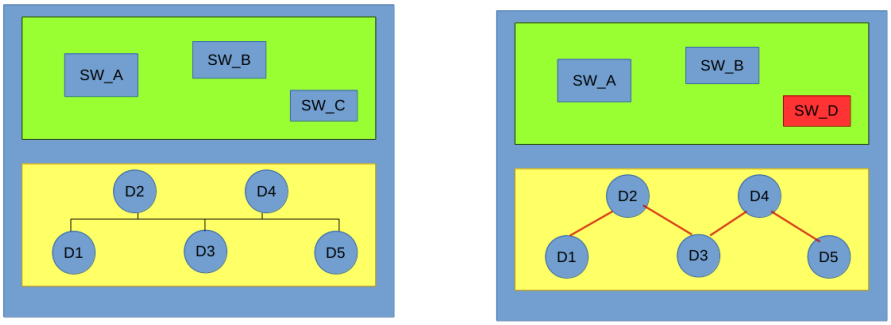
6. การสื่อสารระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์ (SDIPC)

การสื่อสารระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์ (Software-Defined Inter-Processor Communication : SDIPC) คือการประยุกต์ใช้หลักการของ SDN เพื่อใช้ในการสื่อสารระหว่าหน่วยประมวลผล รูปแบบของ SDN ส่วนใหญ่ได้ถูกออกแบบเพื่อใช้ในข่ายงานบริเวณเฉพาะที่ (Local Area Network : LAN) กับคอมพิวเตอร์ส่วนบุคคลและในระบบเครือข่ายทั่วไป เนื่องจากความซับซ้อนในการทำงานดังกล่าว จึงทำให้การนำหลักการของ SDN มาใช้ในการสื่อสารระดับล่างอย่างเช่นการสื่อสารระหว่างหน่วยประมวลผลมีส่วนประกอบที่ไม่จำเป็น (overhead) อย่างมาก ในงานวิจัยนี้จึงมุ่งเน้นในการแก้ไขปัญหาดังกล่าว จึงได้ทำการนิยาม SDIPC ขึ้นมา และใช้ในระบบฝังตัวเป็นสำคัญ เนื่องจากระบบฝังตัว (Embedded System) ซึ่งมีข้อจำกัดทางด้านต้นทุนของฮาร์ดแวร์เป็นหนึ่งในปัจจัยของการออกแบบ ดังนั้นการที่จะใช้รูปแบบการสื่อสารที่มีประสิทธิภาพสูงเกินจำเป็นอย่างเช่น Intel QuickPath Interconnect[9] จึงมักจะไม่เหมาะสม และระบบฝังตัวที่ใช้ microcontroller (MCU) นั้นก็มักจะมีส่วนต่อประสานมาตรฐานให้มาด้วย

จากแนวคิดดังกล่าวสามารถนำมาใช้ประโยชน์ในการสื่อสารระหว่างหน่วยประมวลผลได้ในสองประการได้แก่ ความสามารถในการนำมาใช้ใหม่ของซอฟต์แวร์ (reusability) และ ความสามารถในการปรับตัวของระบบ (adaptability)

6.1. ความสามารถในการนำมาใช้ใหม่ของซอฟต์แวร์ (reusability)

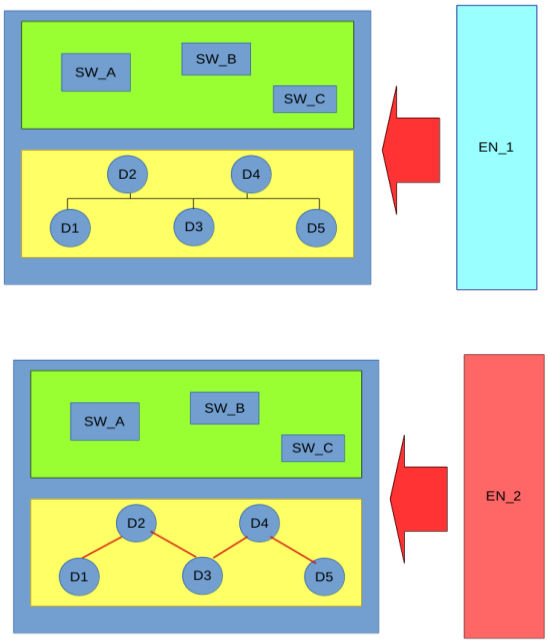
ความสามารถในการนำมาใช้ใหม่ของซอฟต์แวร์นั้น ตัวอย่างเช่น รูปที่ 9 แผนภาพสถานการณ์แสดงถึงความสามารถในการใช้ใหม่ของซอฟต์แวร์ ในกรณีแรก (แผนภาพทางซ้าย) ระบบได้ทำการพัฒนาขึ้นมาแล้วประกอบด้วยส่วนซอฟต์แวร์ SW_A, SW_B, SW_C อยู่บนอุปกรณ์ D1, D2, D3 และเหมาะสมสำหรับระบบดังกล่าว และถ้าระบบต้องนำมาใช้ใหม่ในอีกสถานการณ์หนึ่ง ระบบได้ใช้ SW_A, SW_B ซึ่งมีความสัมพันธ์กับรูปแบบอุปกรณ์เดิม และเพิ่มเติมในส่วน SW_D แต่รูปแบบการเชื่อมต่อของอุปกรณ์เปลี่ยนไป ซึ่งจะส่งผลที่ทำให้ SW_A และ SW_B ไม่สามารถทำงานได้ การที่มีโมเดลที่จะในการอธิบายรูปแบบของการเชื่อมต่อของอุปกรณ์ดังกล่าวและมีตัวจัดการอย่างเช่น Middleware หรือ ส่วนเสริมของตัวแปลภาษา (Compiler)



รูปที่ 9 แผนภาพสถานการณ์แสดงถึงความสามารถในการใช้ใหม่ของซอฟต์แวร์

6.2. ความสามารถในการปรับตัวของระบบ (adaptability)

ความสามารถในการปรับตัวของระบบนั้น ตัวอย่างเช่น รูปที่ 10 ในกรณีแรก (แผนภาพบน) ระบบได้พัฒนาให้เหมาะสมกับสถานการณ์ EN_1 ตามรูปแบบการเชื่อมต่อดังกล่าว แต่เมื่อสถานการณ์เปลี่ยนไปเป็น EN_2 การเชื่อมต่อดังกล่าวนั้นอาจจะไม่เหมาะสม อาจจะเนื่องจากขนาดความยาวของข้อมูลที่ส่ง หรือความถี่ในการส่งข้อมูล ดังนั้นรูปอาจจะควรต้องปรับเปลี่ยนเป็นในรูปแบบอื่น (แผนภาพล่าง) ซึ่งการจะทำเล่นนี้ได้นั้นจะต้องมีโมเดลที่จะในการอธิบายรูปแบบของการเชื่อมต่อของอุปกรณ์ดังกล่าวเช่นกัน และในงานนี้จะทำการสร้างระบบต้นแบบเพื่อใช้ในการแก้ปัญหาดังกล่าวด้วย



รูปที่ 10 แผนภาพสถานการณ์แสดงถึงความสามารถในการปรับตัวของระบบ

7. การสื่อสารแบบหนึ่งอนุกรมสองทาง (FullDOSC)

จากหัวข้อที่ 6 จึงได้ออกแบบแพลตฟอร์ม SDIPC สำหรับอุปกรณ์ที่มีทรัพยากรจำกัดเช่นในระบบอุปกรณ์แบบฝังตัว (Embedded System) ซึ่งมีชื่อว่าการสื่อสารแบบหนึ่งอนุกรมสองทาง (Full-Duplex One Serial Communication : FullDOSC) ซึ่งจะใช้เพียงหนึ่งช่องทางที่มีรูปแบบการสื่อสารแบบสองทาง (Full-Duplex) เช่น UART ซึ่งเป็นช่องทางที่มีในหน่วยประมวลผลทั่วไปและเมื่อประเมินราคาแล้วจะมีราคาที่ถูกที่สุด นอกเหนือจากการใช้ GPIO ตามหัวข้อที่ 2.1

7.1. คำจำกัดความ

เนื่องจากการสื่อสารแบบ FullDOSC เป็นแพลตฟอร์มที่ถูกนิยามขึ้นใหม่ในงานวิจัยนี้ ดังนั้นจึงต้องมีการจำกัดความของศัพท์ที่ใช้ในแพลตฟอร์มนี้ดังนี้

- โหนดอุปกรณ์ (Device node) : คืออุปกรณ์ในแพลตฟอร์ม FullDOSC สำหรับประมวลผลในแต่ละหน่วยในระบบ ซึ่งต้องมีช่องทางการสื่อสารแบบอนุกรมสองทาง (Full-Duplex Serial) เพียง 1 ช่องทาง ซึ่งจะประกอบไปด้วยช่องทางสำหรับส่งข้อมูล (TX) และรับข้อมูล (RX) อุปการณ์เหล่านี้จะเป็นอุปกรณ์ที่จะให้ระบบทำงานตามคุณสมบัติของระบบ
- อุปกรณ์จัดการการเชื่อมต่อ หรือ เราต์เตอร์ (Router) : คืออุปกรณ์ในแพลตฟอร์ม FullDOSC สำหรับจัดการการเชื่อมต่อ ซึ่งอาจจะเป็นวงจรรวมเฉพาะงาน (Application Specific Integrated Circuit : ASIC) แต่เพื่อความสะดวกในการพัฒนาในงานวิจัยนี้จะใช้อุปกรณ์ลอจิกแบบโปรแกรมได้ (Field Programmable Gate Array : FPGA) เนื่องจากตัวจัดการการเชื่อมต่อจะต้องมีการจัดการในระดับล่างผ่านวงจรแบบตรรกะรวม (combinational logic)
- ข่ายงาน (Network) : คือข่ายงานในแพลตฟอร์ม FullDOSC และเป็นไปตามเงื่อนไขของแพลตฟอร์มตามกำหนด
- ระบบ (System) : คือ ระบบที่ใช้แพลตฟอร์ม FullDOSC

7.2. ภาษา FullDOSC Routing Language (FRL)

การออกแบบการสื่อสารที่ใช้ในงานวิจัยนี้จึงจำเป็นต่อควบคุมได้ในระดับล่าง และด้วยเหตุผลเดียวกัน การออกแบบภาษาหรือนิพจน์ที่จะให้กำหนดโครงสร้างหรือทอพอโลยีในงานวิจัยนี้จึงใช้ Regular Language ที่มีรูปแบบเรียบง่ายที่สุด ใช้ทรัพยากรในการประมวลผลต่ำ

นิยามภาษา FRL

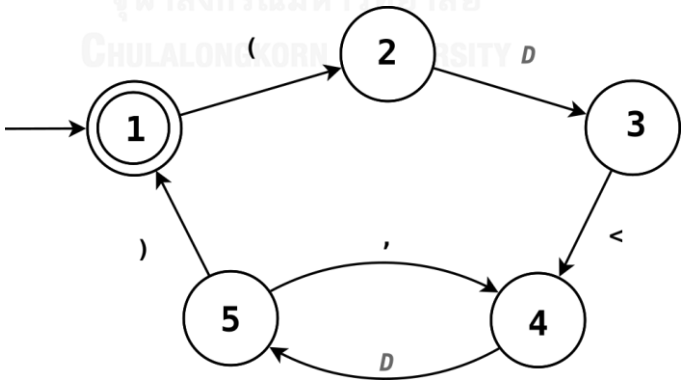
การออกแบบนั้นจึงต้องมุ่งเน้นไปยังการออกแบบที่ง่ายต่อการประมวลผลแปลความหมาย และเพื่อให้สามารถออกแบบได้ในรูปแบบที่หลากหลาย เช่นเดียวกันกับภาษาสำหรับอธิบายฮาร์ดแวร์ (Hardware Description Language : HDL) การออกแบบรูปการอธิบายนี้อิงจากรูปแบบโครงสร้างการเชื่อมต่อของสายสัญญาณต่างๆ ในระบบ สามารถเขียนเป็น Regular Expression และแผนภาพออโตมาต้าสถานะจำกัด (Finite-State Automata Diagram) ได้ดังนี้

กำหนดให้ w คือ ชื่อของสายหรือช่องสัญญาณติดต่อ

b คือ วงเล็บเปิด '(' , d คือ วงเล็บปิด ')'

n คือ ตัวอักษร '<' , c คือ ตัวอักษร ','

นิพจน์ 1	*(bwnw(cw)*d)**



รูปที่ 11 แผนภาพสถานะจำกัดออโตมาต้าของการอธิบายภาษา FRL

จากนิพจน์ 1 ภาษา FRL จะเป็นการอธิบายการเชื่อมต่อทีละโหนดอุปกรณ์ที่เป็นผู้รับสัญญาณ เริ่มต้นด้วยอักขระ ‘(’ และต่อด้วยสัญลักษณ์ของโหนดอุปกรณ์ผู้รับสัญญาณ ตามด้วยอักขระ ‘<’ ซึ่งแสดงเป็นการส่งข้อมูลจากขวาไปซ้าย ซึ่งสัญลักษณ์ของโหนดอุปกรณ์ในงานวิจัยนี้จะใช้อักขระตัวเลขจาก ‘0’ ถึง ‘9’ ตามด้วยกลุ่มของโหนดผู้ส่งสัญญาณ ซึ่งแบ่งด้วยอักขระ ‘,’ และจบด้วยอักขระ ‘)’ สามารถเขียนในรูปแบบ Backus-Naur Form ดังนี้

กำหนดให้

<device_node> คือ สัญลักษณ์โหนดอุปกรณ์

<receive_node> คือ สัญลักษณ์โหนดอุปกรณ์ผู้รับ

<transmit _nodes> คือ สัญลักษณ์โหนดอุปกรณ์หรือกลุ่มโหนดอุปกรณ์ผู้ส่ง

<fr_expression> คือ นิพจน์อธิบายการเชื่อมต่อแบบ FullDOSC (FullDOSC Routing Expression)

<fr_language> คือ ภาษาอธิบายการเชื่อมต่อแบบ FullDOSC (FullDOSC Routing Language)

Backus-Naur Form

<device_node> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<receive_node> ::= <device_node>

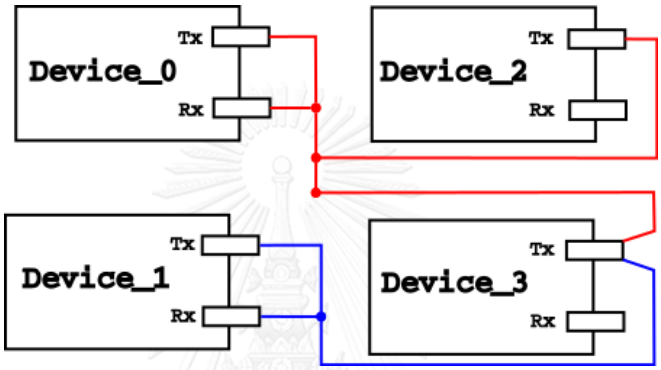
<transmit_nodes> ::= <device_node> |
<device_node>“,”<transmit_nodes>

<fr_expression> ::= “(”<receive_node>“<”<transmit_node>“)”

<fr_language> ::= <fr_expression> | <fr_language><fr_expression>

ตัวอย่าง FRL ซึ่งใช้ในการอธิบายการเชื่อมต่อของข่ายงานซึ่งใช้ในระบบซึ่งต้องการให้ โหนดอุปกรณ์ “0” รับสัญญาณจากโหนดอุปกรณ์ “0”, “2”, และ “3” และโหนดอุปกรณ์ “1” รับสัญญาณจากโหนดอุปกรณ์ “1” และ “3” สามารถเขียนได้เป็นดังในนิพจน์ 2 ซึ่งผลลัพธ์ที่ได้จะเป็นดัง รูปที่ 12

นิพจน์ 2	*(0<0,2,3)(1<1,3)*



รูปที่ 12 ตัวอย่างการเชื่อมต่อแบบ FullDOSC

7.3. ภาษา modified FullDOSC Routing Language (mFRL)

นิยามภาษา mFRL

เช่นเดียวกับภาษา FRL การออกแบบนั้นจึงต้องมุ่งเน้นไปยังการออกแบบที่ง่ายต่อการประมวลผล แปลความหมาย และเพื่อให้สามารถออกแบบได้ในรูปแบบที่หลากหลาย ซึ่งสามารถเขียนเป็น Regular Expression และแผนภาพออโตมาต้าสถานะจำกัด (Finite-State Automata Diagram) ได้ดังนี้

กำหนดให้ w คือ ชื่อของสายหรือช่องสัญญาณติดต่อ

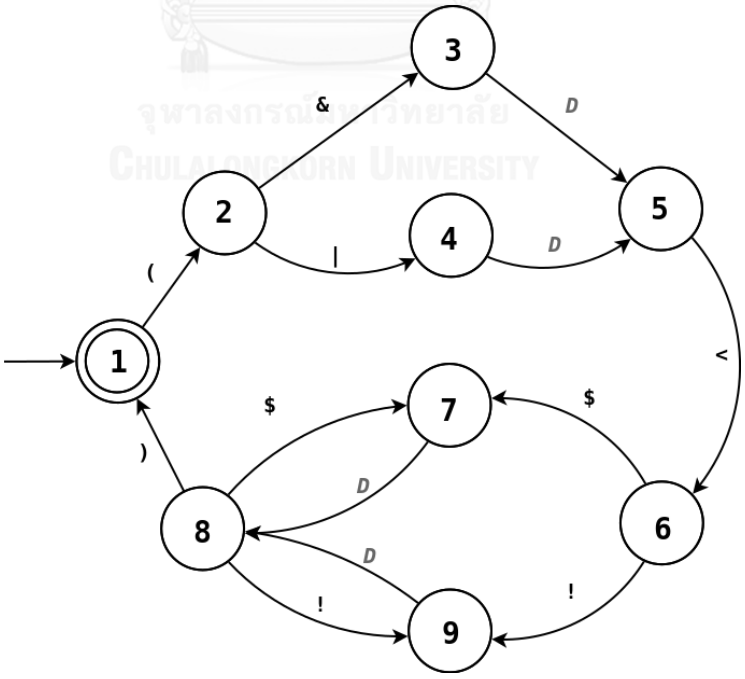
b คือ วงเล็บเปิด '(' , d คือ วงเล็บปิด ')'

n คือ ตัวอักษร '<' , c คือ ตัวอักษร ','

A คือ ตัวอักษร '&' , O คือ ตัวอักษร '|'

N คือ ตัวอักษร '!' , P คือ ตัวอักษร '$'

นิพจน์ 3	*(b(A|O)wn((N|P)w)*d)**



รูปที่ 13 แผนภาพออโตมาต้าสถานะจำกัด (FA) ของการอธิบายภาษา mFRL

จากนิพจน์ 3 ภาษา mFRL ซึ่งเป็นภาษาที่ดัดแปลงมาจากภาษา FRL โดยเพิ่มความสามารถในการกำหนดวิธีการในการรวมข้อมูลสำหรับช่องทางส่งออก และการสลับตรรกะสัญญาณ (invert) ของช่องทางรับเข้า ดังนี้

