การออกแบบส่วนจำเพาะสำหรับการทดสอบแกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรมบาวน์ดารีสแกน

นาย วิชชา เฟื่องจันทร์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2544

ISBN 974-03-1400-7

DESIGN OF TEST MODULE FOR MEL 805X MICROCONTROLLER USING BOUNDARY SCAN ARCHITECTURE

Mr. Witcha Feungchan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2001

ISBN 974-03-1400-7

หัวข้อวิทยานิพนธ์	การออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน
โดย	นาย วิชชา เฟื่องจันทร์
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษา	อาจารย์ ดร.อาทิตย์ ทองทักษ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล)

..... อาจารย์ที่ปรึกษา
(อาจารย์ ดร.อาทิตย์ ทองทักษ์)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สาธิต วงศ์ประทีป)

..... กรรมการ
(อาจารย์ ชัยศิริ ปัณฑิตานนท์)

วิชชา เฟื่องจันทร์ : การออกแบบส่วนจำเพาะสำหรับการทดสอบแกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรมบาวน์ดารีสแกน. (DESIGN OF TEST MODULE FOR MEL 805X MICROCONTROLLER USING BOUNDARY SCAN ARCHITECTURE) อ. ที่ปรึกษา : ดร. อาทิตย์ ทองทักษ์, 123 หน้า. ISBN 974-03-1400-7.

วิทยานิพนธ์นี้เสนอการออกแบบส่วนจำเพาะสำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน โดยใช้ภาษาวีเอชดีแอล (VHDL) ซึ่งเป็นภาษาอธิบายฮาร์ดแวร์ (HDL : Hardware Description Languages) โดยออกแบบวงจรในระดับพฤติกรรม (behavioral level) และในระดับโครงสร้าง (structural level) เพื่อทดสอบการเชื่อมต่อ และทดสอบฟังก์ชันการทำงาน ของแกนไมโครคอนโทรลเลอร์ โดยออกแบบส่วนจำเพาะ สำหรับการทดสอบและสร้างข้อมูลทดสอบ (test data) เพื่อใช้ทดสอบหาความผิดพร่อง (fault) ที่จะเกิดขึ้น โดยกระบวนการทดสอบที่ออกแบบมาเพื่อใช้ทดสอบ จะขึ้นอยู่กับความเหมาะสมและเป้าหมายในการทดสอบ โดยทดสอบด้วยวิธีการจำลองการทำงาน (simulation) และวิเคราะห์ผลด้วยโปรแกรมที่พัฒนาขึ้นมาโดยเฉพาะ

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา .....
ปีการศึกษา 2544

4370492021 : MAJOR COMPUTER SCIENCE

KEY WORD: DESIGN OF TEST MODULE / BOUNDARY SCAN ARCHITECTURE / MICROCONTROLLER / HARDWARE DESCRIPTION LANGUAGES / SIMULATION

WITCHA FEUNGCHAN : DESIGN OF TEST MODULE FOR MEL 805X MICROCONTROLLER USING BOUNDARY SCAN ARCHITECTURE. THESIS ADVISOR : ARTHIT THONGTAK, Ph.D., 123 pp. ISBN 974-03-1400-7.

This Thesis presents the Design of Test Module for MEL 805X Microcontroller using boundary scan architecture. This test module uses VHDL as hardware description language. The Circuit was designed in both behavioral level and structural level to test the interconnection and function of Microcontroller core. The test module and test data were designed to detect possible fault by applying test data. The test method will up to the appropriateness and the target of the test. It uses simulation method to conduct the test and the result will be analyzed by specific purpose developed program.

Department Computer Engineering Student's signature ….....

Field of study Computer Science Advisor's signature .....

Academic year 2001

กิตติกรรมประกาศ

วิทยานิพนธ์ยานิพนธ์ฉบับนี้จะสำเร็จลุล่วงไม่ได้หากไม่ได้รับความกรุณา ช่วยเหลือและให้คำแนะนำในด้านต่าง ๆ เป็นอย่างดียิ่งจาก อาจารย์ ดร.อาทิตย์ ทองทักษ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์

ขอขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ทุกท่าน ที่กรุณาสละเวลาในการให้คำแนะนำต่างๆ ซึ่งทำให้วิทยานิพนธ์ฉบับนี้มีคุณค่า และมีความสมบูรณ์มากยิ่ง

ขอขอบคุณ คุณชำนาญ ปัญญาใส และคุณเจนวิทย์ ศรีหารักษ์ ห้องปฏิบัติการไมโครอิเล็กทรอนิกส์ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ ที่กรุณาให้ข้อมูลต่างๆเป็นอย่างดียิ่ง และเอื้อเฟื้ออุปกรณ์และซอฟแวร์เพื่อใช้ทำการวิจัย

ขอขอบคุณ คุณปัญจภา เรืองสินทรัพย์ และคุณกวี วัฒนะวิรุณ ที่ให้คำแนะนำการใช้โปรแกรมโมเดลซิม

ขอขอบคุณ ห้องปฏิบัติการ Digital System Engineering Laboratory ที่เอื้อเฟื้อสถานที่ในการทำวิจัย

สุดท้ายนี้ผู้วิจัย ขอขอบพระคุณบิดา มารดา คุณชลิดา สุดทัศนทรวง คุณปริญญา เฟื่องจันทร์ และคุณทรงสิริ ศรีสว่างรัตน์ ที่คอยให้คำปรึกษา และให้การสนับสนุนในด้านต่างๆ อีกทั้งยังคอยเป็นกำลังใจอยู่เคียงข้างตลอดเวลาจนผู้วิจัยสำเร็จการศึกษา...ขอขอบพระคุณ

วิชชา เฟื่องจันทร์

9 เมษายน 2545

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญภาพ

สารบัญภาพ (ต่อ)

สารบัญตาราง

บทที่ 1

บทนำ

การออกแบบวงจรดิจิตอล (logic design) ในปัจจุบันถือได้ว่ามีความก้าวหน้าทางเทคโนโลยีสูงเป็นอย่างมาก ทั้งเครื่องมือและซอฟต์แวร์ (software) ที่ช่วยในการออกแบบ ได้ถูกพัฒนาจนสามารถใช้งานได้สะดวกและมีประสิทธิภาพมากขึ้น ทำให้การออกแบวงจรดิจิตอลขนาดใหญ่สามารถทำได้บนเครื่องคอมพิวเตอร์ส่วนบุคคล เช่นการออกแบบไมโครคอนโทรลเลอร์ หรือไมโครโปรเซสเซอร์

ปัจจุบันไมโครคอนโทรลเลอร์ได้ถูกนำมาใช้งานอย่างมาก เพื่อใช้เพิ่มประสิทธิภาพในการทำงานของอุปกรณ์อิเล็กทรอนิกส์ ไมโครคอนโทรลเลอร์ความเร็วสูง MEL 805X ออกแบบโดยคณะวิจัยห้องปฏิบัติการไมโครอิเล็กทรอนิกส์ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ[1] เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ใช้คำสั่งเดียวกับไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทั้งหมดในการทำงาน โดยมีสถาปัตยกรรมภายในที่สามารถทำงานได้เร็วกว่าไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทั่วไปประมาณ 3 เท่า โดยจะใช้สัญญาณนาฬิกา 4 คาบสัญญาณนาฬิกาต่อ 1 รอบของเครื่อง (machine cycle) และทำงานที่สัญญาณนาฬิกาไม่เกิน 12 Mhz. และพัฒนาจนสามารถนำไปผลิตเป็นไอซี (integrated circuit) ต้นแบบและนำมาทดลองใช้งาน

ปัญหาสำคัญของการออกแบบแกนไมโครคอนโทรลเลอร์คือ การทดสอบ ดังนั้นงานวิจัยทางด้านการทดสอบจึงมีความสำคัญมากในงานทางด้านการออกแบบ โดยใช้วิธีจำลองการทำงานและใช้สถาปัตยกรรม บาวน์ดารี สแกน (boundary scan architecture)[2] ซึ่งเป็นวิธีที่ได้รับความนิยม และเป็นที่ยอมรับ เพื่อใช้ในการทดสอบความถูกต้องของระบบที่ออกแบบ โดยการออกแบบส่วนจำเพาะ สำหรับการทดสอบ (design of test module) และสร้างข้อมูลทดสอบ เพื่อใช้ทดสอบหาข้อผิดพลาดที่จะเกิดขึ้น โดยกระบวนการทดสอบที่ออกแบบมาเพื่อใช้ทดสอบ จะขึ้นอยู่กับความเหมาะสมและเป้าหมายในการทดสอบ

ในงานวิจัยนี้จะออกแบบส่วนจำเพาะ สำหรับทดสอบ เพิ่มไว้ในระดับ behavior ซึ่งต่างจากปกติที่ออกแบบในระดับ structure แล้วจึงเพิ่มบาวน์ดารี สแกนเข้าไปหลังจากสังเคราะห์วงจรแล้ว

1.1 ความเป็นมาและความสำคัญของปัญหา

วงจรอิเล็กทรอนิกส์ส่วนใหญ่ จะถูกสร้างให้อยู่ในรูปของวงจรรวม (integrated circuit) ซึ่งมีความซับซ้อนและหนาแน่นมาก ทำให้การทดสอบวงจรรวมมีข้อจำกัดในการทดสอบมากดังนี้

- การทดสอบการเชื่อมต่อ (interconnection test)

  ไม่สามารถทดสอบการเชื่อมต่อของวงจรทำให้ไม่สามารถตรวจสอบได้ว่าถูกต้องหรือไม่

- การทดสอบฟังก์ชัน (functional test)

  สามารถทดสอบฟังก์ชันการทำงานได้โดย การป้อนอินพุต (input) และตรวจสอบเอาต์พุต (output) ไม่สามารถทดสอบการทำงานวงจรย่อยภายในได้ เนื่องจากข้อจำกัดในการใช้ อินพุตและเอาต์พุต ทำให้สามารถทดสอบได้เพียงฟังก์ชันการทำงานโดยรวมไม่สามารถเข้าไปทดสอบการทำงานภายในวงจรได้

- การทดสอบตัวเอง (self test)

  การทดสอบต้องใช้การทดสอบจากภายนอกไม่สามารถทดสอบตัวเองเองได้

จากปัญหาที่กล่าวมาในข้างต้น จึงได้มีการกำหนดมาตรฐานในการเชื่อมต่อวงจรดิจิตอลและเครื่องมือทดสอบรวมทั้ง ข้อมูลทดสอบ (test data) ลักษณะการทดสอบตามมาตรฐาน IEEE Std 1149.1-1990 (Include IEEE Std 1149.1a-1993) IEEE Standard Test Access Port and Boundary scan Architecture [2] มาตรฐานนี้นิยามช่องทางเข้าถึงเพื่อทดสอบ (Test access port) และ สถาปัตยกรรม บาวน์ดารี สแกน สำหรับวงจรรวม โดยจะแนะนำวิธีแก้ปัญหาสำหรับการทดสอบวงจรรวมที่มีความซับซ้อนมาก และมีความหนาแน่นสูง รวมถึงวิธีการเข้าถึง (access) และควบคุม (control) ในลักษณะการออกแบบสำหรับทดสอบ (design for test) ที่ถูกออกแบบรวมในวงจรรวม เช่น เส้นทางสแกนภายใน (internal scan path) และ ฟังก์ชันทดสอบตัวเอง (self test function)