- เพิ่มตัวดำเนินการ AND และ OR โดยใช้สัญลักษณ์ & และ | ตามลำดับไว้หน้าสัญลักษณ์ของโหนดอุปกรณ์ที่เป็นผู้รับสัญญาณ และมีลักษณะเป็นการ pull-up และ pull-down ในวงจรทางไฟฟ้า ตามลำดับ
- เพิ่มตัวดำเนินการในการสลับตรรกะสัญญาณโดยใช้ ! ไว้หน้าสัญลักษณ์โหนดอุปกรณ์ผู้ส่งสัญญาณ และใช้ $ ไว้หน้าสัญลักษณ์โหนดอุปกรณ์ผู้ส่งสัญญาณในกรณีที่ต้องการให้ส่งสัญญาณแบบปกติ

<u>กำหนดให้</u>

<device_node> คือ สัญลักษณ์โหนดอุปกรณ์

<receive_node> คือ สัญลักษณ์โหนดอุปกรณ์ผู้รับ

<transmit _nodes> คือ สัญลักษณ์โหนดอุปกรณ์หรือกลุ่มโหนดอุปกรณ์ผู้ส่ง

<combine_operator> คือ สัญลักษณ์ในการรวมสัญญาณส่งออก

<invert_operator>คือ สัญญาณในการสลับสัญญาณรับเข้า

<fr_expression> คือ นิพจน์อธิบายการเชื่อมต่อแบบ FullDOSC (FullDOSC Routing Expression)

<fr_language> คือ ภาษาอธิบายการเชื่อมต่อแบบ FullDOSC (FullDOSC Routing Language)

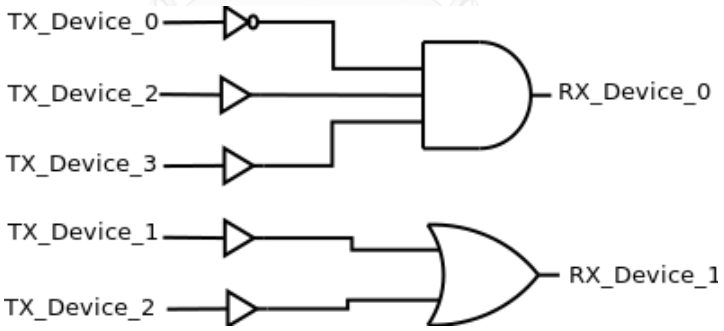
<u>Backus-Naur Form</u>

<device_node>	::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”
<combine_operator>	::= “&” | “|”
<invert_operator>	::= “!” | “$”
<receive_node>	::= <combine_operator><device_node>

<transmit_nodes>	::= <invert_operator><device_node> | <transmit_nodes><transmit_nodes>
<fr_expression>	::= “(”<receive_node>“<”<transmit_node>“)”
<fr_language>	::= <fr_expression> | <fr_language><fr_expression>

ตัวอย่าง FRL ซึ่งใช้ในการอธิบายการเชื่อมต่อของข่ายงานซึ่งใช้ในระบบซึ่งต้องการให้ โหนดอุปกรณ์ “0” รับสัญญาณโดยใช้ตัวดำเนินการ AND ในการรวมสัญญาณจากโหนดอุปกรณ์ “0”, “2”, และ “3” โดยทำการสลับสัญญาณรับเข้าของโหนดอุปกรณ์ “0” และโหนดอุปกรณ์ “1” รับสัญญาณโดยใช้ตัวดำเนินการ OR ในการรวมสัญญาณจากโหนดอุปกรณ์ “1” และ “3” สามารถเขียนได้เป็นดังในนิพจน์ 4 ซึ่งผลลัพธ์ที่ได้จะเป็นดัง รูปที่ 12

นิพจน์ 4	*(&0<!0$2$3)(|1<$1$2)*



รูปที่ 14 แผนภาพเค้าร่างของตัวอย่าง

7.4. พิสูจน์ความสามารของภาษา FRL และ mFRL

ในบทนี้จะเป็นบทพิสูจน์ความสารมารถของภาษา FRL และ mFRL ในการพิสูจน์ความสามารถในการอธิบายความเชื่อมโยงของข่ายงานการสื่อสารแบบหนึ่งอนุกรมสองทาง (FullDOSC network) โดยมีบทตั้งว่า “ภาษา FRL และ mFRL เป็นภาษาที่สามารถอธิบายรูปของการเชื่อมโยงของข่ายงานการสื่อสารแบบหนึ่งอนุกรมสองทางใดๆได้” โดยใช้ตรรกบท (Syllogism) ซึ่งมีนิพจน์ดังรูปที่ 15 โดยให้ A คือ ภาษา FRL/mFRL, B คือ ภาษาที่สามารถอธิบายความสัมพันธ์ของหนึ่งหรือสองโหนดใดๆได้ และ C คือภาษาที่สามารถอธิบายรูปของการเชื่อมโยงของข่ายงานการสื่อสารแบบหนึ่งอนุกรมสองทางใดๆได้ ดังนั้น ถ้าต้องการพิสูจน์ตามบทตั้งที่ได้กล่าวมา สามารถพิสูจน์ได้จากการพิสูจน์ว่า “ภาษา FRL/mFRLเป็นภาษาที่สามารถอธิบายความสัมพันธ์ของหนึ่งหรือสองโหนดใดๆได้” และ “ภาษาที่สามารถอธิบายความสัมพันธ์ของหนึ่งหรือสองโหนดใดๆได้เป็นภาษาที่สามารถอธิบายรูปของการเชื่อมโยงของข่ายงานการสื่อสารแบบหนึ่งอนุกรมสองทางใดๆได้”

Syllogism

$$((A \implies B) \wedge (B \implies C)) \implies (A \implies C) : \text{Tautology}$$

รูปที่ 15 นิพจน์ของตรรกบท (Syllogism)

จากการเชื่อมต่อของระบบแบบหนึ่งอนุกรมสองทาง มีการเชื่อมต่อเป็นรูปแบบหนึ่งของกราฟ (graph) ซึ่งในนิยามของกราฟได้กล่าวว่า “A graph G = (V,E) consist of V, a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoint”[10] ซึ่งจะได้ว่าเราสามารถอธิบายการเชื่อมต่อแบบหนึ่งอนุกรมสองทางซึ่งเป็นรูปแบบหนึ่งของกราฟ โดยการระบุเซตของจุดยอด (vertex) หรือโหนด และความสัมพันธ์ระหว่างหนึ่งหรือสองจุดยอดซึ่งคือเส้นเชื่อม(edge)

ในภาษา FRL และ mFRL จะมีการระบุชื่อของโหนดอุปกรณ์ซึ่งกล่าวได้ว่า ชื่อของโหนดอุปกรณ์ที่ได้กล่าวในชุดคำสั่งเป็นสมาชิกของเซตของจุดยอดในกราฟ

นอกจากการเทียบเคียงกับการเชื่อมต่อในรูปแบบกราฟแล้ว ยังสามารถพิสูจน์โดยการอุปนัยเชิงคณิตศาสตร์ (Mathematical Induction) โดยมีขั้นตอนพื้นฐาน(Basic step) และขั้นตอนอุปนัย (Inductive step) ดังนี้

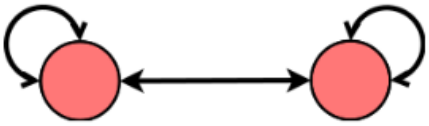
เมื่อกำหนดให้ในข่ายงานมีโหนดเพียงโหนดเดียวสามารถอธิบายได้โดยการอธิบายความสัมพันธ์ของหนึ่งโหนดดังรูปที่ 16

- Basic step : f(n=1) is true.

รูปที่ 16 แผนภาพและการอธิบายความสัมพันธ์ในขั้นตอนพื้นฐาน เมื่อ n=1

เมื่อกำหนดให้ในข่ายงานมีโหนดสองโหนด สามารถอธิบายได้โดยการอธิบายความสัมพันธ์ของหนึ่งโหนด และสองโหนดได้ดังรูปที่ 17

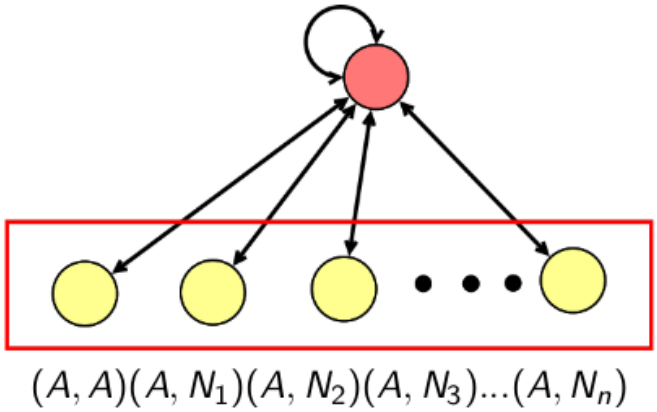
- Basic step : f(n=2) is true.



รูปที่ 17 แผนภาพและการอธิบายความสัมพันธ์ในขั้นตอนพื้นฐาน เมื่อ n=2

เมื่อกำหนดให้ในข่ายงานที่มีโหนด n สามารถอธิบายได้แล้ว สามารถอธิบายข่ายงานที่มีโหนดเพิ่มขึ้นมาอีกหนึ่งโหนดได้โดยการอธิบายความสัมพันธ์ของหนึ่งโหนด และสองโหนดได้ดังรูปที่ 18

- Inductive step : if f(n) is true, f(n+1) is true.



รูปที่ 18 แผนภาพและการอธิบายความสัมพันธ์ในขั้นตอนอุปนัย

จากการพิสูจน์ข้างต้นจะได้ว่า “ภาษาที่สามารถอธิบายความสัมพันธ์ของหนึ่งหรือสองโหนดใดๆ ได้เป็นภาษาที่สามารถอธิบายรูปของการเชื่อมโยงของข่ายงานการสื่อสารแบบหนึ่งอนุกรมสองทางใดๆได้”

การพิสูจน์คุณสมบัติในการอธิบายการเชื่อมต่อหรือการสื่อสารระหว่างหน่วยประมวลผลสองหน่วยใดๆ (รวมถึงการสื่อสารระหว่างตัวเอง) ซึ่งเทียบได้กับเส้นเชื่อมในกราฟมีลักษณะที่เป็นไปได้อยู่ 3 รูปแบบคือ

กำหนดให้

$R_0 ... R_i ... R_n$	คือ	สัญลักษณ์ของโหนดอุปกรณ์ผู้รับ
$T_0 ... T_i ... T_n$	คือ	สัญลักษณ์ของโหนดอุปกรณ์ผู้ส่ง
C	คือ	สัญลักษณ์ของการรวมสัญญาณส่งออก
V	คือ	สัญลักษณ์ของการสลับสัญญาณรับเข้า

- ไม่สามารถติดต่อสื่อสารระหว่างกันได้

  ในกรณีนี้สามารถอธิบายได้โดยไม่ต้องระบุหรือเขียนอธิบาย

- สามารถติดต่อสื่อสารโดยการส่งข้อมูลได้ทางเดียว

  ในกรณีนี้สามารถอธิบายได้ดังในนิพจน์ 5

นิพจน์ 5	$(R_i<T_i)$	*: FRL*
	$(CR_i<VT_i)$	*: mFRL*

ถ้าโหนดอุปกรณ์อื่นๆที่รับสัญญาณจากโหนดอุปกรณ์เดียวกันสามารถอธิบายได้ดังในนิพจน์ 6 โดยกำหนดให้

นิพจน์ 6	$(R_0<T_i)\ (R_1<T_i)\ \ldots\ (R_n<T_i)$	*: FRL*
	$(CR_0<VT_i)\ (CR_1<VT_i)\ \ldots\ (CR_n<VT_i)$	*: mFRL*

ถ้าโหนดอุปกรณ์อื่นๆที่ส่งสัญญาณไปยังโหนดอุปกรณ์เดียวกันสามารถอธิบายได้ดังในนิพจน์ 7 โดยกำหนดให้

นิพจน์ 7	$(R_i<T_0, T_1, \ldots, T_n)$	*: FRL*
	$(CR_i<VT_0VT_1\ \ldots\ VT_n)$	*: mFRL*

- สามารถติดต่อสื่อสารโดยการส่งข้อมูลได้สองทาง

ในกรณีนี้สามารถอธิบายได้โดย การอธิบายในรูปแบบเดียวกับการส่งข้อมูลได้ทางเดียว กับทั้งสองฝั่งดังในนิพจน์ 8

นิพจน์ 8	$(R_i<T_i)\ (T_i<R_i)$	*: FRL*
	$(CR_i<VT_i)\ (CT_i<VR_i)$	*: mFRL*

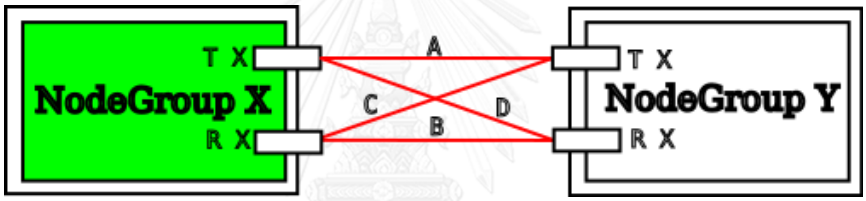
จากทั้ง 3 รูปแบบที่เป็นไปได้ของการสื่อสารจึงสามารถกล่าวได้ว่า “ภาษา FRL/mFRL เป็นภาษาที่สามารถอธิบายความสัมพันธ์ของหนึ่งหรือสองโหนดใดๆได้”

และจากตรรกบทในรูปที่ 15 เมื่อพิสูจน์ได้ว่า “ภาษา FRL/mFRL เป็นภาษาที่สามารถอธิบายความสัมพันธ์ของหนึ่งหรือสองโหนดใดๆได้” และ “ภาษาที่สามารถอธิบายความสัมพันธ์ของหนึ่งหรือสองโหนดใดๆได้เป็นภาษาที่สามารถอธิบายรูปของการเชื่อมโยงของข่ายงานการสื่อสารแบบหนึ่งอนุกรมสองทางใดๆได้” จะสรุปได้ว่า “ภาษา FRL และ mFRL เป็นภาษาที่สามารถอธิบายรูปของการเชื่อมโยงของข่ายงานการสื่อสารแบบหนึ่งอนุกรมสองทางใดๆได้”

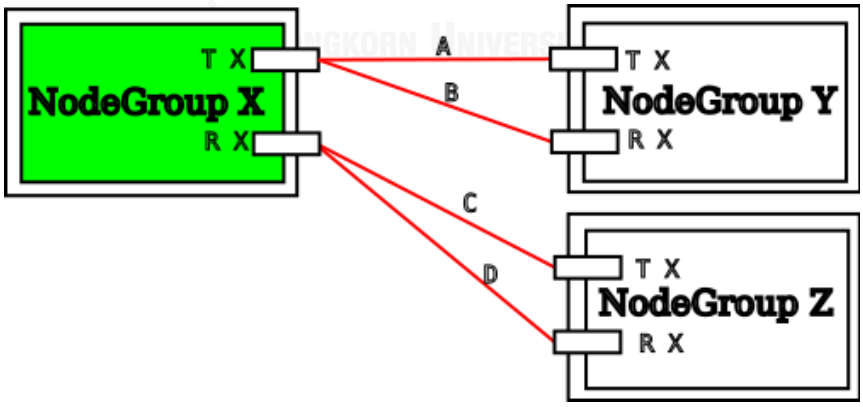
7.5. พิสูจน์รูปแบบเชื่อมต่อโดยทั่วไปของข่ายงาน FullDOSC

จากนิยามของภาษา FRL ในหัวข้อ 7.2 และ mFRL ในหัวข้อ 7.3 จะเห็นได้ว่ามีรูปแบบการเชื่อมต่อหรือทอพอโลยีร่วมกัน ดังนั้นในงานวิจัยนี้จึงจะขอกำหนดรูปแบบการเชื่อมต่อทั่วไป ซึ่งจะครอบคลุมรูปแบบ master-slave , multi-drop และ ring network จากหัวข้อ 2.2

สำหรับการพิสูจน์ทำได้โดย กำหนดให้กลุ่มโหนด หนึ่ง คือ หนึ่งหรือกลุ่มของหน่วยประมวลผล (Node Group) ซึ่งมี TX เป็นช่องทางสำหรับการส่งข้อมูล และ RX เป็นช่องทางสำหรับรับข้อมูล และสามารถติดต่อผ่านกลุ่มโหนด ภายนอกโดยใช้ช่องทางดังกล่าวร่วมกัน ในกรณีนี้เราจะพิจารณาการส่งโดยการใช้ TX และ RX หนึ่งชุด เมื่อกำหนดกลุ่มโหนด ที่พิจารณา(กลุ่มโหนดสีเขียว) แล้วจะแบ่งการเชื่อมต่อกับกลุ่มโหนดอื่นได้เป็นสองกรณีหลักคือ TX และ RX ของกลุ่มโหนด ที่พิจารณาเชื่อมต่อไปยังกลุ่มโหนดอื่น กลุ่มโหนดเดียวกัน และ ต่างกันซึ่งเป็นไปได้เพียงแค่สองกลุ่มโหนด ดังรูป



รูปที่ 19แผนภาพแสดงการเชื่อมต่อไปยังโหนดเดียวกัน



รูปที่ 20 แผนภาพแสดงการเชื่อมต่อไปยังโหนดต่างกัน

กรณีที่ 1 (รูปที่ 19)

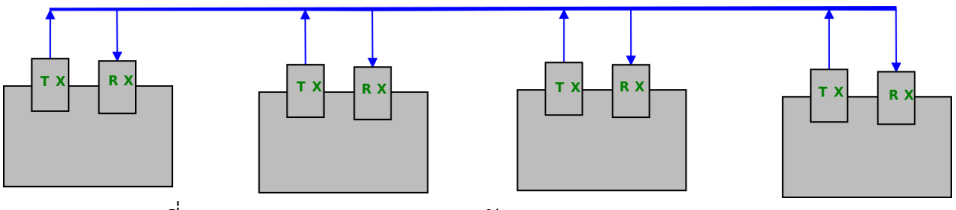
- ไม่ตัดเส้นเชื่อมต่อใดๆ เลย จะทำให้ทุกจุดเชื่อมต่อกันเหมือนเป็นเส้นเดียวกันกำหนดให้รูปแบบนี้เป็น C1
- ตัดเส้น A, B, C, หรือ D เส้นใดเส้นหนึ่ง จะทำให้ได้ลักษณะเหมือนกับ C1 เนื่องจากทุกจุดยังคงเชื่อมต่อเสมือนเป็นจุดเดียวเหมือนเดิม
- ตัดเส้น A และ B จะทำให้ TX และ RX ของ Node ที่พิจารณา ต่อเข้ากับ RX และ TX ของ Node อื่นตามลำดับ ซึ่งจะทำให้สามารถส่งข้อมูลหากันของทั้งสองกลุ่มนี้ได้ กำหนดให้รูปแบบนี้เป็น C2
- ตัดเส้น C และ D จะทำให้ไม่สามารถส่งข้อมูลหากันได้
- การตัดเส้น A และ D หรือ C และ B จะทำให้กลุ่มโหนดที่พิจารณาไม่สามารถส่งหรือรับข้อมูลได้ตามลำดับ
- การตัดเส้นมากกว่าสองเส้นขึ้นไปทำให้การติดต่อกันไม่สมบูรณ์ตามข้อก่อนหน้านี้