การออกแบบวงจรรวมดิจิตอลสามารถใช้ภาษาอธิบายฮาร์ดแวร์[3,4] เช่นภาษาวีเอชดีแอลและภาษาเวอริลอก (Verilog) มาทำการเขียนอธิบายการทำงานของวงจรที่ต้องการ โดยทั้งสองภาษานี้มีข้อดี ตรงที่ไม่ขึ้นกับแพลตฟอร์ม (platform) พร้อมทั้งสามารถที่จะทำการออกแบบและทดสอบการทำงานของวงจรที่ออกแบบ ด้วยการจำลองการทำงาน โดยการทำงานอยู่บนเครื่องคอมพิวเตอร์ส่วนบุคคล และยังมีซอฟต์แวร์ที่สนับสนุนการออกแบบที่มีประสิทธิภาพ

จากที่กล่าวมาข้างต้น วิทยานิพนธ์นี้มุ่งประเด็นไปที่การออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน เพื่อทดสอบการเชื่อมต่อ และทดสอบฟังก์ชันการทำงาน ด้วยการใช้ภาษาวีเอชดีแอล ออกแบบวงจรในระดับพฤติกรรม และระดับโครงสร้าง และทดสอบด้วยวิธีการจำลองการทำงานของวงจร

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อหาวิธีการออกแบบส่วนจำเพาะสำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน

1.2.2 เพื่อเพิ่มความสามารถ และสนับสนุนงานวิจัยทางด้านการออกแบบ และพัฒนาวงจรทดสอบในประเทศไทย

1.3 ขอบเขตของการวิจัย

1.3.1 ออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน เพื่อให้สามารถ

1 ทดสอบการเชื่อมต่อ

2 ทดสอบฟังก์ชันของ หน่วยคำนวณและตรรกะ (arithmetic-logic unit)

3 ทดสอบฟังก์ชันของ ตัวถอดรหัส (decoder) ใน คอนโทรลพาท (control path)

1.3.2 ออกแบบด้วยการใช้ภาษาวีเอชดีแอล ออกแบบวงจรในระดับพฤติกรรม และระดับโครงสร้าง และสามารถทดสอบด้วยวิธีจำลองการทำงาน

1.4 วิธีดำเนินการวิจัย

1.4.1 ศึกษามาตรฐาน IEEE Std 1149.1-1990 และการออกแบบส่วนจำเพาะ สำหรับการทดสอบ และทดลองออกแบบส่วนจำเพาะ สำหรับการทดสอบด้วยภาษาวีเอชดีแอล ให้สามารถจำลองการทำงานได้อย่างถูกต้อง เพื่อทดสอบการทำงานให้ตรงตามมาตรฐาน IEEE Std 1149.1-1990 รวมทั้งหาข้อจำกัดและแนวทางแก้ไข

1.4.2 ออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X

1.4.3 ออกแบบข้อมูลทดสอบเพื่อใช้ทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X

1.4.4 ทดสอบส่วนจำเพาะ สำหรับการทดสอบ ด้วยวิธีจำลองการทำงาน ดังนี้

1 ทดสอบการเชื่อมต่อ

2 ทดสอบฟังก์ชันของ หน่วยคำนวณและตรรกะ

3 ทดสอบฟังก์ชันของ ตัวถอดรหัสใน คอนโทรลพาท

1.4.5 โดยการทดสอบจะทดสอบทั้ง แกนไมโครคอนโทรลเลอร์ MEL 805X ที่ถูกต้องและแกนที่ดัดแปลงให้ผิดพลาด

1.5 ประโยชน์ที่ได้รับ

1.5.1 เป็นแนวทางเริ่มต้นในการวิจัยและพัฒนา เรื่องของการออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน

1.5.2 สามารถนำหลักการ และขั้นตอนการออกแบบส่วนจำเพาะ สำหรับการทดสอบแกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน ไปใช้ในการออกแบบส่วนจำเพาะ สำหรับการทดสอบ สำหรับวงจรอื่นๆได้

1.6 ลำดับขั้นตอนในการเสนอผลการวิจัย

ในบทที่ 2 จะเป็นการนำเสนอแนวคิดและทฤษฎี ที่เกี่ยวกับหลักการออกแบบวงจรดิจิตอล หลักการทดสอบ และการจำลองการทำงานรวมทั้งนำเสนอ โครงสร้างสถาปัตยกรรม บาวน์ดารี สแกน ซึ่งเป็นส่วนสำคัญสำหรับการออกแบบส่วนจำเพาะ และยังอธิบายถึงไมโครคอนโทรลเลอร์

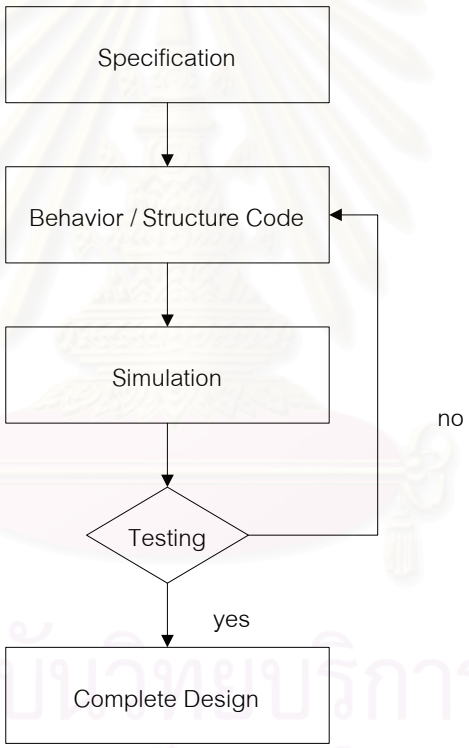
MEL 805X ที่ได้นำมาใช้ในงานวิจัยครั้งนี้ บทที่ 3 กล่าวถึงขั้นตอนการออกแบบ และการทำงานของส่วนจำเพาะ บทที่ 4 จะอธิบาย แบบจำลองความผิดพร่อง (fault model)[5,6] และหลักการออกแบบข้อมูลทดสอบ กระบวนการทดสอบรวมทั้ง การตรวจสอบหาข้อผิดพลาด ส่วนในบทที่ 5 การทดสอบส่วนจำเพาะ สำหรับการทดสอบ บทที่ 6 เป็นบทแสดง ผลและการวิเคราะห์ผลการทดสอบส่วนจำเพาะ สำหรับการทดสอบ และบทที่ 7 เป็นบทสรุปและข้อเสนอแนะ

บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดและทฤษฎี

2.1.1 ขั้นตอนการออกแบบ



รูปที่ 2.1 ขั้นตอนการออกแบบ

2.1.1.1 ขั้นตอนการสร้างข้อกำหนด

การสร้างข้อกำหนดจะสร้างตามความต้องการและวิเคราะห์ระบบ เพื่อหาแนวความคิดหลักการ และวิธีการแก้ไขปัญหา

2.1.1.2 ขั้นตอนการเขียนโปรแกรมต้นฉบับด้วยภาษาวีเอชดีแอล[3,4]

จากการออกแบบตามข้อกำหนดของความต้องการสามารถเขียนอธิบายการทำงานของระบบ โดยมองเป็นลักษณะแต่ละชิ้นส่วน (component) โดยสามารถเขียนแทนการทำงานได้โดยที่เขียนแทนด้วยเอนทิตี (entity) กระบวนการ (process) หรือการใช้โปรแกรมสำเร็จ (package) ซึ่งเป็นวากยสัมพันธ์ (syntax) ของภาษาวีเอชดีแอล ซึ่งจะมีรูปแบบและลำดับการเรียกใช้ดังนี้

1. โปรแกรมสำเร็จ

ใช้สำหรับเก็บข้อมูลในลักษณะโปรแกรมสำเร็จรูปเพื่อเรียกมาใช้

2. คลัง (library)

เป็นคลังเก็บวงจรที่ผ่านการแปลโปรแกรม (complie) จากการออกแบบที่ผ่านมา

3. เอนทิตี

ส่วนของการแสดงชื่อในแต่ละชิ้นส่วน และขาสัญญาณที่ใช้ติดต่อกับภายนอกรวมทั้งอธิบายลักษณะของขาสัญญาณ

4. สถาปัตยกรรม (architecture)

โครงสร้างของวงจร

5. กระบวนการ (process)

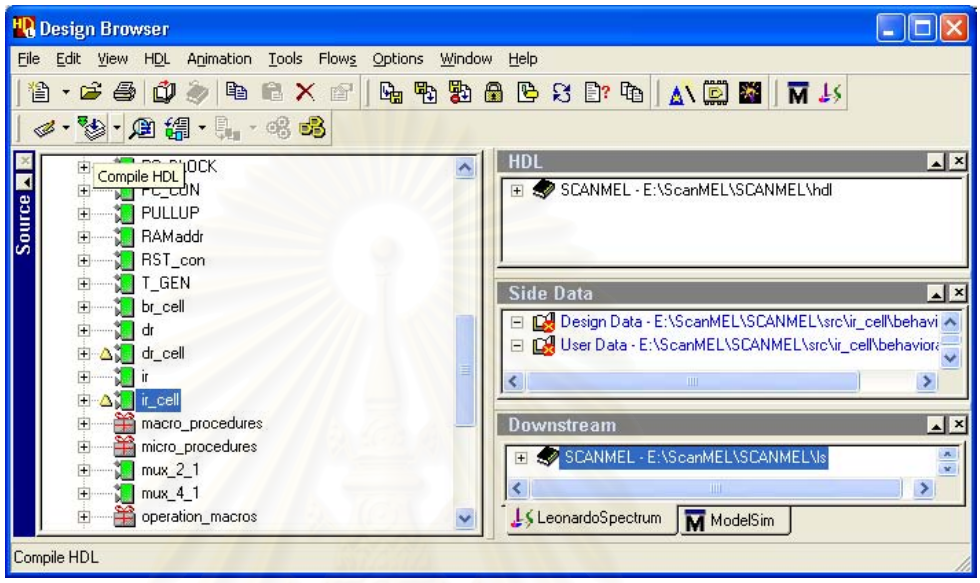
ใช้อธิบายกระบวนการทำงานในแต่ละชิ้นส่วนย่อย ซึ่งสามารถเชื่อมโยงการทำงานกัน