กรณีที่ 2 (รูปที่ 20)

- การเชื่อมต่อของเส้น A และ D ไม่เกิดผลในการส่งข้อมูลจึงสมควรตัดออกจากการพิจารณา
- ตัดเส้น A และ D จะทำให้ TX และ RX ของกลุ่มโหนดที่พิจารณาเชื่อมต่อกับ TX และ RX ของ Node อื่นที่ต่างกันกำหนดให้รูปแบบนี้เป็น C3

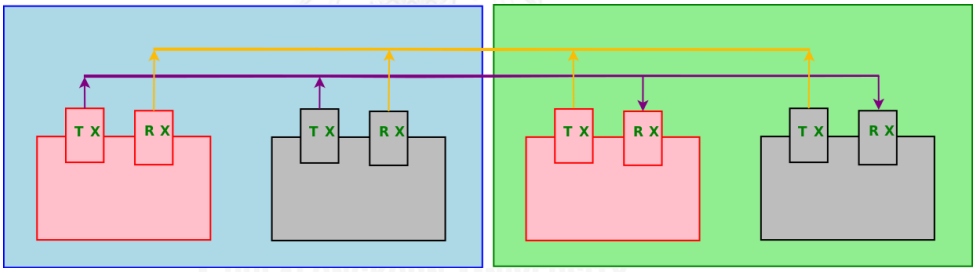
จากการพิสูจน์ดังกล่าวจึงสามารถแบ่งรูปแบบการเชื่อมต่อการสื่อสารรูปแบบเชื่อมต่อโดยทั่วไปของการสื่อสารหนึ่งอนุกรมสองทางได้ 3 รูปแบบคือ C1 และ C2 จากกรณีที่ 1 และ C3 จากกรณีที่ 2 ซึ่งเมื่อพิจารณาเทียบเคียงจากหัวข้อ 2.2 C1 มีการเชื่อมต่อลักษณะเดียวกับ รูปแบบบัสหลายจุด (Multi-Drop Bus) และ C3 มีการเชื่อมต่อแบบวงแหวน (Ring) แต่ในกรณี master-slave bus มีลักษณ์เป็นเซตย่อยของ C2 ดังนั้นจึงนิยามชื่อของรูปแบบนี้เป็น รูปแบบบัสสองฝั่ง (Two-Side Bus) เนื่องจากมีลักษณะควบคุมเป็นสองฝั่ง และ master-slave bus คือรูปแบบบัสสองฝั่งที่ฝั่งหนึ่งมีโหนดอุปกรณ์เพียงหนึ่งตัว

- C1 : รูปแบบบัสหลายจุด (Multi-Drop Bus)



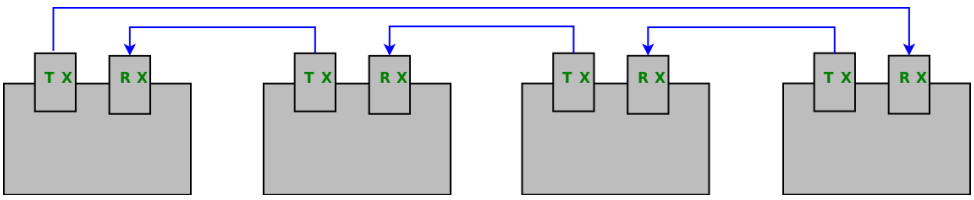
รูปที่ 21 แผนภาพแสดงรูปแบบบัสหลายจุด (Multi-Drop Bus)

- C2 : รูปแบบบัสสองฝั่ง (Two-Side Bus)



รูปที่ 22 แผนภาพแสดงรูปแบบบัสสองฝั่ง (Two-Side Bus)

- C3 : รูปแบบวงแหวน (Ring)



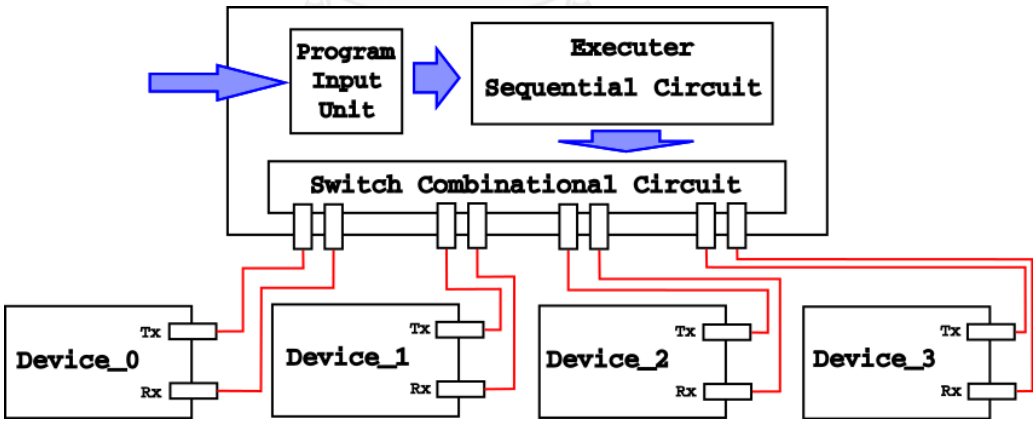
รูปที่ 23 แผนภาพแสดงรูปแบบวงแหวน (Ring)

8. ระบบสื่อสารแบบหนึ่งอนุกรมสองทาง (FullDOSC System)

8.1. การออกแบบอุปกรณ์จัดการการเชื่อมต่อ (FullDOSC Router Design)

ในงานวิจัยนี้ ได้ทำการออกแบบโครงสร้างของอุปกรณ์จัดการการเชื่อมต่อ ซึ่งแบ่งเป็นได้เป็นสามส่วนหลักซึ่งได้แก่ ส่วนรับข้อมูลของโปรแกรม (Program Input Unit), ส่วนวงจรต่อเนื่องสำหรับแปลภาษา (Executer Sequential Circuit : ESC) และ ส่วนวงจรรวมสลับสัญญาณ (Switch Combinational Circuit : SCC)

- ส่วนรับข้อมูลของโปรแกรม (Program Input Unit) คือ ส่วนรับข้อมูลของโปรแกรม FRL เข้ามาซึ่งอาจเป็น I^2C Controller หรืออาจจะเป็น bus ข้อมูล 8 บิท ซึ่งไม่จำเป็นต้องมีส่วนนี้
- ส่วนวงจรต่อเนื่องสำหรับแปลภาษา (Executer Sequential Circuit : ESC) คือ ส่วนที่ทำการประมวลผลสายอักขระภาษา FRL และควบคุมการทำงานของ SCC
- ส่วนวงจรรวมสลับสัญญาณ (Switch Combinational Circuit : SCC) คือ ส่วนที่ทำหน้าที่เชื่อมโยงสัญญาณ



รูปที่ 24 โครงสร้างตัวอย่างของระบบ FullDOSC

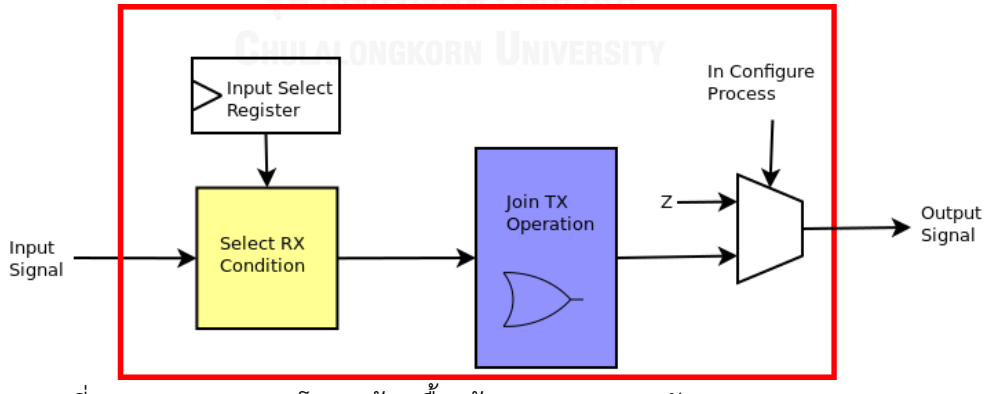
อุปกรณ์จัดการเชื่อมต่อสำหรับภาษา FRL (FullDOSC Router ESC)

จากที่ได้กล่าวไป ในงานวิจัยนี้ จะทำการสร้างอุปกรณ์จัดการการเชื่อมต่อ โดยใช้ Verilog HDL ในการอธิบายวงจรโครงสร้างเป็น module fr ดังใน รูปที่ 27 ถึง รูปที่ 30 ซึ่งเป็นการออกแบบโดยกำหนดจำนวนของโหนดอุปกรณ์ผ่าน NUM_DEVICE มีข้อมูลรับเข้าได้แก่ สัญญาณนาฬิกา (clock), สัญญาณเริ่มใหม่ (reset), บัสข้อมูลสำหรับโปรแกรม FRL ตามรหัส ASCII ขนาด 8 บิท (str_prog) และช่องทางรับข้อมูลของจากโหนดอุปกรณ์ขนาดตามจำนวนโหนดอุปกรณ์ (rx) และข้อมูลส่งออกได้แก่ ช่องทางส่งข้อมูลของไปยังโหนดอุปกรณ์ขนาดตามจำนวนโหนดอุปกรณ์ (tx)

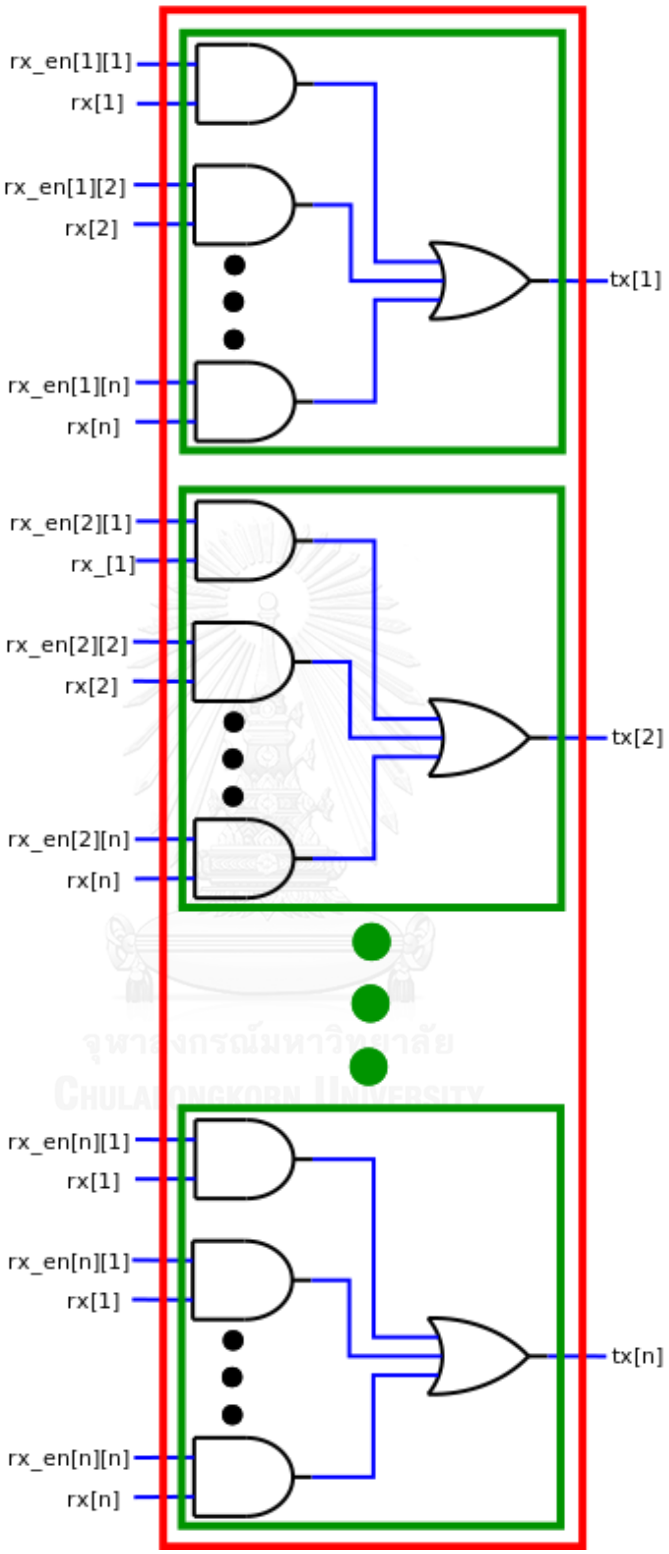
จากรูปที่ 27 และ รูปที่ 28 ในบรรทัดที่ 33 ถึง 44 เป็นส่วนวงจรรวมสลับสัญญาณ (Switch Combinational Circuit) ซึ่งสัญญาณรับเข้าในทุกช่องทางจะถูกส่งไปยังทุกวงจรสำหรับขาออกโดยมีเรจิสเตอร์ (register) สำหรับเปิด-ปิดสัญญาณในแต่ละเส้นทางของสัญญาณ และข้อมูลส่งออกได้จากการดำเนินการ OR ของข้อมูลรับเข้าที่เปิดสัญญาณซึ่งมีโครงสร้างเป็นดังแผนภาพในรูปที่ 25 และสามารถเขียนเป็นวงจรทางตรรกะได้ดังแผนภาพในรูปที่ 26

จากรูปที่ 28 และ รูปที่ 29 ในบรรทัดที่ 46 ถึง 100 เป็นวงจรต่อเนื่องสำหรับแปลภาษา (Executer Sequential Circuit) ซึ่งสร้างจากแผนภาพในรูปที่ 11 ซึ่งมีสถานะ (state) อยู่ 5 สถานะ และมีการเปลี่ยนตามข้อมูลรอบข้างดังกล่าว

จากรูปที่ 29 และ รูปที่ 30 ในบรรทัด 111 ถึง 161 เป็นส่วนตรรกะส่งออกเพื่อไปควบคุมส่วนวงจรรวมสลับสัญญาณ



รูปที่ 25 แผนภาพแสดงโครงสร้างเบื้องต้นของวงจรรวมสลับ (SCC) ของ FullDOSC



รูปที่ 26 แผนภาพเค้าร่างโดยขยายของวงจรรวมสลับ (SCC)

รหัสคำสั่งอธิบายโครงสร้าง FullDOSC Router

```
module fr
#(
    parameter NUM_DEVICE = 4
)
(
    input clock,
    input reset,
    input [7:0] str_prog,
    input [NUM_DEVICE-1:0] rx,
    output [NUM_DEVICE-1:0] tx

);

    parameter [2:0] STATE_1 = 3'b001;
    parameter [2:0] STATE_2 = 3'b010;
    parameter [2:0] STATE_3 = 3'b011;
    parameter [2:0] STATE_4 = 3'b100;
    parameter [2:0] STATE_5 = 3'b101;

    reg [2:0] state, next_state;

    reg [7:0] select_tx_node;
    reg [7:0] select_rx_node;
    wire [3:0] select_tx_w;
    wire [3:0] select_rx_w;

    reg in_config;

    reg [NUM_DEVICE-1:0] rx_en[NUM_DEVICE-1:0];

    genvar i,j;

//// Switch Combinational Circuit
    for(i=0;i<NUM_DEVICE;i=i+1)
    begin
        wire [NUM_DEVICE-1:0] sel_rx;
        wire out;
        for(j=0;j<NUM_DEVICE;j=j+1)
        begin
            assign sel_rx[j] = rx_en[i][j] & rx[j];
```

รูปที่ 27 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย FullDOSC Router (1)

```
41          end
42          assign out = |sel_rx;
43          assign tx[i] = (in_config) ? 1'bz : out;
44      end
45
46  //// Executer Sequential Circuit
47
48      always @ (state or str_prog) begin
49          if(reset) begin
50              next_state = 0;
51          end else begin
52              case (state)
53                  STATE_1 : begin
54                      if (str_prog == "(") begin
55                          next_state = STATE_2;
56                      end else begin
57                          next_state = STATE_1;
58                      end
59                  end
60
61                  STATE_2 : begin
62                      if(str_prog >= "0" && str_prog <= "9") begin
63                          next_state = STATE_3;
64                      end else begin
65                          next_state = STATE_2;
66                      end
67                  end
68
69                  STATE_3 : begin
70                      if (str_prog == "<") begin
71                          next_state = STATE_4;
72                      end else begin
73                          next_state = STATE_3;
74                      end
75                  end
76
77                  STATE_4 : begin
78                      if(str_prog >= "0" && str_prog <= "9") begin
79                          next_state = STATE_5;
80                      end else begin
```


รูปที่ 28 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย FullDOSC Router (2)

```
81                          next_state = STATE_4;
82                      end
83                  end
84
85                  STATE_5 : begin
86                      if (str_prog == "," ) begin
87                          next_state = STATE_4;
88                      end else if (str_prog == ")") begin
89                          next_state = STATE_1;
90                      end else begin
91                          next_state = STATE_5;
92                      end
93                  end
94
95                  default : begin
96                      next_state = STATE_1;
97                  end
98              endcase
99          end
100     end
101
102     //// Sequential Logic
103     always @ ( posedge clock) begin
104         if (reset) begin
105             state <= STATE_1;
106         end else begin
107             state <= next_state;
108         end
109     end
110
111     //// Output Logic
112
113     integer k;
114
115     always @ ( posedge clock) begin
116         if (reset) begin
117             for(k=0;k<NUM_DEVICE;k=k+1)
118                 rx_en[k] <= 0;
119             select_tx_node <= 0;
120             select_rx_node <= 0;
```

รูปที่ 29 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย FullDOSC Router (3)

```
121         end else begin
122             case (state)
123                 STATE_1 : begin
124                      in_config <= 0;
125                 end
126 
127                 STATE_2 : begin
128                     in_config <= 1;
129                     if(str_prog - "0" > NUM_DEVICE) begin
130                         select_tx_node <= NUM_DEVICE-1+"0";
131                     end else begin
132                         select_tx_node <= str_prog;
133                     end
134                 end
135 
136                 STATE_3 : begin
137                     rx_en[select tx_w] <= 0;
138                 end
139 
140                 STATE_4 : begin
141                     if(str_prog - "0" > NUM_DEVICE) begin
142                         select_rx_node <= NUM_DEVICE-1+"0";
143                     end else begin
144                         select_rx_node <= str_prog;
145                     end
146                 end
147 
148                 STATE_5 : begin
149                     for(k=0;k<NUM_DEVICE;k=k+1) begin
150                         rx_en[k][select_rx_w] <=
151                             (select_tx_w == k) ?
152                                     1 :
153                                     rx_en[k][select_rx_w];
154                     end
155                 end
156             endcase
157         end
158     end
159 
160     assign select_tx_w = select_tx_node - "0";
161     assign select_rx_w = select_rx_node - "0";
162 
163 endmodule
```