ในงานวิจัยนี้ใช้โปรแกรมโคดไรท์ (Code Wright) ในการเขียนโปรแกรมภาษาวีเอชดีแอล เนื่องจากโปรแกรม สามารถที่จะตรวจสอบรูปแบบของภาษาวีเอชดีแอลได้และโปรแกรมเอชดีแอลดีไซเนอร์ (HDL designer) ที่รวมอยู่ในโปรแกรมเอฟพีจีเอแอดแวนเทจ (FPGA advantage) เพื่อใช้แปลงโปรแกรมต้นฉบับภาษาวีเอชดีแอล ให้เป็นรูปแบบกราฟิก (graphic) และสามารถดัดแปลงแก้ไขหรือสร้างโปรแกรมต้นฉบับ (generate source program) ได้

2.1.1.3 ขั้นตอนการจำลองการทำงาน

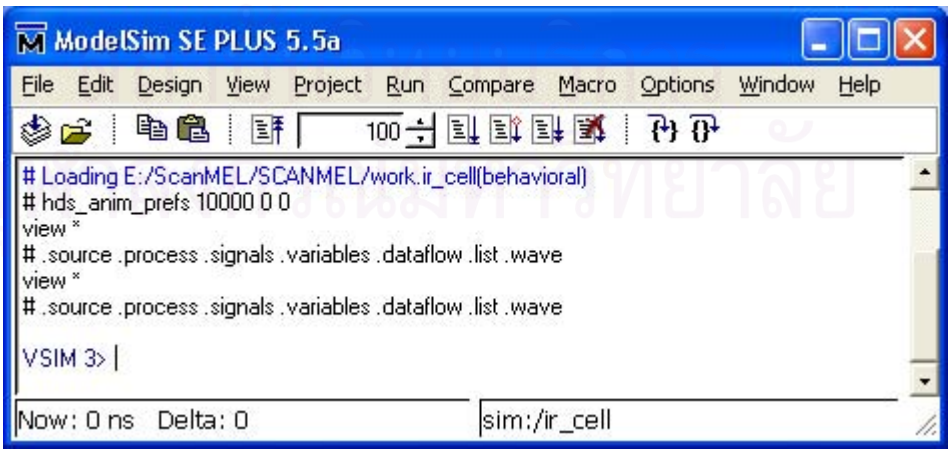
เมื่อทำการแปลโปรแกรมด้วยโปรแกรมโมเดลซิม (ModelSim) ที่รวมอยู่ในโปรแกรม เอฟพีจีเอแอดแวนเทจ แล้วจะเริ่มตรวจสอบการทำงานของระบบ โดยใช้การจำลองการทำงานบนโปรแกรมโมเดลซิมโดยมีขั้นตอนการทำงานดังนี้

1. เมื่อได้โปรแกรมต้นฉบับที่ต้องการ จากการเขียนเรียบร้อยแล้วจะนำมาทำการแปลโปรแกรมโดยจะเลือกเอนทีตี ที่ต้องการแปลโปรแกรมก่อน แล้วจึงเลือกใช้เมนูคอมไพล์เอชดีแอล (compile HDL) จากหน้าจอของโปรแกรมเพื่อทำการแปลโปรแกรม ดังรูปที่ 2.2



รูปที่ 2.2 การใช้งานโปรแกรม เอฟพีจีเอแอดแวนเทจ ในการแปลโปรแกรม

2. เมื่อโปรแกรมต้นฉบับวีเอชดีแอล ไม่มีปัญหาหรือไม่มีข้อผิดพลาดในเรื่องของรูปแบบก็สามารถจำลองการทำงานได้ ซึ่งสามารถเลือกใช้เมนูซิมมูเลชันโฟลว (simulation flow) แล้วจะเกิดหน้าจอการจำลองการทำงานดังรูปที่ 2.3



รูปที่ 2.3 ผลที่ได้จากการใช้คำสั่งซิมมูเลชันโฟลว

จากนั้นป้อนค่าสัญญาณต่างๆที่ต้องการแล้วดูผลลัพธ์ว่าได้ออกมาตรงกับความต้องการและการทำงานของระบบถูกต้องหรือไม่ โดยการป้อนค่าจะเขียนโปรแกรมต้นฉบับ ขึ้นมาเพื่อป้อนค่าต่างๆ หากมีข้อผิดพลาดต้องแก้ไขโปรแกรมต้นฉบับ และต้องทำการแปลโปรแกรมใหม่อีกจนกว่าจะได้ค่าตามความต้องการ

2.1.2 ข้อกำหนดรายละเอียดในการอธิบายการทำงาน (specification and description)

การกำหนดรายละเอียดและอธิบายการทำงานของฮาร์ดแวร์ ถือได้ว่าเป็นหลักการที่มีความสำคัญสำหรับการออกแบบวงจรเพื่อให้ถูกต้อง และการออกแบบสามารถแบ่งได้หลายระดับ ซึ่งแต่ละระดับมีวิธีการออกแบบที่ต่างกันตามลักษณะการเขียน ด้วยภาษาวีเอชดีแอล แบ่งได้เป็น

1. ระดับพฤติกรรม

ระดับพฤติกรรม เป็นรูปแบบที่ใช้บรรยายพฤติกรรมของวงจรดิจิตอล ในส่วนของการบรรยายจะมีโครงสร้างคล้ายกับภาษาระดับสูง (high level language) ทั่วไปเช่นภาษา ซี(C) หรือภาษา ปาสกาล(Pascal) ซึ่งในการจำลองการทำงาน คำสั่งแต่ละคำสั่ง (statement) จะถูกประเมินผลไปตามลำดับ (sequential) จากบนลงล่าง และสามารถเรียกใช้โปรแกรมย่อย มีการใช้การวน (loop) สามารถบรรยายพฤติกรรมของวงจรและรายละเอียดความสัมพันธ์ระหว่าง อินพุต และเอาท์พุต ได้อย่างดี

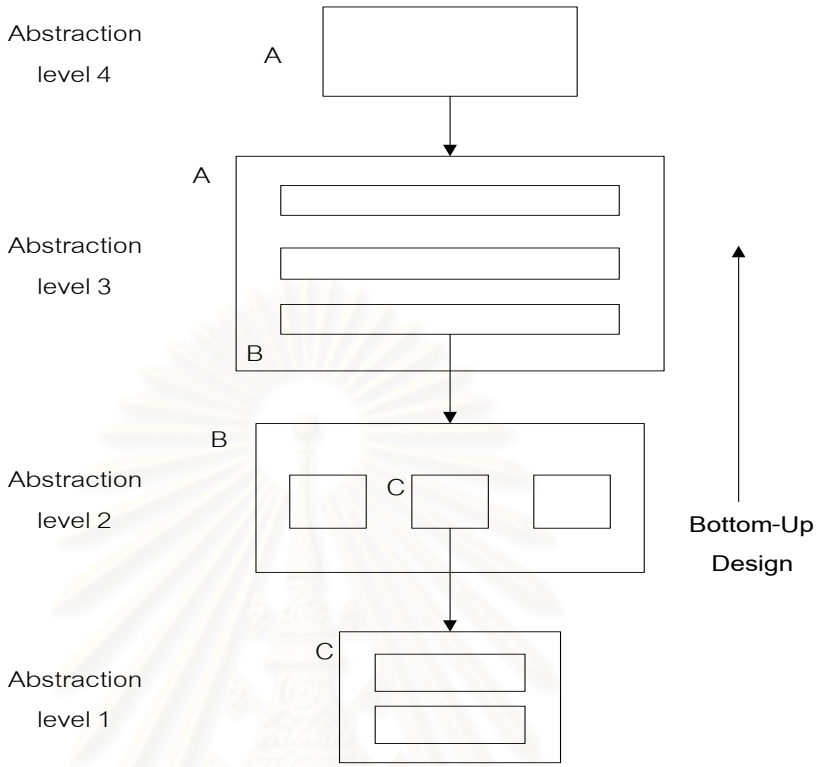
2. ระดับโครงสร้าง

ในระดับนี้จะแสดงการเชื่อมต่อระหว่างชิ้นส่วนต่างๆ ที่นำมาประกอบขึ้นเป็นวงจรหรือระบบดิจิตอล ซึ่งเป็นการเขียนอธิบายให้เห็นถึงโครงสร้างของวงจร ว่ามีการเชื่อมต่อกันอย่างไร

ซึ่งในการออกแบบวงจรส่วนมากจะใช้ทั้งสองระดับผสมกัน และเนื่องจากคุณสมบัติของภาษาวีเอชดีแอล ทำให้สามารถเขียนอธิบาย ได้ทั้งสองรูปแบบเพื่อบรรยายวงจรหรือระบบดิจิตอลเดียวกันได้

2.1.3 การสร้างรายละเอียดในการอธิบายการทำงาน

ในรูปที่ 2.4 เป็นการแสดงถึงลำดับการออกแบบอย่างคร่าวๆ ซึ่งเป็นการออกแบบจากล่างขึ้นบน โดยจะนำมาใช้สำหรับการออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ในการทำงานวิจัยครั้งนี้ ซึ่งสามารถแบ่งออกเป็น 3 ขั้นตอนดังนี้



รูปที่ 2.4 การออกแบบด้วยวิธีการจากล่างขึ้นบน[3]

2.1.3.1 ข้อกำหนดคุณลักษณะและการอธิบายการทำงาน

ในส่วนการอธิบายการทำงานของส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน และยึดตามมาตรฐาน IEEE Std 1149.1-1990 ซึ่งได้นำตรรกะทดสอบ (test logic) ของนายปีเตอร์ เอ็ม แคมป์เบลล์ (Peter M. Campbell) มาปรับเปลี่ยนให้เหมาะสม โดยการอธิบายวงจรสามารถใช้ทั้ง ระดับพฤติกรรมและระดับโครงสร้าง ผสมกันเนื่องจากการอธิบายวงจรในแต่ละแบบมีข้อดีข้อเสียแตกต่างกัน ดังนี้ การอธิบายระดับพฤติกรรมของวงจร จะทำให้สามารถเข้าใจการทำงานของวงจรนั้นได้ง่ายกว่าการอธิบายระดับโครงสร้างของวงจร แต่ในการอธิบายพฤติกรรมของวงจรในบางรูปแบบไม่สามารถนำไปสังเคราะห์ (synthesis) เป็นวงจรจริงได้ เช่น คำสั่งรอหลังจาก (wait until) เป็นต้น

2.1.3.2 การรวมส่วนการออกแบบ (integrated design)

ในการรวมชิ้นส่วนต่างสำหรับการออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X จะประกอบตรรกะทดสอบต่างๆเข้าด้วยกัน จาก

ระดับ แสกนเซลล์ (scan cell) ประกอบขึ้นมาจากล่างขึ้นบนจนเป็นส่วนจำเพาะ สำหรับการทดสอบรวมเข้ากับ แกนไมโครคอนโทรลเลอร์ MEL 805X