รูปที่ 30 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย FullDOSC Router (4)

อุปกรณ์จัดการเชื่อมต่อสำหรับภาษา mFRL (modified FullDOSC Router ESC)

การนำภาษา mFRL ให้ใช้ในระบบแทน FRL จะยึดโครงสร้างตามรูปที่ 24 โดยมีการเปลี่ยนรายละเอียดของวงจรต่อเนื่องสำหรับแปลภาษา (ESC) โดยใช้การเปลี่ยนแปลงสถานะของระบบเป็นไปตามรูปที่ 13 และเปลี่ยนโครงสร้างของวงจรรวมสลับ (SCC) เพื่อให้รองรับการทำงานที่เพิ่มขึ้นดังที่ได้อธิบายไว้ในนิยามภาษาของ mFRL สามารถเขียนเป็นตารางของตรรกะได้โดยมีตรรกะขาเข้า คือ สัญญาณเลือกตัวดำเนินการรวม (Combine Select) ซึ่งให้ 1 คือตัวดำเนินการ AND และ 0 คือตัวดำเนินการ OR สัญญาณรับเข้า (RX) สัญญาณเปิดสัญญาณรับเข้า (RX Enable) และสัญญาณการสลับสัญญาณรับเข้า (Invert RX) ซึ่งให้ 1 เป็นการเลือกให้สลับสัญญา ดังในตารางที่ 1 และสามารถเขียนในรูป Karnaugh map ได้ดังรูปที่ 31

Combine Select (A)	RX (B)	RX Enable (C)	Invert RX (D)	**Output**
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**1**
0	1	1	1	**0**
1	0	0	0	**1**
1	0	0	1	**1**
1	0	1	0	**0**
1	0	1	1	**1**
1	1	0	0	**1**
1	1	0	1	**1**
1	1	1	0	**1**
1	1	1	1	**0**

ตารางที่ 1 ตารางแสดงค่าสัญญานของ SCC สำหรับ mFullDOSC

AB \ CD	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	1	1	0	1
10	1	1	1	0

รูปที่ 31 Karnaugh map ของ SCC สำหรับ mFullDOSC

จากรูปที่ 31 สามารถเขียนเป็นผลรวมของผลคูณ (Sum Of Product : SOP) ได้ดังนี้

	Output	= AC’ + B’CD + BCD’
นิพจน์ 9		= AC’ + C(B’D + BD’)

จากนิยามของออร์เฉพาะ (Exclusive OR : XOR) ซึ่งใช้สัญลักษณ์เป็น ^ คือ A^B จะมีค่าเป็นจริง(1) เมื่อ A หรือ B ตัวใดตัวหนึ่งมีค่าเป็นจริงเท่านั้น เพื่อให้ตรงกับสัญลักษณ์ในภาษา Verilog HDL จึงเปลี่ยนสัญลักษณ์ได้จากนิพจน์ 9 เป็นดังนี้

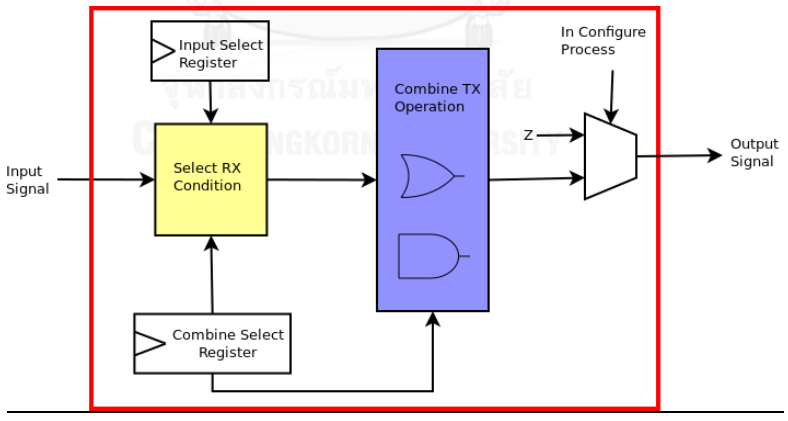
	Output	= (A & ~C) | (C & ((~B & D) | (B & ~D))
นิพจน์ 10		= (A & ~C) | (C & (B ^ D))

จากการนำไปใช้ของ FullDOSC Router เดิม ได้มีการปรับปรุงอุปกรณ์จัดการการเชื่อมต่อ โดยใช้ Verilog HDL ในการอธิบายวงจรโครงสร้างเป็น module fr ดังในรูปที่ 33 ถึง รูปที่ 37 ซึ่งเป็นการออกแบบโดยกำหนดจำนวนของโหนดอุปกรณ์ผ่าน NUM_DEVICE มีข้อมูลรับเข้าได้แก่ สัญญาณนาฬิกา (clock), สัญญาณเริ่มใหม่ (reset), บัสข้อมูลสำหรับโปรแกรม FRL ตามรหัส ASCII ขนาด 8 บิท (str_prog) และช่องทางรับข้อมูลของจากโหนดอุปกรณ์ขนาดตามจำนวนโหนดอุปกรณ์ (rx) และข้อมูลส่งออกได้แก่ ช่องทางส่งข้อมูลของไปยังโหนดอุปกรณ์ขนาดตามจำนวนโหนดอุปกรณ์ (tx)

จากรูปที่ 33 และ รูปที่ 34 ในบรรทัดที่ 45 ถึง 63 เป็นส่วนวงจรรวมสลับสัญญาณ (Switch Combinational Circuit) ซึ่งสัญญาณรับเข้าในทุกช่องทางจะถูกส่งไปยังทุกวงจรสำหรับขาออกโดยมีเรจิสเตอร์ (register) สำหรับเปิด-ปิดสัญญาณในแต่ละเส้นทางของสัญญาณ (rx_en), สลับสัญญาณ (inv_rx_en) และเลือกตัวรวมสัญญาณส่งออก (comb_tx) และใช้สัญลักษณ์แทนตามตารางที่ 1 และใช้นิพจน์ 10 เพื่อสร้างวงจรเผื่อให้ได้ผลลัพธ์ตามที่ออกแบบไว้

จากรูปที่ 34 และ รูปที่ 36 ในบรรทัดที่ 46 ถึง 100 เป็นวงจรต่อเนื่องสำหรับแปลภาษา (Executer Sequential Circuit) ซึ่งสร้างจากแผนภาพในรูปที่ 13 ซึ่งมีสถานะ (state) อยู่ 9 สถานะ และมีการเปลี่ยนตามข้อมูลรอบข้างดังกล่าว

จากรูปที่ 36 และ รูปที่ 37 ในบรรทัด 168 ถึง 250 เป็นส่วนตรรกะส่งออกเพื่อไปควบคุมส่วนวงจรรวมสลับสัญญาณ



รูปที่ 32 แผนภาพแสดงโครงสร้างเบื้องต้นของวงจรรวมสลับ (SCC) ของ mFullDOSC

รหัสคำสั่งอธิบายโครงสร้าง modified FullDOSC Router

```
module fr
#(
    parameter NUM_DEVICE = 4
)
(
    input clock,
    input reset,
    input [7:0] str_prog,
    input [NUM_DEVICE-1:0] rx,
    output [NUM_DEVICE-1:0] tx
);
    parameter [3:0] STATE_1 = 4'b0001;
    parameter [3:0] STATE_2 = 4'b0010;
    parameter [3:0] STATE_3 = 4'b0011;
    parameter [3:0] STATE_4 = 4'b0100;
    parameter [3:0] STATE_5 = 4'b0101;

    parameter [3:0] STATE_6 = 4'b0110;
    parameter [3:0] STATE_7 = 4'b0111;
    parameter [3:0] STATE_8 = 4'b1000;
    parameter [3:0] STATE_9 = 4'b1001;

    reg [3:0] state, next_state;

    reg [7:0] select_tx_node;
    reg [7:0] select_rx_node;
    wire [3:0] select_tx_w;
    wire [3:0] select_rx_w;

    reg in_config;

    reg [NUM_DEVICE-1:0] rx_en[NUM_DEVICE-1:0];

    reg [NUM_DEVICE-1:0] inv_rx_en[NUM_DEVICE-1:0];
    reg inv_tmp;

    reg [NUM_DEVICE-1:0] comb_tx;
    reg comb_tmp;

    wire [NUM_DEVICE-1:0] or_tx;
    wire [NUM_DEVICE-1:0] and_tx;

    genvar i,j;

//// Switch Combinational Circuit
    for(i=0;i<NUM_DEVICE;i=i+1)
    begin
        wire [NUM_DEVICE-1:0] sel_rx;
        wire out;
        for(j=0;j<NUM_DEVICE;j=j+1)
```

รูปที่ 33 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย modified FullDOSC Router (1)

```
51          begin
52              wire a,b,c,d;
53              assign a = comb_tx[i];
54              assign b = rx[j];
55              assign c = rx_en[i][j];
56              assign d = inv_rx_en[i][j];
57              assign sel_rx[j] = (a & ~c) |(c & (b ^  d));
58          end
59          assign or_tx[i] = |sel_rx;
60          assign and_tx[i] = &sel_rx;
61          assign out = (comb_tx[i]) ? and_tx[i] : or_tx[i];
62          assign tx[i] = (in_config) ? 1'bz : out;
63      end
64
65  //// Executer Sequential Circuit
66
67      always @ (state or str_prog) begin
68          if(reset) begin
69              next_state = 0;
70          end else begin
71              case (state)
72                  STATE_1 : begin
73                      if (str_prog == "(") begin
74                          next_state = STATE_2;
75                      end else begin
76                          next_state = STATE_1;
77                      end
78                  end
79
80                  STATE_2 : begin
81                      if(str_prog == "&") begin
82                          next_state = STATE_3;
83                      end else if(str_prog == "|") begin
84                          next_state = STATE_4;
85                      end else begin
86                          next_state = STATE_2;
87                      end
88                  end
89
90                  STATE_3 : begin
91                      if(str_prog >= "0" && str_prog <= "9") begin
92                          next_state = STATE_5;
93                      end else begin
94                          next_state = STATE_3;
95                      end
96                  end
97
98                  STATE_4 : begin
99                      if(str_prog >= "0" && str_prog <= "9") begin
100                         next_state = STATE_5;
```

รูปที่ 34 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย modified FullDOSC Router (2)

```
101                     end else begin
102                         next_state = STATE_4;
103                     end
104                 end
105 
106                 STATE_5 : begin
107                     if (str_prog == "<") begin
108                         next_state = STATE_6;
109                     end else begin
110                         next_state = STATE_5;
111                     end
112                 end
113 
114                 STATE_6 : begin
115                     if (str_prog == "$") begin
116                         next_state = STATE_7;
117                     end else if(str_prog == "!") begin
118                         next_state = STATE_9;
119                     end else begin
120                         next_state = STATE_6;
121                     end
122                 end
123 
124                 STATE_7 : begin
125                     if(str_prog >= "0" && str_prog <= "9") begin
126                         next_state = STATE_8;
127                     end else begin
128                         next_state = STATE_7;
129                     end
130                 end
131 
132                 STATE_8 : begin
133                     if (str_prog == "$" ) begin
134                         next_state = STATE_7;
135                     end else if (str_prog == "!") begin
136                         next_state = STATE_9;
137                     end else if (str_prog == ")") begin
138                         next_state = STATE_1;
139                     end else begin
140                         next_state = STATE_8;
141                     end
142                 end
143 
144                 STATE_9 : begin
145                     if(str_prog >= "0" && str_prog <= "9") begin
146                         next_state = STATE_8;
147                     end else begin
148                         next_state = STATE_9;
149                     end
150                 end
```

รูปที่ 35 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย modified FullDOSC Router (3)

```
151
152                 default : begin
153                     next_state = STATE_1;
154                 end
155             endcase
156         end
157     end
158
159     //// Sequential Logic
160     always @ ( posedge clock) begin
161        if (reset) begin
162             state <= STATE_1;
163         end else begin
164             state <= next_state;
165         end
166     end
167
168     //// Output Logic
169
170     integer k;
171
172     always @ ( posedge clock) begin
173         if (reset) begin
174             in_config <= 0;
175             comb_tx <= 0;
176             for(k=0;k<NUM_DEVICE;k=k+1) begin
177                 rx_en[k] <= 0;
178                 inv_rx_en[k] <= 0;
179             end
180             select_tx_node <= 0;
181             select_rx_node <= 0;
182
183         end else begin
184             case (state)
185                 STATE_1 : begin
186                     in_config <= 0;
187                 end
188
189                 STATE_2 : begin
190                     in_config <= 1;
191                 end
192
193                 STATE_3 : begin
194                     comb_tmp <= 1;
195                     if(str_prog > NUM_DEVICE + "0") begin
196                         select_tx_node <= NUM_DEVICE-1+"0";
197                     end else begin
198                         select_tx_node <= str_prog;
199                     end
200                 end
```

รูปที่ 36 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย modified FullDOSC Router (4)

```
201
202                 STATE_4 : begin
203                     comb_tmp <= 0;
204                     if(str_prog > NUM_DEVICE + "0" ) begin
205                         select_tx_node <= NUM_DEVICE-1+"0";
206                     end else begin
207                         select_tx_node <= str_prog;
208                     end
209                 end
210
211                 STATE_5 : begin
212                     rx_en[select_tx_w] <= 0;
213                     comb_tx[select_tx_w] <= comb_tmp;
214                 end
215
216                 STATE_7 : begin
217                     inv_tmp <= 0;
218                     if(str_prog > NUM_DEVICE + "0") begin
219                         select_rx_node <= NUM_DEVICE-1+"0";
220                     end else begin
221                         select_rx_node <= str_prog;
222                     end
223                 end
224
225                 STATE_8 : begin
226                     for(k=0;k<NUM_DEVICE;k=k+1) begin
227                         rx_en[k][select_rx_w] <= (select_tx_w == k) ?
228                                 1 :
229                                 rx_en[k][select_rx_w];
230                         inv_rx_en[k][select_rx_w] <= (select_tx_w == k) ?
231                                 inv_tmp :
232                                 inv_rx_en[k][select_rx_w];
233                     end
234                 end
235
236                 STATE_9 : begin
237                     inv_tmp <= 1;
238                     if(str_prog > NUM_DEVICE + "0") begin
239                         select_rx_node <= NUM_DEVICE-1+"0";
240                     end else begin
241                         select_rx_node <= str_prog;
242                     end
243                 end
244             endcase
245         end
246     end
247
248     assign select_tx_w = select_tx_node - "0";
249     assign select_rx_w = select_rx_node - "0";
250 endmodule
```

รูปที่ 37 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย modified FullDOSC Router (5)

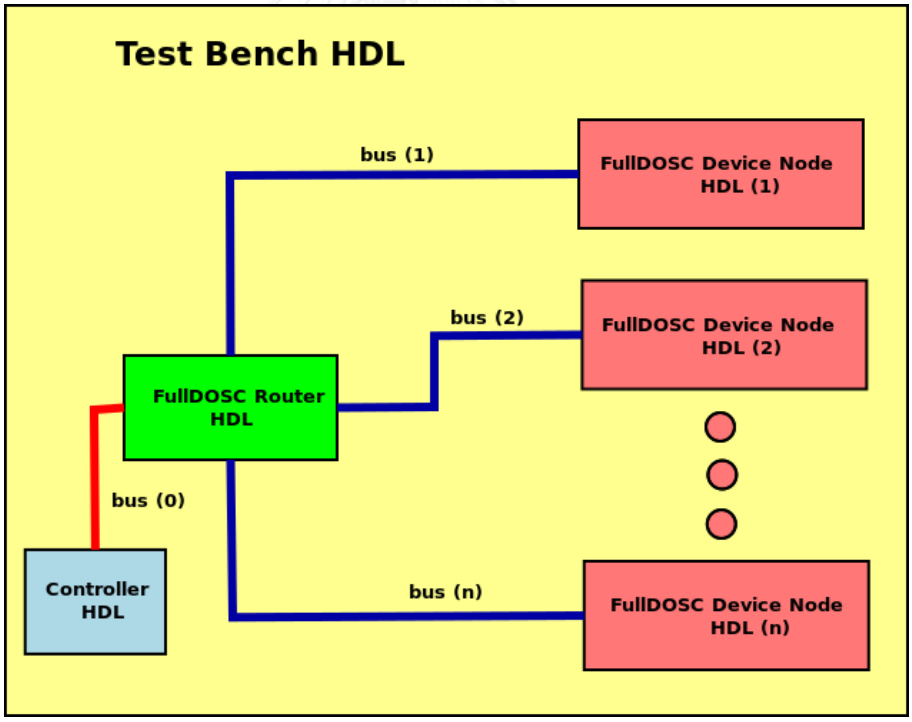
8.2. การนำระบบไปใช้งาน (FullDOSC System Implementation)

การจำลองการระบบทำการทำงาน (Simulation)

เนื่องจากในงานออกมักจะเกิดกรณีที่ไม่มีอุปกรณ์จริงเพื่อทดลองการใช้งาน ดังนั้นการจำลองระบบสื่อสารแบบอนุกรมสองทางจึงเป็นสิ่งที่จำเป็น ซึ่งในงานวิจัยนี้ได้นำเสนอไว้สองวิธีคือ การจำลองโดยใช้กระบวนการเดี่ยว และใช้กระบวนการแยกกัน

การจำลองโดยใช้กระบวนการเดี่ยว

เป็นการจำลองโดยสร้างตัวตรวจสอบหรือ test bench คลุมทั้งระบบซึ่งได้แก่ อุปกรณ์จัดการการเชื่อมต่อ (FullDOSC Router), ตัวควบคุมหรือส่วนที่จะส่งโปรแกรม FRL/mFRL ไปยังอุปกรณ์จัดการการเชื่อมต่อ และ โหนดอุปกรณ์ ตามรูปที่ 38



รูปที่ 38 แผนภาพแสดงการจำลองการทำงานโดยใช้ test bench

วิธีนี้มีข้อดีคือง่ายต่อการสร้างและจัดการ และเป็นวิธีที่นิยมใช้โดยทั่วไป แต่มีข้อเสียคือ ฝั่งโหนดอุปกรณ์จะต้องถูกอธิบายโดย HDL เท่านั้น และการเข้าถึงโหนดอุปกรณ์แต่ละตัวนั้นจะทำได้ค่อนข้างลำบากเนื่องจากถูกคลุมโดย Test Bench รวม