2.1.3.3 การทดสอบความถูกต้อง

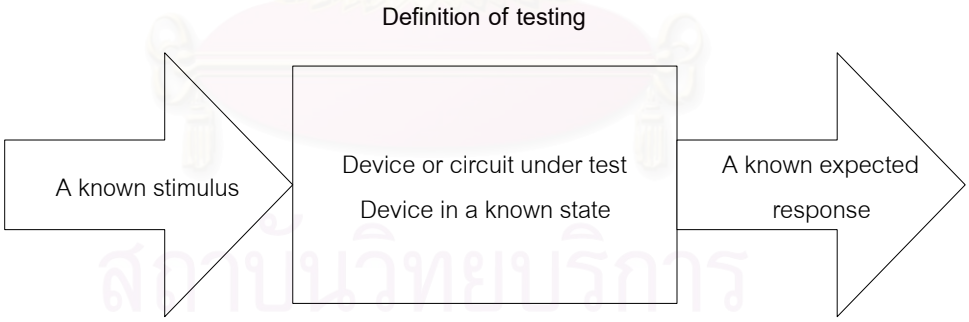
ในการทดสอบความถูกต้องเมื่อได้ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X แล้วนำมาจำลองการทำงานเพื่อทดสอบความถูกต้องของวงจร

2.1.4 การอธิบายโครงสร้างและพฤติกรรมการทำงานของวงจร

การออกแบบวงจรในปัจจุบัน สามารถใช้ภาษาอธิบายฮาร์ดแวร์ เพื่ออธิบายการทำงานของวงจร ซึ่งงานวิจัยนี้ใช้ภาษาวีเอชดีแอล ซึ่งมีโครงสร้างของภาษาอยู่บนพื้นฐานภาษาปาสกาล โดยจะใช้เพื่อ อธิบายโครงสร้างและพฤติกรรมของส่วนจำเพาะ สำหรับการทดสอบ

2.1.5 หลักการทดสอบวงจร[5]

หลักการทดสอบโดยทั่วไปการทดสอบวงจรส่วนมาก จะให้นิยามการทดสอบวงจรที่คล้ายๆกันโดยที่การทดสอบคือรู้การทำงานต่างๆของวงจรที่ต้องการทดสอบ ป้อนอินพุตที่เราต้องการ เพื่อทดสอบ และตรวจสอบกับเอาท์พุตที่ได้ออกมาเพื่อตรวจสอบความถูกต้องดังรูปที่ 2.5



รูปที่ 2.5 นิยามการทดสอบ[5]

2.1.6โครงสร้างสถาปัตยกรรม บาวน์ดารี สแกน[2]

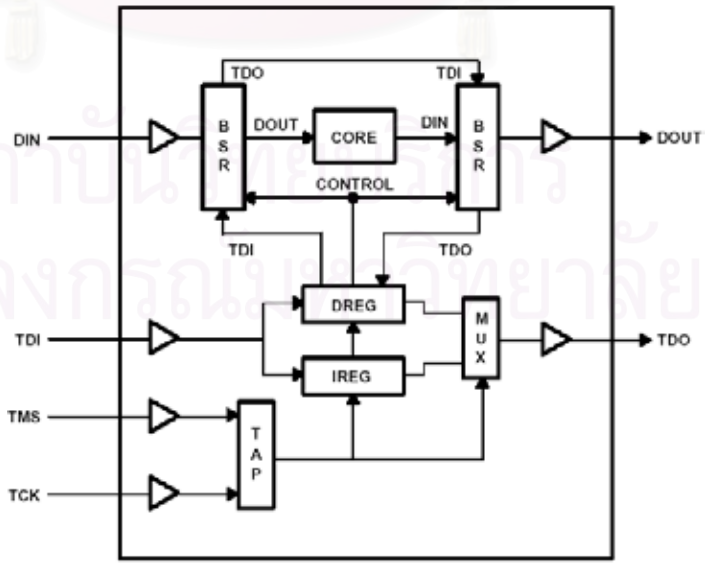
โครงสร้างสถาปัตยกรรมจะอิง IEEE Std 1149.1 เป็นมาตรฐานนิยาม ช่องทางเข้าถึงเพื่อทดสอบและ สถาปัตยกรรม บาวน์ดารี สแกน สำหรับวงจรรวมโดยแนะนำวิธีแก้ปัญหาสำหรับการทดสอบวงจรรวมที่มีความซับซ้อนมากและมีความหนาแน่นสูง โดยหมายถึงการเข้าถึงและควบ

คุม ในลักษณะการออกแบบเพื่อทดสอบ ที่สามารถออกแบบวงจรทดสอบรวมอยู่ในวงจรรวม โดยมาตรฐานนี้จะนิยาม ตรรกะทดสอบ ที่ออกแบบรวมไว้ในวงจรรวมเพื่อ

- ทดสอบการเชื่อมต่อภายในวงจรรวม
- ทดสอบวงจรรวมด้วยตัวเอง
- สังเกตและแก้ไขการทำงานของวงจรในสภาวการณ์ทำงานปกติ

โดยตรรกะทดสอบ ประกอบด้วย บาวน์ดารี สแกนและบล็อก (block) อื่นๆดังรูปที่ 2.6 เข้าถึงผ่านช่องทางเข้าถึงเพื่อทดสอบ และวงจรที่นิยามในมาตรฐานนี้ อนุญาตให้ใช้คำสั่งทดสอบ (test instruction) และข้อมูลทดสอบที่เกี่ยวข้อง ป้อนเข้าสู่ชิ้นส่วน หรือผลลัพธ์ที่ได้ออกมา ผ่านทางรูปแบบการสื่อสารแบบอนุกรมเท่านั้น และลำดับของชุดคำสั่งทดสอบที่ใช้ทดสอบ จะถูกควบคุมโดยบัสมาสเตอร์ (Bus master) ซึ่งอาจจะเป็นเครื่องทดสอบอัตโนมัติ (automatic test equipment) หรือชิ้นส่วนที่ใช้ควบคุม โดยที่ตัวควบคุมจะทำการควบคุมสัญญาณเลือกวิธีทดสอบ (TMS : test mode select) และสัญญาณนาฬิกาทดสอบ (TCK : test clock) ซึ่งจะต่อกับบัสมาสเตอร์ ซึ่งปกติจะอยู่ในสภาพไม่ทำงาน

ในการทำงาน ขั้นตอนแรกจะอ่านข้อมูลแบบอนุกรมเข้าสู่ส่วนจำเพาะ สำหรับการทดสอบเพื่อดู รหัสคำสั่ง (instruction code) ว่าจะต้องกระทำการดำเนินการ (operation) อะไรเพื่อทำการควบคุมการทำงานของบล็อกต่างๆ ในส่วนจำเพาะ และเมื่อกระทำการดำเนินการ ต่างๆกับข้อมูลทดสอบแล้ว จะได้ผลลัพธ์ออกมาจากส่วนจำเพาะ ส่งให้ บัสมาสเตอร์ต่อไป



รูปที่ 2.6 สถาปัตยกรรมตรรกะทดสอบ (test logic architecture)[7]

2.1.6.1 เทคนิคการใช้ IEEE 1149.1[2]

เทคนิคสำคัญในการทดสอบวงจรที่ใช้ใน มาตรฐาน IEEE1149.1 คือการใช้สแกนดีไซน์ (scan design) ซึ่งอนุญาตให้สามารถสังเกตและควบคุมวงจรภายใน ได้ในจุดต่างๆ โดยไม่จำเป็นต้องใช้ ขาอินพุต และเอาท์พุตจำนวนมากซึ่งวิธีนี้ใช้ เรจิสเตอร์แทนโดยเรียกว่า สแกน เรจิสเตอร์ (scan register) ซึ่งจะประกอบด้วย บาวน์ดารี สแกนเซลล์ (boundary scan cell) ที่สามารถเลื่อน (shift) และ บรรจุแบบขนาน (parallel load) ซึ่งสามารถนำไปใส่ไว้ในจุดต่างๆในวงจรเพื่อสังเกตและควบคุม และมี บาวน์ดารี สแกน ซึ่งเป็นสแกนดีไซน์ประเภทหนึ่งซึ่งจะนำบาวน์ดารี สแกนเซลล์ ต่อไว้ที่ขาอินพุต และขาเอาท์พุตของ มอดูล (module) เชื่อมต่อกันสร้างเป็น บาวน์ดารี สแกน เรจิสเตอร์ (BSR : boundary scan register)

การใช้เทคนิคบาวน์ดารี สแกน สัญญาณต่างๆที่ ขาอินพุต และขาเอาท์พุต จะถูกตรวจสอบสอบโดยการเลื่อนค่าต่างๆผ่านทางบาวน์ดารี สแกน เรจิสเตอร์ และจะทดสอบมอดูล ได้โดยเลื่อนข้อมูลทดสอบไปให้แต่ละบาวน์ดารี สแกนเซลล์ และนอกจากนั้นยังสามารถทดสอบการเชื่อมต่อภายในของวงจร โดยการเลื่อนค่าไปไว้ที่ขาเอาท์พุต ของมอดูลและทำการชักตัวอย่าง (sampling) ค่าที่ขาอินพุต ของอีกมอดูล ที่เชื่อมต่อกันเพื่อทดสอบการเชื่อมต่อ และนอกจากนี้มาตรฐาน IEEE 1149.1 ยังอนุญาต ให้สามารถวางบาวน์ดารี สแกนเซลล์ ไว้ในตำแหน่งต่างๆในวงจรนอกจากที่ ขาอินพุต และขาเอาท์พุต เพื่อความสะดวกในการทดสอบและเพื่อให้สามารถแบ่งมอดูล ออกเป็นส่วนย่อยเพื่อให้ทดสอบง่าย

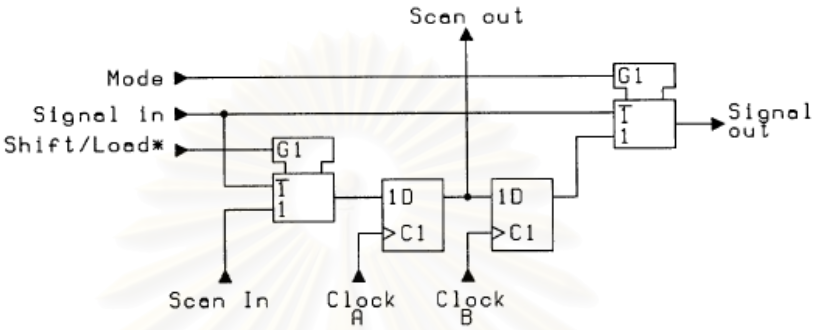
ในการสร้างวงจรทดสอบนี้จะยึดตามมาตรฐาน IEEE1149.1 โดยจะใช้ภาษาวีเอชดีแอล ซึ่งเป็นภาษาที่ใช้ในการออกแบบวงจรรวมมาใช้ในการออกแบบวงจรทดสอบทั้งหมด

2.1.6.2 บาวน์ดารี สแกน

เทคนิคบาวน์ดารี สแกน จะเกี่ยวกับการรวมบาวน์ดารี สแกนเซลล์ ที่วางติดกับขาของแต่ละมอดูล โดยที่สัญญาณที่ขาต่างๆตามบริเวณขอบของมอดูล สามารถควบคุมได้โดยใช้หลักการ ทดสอบสแกน

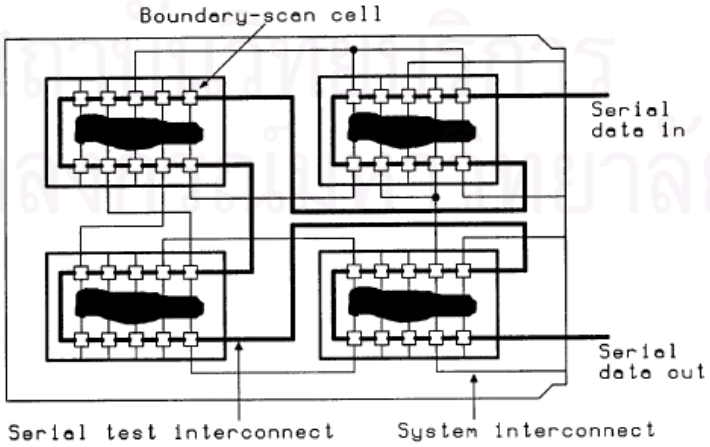
แสดงตัวอย่างการสร้างบาวน์ดารี สแกนเซลล์ ซึ่งสามารถใช้กับ การเชื่อมต่อของอินพุตหรือเอาท์พุตเข้ากับวงจรรวม โดยจะขึ้นอยู่กับสัญญาณควบคุม ที่ใช้กับอุปกรณ์รวมส่งสัญญาณ (multiplexer) ข้อมูลสามารถอ่านเข้าสู่ บาวน์ดารี สแกน เรจิสเตอร์ จาก ช่องทางเข้าอนุกรม หรือขับออกจากเรจิสเตอร์ไปยังช่องทางออกอนุกรม และสำหรับฟลิปฟล็อป (flip-flop) ตัวที่สองควบ

คุมด้วย สัญญาณนาฬิกาบี (clock B) ใช้สำหรับคงค่าที่ขับออกไปให้คงอยู่เมื่อมีข้อมูลใหม่อ่านเข้ามาในเซลล์ โดยใช้ สัญญาณนาฬิกาเอ (clock A) ในการควบคุม ซึ่ง Flip-flop ตัวนี้ไม่จำเป็นสำหรับทุกกรณีในดังรูปที่ 2.7



รูปที่ 2.7 โครงสร้าง บาวน์ดารี สแกน เซลล์[2]

บาวน์ดารี สแกนเซลล์ สำหรับแต่ละขาของ มอดูล นั้นจะมีการเชื่อมต่อกันเพื่อสร้าง ลูกโซ่บาวน์ดารี สแกน (boundary scan register chain) รอบๆขอบของการออกแบบ ดังรูปที่ 2.8 โดยเส้นทางนี้จะมีทั้งอินพุต และเอาท์พุตแบบอนุกรม รวมทั้งสัญญาณควบคุม และสัญญาณนาฬิกาที่เชื่อมต่อกันเป็นเส้นทางเดียวทั้งหมด ซึ่งทำให้สามารถเชื่อมต่อระหว่างแต่ละมอดูล ได้เพียงเพื่อส่งข้อมูลทดสอบไปยังแต่ละมอดูล และสามารถตรวจสอบผลลัพธ์ได้ อีกทั้งยังสามารถตรวจสอบการทำงานแต่ละมอดูล ได้โดยตรงโดยที่ปราศจากการรบกวน



รูปที่ 2.8 การเชื่อมต่อของ บาวน์ดารี สแกน เซลล์ ในวงจร[2]

2.1.6.3 ตรรกะทดสอบ

จะมีส่วนที่จำเป็นทั้งหมด 4 ส่วน ประกอบกันเป็นตรรกะทดสอบดังนี้

1. ช่องทางเข้าถึงเพื่อทดสอบ

เป็นมาตรฐานในการติดต่อสื่อสารระหว่างตรรกะทดสอบกับเครื่องมือทดสอบภายนอก (external test equipment) หรือบัส ซึ่งจะประกอบด้วยอย่างน้อยที่สุด 3 อินพุต และ 1 เอาท์พุตหรืออาจจะมีอินพุตเพื่อใช้ตั้งใหม่ (reset)ในแบบที่ไม่ขึ้นกับสัญญาณนาฬิกา ช่องทางเข้าถึงเพื่อทดสอบ จะประกอบด้วย

อินพุต

1. สัญญาณนาฬิกาทดสอบ เป็นสัญญาณนาฬิกาเพื่อให้ทำงานเข้าจังหวะ

2. สัญญาณเลือกวิธีทดสอบ เป็นสัญญาณที่จะถูกนำไปถอดรหัส (decode) ในตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ (TAP controller) โดยสัญญาณ ถูกควบคุมโดยบัสมาสเตอร์

3. สัญญาณอินพุตข้อมูลทดสอบ (TDI : test data input) คำสั่งทดสอบ และข้อมูลทดสอบจะถูกส่งมาแบบอนุกรมที่อินพุตข้อมูลทดสอบ

4. สัญญาณตั้งใหม่ทดสอบ (TRST : test reset) เป็นสัญญาณเพิ่มเพื่อใช้ตั้งใหม่ในแบบที่ไม่ขึ้นกับสัญญาณนาฬิกา

เอาท์พุต

1. สัญญาณเอาท์พุตข้อมูลทดสอบ (TDO : test data output) เป็นผลลัพธ์ของคำสั่งทดสอบและข้อมูลทดสอบที่ได้จากตรรกะทดสอบ

2. ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ

ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบจะเป็นตัวสร้างสัญญาณนาฬิกาภายในและสัญญาณต่างๆที่ใช้สำหรับวงจรทดสอบ ซึ่งทำงานแบบเครื่องสถานะจำกัดแบบประสานเวลา (synchronous finite state machine) ประกอบด้วย 16 สถานะ โดยที่ ณ ขณะเวลาหนึ่งจะทำงานที่สถานะใดสถานะหนึ่งเท่านั้น และการเปลี่ยนสถานะจะขึ้นอยู่กับค่าสัญญาณเลือกวิธีทดสอบและเกิดขึ้นที่ขอบขาขึ้นของสัญญาณนาฬิกาทดสอบเท่านั้น

3. เรจิสเตอร์คำสั่ง (IR : instruction register)

เรจิสเตอร์คำสั่ง ทำหน้าที่เก็บค่าสำหรับคำสั่งที่ใช้เพื่อทดสอบหรือเข้าถึงเรจิสเตอร์ข้อมูลโดยที่ เรจิสเตอร์คำสั่งจะประกอบด้วยเรจิสเตอร์คำสั่งเซลล์ (instruction register cell) จำนวนสองเซลล์หรือมากกว่าเชื่อมต่อกัน โดยอนุญาตให้ข้อมูลสามารถบรรจุแบบอนุกรมผ่านทาง สัญญาณอินพุตข้อมูลทดสอบได้

ข้อมูลจะถูกแลตช์ (latch) ไว้ในเรจิสเตอร์คำสั่งในสถานะแคปเทอร์ไออาร์ (capture-IR) และข้อมูลจะถูก แลตช์ไว้ในเรจิสเตอร์คำสั่งเอาท์พุต ในสถานะอัปเดตไออาร์ (update-IR)และ เอาท์พุตของเรจิสเตอร์คำสั่งจะมีคำสั่งปัจจุบันอยู่ และมีข้อสังเกตคือ อย่างน้อยเรจิสเตอร์คำสั่ง ต้องมี 2 เรจิสเตอร์คำสั่งเซลล์ เพื่อเก็บ 2 บิตแรกของ เรจิสเตอร์คำสั่งและต้องบรรจุค่า ทวิภาค (binary) "01" เมื่ออยู่ในสถานะอัปเดตไออาร์และในการออกแบบต้องมี เรจิสเตอร์คำสั่ง เพียงตัวเดียวสำหรับทุกๆ ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ

4. เรจิสเตอร์ข้อมูล

กลุ่มของเรจิสเตอร์ข้อมูล จะต้องประกอบด้วย เรจิสเตอร์ผ่าน (BR : bypass register) และบาวน์ดารี สแกน เรจิสเตอร์ จะมีเพียงเรจิสเตอร์ผ่าน เพียงหนึ่งเดียวซึ่ง เป็นเรจิสเตอร์เลื่อน (shift register) ซึ่งใช้เพื่ออนุญาตให้ทำงานผ่านเรจิสเตอร์ข้อมูล โดยต่อตรงกับอินพุตข้อมูลทดสอบ และเอาท์พุตข้อมูลทดสอบ ได้ทำให้เส้นทางระหว่างอินพุตข้อมูลทดสอบ และเอาท์พุตข้อมูลทดสอบสั้นลง

2.1.6.4 ชุดคำสั่ง (instruction set)

ชุดคำสั่งต่างๆ ใช้เพื่อเลือก ฟังก์ชันในการจัดการกับเรจิสเตอร์ ซึ่งจะมีอยู่ 3 คำสั่งหลักๆคือคำสั่งผ่าน (BYPASS) ,คำสั่งแซมเปิล/พรีโลด (SAMPLE/PRELOAD) และคำสั่งเอกซ์เทส (EXTEST) และคำสั่งอื่นๆสามารถนิยามตามการออกแบบ

1. คำสั่งผ่าน

คำสั่งผ่านเป็นคำสั่งเดียวที่ใช้กับ เรจิสเตอร์ผ่านซึ่งมี รหัสทวิภาค (Binary code) เป็น "11...1" คือมีค่า 1 ในทุกๆเรจิสเตอร์คำสั่งเซลล์ และคำสั่งนี้ต้องถูกเก็บไว้ในเอาท์พุตเรจิสเตอร์คำสั่งเซลล์ เมื่อเข้าสู่สถานะ Run-Test-Idle controller

2. คำสั่งแซมเปิล/พรีโลด

คำสั่งแซมเปิล/พรีโลด จะชักตัวอย่างข้อมูล ที่อินพุตแบบขนาน (parallel input) แล้วเลือกบาวน์ดารี สแกน เรจิสเตอร์ และอนุญาตให้เลื่อนข้อมูลไว้ใน บาวน์ดารี สแกน เรจิสเตอร์ที่ถูกเลือกไว้ ข้อมูลที่ขาของระบบ (system pin) จะบรรจุไว้ใน บาวน์ดารี สแกน เรจิสเตอร์ในสถานะแคปเทอร์ไออาร์และข้อมูลในเรจิสเตอร์ จะถูกเลื่อนในสถานะชิพท์ดีอาร์ (shift-DR) และแลตช์ไว้ในที่พักเอาท์พุตแบบขนาน (parallel output buffer) ของเรจิสเตอร์ในสถานะอัปเดตดีอาร์ (update-DR) ค่าทวิภาคของคำสั่งแซมเปิล/พรีโลด อาจกำหนดโดยผู้ออกแบบ