รหัสคำสั่งอธิบายส่วนทดสอบของ FullDOSC Router

ในส่วนการทดสอบระบบนั้นจะทำการแปลภาษาโดยใช้ Icarus Verilog และใช้ Impulse จาก toem ในการแสดงผล (waveform) และทำการสร้างรหัสคำสั่งภาษา Verilog HDL สำหรับทดสอบระบบ (Test bench) โดยอ้างอิงจากทอพอโลยีในรูปที่ 6, รูปที่ 7 และรูปที่ 8

```
1  `timescale 1ns / 1ps
2
3  module fr_tb(
4
5      );
6          parameter NUM_DEVICE_TB = 4;
7
8          reg clock, reset;
9          reg [7:0] str_prog;
10         reg [NUM_DEVICE_TB-1:0]d_tx;
11         wire [NUM_DEVICE_TB-1:0]d_rx;
12
13         initial begin
14             $dumpfile("fr_dump.vcd");
15             $dumpvars(0,fr_tb);
16             clock = 0;
17             reset = 1;
18             d_tx = 0;
19             str_prog = 8'h00;
20             #100 reset = 0;
21
22             ///// Program Route Path
23
24             ////////Multi-Drop
25             #100 str_prog = "(";
26             #100 str_prog = "0";
27             #100 str_prog = "<";
28             #100 str_prog = "0";
29             #100 str_prog = ",";
30             #100 str_prog = "1";
31             #100 str_prog = ",";
32             #100 str_prog = "2";
33             #100 str_prog = ",";
34             #100 str_prog = "3";
35             #100 str_prog = ")";
36
37             #100 str_prog = "(";
38             #100 str_prog = "1";
39             #100 str_prog = "<";
40             #100 str_prog = "0";
41             #100 str_prog = ",";
42             #100 str_prog = "1";
43             #100 str_prog = ",";
44             #100 str_prog = "2";
45             #100 str_prog = ",";
```

รูปที่ 39 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ FullDOSC Router (1)

```
46              #100 str_prog = "3";
47              #100 str_prog = ")";
48
49              #100 str_prog = "(";
50              #100 str_prog = "2";
51              #100 str_prog = "<";
52              #100 str_prog = "0";
53              #100 str_prog = ",";
54              #100 str_prog = "1";
55              #100 str_prog = ",";
56              #100 str_prog = "2";
57              #100 str_prog = ",";
58              #100 str_prog = "3";
59              #100 str_prog = ")";
60
61              #100 str_prog = "(";
62              #100 str_prog = "3";
63              #100 str_prog = "<";
64              #100 str_prog = "0";
65              #100 str_prog = ",";
66              #100 str_prog = "1";
67              #100 str_prog = ",";
68              #100 str_prog = "2";
69              #100 str_prog = ",";
70              #100 str_prog = "3";
71              #100 str_prog = ")";
72
73              ///// Transfer Data
74              #100 d_tx[0] = 1;
75              #100 d_tx[0] = 0;
76              #100 d_tx[1] = 1;
77              #100 d_tx[1] = 0;
78              #100 d_tx[2] = 1;
79              #100 d_tx[2] = 0;
80              #100 d_tx[3] = 1;
81              #100 d_tx[3] = 0;
82
83              //#10 reset = 1;
84              //#10 reset = 0;
85
86              ///// Program Route Path
87
88              ////////Master-Slave
89              #100 str_prog = "(";
90              #100 str_prog = "0";
```

รูปที่ 40 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ FullDOSC Router (2)

```
 91              #100 str_prog = "<";
 92              #100 str_prog = "1";
 93              #100 str_prog = ",";
 94              #100 str_prog = "2";
 95              #100 str_prog = ",";
 96              #100 str_prog = "3";
 97              #100 str_prog = ")";
 98
 99              #100 str_prog = "(";
100              #100 str_prog = "1";
101              #100 str_prog = "<";
102              #100 str_prog = "0";
103              #100 str_prog = ")";
104
105              #100 str_prog = "(";
106              #100 str_prog = "2";
107              #100 str_prog = "<";
108              #100 str_prog = "0";
109              #100 str_prog = ")";
110
111              #100 str_prog = "(";
112              #100 str_prog = "3";
113              #100 str_prog = "<";
114              #100 str_prog = "0";
115              #100 str_prog = ")";
116
117              ///// Transfer Data
118              #100 d_tx[0] = 1;
119              #100 d_tx[0] = 0;
120              #100 d_tx[1] = 1;
121              #100 d_tx[1] = 0;
122              #100 d_tx[2] = 1;
123              #100 d_tx[2] = 0;
124              #100 d_tx[3] = 1;
125              #100 d_tx[3] = 0;
126
127              //#10 reset = 1;
128              //#10 reset = 0;
129
130              ///// Program Route Path
131
132              ////////Ring
133              #100 str_prog = "(";
134              #100 str_prog = "0";
135              #100 str_prog = "<";
```

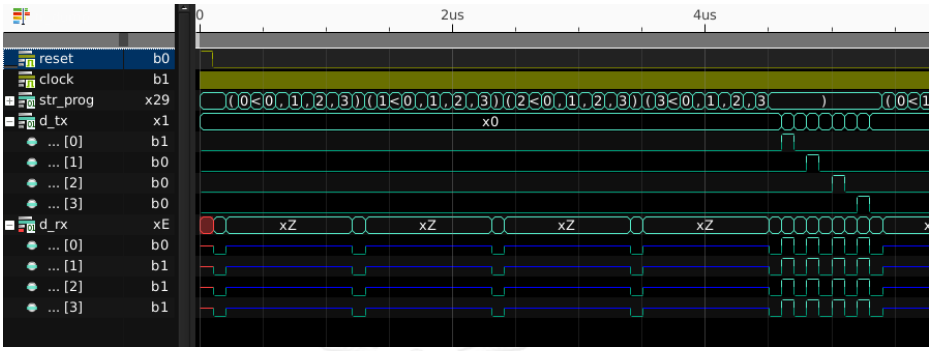
รูปที่ 41 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ FullDOSC Router (3)

```
136             #100 str_prog = "1";
137             #100 str_prog = ")";
138
139             #100 str_prog = "(";
140             #100 str_prog = "1";
141             #100 str_prog = "<";
142             #100 str_prog = "2";
143             #100 str_prog = ")";
144
145             #100 str_prog = "(";
146             #100 str_prog = "2";
147             #100 str_prog = "<";
148             #100 str_prog = "3";
149             #100 str_prog = ")";
150
151             #100 str_prog = "(";
152             #100 str_prog = "3";
153             #100 str_prog = "<";
154             #100 str_prog = "0";
155             #100 str_prog = ")";
156
157             ///// Transfer Data
158             #100 d_tx[0] = 1;
159             #100 d_tx[0] = 0;
160             #100 d_tx[1] = 1;
161             #100 d_tx[1] = 0;
162             #100 d_tx[2] = 1;
163             #100 d_tx[2] = 0;
164             #100 d_tx[3] = 1;
165             #100 d_tx[3] = 0;
166
167             #10 $finish;
168         end
169
170         always begin
171             #2 clock = ~clock;
172         end
173
174         fr#(NUM_DEVICE_TB) U_fr(
175             clock,
176             reset,
177             str_prog,
178             d_tx,
179             d_rx );
180     endmodule
```

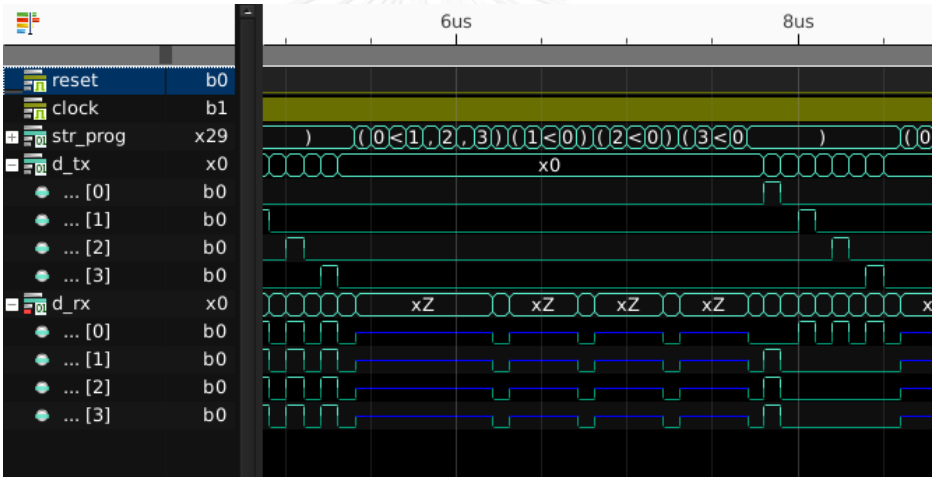
รูปที่ 42 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ FullDOSC Router (4)

ผลการจำลองการทำงาน

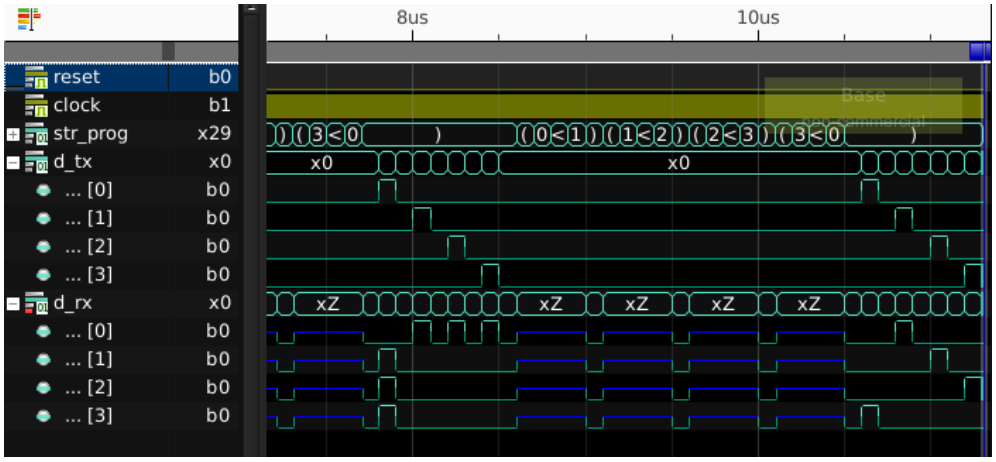
จากการนำรหัสคำสั่งภาษา Verilog HDL ของ test bench และ FullDOSC Router มาจำลองการทำงาน (Simulation) โดยใช้ Icarus Verilog และแสดงผลโดยใช้ Impulse และได้ผลลัพธ์ในการทดสอบระบบได้ดังรูปที่ 43 ถึง รูปที่ 45



รูปที่ 43 รูปแบบของคลื่นของสัญญาณในการทดสอบระบบ FullDOSC (1)



รูปที่ 44 รูปแบบของคลื่นของสัญญาณในการทดสอบระบบ FullDOSC (2)



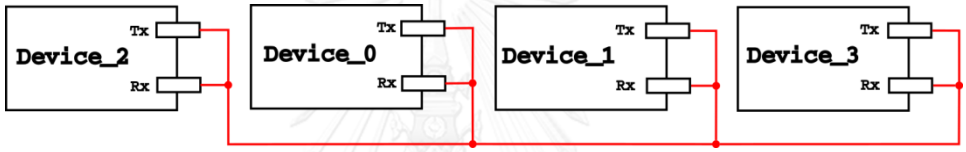
รูปที่ 45 รูปแบบของคลื่นของสัญญาณในการทดสอบระบบ FullDOSC (3)

สัญญาณในรูปที่ 43 ถึง รูปที่ 45 สัญญาณ d_tx คือ สัญญาณที่ส่งมาจากโหนดอุปกรณ์ d_rx คือ สัญญาณที่ส่งมาจากโหนดอุปกรณ์ และรับข้อมูลของโปรแกรมภาษา FRL ผ่านช่องทาง str_prog

จากการจำลองการทำงานโดยซึ่งแบ่งเป็น 3 ส่วนตามทอพอโลยีซึ่งได้แก่

- ทอพอโลยี multi-drop โดยส่วนรหัสคำสั่งในการทดสอบ รูปที่ 39 และ รูปที่ 40 บรรทัดที่ 25 ถึง 81 โดยจะส่งสายอักขระในการกำหนดรูปแบบการเชื่อมต่อดังในนิพจน์ (3) และผลลัพธ์ที่ได้ แสดงในรูปแบบคลื่นสัญญาณใน รูปที่ 43 แสดงการทำงานที่ถูกต้องตามทอพอโลยี คือเมื่อเมื่อมีสัญญาณส่งออกจากโหนดอุปกรณ์ใดๆ จะทำให้เกิดสัญญาณส่งไปยังทุกโหนดอุปกรณ์

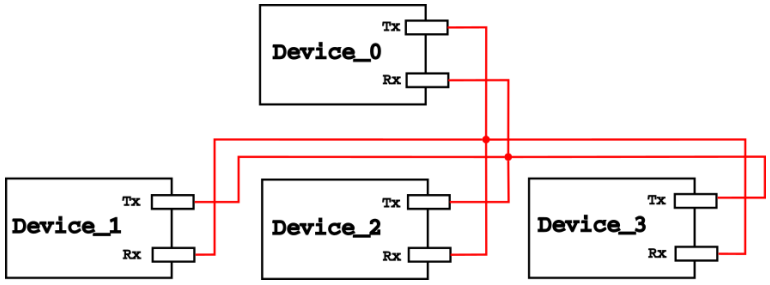
(0<0,1,2,3) (1<0,1,2,3) (2<0,1,2,3) (3<0,1,2,3) -------- (1)



รูปที่ 46 การเชื่อมต่อแบบ multi-drop ตามตัวอย่าง

- ทอพอโลยี master-slave โดยส่วนรหัสคำสั่งในการทดสอบ รูปที่ 40 และ รูปที่ 41 บรรทัดที่ 89 ถึง 125 โดยจะส่งสายอักขระในการกำหนดรูปแบบการเชื่อมต่อดังในนิพจน์ (4) และผลลัพธ์ที่ได้ แสดงในรูปแบบคลื่นสัญญาณใน รูปที่ 44 แสดงการทำงานที่ถูกต้องตามทอพอโลยี คือเมื่อสัญญาณส่องออกจากโหนดอุปกรณ์ 0 ซึ่งกำหนดให้เป็น master จะส่งไปยังโหนดอื่นๆซึ่งกำหนดให้เป็น slave และในทางกลับกันโหนดที่เป็น slave จะส่งให้ master ได้เท่านั้น

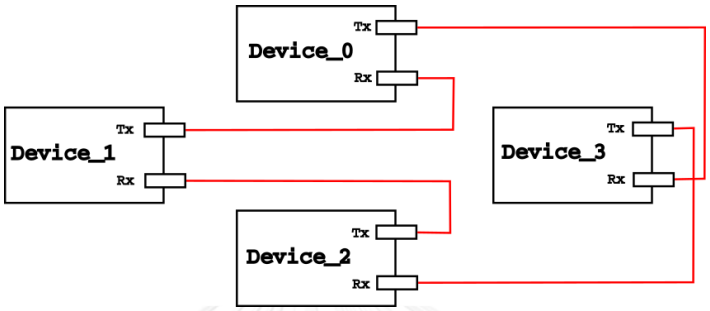
(0<1,2,3) (1<0) (2<0) (3<0) -------- (4)



รูปที่ 47 การเชื่อมต่อแบบ master-slave ตามตัวอย่าง

- ทอพอโลยี ring โดยส่วนรหัสคำสั่งในการทดสอบ รูปที่ 41 และ รูปที่ 42 บรรทัดที่ 132 ถึง 165 โดยจะส่งสายอักขระในการกำหนดรูปแบบการเชื่อมต่อดังในนิพจน์ (5) และผลลัพธ์ที่ได้แสดงในรูปแบบคลื่นสัญญาณใน รูปที่ 45 แสดงการทำงานที่ถูกต้องตามทอพอโลยี คือ สัญญาณจะถูกส่งไปยังโหนดอุปกรณ์หนึ่งวนเป็นวงแหวน

(0<1) (1<2) (2<3) (3<0) -------- (5)	



รูปที่ 48 การเชื่อมต่อแบบ ring ตามตัวอย่าง

รหัสคำสั่งอธิบายส่วนทดสอบของ modified FullDOSC Router

ในส่วนการทดสอบระบบนั้นจะทำการแปลภาษาโดยใช้ Icarus Verilog และใช้ Impulse จาก toem ในการแสดงผล (waveform) และทำการสร้างรหัสคำสั่งภาษา Verilog HDL สำหรับทดสอบระบบ (Test bench) โดยอ้างอิงจากทอพอโลยีในรูปที่ 49 ถึง รูปที่ 54

```
1  `timescale 1ns / 1ps
2
3  module fr_tb(
4
5  );
6      parameter NUM_DEVICE_TB = 4;
7
8      reg clock, reset;
9      reg [7:0] str_prog;
10     reg [NUM_DEVICE_TB-1:0]d_tx;
11     wire [NUM_DEVICE_TB-1:0]d_rx;
12
13     initial begin
14         $dumpfile("fr_dump.vcd");
15         $dumpvars(0,fr_tb);
16
17         clock = 0;
18         reset = 1;
19         d_tx = 0;
20         str_prog = 8'h00;
21         #100 reset = 0;
22
23         ///// Program Route Path
24         #100 str_prog = "(";
25         #100 str_prog = "|";
26         #100 str_prog = "0";
27         #100 str_prog = "<";
28         #100 str_prog = "$";
29         #100 str_prog = "3";
30         #100 str_prog = ")";
31
32         #100 str_prog = "(";
33         #100 str_prog = "|";
34         #100 str_prog = "1";
35         #100 str_prog = "<";
36         #100 str_prog = "$";
37         #100 str_prog = "3";
38         #100 str_prog = "$";
39         #100 str_prog = "2";
40         #100 str_prog = ")";
41
42         #100 str_prog = "(";
43         #100 str_prog = "|";
44         #100 str_prog = "2";
45         #100 str_prog = "<";
46         #100 str_prog = "$";
47         #100 str_prog = "3";
48         #100 str_prog = "$";
49         #100 str_prog = "2";
50         #100 str_prog = "$";
```

รูปที่ 49 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ modified FullDOSC Router (1)

```
51          #100 str_prog = "1";
52          #100 str_prog = ")";
53
54          #100 str_prog = "(";
55          #100 str_prog = "|";
56          #100 str_prog = "3";
57          #100 str_prog = "<";
58          #100 str_prog = "$";
59          #100 str_prog = "3";
60          #100 str_prog = "$";
61          #100 str_prog = "2";
62          #100 str_prog = "$";
63          #100 str_prog = "1";
64          #100 str_prog = "$";
65          #100 str_prog = "0";
66          #100 str_prog = ")";
67
68          ///// Transfer Data
69          #100 d_tx = 4'b0000;
70          #100 d_tx = 4'b0001;
71          #100 d_tx = 4'b0000;
72          #100 d_tx = 4'b0011;
73          #100 d_tx = 4'b0000;
74          #100 d_tx = 4'b0111;
75          #100 d_tx = 4'b0000;
76          #100 d_tx = 4'b1111;
77          #100 d_tx = 4'b0000;
78          #100 d_tx = 4'b1110;
79          #100 d_tx = 4'b0000;
80          #100 d_tx = 4'b1100;
81          #100 d_tx = 4'b0000;
82          #100 d_tx = 4'b1000;
83          #100 d_tx = 4'b0000;
84
85
86          ///// Program Route Path
87          #100 str_prog = "(";
88          #100 str_prog = "|";
89          #100 str_prog = "0";
90          #100 str_prog = "<";
91          #100 str_prog = "!";
92          #100 str_prog = "3";
93          #100 str_prog = ")";
94
95          #100 str_prog = "(";
96          #100 str_prog = "|";
97          #100 str_prog = "1";
98          #100 str_prog = "<";
99          #100 str_prog = "!";
100         #100 str_prog = "3";
```

รูปที่ 50 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ modified FullDOSC Router (2)

```
101         #100 str_prog = "!";
102         #100 str_prog = "2";
103         #100 str_prog = ")";
104
105         #100 str_prog = "(";
106         #100 str_prog = "|";
107         #100 str_prog = "2";
108         #100 str_prog = "<";
109         #100 str_prog = "!";
110         #100 str_prog = "3";
111         #100 str_prog = "!";
112         #100 str_prog = "2";
113         #100 str_prog = "!";
114         #100 str_prog = "1";
115         #100 str_prog = ")";
116
117         #100 str_prog = "(";
118         #100 str_prog = "|";
119         #100 str_prog = "3";
120         #100 str_prog = "<";
121         #100 str_prog = "!";
122         #100 str_prog = "3";
123         #100 str_prog = "!";
124         #100 str_prog = "2";
125         #100 str_prog = "!";
126         #100 str_prog = "1";
127         #100 str_prog = "!";
128         #100 str_prog = "0";
129         #100 str_prog = ")";
130
131         ///// Transfer Data
132         #100 d_tx = 4'b1111;
133         #100 d_tx = 4'b1110;
134         #100 d_tx = 4'b1111;
135         #100 d_tx = 4'b1100;
136         #100 d_tx = 4'b1111;
137         #100 d_tx = 4'b1000;
138         #100 d_tx = 4'b1111;
139         #100 d_tx = 4'b0000;
140         #100 d_tx = 4'b1111;
141         #100 d_tx = 4'b0111;
142         #100 d_tx = 4'b1111;
143         #100 d_tx = 4'b0011;
144         #100 d_tx = 4'b1111;
145         #100 d_tx = 4'b0001;
146         #100 d_tx = 4'b1111;
147         #100 d_tx = 4'b0000;
148         #100 d_tx = 4'b1111;
149
```

รูปที่ 51 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ modified FullDOSC Router (3)

```
150         ///// Program Route Path
151         #100 str_prog = "(";
152         #100 str_prog = "&";
153         #100 str_prog = "0";
154         #100 str_prog = "<";
155         #100 str_prog = "$";
156         #100 str_prog = "3";
157         #100 str_prog = ")";
158
159         #100 str_prog = "(";
160         #100 str_prog = "&";
161         #100 str_prog = "1";
162         #100 str_prog = "<";
163         #100 str_prog = "$";
164         #100 str_prog = "3";
165         #100 str_prog = "$";
166         #100 str_prog = "2";
167         #100 str_prog = ")";
168
169         #100 str_prog = "(";
170         #100 str_prog = "&";
171         #100 str_prog = "2";
172         #100 str_prog = "<";
173         #100 str_prog = "$";
174         #100 str_prog = "3";
175         #100 str_prog = "$";
176         #100 str_prog = "2";
177         #100 str_prog = "$";
178         #100 str_prog = "1";
179         #100 str_prog = ")";
180
181         #100 str_prog = "(";
182         #100 str_prog = "&";
183         #100 str_prog = "3";
184         #100 str_prog = "<";
185         #100 str_prog = "$";
186         #100 str_prog = "3";
187         #100 str_prog = "$";
188         #100 str_prog = "2";
189         #100 str_prog = "$";
190         #100 str_prog = "1";
191         #100 str_prog = "$";
192         #100 str_prog = "0";
193         #100 str_prog = ")";
194
195         ///// Transfer Data
196         #100 d_tx = 4'b1111;
197         #100 d_tx = 4'b1110;
198         #100 d_tx = 4'b1111;
199         #100 d_tx = 4'b1100;
200         #100 d_tx = 4'b1111;
```

รูปที่ 52 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ modified FullDOSC Router (4)

```
201         #100 d_tx = 4'b1000;
202         #100 d_tx = 4'b1111;
203         #100 d_tx = 4'b0000;
204         #100 d_tx = 4'b1111;
205         #100 d_tx = 4'b0111;
206         #100 d_tx = 4'b1111;
207         #100 d_tx = 4'b0011;
208         #100 d_tx = 4'b1111;
209         #100 d_tx = 4'b0001;
210         #100 d_tx = 4'b1111;
211         #100 d_tx = 4'b0000;
212         #100 d_tx = 4'b1111;
213
214         ///// Program Route Path
215         #100 str_prog = "(";
216         #100 str_prog = "&";
217         #100 str_prog = "0";
218         #100 str_prog = "<";
219         #100 str_prog = "!";
220         #100 str_prog = "3";
221         #100 str_prog = ")";
222
223         #100 str_prog = "(";
224         #100 str_prog = "&";
225         #100 str_prog = "1";
226         #100 str_prog = "<";
227         #100 str_prog = "!";
228         #100 str_prog = "3";
229         #100 str_prog = "!";
230         #100 str_prog = "2";
231         #100 str_prog = ")";
232
233         #100 str_prog = "(";
234         #100 str_prog = "&";
235         #100 str_prog = "2";
236         #100 str_prog = "<";
237         #100 str_prog = "!";
238         #100 str_prog = "3";
239         #100 str_prog = "!";
240         #100 str_prog = "2";
241         #100 str_prog = "!";
242         #100 str_prog = "1";
243         #100 str_prog = ")";
244
245         #100 str_prog = "(";
246         #100 str_prog = "&";
247         #100 str_prog = "3";
248         #100 str_prog = "<";
249         #100 str_prog = "!";
250         #100 str_prog = "3";
```

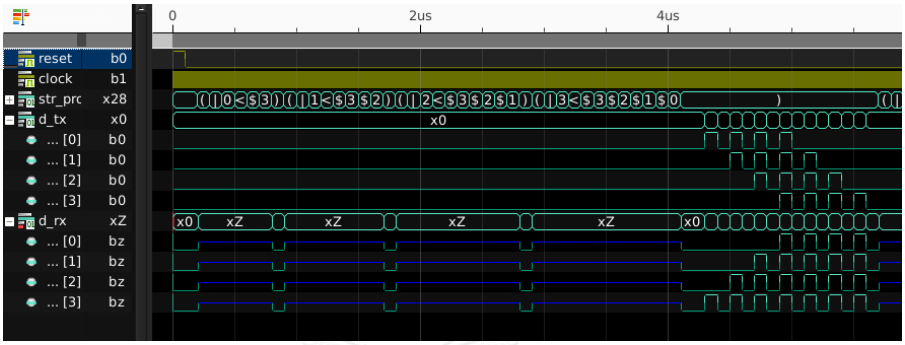
รูปที่ 53 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ modified FullDOSC Router (5)

```
251         #100 str_prog = "!";
252         #100 str_prog = "2";
253         #100 str_prog = "!";
254         #100 str_prog = "1";
255         #100 str_prog = "!";
256         #100 str_prog = "0";
257         #100 str_prog = ")";
258
259         ///// Transfer Data
260         #100 d_tx = 4'b0000;
261         #100 d_tx = 4'b0001;
262         #100 d_tx = 4'b0000;
263         #100 d_tx = 4'b0011;
264         #100 d_tx = 4'b0000;
265         #100 d_tx = 4'b0111;
266         #100 d_tx = 4'b0000;
267         #100 d_tx = 4'b1111;
268         #100 d_tx = 4'b0000;
269         #100 d_tx = 4'b1110;
270         #100 d_tx = 4'b0000;
271         #100 d_tx = 4'b1100;
272         #100 d_tx = 4'b0000;
273         #100 d_tx = 4'b1000;
274         #100 d_tx = 4'b0000;
275
276         #100 $finish;
277     end
278
279     always begin
280         #2 clock = ~clock;
281     end
282
283     fr#(NUM_DEVICE_TB) U_fr(
284         clock,
285         reset,
286         str_prog,
287         d_tx,
288         d_rx );
289 endmodule
```

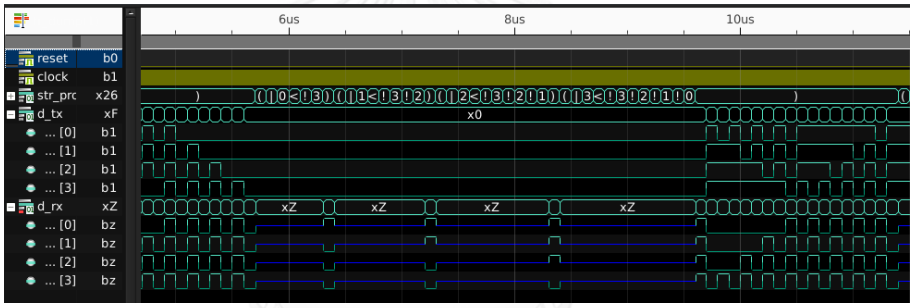
รูปที่ 54 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ modified FullDOSC Router (6)

ผลการจำลองการทำงาน

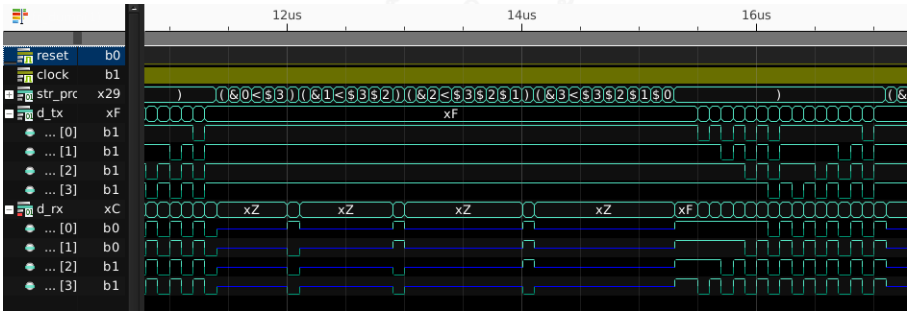
จากการนำรหัสคำสั่งภาษา Verilog HDL ของ test bench และ modified FullDOSC Router มาจำลองการทำงาน (Simulation) โดยใช้ Icarus Verilog และแสดงผลโดยใช้ Impulse และได้ผลลัพธ์ในการทดสอบระบบได้ดังรูปที่ 55 ถึง รูปที่ 58



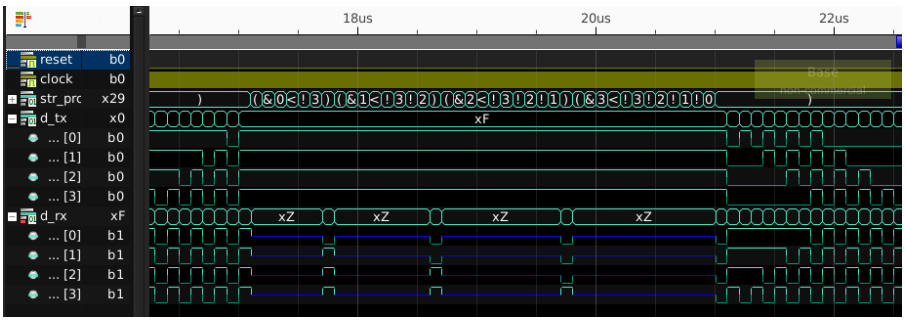
รูปที่ 55 รูปแบบของคลื่นของสัญญาณในการทดสอบระบบ mFullDOSC (1)



รูปที่ 56 รูปแบบของคลื่นของสัญญาณในการทดสอบระบบ mFullDOSC (2)



รูปที่ 57 รูปแบบของคลื่นของสัญญาณในการทดสอบระบบ mFullDOSC (3)



รูปที่ 58 รูปแบบของคลื่นของสัญญาณในการทดสอบระบบ mFullDOSC (4)

สัญญาณในรูปที่ 55 ถึง รูปที่ 58 สัญญาณ d_tx คือ สัญญาณที่ส่งมาจากโหนดอุปกรณ์ d_rx คือ สัญญาณที่ส่งมาจากโหนดอุปกรณ์ และรับข้อมูลของโปรแกรมภาษา FRL ผ่านช่องทาง str_prog

จากการจำลองการทำงานเพื่อแสดงความสามารถเพิ่มเติมของ modified FullDOSC จาก FullDOSC ซึ่งคือความสามารถในการสลับสัญญาณรับเข้าและเลือกวิธีรวมข้อมูล ดังนั้นจึงสร้างการเชื่อมตัวต่ออย่างดังรูปที่ 59 มีทั้งหมด 4 นิพจน์ซึ่งอธบายการเชื่อมต่อของสายสัญญาณสีเขียว, น้ำเงิน , ม่วง และแดง ตามลำดับ และแบ่งออกเป็น 4 กรณีได้แก่

- กรณีไม่สลับสัญญาณรับเข้า และรวมสัญญาณแบบ OR หรือดึงสัญญาณลง (pull-down) ในสัญญาณส่งออกของอุปกรณ์จัดการการเชื่อมต่อ ซึ่งสามารถเขียนเป็นภาษา mFRL ได้ดังนิพจน์ (10)

(|0<$3) (|1<$2$3) (|2<$1$2$3) (|3<$0$1$2$3) -------- (10)	

- กรณีสลับสัญญาณรับเข้า และรวมสัญญาณแบบ OR หรือดึงสัญญาณลง (pull-down) ในสัญญาณส่งออกของอุปกรณ์จัดการการเชื่อมต่อ ซึ่งสามารถเขียนเป็นภาษา mFRL ได้ดังนิพจน์ (11)

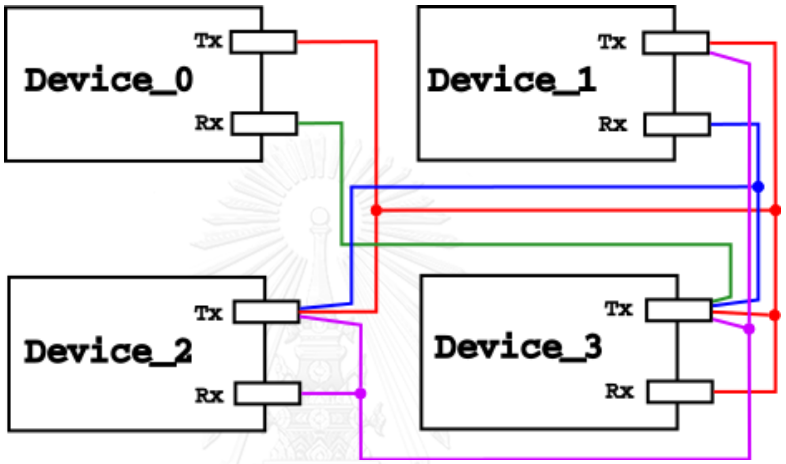
(|0<!3) (|1<!2!3) (|2<!1!2!3) (|3<!0!1!2!3) -------- (11)	

- กรณีไม่สลับสัญญาณรับเข้า และรวมสัญญาณแบบ AND หรือดึงสัญญาณลง (pull-up) ในสัญญาณส่งออกของอุปกรณ์จัดการการเชื่อมต่อ ซึ่งสามารถเขียนเป็นภาษา mFRL ได้ดังนิพจน์ (12)

(&0<$3) (&1<$2$3) (&2<$1$2$3) (&3<$0$1$2$3) -------- (12)	

- กรณีสลับสัญญาณรับเข้า และรวมสัญญาณแบบ AND หรือดึงสัญญาณลง (pull-up) ในสัญญาณส่งออกของอุปกรณ์จัดการการเชื่อมต่อ ซึ่งสามารถเขียนเป็นภาษา mFRL ได้ดังนิพจน์ (13)

(&0<!3) (&1<!2!3) (&2<!1!2!3) (&3<!0!1!2!3) -------- (13)	



รูปที่ 59 การเชื่อมต่อเพื่อทดสอบระบบตัวอย่าง

การจำลองโดยใช้กระบวนการแยกกัน

เป็นวิธีการที่นำเสนอในงานวิจัยนี้ เนื่องจากปัญหาในการจัดการโหนดอุปกรณ์แต่ละตัว ซึ่งสามารถนำเอาแกนหน่วยประมวลผลที่สมบูรณ์ในตัวแล้วเช่น OpenMSP430 และ Amber core จาก OpenCore หรือโปรแกรมจำลองการทำงานซึ่งอาจถูกสร้างขึ้นโดยภาษาอื่นเช่น C หรือ Python ซึ่งไม่สามารถถูกคลุมโดย Test Bench ซึ่งถูกอธิบายโดยใช้ภาษา HDL ได้

จากปัญหาดังกล่าวสามารถแก้ปัญหาได้โดยการแยกหน่วยจำลองภาษา Verilog HDL ซึ่งในงานนี้ใช้ Verification and Validation Plan (VVP) ซึ่งในงานวิจัยนี้ใช้ Icarus Verilog vvp runtime engine แยกกระบวนการ (process) ในแต่ละโหนดอุปกรณ์ ซึ่งจะทำให้สามารถจัดการจำลองซึ่งรวมถึงการโปรแกรมเข้าไปในแต่ละโหนด หรือเปิดปิดโหนด หรืออาจจะสร้างโหนดอุปกรณ์โดยใช้ภาษาอื่น และสร้างช่องทางการเชื่อมโดยใช้ Multipurpose relay (SOcket CAT) หรือคำสั่ง socat ดังตัวอย่างในรูปที่ 61 ถึง รูปที่ 63 ซึ่งสามารถทำงานร่วมกับ standard I/O ฟังก์ชั่นของ Verilog HDL และ C รวมถึงภาษาอื่นที่สามารถเขียนและอ่านไฟล์ได้ เพื่อสร้างโครงสร้างในรูปที่ 64

```
# socat -d -d pty,raw,echo=0 pty,raw,echo=0
2016/11/11 07:00:47 socat[7454] N PTY is /dev/pts/5
2016/11/11 07:00:47 socat[7454] N PTY is /dev/pts/6
2016/11/11 07:00:47 socat[7454] N starting data transfer lo
op with FDs [5,5] and [7,7]
```