3. คำสั่งเอกซ์เทส

คำสั่งเอกซ์เทสสามารถใช้เพื่อทดสอบการเชื่อมต่อ โดยข้อมูลทดสอบจะเก็บไว้ในเอาท์พุตของ บาวน์ดารี สแกน เซลล์ และข้อมูลที่ขาอินพุตจะแลตช์ไว้ในเรจิสเตอร์โดยปกติจะบรรจุโดยใช้คำสั่งแซมเปิล/พรีโลด

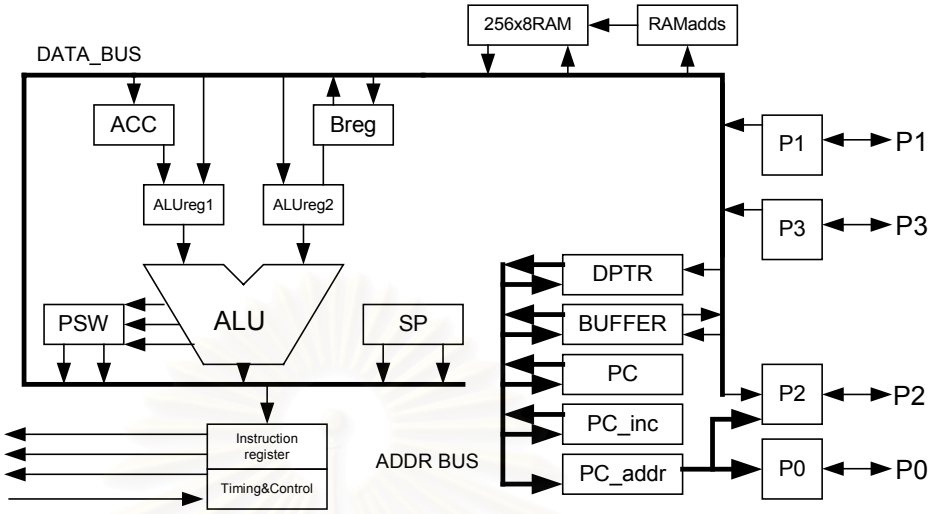
2.1.7 ไมโครคอนโทรลเลอร์ความเร็วสูง MEL805x[1]

ไมโครคอนโทรลเลอร์ MEL 805X เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ใช้ชุดคำสั่งเดียวกับไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทั้งหมดในการทำงาน โดยมีสถาปัตยกรรมภายในที่สามารถทำงานได้เร็วกว่าไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทั่วไปประมาณ 3 เท่า โดยจะใช้สัญญาณนาฬิกา 4 คาบสัญญาณนาฬิกาต่อ 1 รอบของเครื่อง

สามารถอ้างถึงหน่วยความจำข้อมูลภายในได้ 256 ตำแหน่ง ไม่มีหน่วยความจำโปรแกรมภายในต้องต่อใช้จากภายนอกเท่านั้น โดยสามารถต่อแรม (RAM) และรอม (ROM) ภายนอกได้ถึง 64 กิโลไบท์ พอร์ตอินพุตและเอาท์พุต มีขนาด 8 บิตจำนวน 4 พอร์ต สามารถใช้งานเป็นได้ 3 พอร์ต เนื่องจากพอร์ต 0 ต้องใช้ในการอ้างถึงแอดเดรสของหน่วยความจำโปรแกรมภายนอกเท่านั้น

2.1.7.1 โครงสร้างของวงจร

จากรูปที่ 2.9 แบ่งการทำงานได้เป็น 2 ส่วนคือส่วนที่รับส่งข้อมูลผ่านบัสข้อมูล(data bus) ขนาด 8 บิต และส่วนที่รับส่งข้อมูลผ่านบัสแอดเดรส (address bus)ขนาด 16 บิต โดยจะมีวงจรที่พักเพื่อทำการแลกเปลี่ยนข้อมูลระหว่างบัสข้อมูลและบัสแอดเดรส การแบ่งช่วงเวลาการใช้งานบัสข้อมูลจะขึ้นอยู่กับแต่ละคำสั่ง ว่ามีการเคลื่อนย้ายข้อมูลอย่างไร ใช้เวลากี่รอบของเครื่อง ส่วนการใช้งานบัสแอดเดรสจะขึ้นอยู่กับว่าเป็นการดำเนินการปกติ (normal operation) หรือการดำเนินการก้าวกระโดด (branch operation) ถ้ากำลังทำงานที่การดำเนินการปกติ วงจรภายในบัสแอดเดรสจะทำการเพิ่มค่าในตัวนับระบุตำแหน่งคำสั่ง (PC : program counter) แล้วส่งข้อมูลสำหรับชี้ ตำแหน่งคำสั่งในหน่วยความจำโปรแกรม (program memory)ไปที่พอร์ต 0 และพอร์ต 2 ถ้าหากเป็นการดำเนินการก้าวกระโดด ซึ่งจะมีการส่งข้อมูลจากบัสข้อมูลไปยังบัสแอดเดรสเพื่อเปลี่ยนแปลงค่าในตัวนับระบุตำแหน่งคำสั่ง โดยวงจรภายในจะแบ่งเป็น 2 ส่วนคือ ดาตาพาท (data path) และคอนโทรลพาท



รูปที่ 2.9 แผนภาพบล็อกทั้งหมดของ แกนไมโครคอนโทรลเลอร์ MEL805X[1]

2.1.7.2 ดาตาพาท

ดาตาพาทจะประกอบด้วยวงจร ฟลิปฟล็อป หรือ เรจิสเตอร์ แทนการทำงานของส่วนต่างๆ ซึ่งการทำงานจะถูกควบคุมโดยสัญญาณควบคุมซึ่งมาจากวงจรส่วนควบคุม

- หน่วยคำนวณและตรรกะ หน้าที่ประมวลผลทางคณิตศาสตร์ สามารถทำคำสั่งคูณหารขนาด 8 บิตได้ผลลัพธ์ขนาด 16 บิตได้ และคำนวณทางลอจิกได้
- ตัวสะสม (accumulator) หรือ เรจิสเตอร์เอ (A register) ซึ่งเป็นเรจิสเตอร์หลักสามารถส่งข้อมูลให้กับหน่วยคำนวณและตรรกะได้ โดยไม่ต้องผ่านบัสข้อมูลเพื่อลดระยะเวลาในการประมวลผลของวงจร
- เรจิสเตอร์บี (B register) ซึ่งเป็นเรจิสเตอร์สำหรับประมวลผล คำสั่งคูณและหาร และเป็นรีจีสเตอร์ใช้งานทั่วไป
- เรจิสเตอร์ตัวดำเนินการ 1 (operand register 1) และ เรจิสเตอร์ตัวดำเนินการ 2 (operand register 2) เป็นเรจิสเตอร์สำหรับรับข้อมูลจากส่วนต่างๆที่ส่งเข้ามาเพื่อประมวลผลโดยผ่านบัสข้อมูล แล้วส่งให้หน่วยคำนวณและตรรกะ เพื่อประมวลผล
- ตัวชี้กองซ้อน (stack pointer) เป็นเรจิสเตอร์ที่ชี้หน่วยความจำแบบกองซ้อน
- เรจิสเตอร์ สถานภาพ (status register) ทำหน้าที่เก็บสถานะต่างๆในการคำนวณ
- แรมแอดเดรส (ram address) เป็นวงจรทำหน้าที่สำหรับกำหนดค่าแอดเดรสให้กับหน่วยความจำภายใน เมื่อมีการอ้างถึงหน่วยความจำในลักษณะต่างๆ
- ที่พัก ทำหน้าที่ส่งผ่านข้อมูลระหว่างบัสข้อมูลและแอดเดรสบัส

- ดีพีทีอาร์ (DPTR) เป็นรีจีสเตอร์สำหรับเก็บแอดเดรสสำหรับการดำเนินการที่มีการส่งผ่านข้อมูลกับอุปกรณ์ภายนอก
- ตัวนับระบุตำแหน่งคำสั่ง เป็นรีจีสเตอร์ที่ทำงานในการเพิ่มค่าและเปลี่ยนแปลงค่าตำแหน่งแอดเดรส
- พอร์ต 0 ถึงพอร์ต 3 เป็นพอร์ตขนานแบบสองทิศทาง(bidirectional IO port) สามารถใช้งานทีละบิตได้

2.1.7.3 คอนโทรลพาท

ในส่วนคอนโทรลพาทเป็นการสร้างสัญญาณเพื่อไปควบคุมส่วนต่างๆของส่วนดาตาพาทโดยส่วนใหญ่จะเป็นสัญญาณเปิดทาง (enable) เรจิสเตอร์หรือ เลือกอุปกรณ์รวมส่งสัญญาณเป็นต้น โดยสัญญาณเหล่านี้จะมีจังหวะการทำงานขึ้นอยู่กับว่ากำลังดำเนินการ คำสั่งอะไร มีกี่รอบของเครื่อง แบ่งการทำงานแต่ละส่วนได้ดังนี้

- เรจิสเตอร์คำสั่ง ทำหน้าที่รับรหัสคำสั่งจากบัสข้อมูลซึ่งส่งมาจากหน่วยความจำโปรแกรมโดยมีสัญญาณ INSreg_in_en จากส่วนควบคุมการไปนำคำสั่งมา (fetch instruction control) ทำหน้าที่กำหนดว่าจะ ไปนำคำสั่งมาเมื่อเวลาใด
- นัมเบอร์ออฟไซเคิล (NUM_of_cycle) เมื่อตัวถอดรัสได้รับรหัสคำสั่ง จากเรจิสเตอร์คำสั่ง แล้วทำการ ถอดรหัสว่าจะต้องกระทำการ (execute) ในกี่รอบของเครื่องเพื่อส่งให้ส่วนควบคุมการไปนำคำสั่งมา เพื่อสร้างสัญญาณไปกำหนดเวลาในการไปนำคำสั่งมาในครั้งต่อไป
- ทีเจน (T_GEN)ทำหน้าที่สร้างสัญญาณ ทีหนึ่ง (T_1) ถึงทียี่สิบ (T_20) ซึ่งจะเป็นค่า 1 ในช่วงสถานะหนึ่งๆตามหมายเลขชื่อ เพื่อกำหนดจังหวะการทำงานของสัญญาณควบคุมต่างๆ และกำหนดจังหวะการไปนำคำสั่งมาจากบัสข้อมูล
- ตัวถอดรหัสควบคุม (control decoder) ทำหน้าที่นำเอารหัสคำสั่งจากเรจิสเตอร์คำสั่ง มาถอดรหัสเป็นสัญญาณควบคุม ของส่วนต่างๆในดาตาพาท