รูปที่ 60 ตัวอย่างการใช้คำสั่ง socat ในการสร้างช่องทางเชื่อมต่อ

รูปที่ 61 แผนภาพแสดงการใช้งาน vvp ร่วมกับ socat

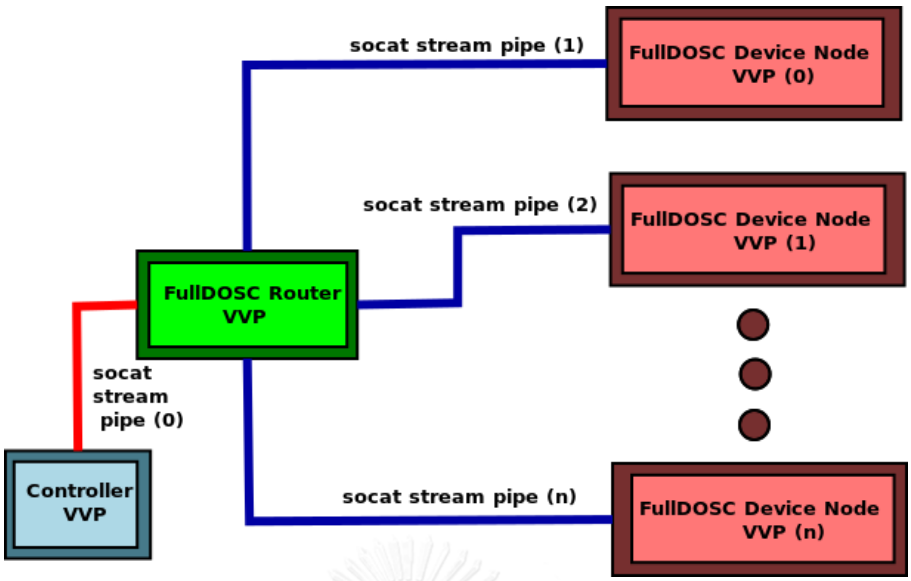
```
f = $fopen("/dev/pts/5","r+");
f = $fopen("/dev/pts/6","r+");
```

รูปที่ 62 รหัสคำสั่งภาษา Verilog HDL เพื่อเปิดไฟล์ หรือเปิดช่องทางการเชื่อมต่อ

```
in = $fgetc(f);

$fwrite(f,"%c",in);
```

รูปที่ 63 รหัสคำสั่งภาษา Verilog HDL เพื่ออ่านและเขียนลงไฟล์เสมือนรับและส่งข้อมูล



รูปที่ 64 แผนภาพแสดงการใช้ vvp ร่วมกับ socat ในการจำลองการทำงาน

จากหลักการที่ได้กล่าวมาแล้ว ในงานวิจัยนี้ได้ทำการจำลองการทำงานบางส่วนของระบบ FullDOSC โดยใช้กระบวนการแยกกัน โดยทำการสร้าง test bench ภาษา Verilog HDL ครอบส่วนของ FullDOSC Router ดังรูปที่ 67 และ รูปที่ 68 ซึ่งจะเป็นส่วนที่ทำหน้าที่เชื่อมต่อ FullDOSC router เข้ากับช่องทางเสมือนที่สร้างด้วยคำสั่ง socat สำหรับส่วนที่จะมาต่อนั้นจะเขียนเป็นโปรแกรมภาษา C ได้แก่ Tick (รูปที่ 66) เพื่อทำการส่งสัญญาณให้กับ FullDOSC router เพื่อให้ระบบสามารถทำงานต่อเนื่องไปได้ เนื่องจาก I/O interface ใน Verilog HDL เป็นแบบ Synchronous และ Controller (รูปที่ 65) สำหรับส่งชุดสายอักระภาษา FRL ไปยัง test bench ดังกล่าว

```c
#include<stdio.h>
#include<string.h>

char str[256];

void main(){
    FILE* pf;
    int i;
    while(1){
        scanf("%s",str);
        for(i=0;i<strlen(str);i++){
            pf = fopen("/dev/pts/7","r+");
            fputc(str[i],pf);
            fclose(pf);
        }
        printf("SEND : %s\n",str);
    }
}
```

รูปที่ 65 รหัสคำสั่งภาษา C สำหรับสำหรับโปรแกรม Controller

```c
#include<stdio.h>
#include<string.h>

int n;

void main(){
    FILE* pf;
    int i;
    while(1){
        scanf("%d",&n);
        for(i=0;i<n;i++){
            pf = fopen("/dev/pts/7","r+");
            fputc('.',pf);
            fclose(pf);
        }
    }
}
```

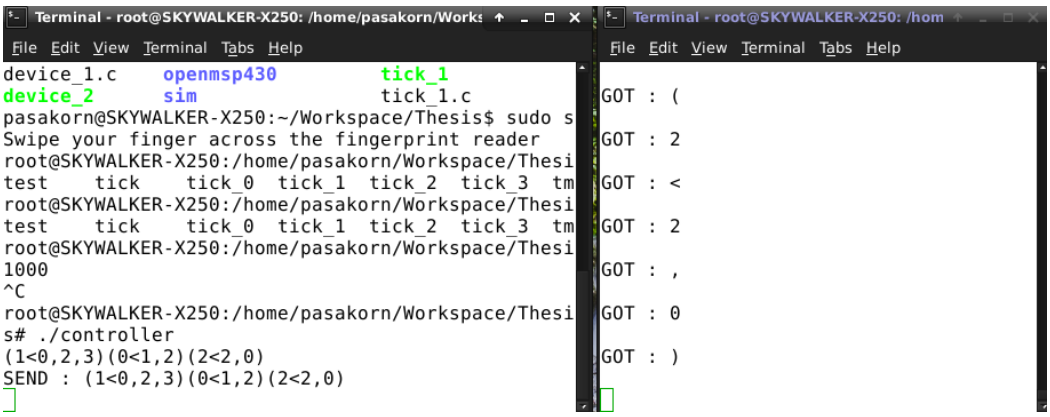
รูปที่ 66 รหัสคำสั่งภาษา C สำหรับสำหรับโปรแกรม Tick

```
1  `timescale 1ns / 1ps
2
3  module fr_ds
4  (
5  );
6      parameter NUM_DEVICE_TB = 4;
7      parameter STR_BUFF_MAX = 255;
8
9      reg clock, reset;
10     reg [7:0] str_prog;
11     reg [NUM_DEVICE_TB-1:0]d_tx;
12     wire [NUM_DEVICE_TB-1:0]d_rx;
13
14     integer f,i;
15     integer f_in[NUM_DEVICE_TB-1:0];
16     integer f_out[NUM_DEVICE_TB-1:0];
17     reg [7:0]   out_s,in_s;
18     reg [7:0]   out_d[NUM_DEVICE_TB-1:0];
19     reg [7:0]   in_d[NUM_DEVICE_TB-1:0];
20     reg [7:0]   str_buff[STR_BUFF_MAX:0];
21     integer start_buff, end_buff;
22
23     initial begin
24         $dumpfile("fr_ds_dump.vcd");
25         $dumpvars(0,fr_ds);
26         start_buff = 0;
27         end_buff = 0;
28         d_tx = 4'b0000;
29         reset = 1;
30         f = $fopen("/dev/pts/6","r");
31
32         f_in[0] = $fopen("/dev/pts/9","r");
33         f_in[1] = $fopen("/dev/pts/12","r");
34         f_in[2] = $fopen("/dev/pts/15","r");
35         f_in[3] = $fopen("/dev/pts/18","r");
36
37         f_out[0] = $fopen("/dev/pts/21","w");
38         f_out[1] = $fopen("/dev/pts/24","w");
39         f_out[2] = $fopen("/dev/pts/27","w");
40         f_out[3] = $fopen("/dev/pts/30","w");
```

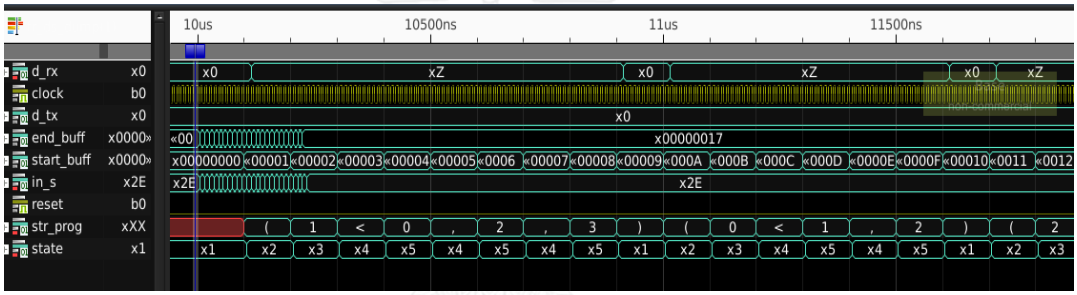
รูปที่ 67 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ FullDOSC Router ในการจำลองโดยใช้กระบวนการแยกกัน (1)

```
41          #100 reset = 0;
42      end
43
44      initial begin
45          clock = 0;
46          forever #5 clock = ~clock;
47      end
48
49      always@(posedge clock) begin
50          in_s = $fgetc(f);
51          if(in_s == "q") begin
52              $finish;
53          end
54          else if(in_s == ".") begin
55             $display(".");
56          end
57          else  begin
58              $display("GOT : %c\n",in_s);
59              str_buff[end_buff] = in_s;
60              end_buff = end_buff > STR_BUFF_MAX - 1 ?
61                                          0 : end_buff + 1;
62          end
63      end
64
65      always begin
66          #100 if(start_buff != end_buff) begin
67              str_prog = str_buff[start_buff];
68              start_buff = start_buff > STR_BUFF_MAX - 1 ?
69                                      0 : start_buff + 1;
70          end
71      end
72
73
74      fr#(NUM_DEVICE_TB) U_fr(
75              clock,
76              reset,
77              str_prog,
78              d_tx,
79              d_rx );
80  endmodule
```

รูปที่ 68 รหัสคำสั่งภาษา Verilog HDL สำหรับการอธิบาย Test bench ของ FullDOSC Router ในการจำลองโดยใช้กระบวนการแยกกัน (2)



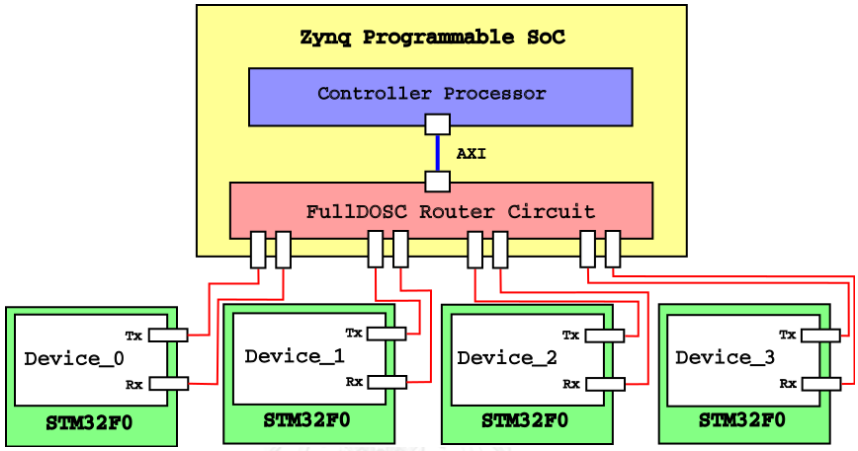
รูปที่ 69 หน้าต่างแสดงการทำงานเชิงโต้ตอบโดยใช้การจำลองโดยใช้กระบวนการแยกกัน



รูปที่ 70 ผลการทำงานของการจำลองโดยใช้กระบวนการแยกกัน

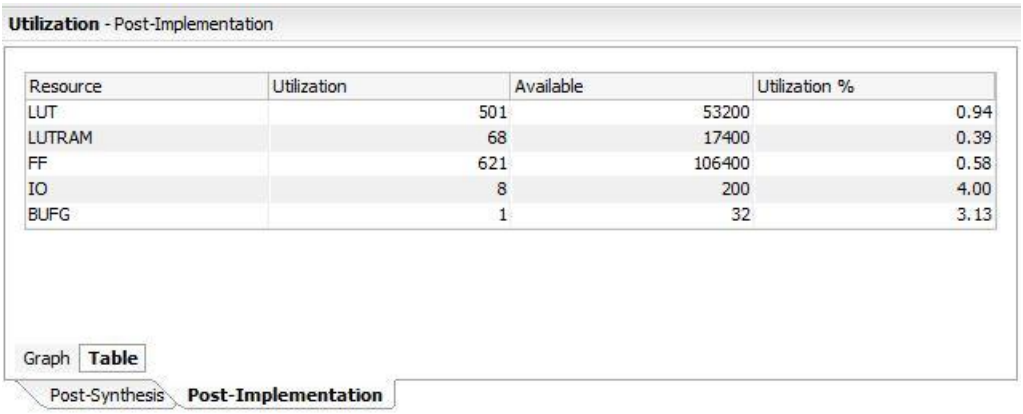
การใช้งานระบบในอุปกรณ์จริง (Physical Hardware Implementation)

งานวิจัยนี้ได้ทำการใช้งานระบบสื่อสารแบบอนุกรมสองทางในอุปกรณ์จริง โดยใช้ Xilinx Zynq SoC ซึ่งภายในจะประกอบไปด้วย FPGA ซึ่งจะถูกนำมาใช้เป็นส่วนจัดการการ (FullDOSC Router Circuit) และ ARM Cortex-A9 Processor core ซึ่งจะใช้เป็นตัวควบคุมหรือส่วนที่จะส่งโปรแกรม FRL/mFRL ไปยังส่วนจัดการการเชื่อมต่อ มีโครงสร้างดังรูปที่ 71



รูปที่ 71 โครงสร้างตัวอย่างของระบบในอุปกรณ์จริง

เมื่อทำการออกแบบโครงสร้างของระบบที่ต้องการแล้ว จึงได้เพิ่มเติมในส่วนรายละเอียดภายในชิปโดยรวมสำหรับใช้ในอุปกรณ์ดังรูปที่ 74 ซึ่งภายในนั้นจะประกอบไปด้วยโครงสร้างหลักได้แก่ ZYNQ7 Processor system ซึ่งทำหน้าที่เป็นตัวควบคุมส่งชุดคำสั่งภาษา FRL/mFRL ไปยังส่วนจัดการการเชื่อมต่อคือ fulldosc_router_v1_0 ซึ่งรับข้อมูลจากโหนดอุปกรณ์ผ่านทาง rx[3:0] และส่งข้อมูลไปยังโนดอุปกรณ์ผ่านทาง tx[3:0] และมีส่วนประกอบอื่นๆเป็นส่วนเสริม (สร้างโดยใช้คำสั่ง Auto-Generate ใน Xilinx Vivado) การออกแบบดังกล่าวนี้มีการใช้ทรัพยากรสำหรับภาษา FRL ดังรูปที่ 72 และภาษา mFRL ดังรูปที่ 73



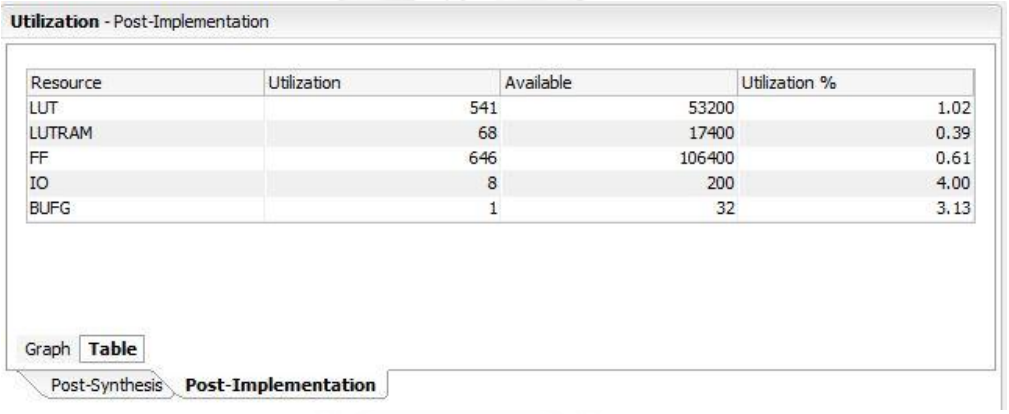
Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
LUT	501	53200	0.94
LUTRAM	68	17400	0.39
FF	621	106400	0.58
IO	8	200	4.00
BUFG	1	32	3.13

Graph **Table**

Post-Synthesis **Post-Implementation**

รูปที่ 72 การใช้ทรัพยากรของ FullDOSC Router รวมกับส่วนควบคุมบน Xilinx Zynq-7000 สำหรับจัดการโหนดอุปกรณ์ 4 โหนด



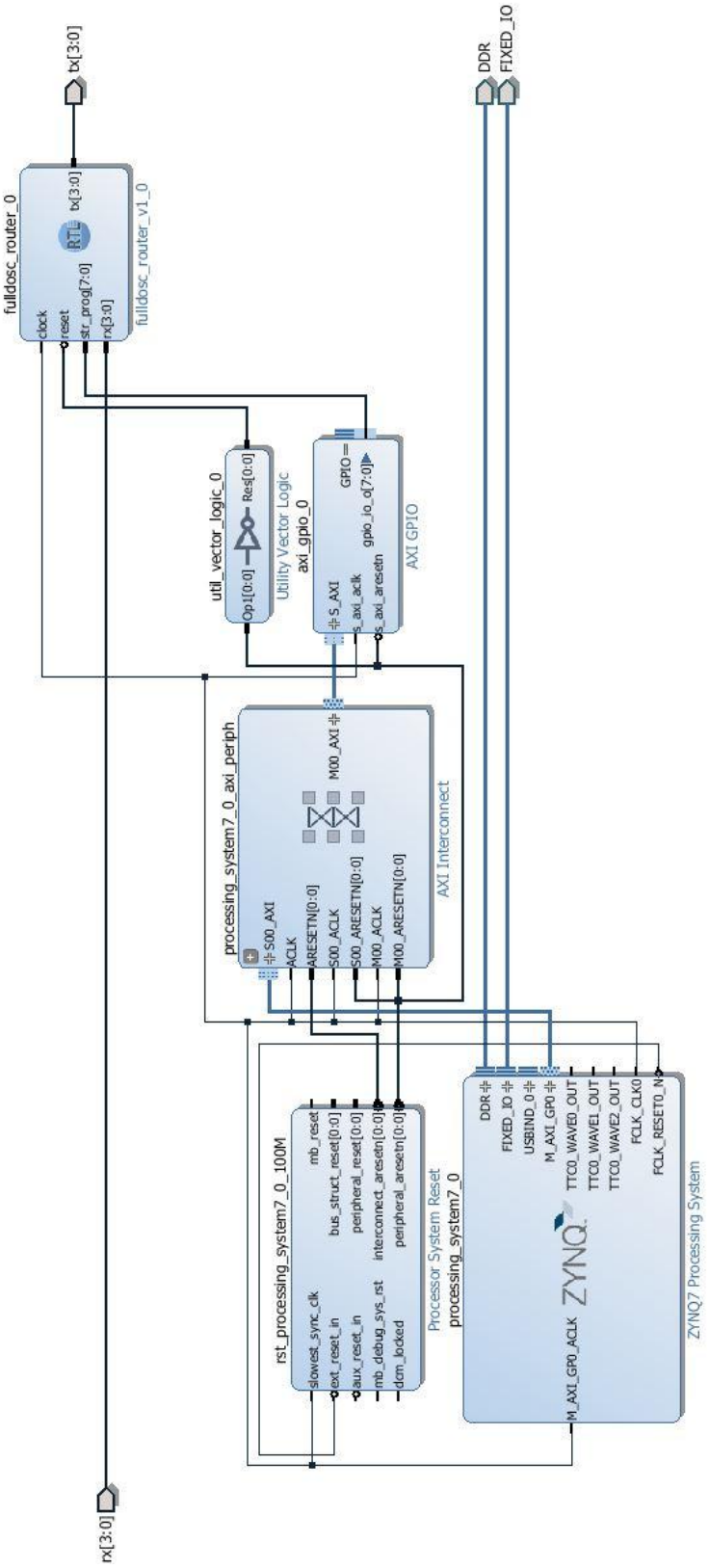
Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
LUT	541	53200	1.02
LUTRAM	68	17400	0.39
FF	646	106400	0.61
IO	8	200	4.00
BUFG	1	32	3.13