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

ปี ค .ศ . 1990-1993 The Institute of Electrical and Electronics Engineers (IEEE)[2] ได้กำหนดมาตรฐาน IEEE Std 1149.1-1990 ออกมาเป็นมาตรฐานในการเชื่อมต่อวงจรและเครื่องมือทดสอบรวมทั้งข้อมูลสอบ อีกทั้งรวมบาวน์ดารี สแกน เรจิสเตอร์ ซึ่งเป็นส่วนที่สามารถตอบสนองกับบางชุดคำสั่งที่ออกแบบมาเพื่อช่วยในการทดสอบวงจร ทำให้แนวทางการออกแบบการทดสอบต่างๆเป็นไปในแนวทางเดียวกัน และใช้สำหรับการพัฒนาจนถึงปัจจุบัน

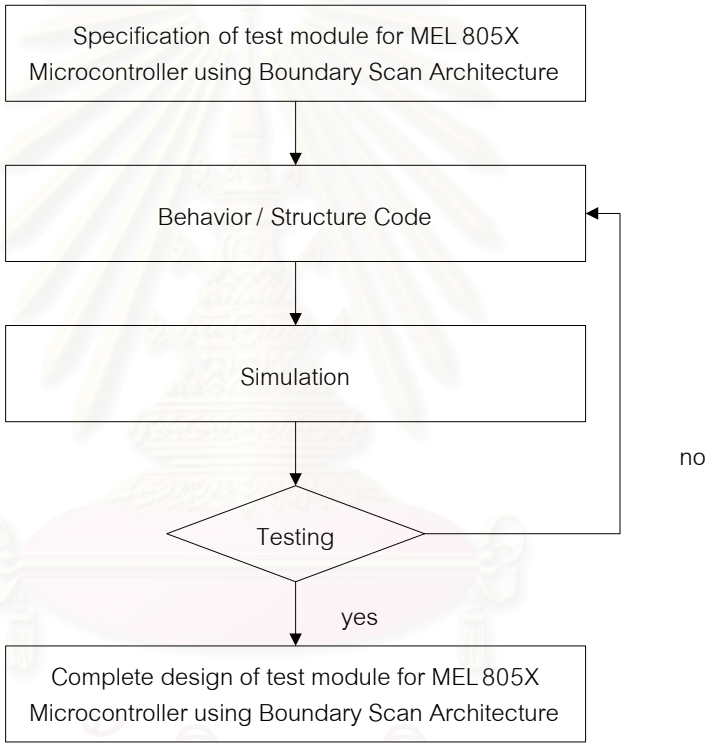
ปีค.ศ. 1992 นายปีเตอร์ เอ็ม แคมป์เบลล์ และคณะวิจัย[8]ได้เสนองานวิจัย Implementation of IEEE Std 1149.1-1990 in VHDL ที่ งาน The Spring 1992 VHDL International User's Forum Conference โดยได้พัฒนาออกแบบวงจรทดสอบ ตามมาตรฐาน IEEE 1149.1 โดยการใช้ภาษาวีเอชดีแอล ในการอธิบายวงจรทดสอบ และสามารถจำลองการทำงานให้เห็นภาพการทำงาน และเสนอวิธีการนำไปใช้งาน ซึ่งสามารถนำการออกแบบนี้ไปใช้ในการออกแบบส่วนจำเพาะ สำหรับทดสอบได้ แต่ยังต้องทำการปรับให้เข้ากับแต่ละวงจร

เจนวิทย์ ศรีหารักษา และคณะวิจัย[1]ห้องปฏิบัติการไมโครอิเล็กทรอนิกส์ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ นำเสนองานวิจัยการออกแบบ ไมโครคอนโทรลเลอร์ความเร็วสูง MEL805x ขนาด 8 บิต โดยมีชุดคำสั่งและสัญญาณสำหรับติดต่อกับอุปกรณ์รอบข้างเหมือนกับไมโครคอนโทรลเลอร์ตระกูล MCS-51 แต่มีจังหวะการทำงานเร็วกว่าไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทั่วไปประมาณ 3 เท่า โดยใช้ภาษาวีเอชดีแอลในการออกแบบ และพัฒนาจนสามารถนำไปผลิตเป็นไอซีต้นแบบ

บทที่ 3

การออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน

3.1 ขั้นตอนการออกแบบส่วนจำเพาะ สำหรับการทดสอบ

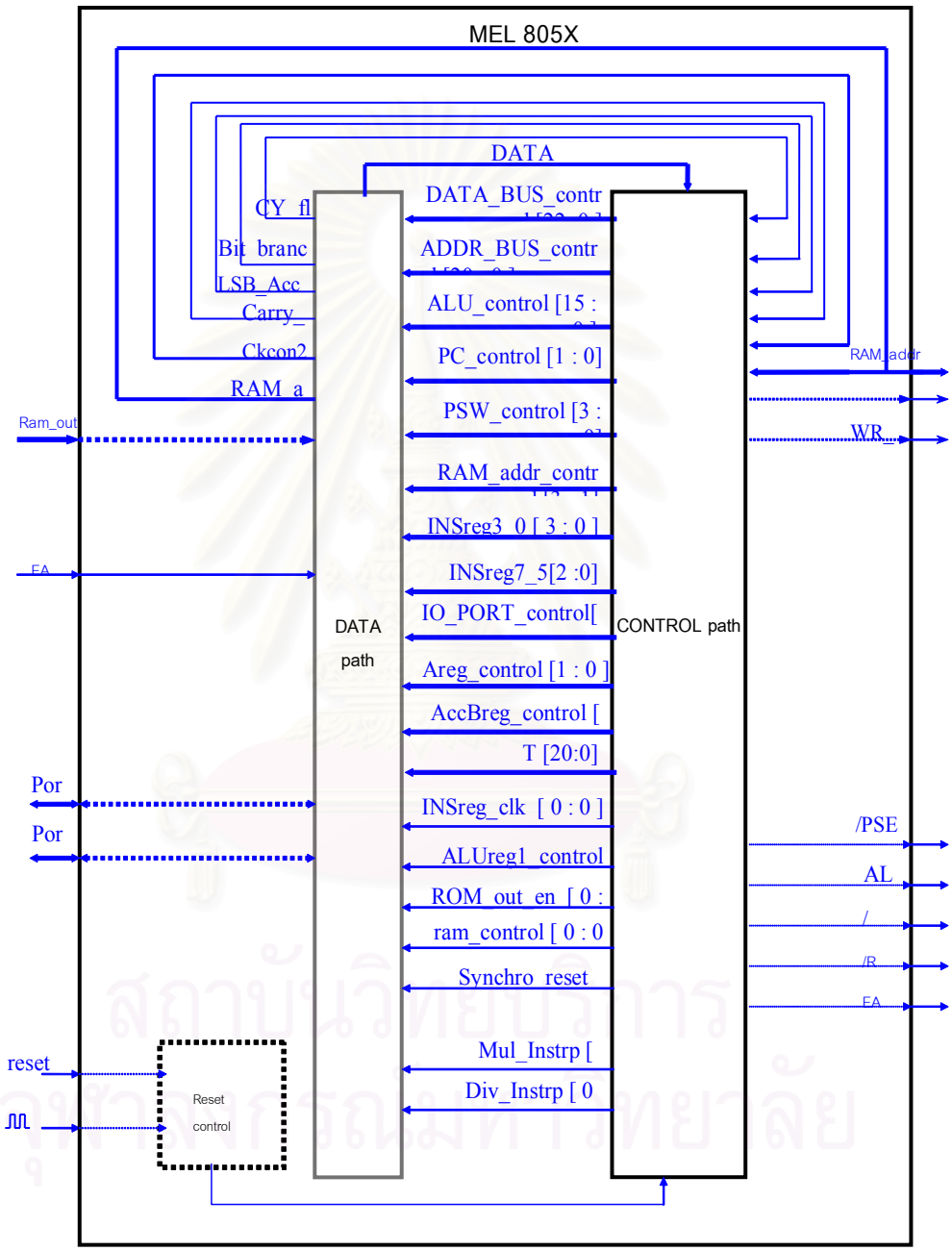


รูปที่ 3.1 ขั้นตอนการออกแบบออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน

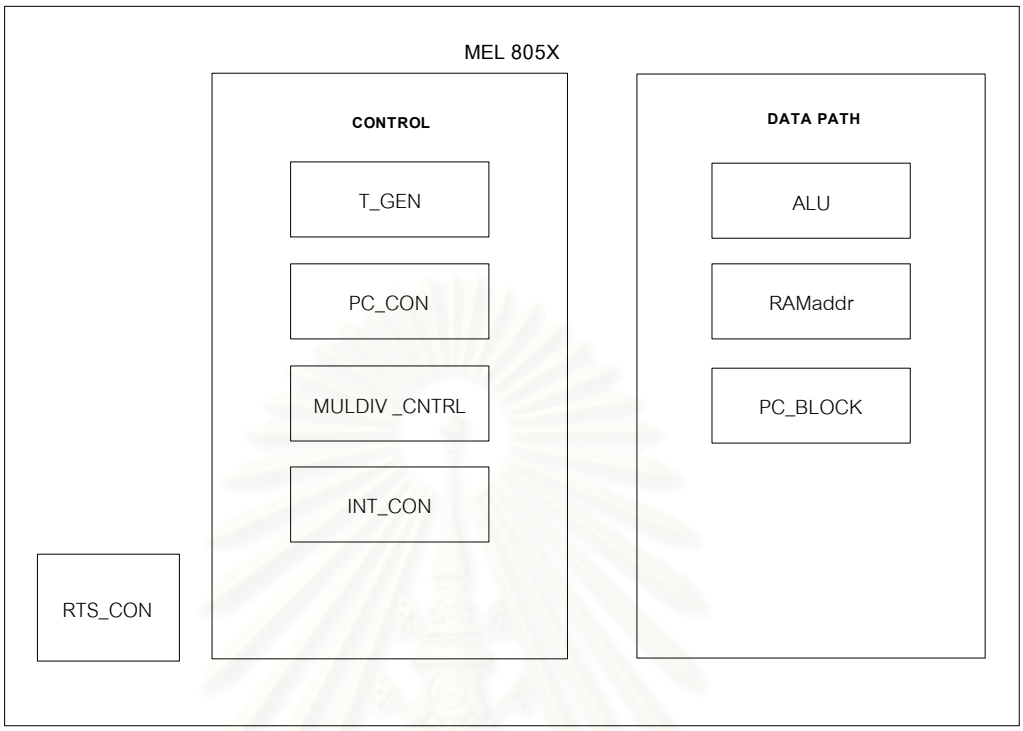
3.1.1 กำหนดคุณลักษณะของส่วนจำเพาะ สำหรับการทดสอบ