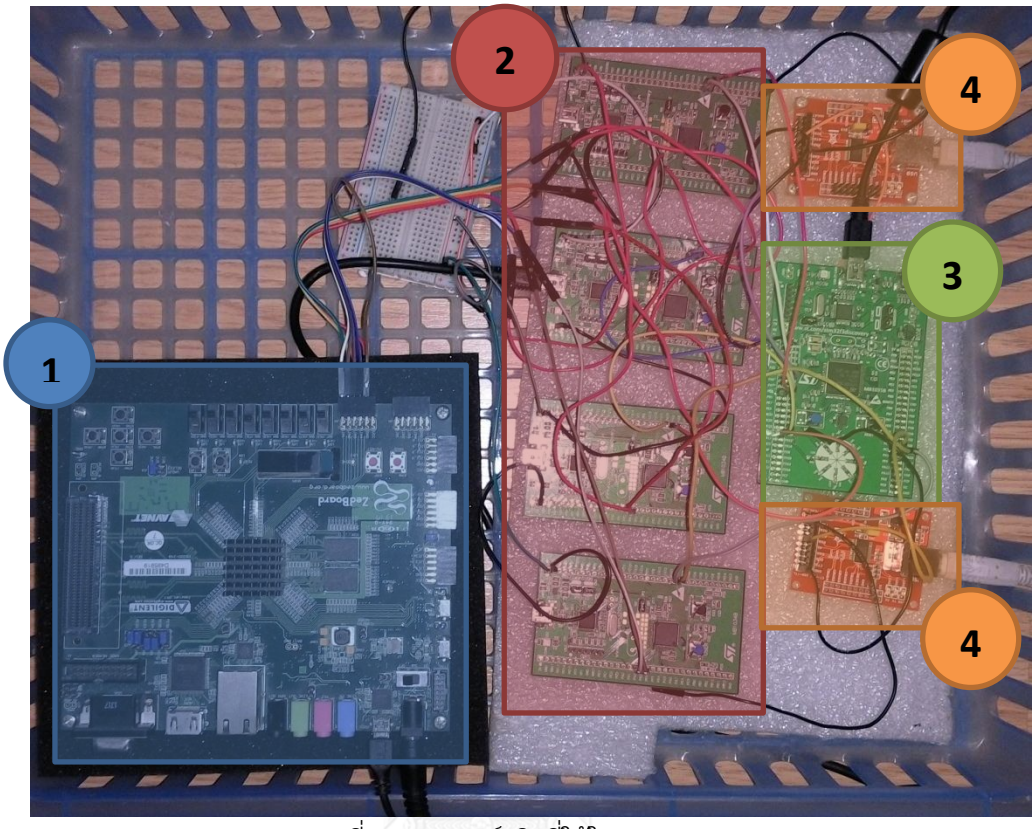
Graph **Table**

Post-Synthesis **Post-Implementation**

รูปที่ 73 การใช้ทรัพยากรของ modified FullDOSC Router รวมกับส่วนควบคุมบน Xilinx Zynq-7000 สำหรับจัดการโหนดอุปกรณ์ 4 โหนด



รูปที่ 74 แผนภาพแสดงการออกแบบโครงสร้าง FullDOSC Router บน Xilinx Zynq SoC ด้วย Xilinx Vivado



รูปที่ 75 อุปกรณ์จริงที่ใช้ในการทดสอบ

จากโครงสร้างที่ออกแบบไว้ในรูปที่ 71 ได้ทำการเชื่อมต่ออุปกรณ์จริงในงานวิจัยนี้เพื่อทดสอบการทำงานได้ดังรูปที่ 75 ซึ่งประกอบไปด้วย

1. Digilent FPGA ZedBoard Zynq-7000 ARM/FPGA SoC Development Board ทำหน้าที่เป็นอุปกรณ์จัดการการเชื่อมต่อ และมีตัวควบคุมเป็น ARM processor core อยู่ภายใน
2. STM32F0 Discovery Board ทำหน้าที่เป็นโหนดอุปกรณ์ 4 โหนดในระบบ
3. STM32F3 Discovery Board ทำหน้าที่เป็นตัวควบคุมและรับข้อมูลโหนดอุปกรณ์เพื่อช่วยในการสั่งงานโหนดอุปกรณ์ ซึ่งรับคำสั่งผ่านคอมพิวเตอร์
4. ETT-Mini USB-TTL Board เป็นตัวส่งข้อมูลที่ได้รับจากคอมพิวเตอร์ซึ่งผ่านพอร์ต USB ไปยังอุปกรณ์ในข้อ 3 และช่วยในการตรวจสอบข้อมูลที่ส่งระหว่างบอร์ด

9. วิเคราะห์ผลจากงานวิจัย

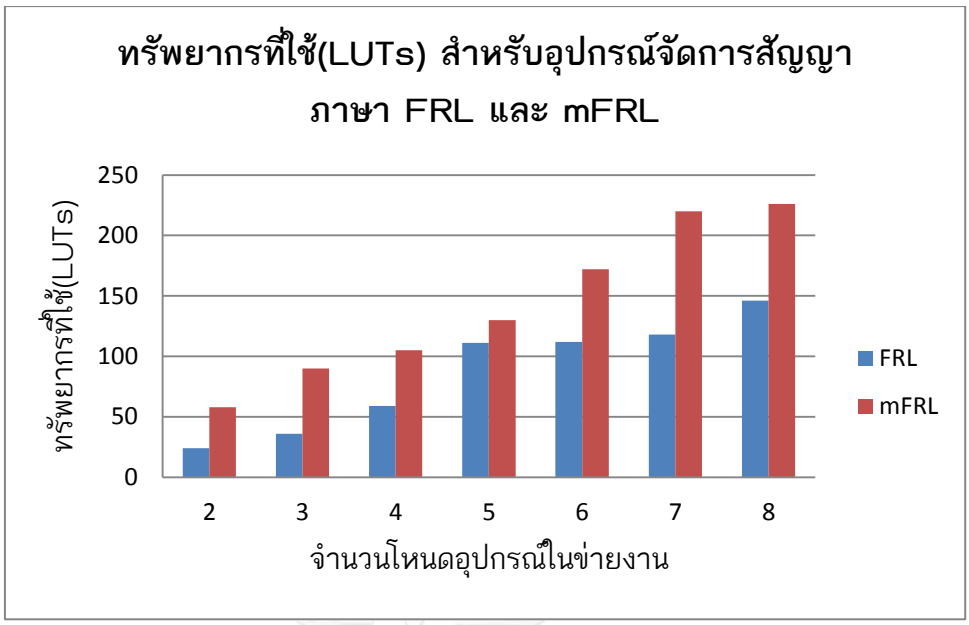
9.1 การใช้ทรัพยากรของอุปกรณ์เชื่อมต่อสัญญาณ (FullDOSC Router)

จากการนำภาษา FRL และ mFRL มาใช้ในอุปกรณ์จัดการการเชื่อมต่อในหัวข้อ 7.2 และ 7.3 มาออกแบบในโปรแกรม Xilinx Vivado 2016.2 โดยเลือกอุปกรณ์เป็น Zynq-7000 ซึ่งมี LUT เป็นแบบ 6-input LUT และทำการใช้คำสั่ง Synthesis และ Implement ซึ่งเป็นการจำลองการทำงานโดยคิดปัจจัยในอุปกรณ์จริงด้วย และได้ผลดังตารางที่ 2

จำนวนโหนดอุปกรณ์	ทรัพยากรที่ใช้ (LUTs)		%
	FRL	mFRL	
2	24	58	141.67
3	36	90	150.00
4	59	105	77.96
5	111	130	17.12
6	112	172	53.57
7	118	220	86.44
8	146	226	54.79
		% Average	83.08

ตารางที่ 2 ทรัพยากรที่ใช้(LUTs) สำหรับอุปกรณ์จัดการสัญญาณภาษา FRL และ mFRL

จากตารางที่ 2 จะเห็นได้ว่าการใช้ทรัพยากรของ mFRL Router มากกว่า FRL อยู่ถึง 83.08% เมื่อคิดค่าเฉลี่ยจากจำนวนโหนดอุปกรณ์จำนวน 2 ถึง 8 โหนด ดังนั้นถ้าการส่งสัญญาณในระบบเป็นรูปแบบที่สามารถรองรับได้ด้วย FRL เช่น UART จึงควรใช้ FRL ในการอธิบายการเชื่อมต่อ แต่ถึงอย่างไรก็ตาม การใช้ทรัพยากรของทั้งสองรูปแบบนี้ก็ถือว่าใช้น้อยมากเมื่อเทียบกับทรัพยากรใน FPGA ปัจจุบัน (Spartan-7 มี 24,000 LUT ขึ้นไป) ซึ่งใช้แค่ 146 และ 226 LUT สำหรับระบบที่มี 8 โหนดอุปกรณ์



แผนภูมิที่ 1 ทรัพยากรที่ใช้(LUTs) สำหรับอุปกรณ์จัดการสัญญาภาษา FRL และ mFRL

9.2 เปรียบเทียบการใช้ทรัพยากรของ FullDOSC Router กับระบบอื่น

เปรียบเทียบกับ Crossbar ใน FPGA

จากงานของ David Bafumba-Lokilo ชื่อว่า Generic Crossbar Network on Chip for FPGA MPSoCs[11] ซึ่งเป็นการสร้าง Crossbar รูปแบบทั่วไป และในงานนี้ได้สรุปการใช้ทรัพยากรดังในตารางที่ 3 เมื่อเทียบกับ FullDOSC Router จะสามารถใช้ NxN Crossbar กับ FullDOSC router ที่รองรับ N โหนดอุปกรณ์

NxN	Estimated freq. (MHz)	LUTs	Registers
2x2	243,4	228	146
3x3	187,3	553	292
4x4	139,5	1604	336
5x5	131,2	2278	557
6x6	126	3467	734
7x7	118,2	3676	843
8x8	111,1	4956	986

ตารางที่ 3 การใช้ทรัพยากรของ Generic Crossbar Network ใน 4-Input LUT FPGA

จากงานของ Pongyupinpanich Surapon ชื่อว่า TDM Switch-based Crossbar Architecture for SoC on FPGA[12] ซึ่งเป็นระบบที่สร้าง Crossbar ใน FPGA ในรูปแบบ Time Division Multiplexer (TDM)

Design	Slice Flip-Flop	Slice LUTs	Estimation Freq. (MHz)
6x6 TDM switch-based	912 [3.33 %]	4,398 [16.05 %]	113.85
8x8 generic	734 [2.65 %]	3,467 [12.65 %]	126.00
8x8 BRCS-NR	3,444 [12.57 %]	2,665 [9.72 %]	-
8x8 PLB buses	229 [0.83 %]	1,296 [4.73 %]	228.40

ตารางที่ 4 การใช้ทรัพยากรของ TDM Switch-based Crossbar เปรียบเทียบกับ Crossbar รูปแบบอื่นๆ ใน 4-Input LUT FPGA

จากการใช้ทรัพยากรของ crossbar ใน FPGA จากงานวิจัยอื่นแล้ว การใช้ FullDOSC router นั้นใช้ทรัพยากรที่น้อยกว่ามาก ดังนั้นถ้าต้องการจะสร้างระบบที่สามารถจะปรับเปลี่ยนการเชื่อมต่อได้ และใช้ FullDOSC แพลตฟอร์ม ซึ่งมีการออกแบบให้รองรับ SDIPC ที่มุ่งเน้นไปยังการใช้ทรัพยากรของระบบให้น้อยที่สุด การใช้ FullDOSC router ตามที่งานวิจัยนี้ได้ออกแบบไว้ จึงเหมาะสมกว่าการนำ crossbar ในรูปแบบอื่นๆ มาใช้งาน

10. บทสรุป

10.1. สรุปผลงานวิจัย

งานวิจัยนี้ได้นิยามการสื่อสารระหว่างหน่วยประมวลผลที่กำหนดได้ด้วยซอฟต์แวร์ (Software-Defined Inter-Processor Communication : SDIPC) ซึ่งเป็นการประยุกต์ใช้หลักการของข่ายงานที่กำหนดได้ด้วยซอฟต์แวร์ (Software-Defined Networking) เพื่อใช้ในการสื่อสารระหว่าหน่วยประมวลผล

ต่อเนื่องจากการนิยาม SDIPC จึงได้ออกแบบแพลตฟอร์ม SDIPC สำหรับอุปกรณ์ที่มีทรัพยากรจำกัดเช่นในระบบอุปกรณ์แบบฝังตัว (Embedded System) ซึ่งมีชื่อว่าการสื่อสารแบบหนึ่งอนุกรมสองทาง (Full-Duplex One Serial Communication : FullDOSC) ซึ่งจะใช้เพียงหนึ่งช่องทางที่มีรูปแบบการสื่อสารแบบสองทาง (Full-Duplex) และในงานวิจัยนี้ได้พัฒนา FullDOSC ให้สามารถอธิบายรูปแบบเพื่อรองรับความต้องการได้มากขึ้นชื่อ modified FullDOSC

จากการออกแบบแพลตฟอร์ม FullDOSC จึงได้ออกแบบภาษา FullDOSC Routing Language (FRL) และ modified FullDOSC Routing Language (mFRL) เพื่อใช้ในการอธิบายโครงสร้างการเชื่อมต่อในระบบ FullDOSC และพิสูจน์การใช้งานทั้งในทางทฤษฎีและนำไปใช้งานในอุปกรณ์จริง

สุดท้ายนี้ ได้ทำการออกแบบระบบ FullDOSC ตัวอย่างเพื่อสาธิตการทำงานของแพลตฟอร์ม ซึ่งอยู่ในรูปแบบการจำลองการใช้งาน และในอุปกรณ์จริงซึ่งใช้ Xilinx Zynq SoC เป็นอุปกรณ์จัดการการเชื่อมต่อและส่วนควบคุม และ STM32 Microcontroller เป็นโหนดอุปกรณ์ที่จะเชื่อมต่อในระบบ

10.2. แนวทางการวิจัยในอนาคต

จากงานวิจัยนี้ได้เน้นไปยังการออกแบบแพลตฟอร์มการเชื่อมต่อแบบหนึ่งอนุกรมสองทาง ซึ่งได้แก่ส่วนภาษาที่ใช้ในการอธิบายและอุปกรณ์จัดการการเชื่อมต่อ แต่ระบบนั้นจะทำงานได้อย่างสมบูรณ์จำเป็นจะต้องมีการออกแบบส่วนจัดการการส่งข้อมูลภายในโหนดอุปกรณ์แต่ละตัวด้วย ซึ่งอาจสร้างเป็นมิดเดิลแวร์ (Middleware) หรือ ชั้นเชื่อมต่อฮาร์ดแวร์ (Hardware Abstraction Layer : HAL) หรืออาจเพิ่มเข้าไปเป็นส่วนหนึ่งระบบปฏิบัติการแบบทันการ (Real-Time Operating System : RTOS) ซึ่งนิยมใช้ในระบบฝังตัว (Embedded System)

ในกรณีที่มีระบบการจัดการการสื่อสารในรูปแบบหนึ่งอนุกรมสองทางที่ดีพอแล้วจะทำให้สามารถวัดผลการสื่อสารในแต่ระรูปแบบเช่น เวลาแฝง (latency), ความเร็ว (speed) , อัตราความผิดพลาด (error rate) หรือคุณสมบัติอื่นๆ ในการส่งข้อมูลในแต่ละรูปแบบและสภาพแวดล้อมต่างๆ

จากการที่มีความสามารถในการเปลี่ยนแปลงรูปแบบการเชื่อมต่อได้ในเวลาทำงาน จึงทำให้สามารถออกแบบและพัฒนาขั้นตอนวิธี (algorithm) ในการเปลี่ยนแปลงรูปแบบให้เหมาะสมกับสภาพแวดล้อมต่างๆ

รายการอ้างอิง

[1] T. S. Ramteke, *LANs: Basic Concept in NETWORKS*, 2 ed., New Jersey: Prentice-Hall, Inc., 2001.

[2] J. W. Valvano, *Embedded Microcomputer Systems: Real Time Interfacing*, 3rd ed.: CL-Engineering, 2012.

[3] Z. Liu, N. Li, and P. Xiyuan, "A Design of Multiprotocol Asynchronous Serial Communication Based on M Module." pp. 377-380.

[4] L. Keqiu, M. Yuanping, L. Keqin, and M. Geyong, "Exchanged Crossed Cube: A Novel Interconnection Network for Parallel Computation," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 24, no. 11, pp. 2211-2219, 2013.

[5] L. Ching-Kai, and C. Kwang-Cheng, "A green software-defined communication processor for dynamic spectrum access." pp. 774-779.

[6] Z. Jingyao, S. Iyer, Z. Xiangwei, P. Schaumont, and Y. Yaling, "Hardware-software co-design for heterogeneous multiprocessor sensor nodes." pp. 20-25.

[7] Y. Ando, Y. Ishida, S. Honda, H. Takada, and M. Edahiro, "Automatic synthesis of inter-heterogeneous-processor communication implementation for programmable system-on-chip." pp. 1-6.

[8] A. Pinto, L. P. Carloni, and A. Sangiovanni-Vincentelli, "COSI: A Framework for the Design of Interconnection Networks," *Design & Test of Computers, IEEE,* vol. 25, no. 5, pp. 402-415, 2008.

[9] Intel, *An Introduction to the Intel ®QuickPath Interconnect,* 320412-001US, 2009.

[10] K. H. Rosen, *Discrete Mathematics and Its Applications*: McGraw-Hill Higher Education, 2007.

[11] D. Bafumba-Lokilo, Y. Savaria, and J. P. David, "Generic crossbar network on chip for FPGA MPSoCs." pp. 269-272.

[12] P. Surapong, and M. Glesner, "TDM switch-based crossbar architecture for SoC on FPGA." pp. 1-4.

ภาคผนวก

ประวัติผู้เขียนวิทยานิพนธ์

นายภาสกร ทองสันตติ์ เกิดเมื่อวันที่ 4 เมษายน พ.ศ. 2533 สำเร็จการศึกษาระดับปริญญา วิศวกรรมศาสตร์บัณฑิต สาขาวิชา วิศวกรรมคอมพิวเตอร์ จากคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2556