การกำหนดคุณลักษณะ เป็นการกำหนดโครงสร้างของวงจรส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน โดยที่การออกแบบนั้นจะใช้ บาวน์ดารี สแกน เรจิสเตอร์ กับทุกขั้นส่วนของแกนไมโครคอนโทรลเลอร์

MEL 805X ซึ่งประกอบด้วย ชิ้นส่วนใหญ่อยู่ 2 ชิ้นคือ ส่วนคอนโทรลพาทและดาตาพาท ดังรูปที่ 3.2 ซึ่งภายในแต่ละชิ้นส่วนจะประกอบไปด้วยชิ้นส่วนย่อยต่างๆ ดังรูปที่ 3.3



รูปที่ 3.2 ชิ้นส่วนดาตาพาทและคอนโทรลพาท[1]



รูปที่ 3.3 ชิ้นส่วนภายในของแกนไมโครคอนโทรลเลอร์ MEL 805X[1]

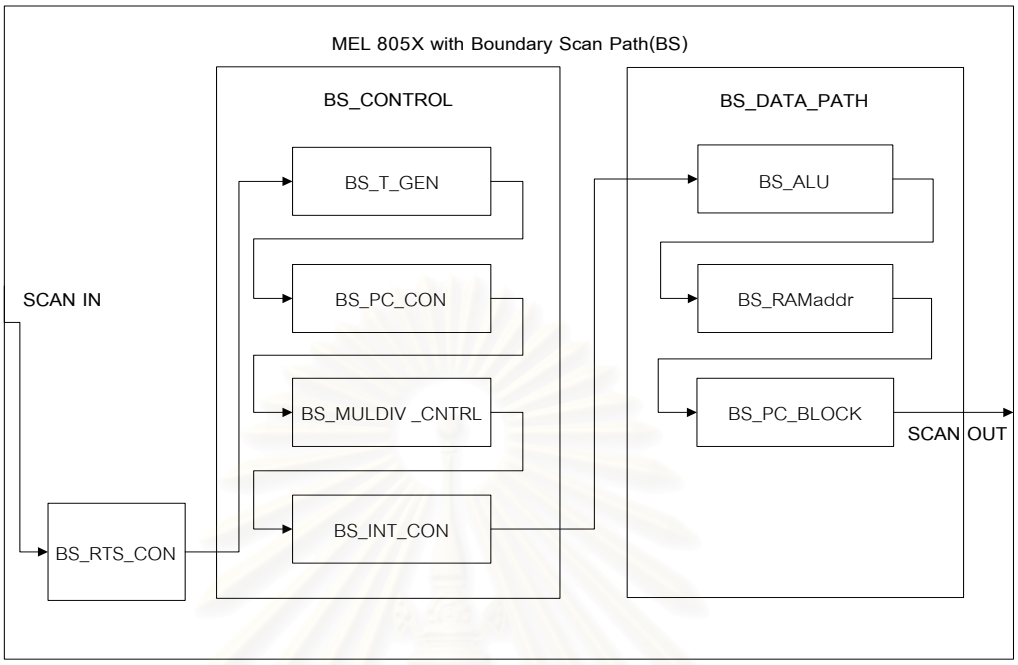
ดังนั้นการออกแบบจึงจำเป็นต้องออกแบบให้มี 2 สแกนพาท (scan path) เพื่อให้ครอบคลุมทุกชิ้นส่วนดังนี้

3.1.1.1 บาวน์ดารี สแกนพาท (BS : boundary scan path)

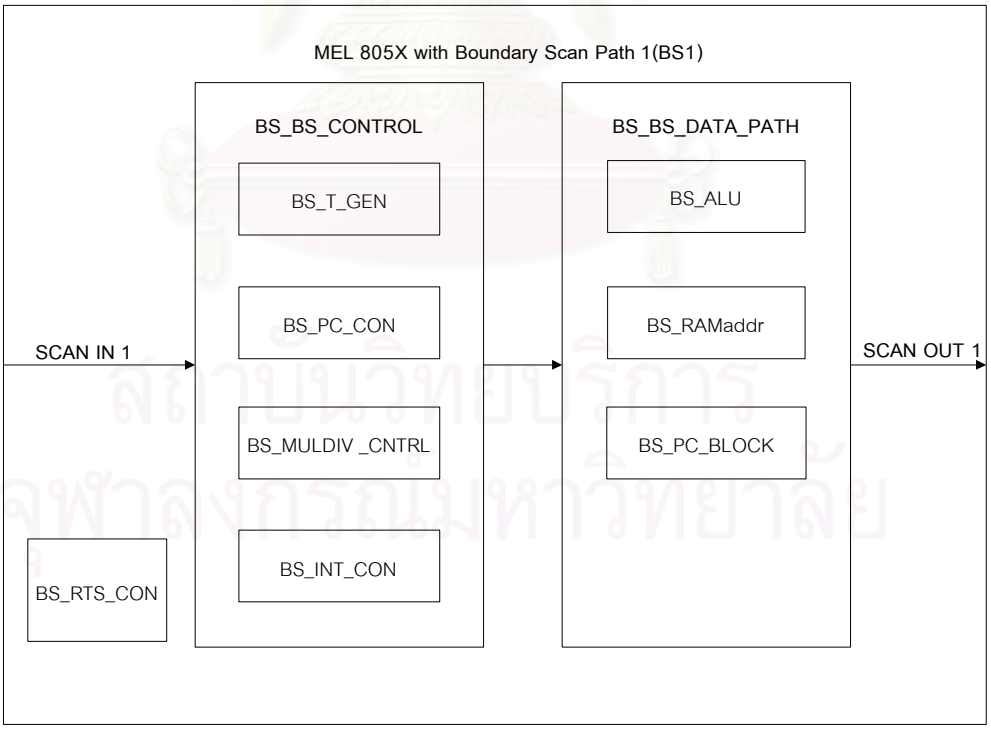
บาวน์ดารี สแกนพาท จะเชื่อมต่อทุกชิ้นส่วนที่อยู่ภายในส่วนคอนโทรลพาทและดาตาพาท ดังรูปที่ 3.4 ซึ่งจะใช้สำหรับทดสอบการเชื่อมต่อ และสำหรับทดสอบฟังก์ชันของ หน่วยคำนวณและตรรกะ ประกอบไปด้วย บาวน์ดารี สแกน เรจิสเตอร์ BS_RTS_con ,BS_T_GEN ,BS_PC_CON ,BS_muldiv_cntrl ,BS_INT_con ,BS_ALU ,BS_RAMaddr และ BS_PC_BLOCK เชื่อมต่อกัน

3.1.1.2 บาวน์ดารี สแกนพาทหนึ่ง (BS1 : boundary scan path1)

บาวน์ดารี สแกนพาทหนึ่ง จะเชื่อมต่อระหว่างชิ้นส่วนคอนโทรลพาทและดาตาพาท ดังรูปที่ 3.5 ซึ่งจะใช้สำหรับทดสอบการเชื่อมต่อ และสำหรับทดสอบฟังก์ชันของ ตัวถอดรหัสประกอบไปด้วย บาวน์ดารี สแกน เรจิสเตอร์ BS_BS_control และ BS_BS_data_path เชื่อมต่อกัน



รูปที่ 3.4 บาวน์ดารี สแกนพาท



รูปที่ 3.5 บาวน์ดารี สแกนพาทหนึ่ง

โดยบาวน์ดารี สแกนพาท ทั้งสองจะอิสระต่อกันไม่เกี่ยวกันทำให้ ความซับซ้อนลดน้อยลง และ การออกแบบส่วนจำเพาะ สำหรับทำการทดสอบได้ง่ายขึ้น

3.1.2 การเขียนโปรแกรมต้นฉบับอธิบายพฤติกรรมและโครงสร้าง

การเขียนโปรแกรมต้นฉบับอธิบายพฤติกรรมและโครงสร้าง เป็นการออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรมบาวน์ดารี สแกน โดยใช้ภาษาวีเอชดีแอล ในการอธิบายการทำงานของวงจรโดยจะอธิบายพฤติกรรมและโครงสร้างของวงจร อ่านได้จาก ภาคผนวก ข.

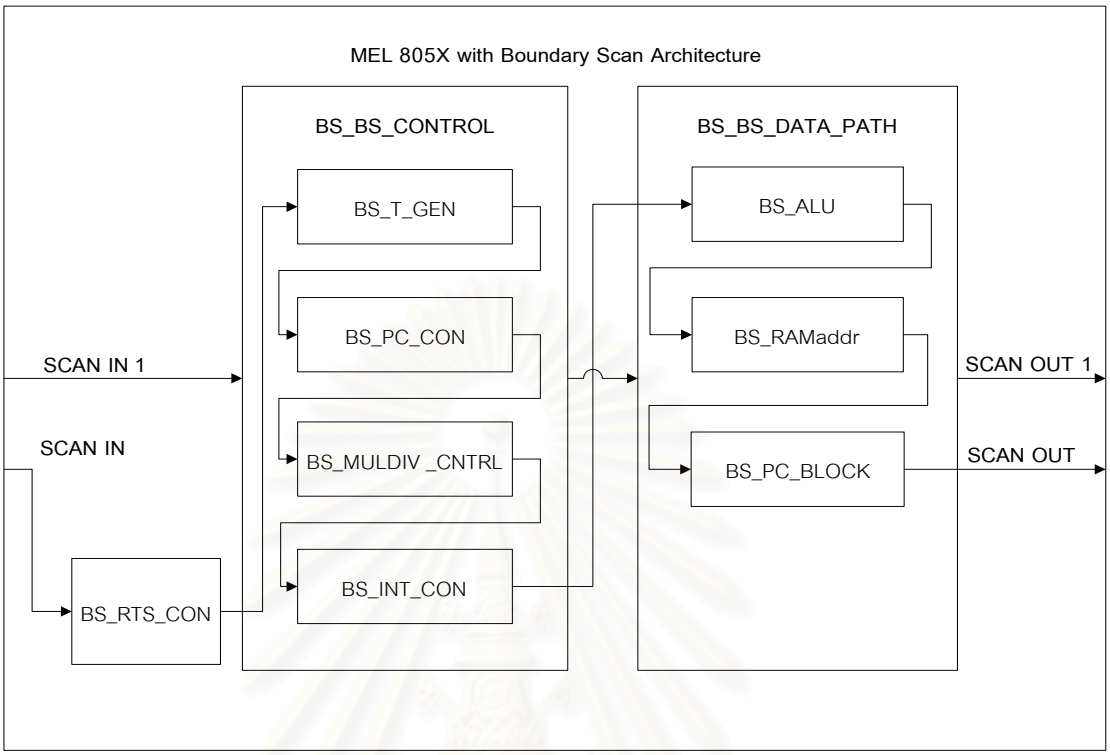
3.1.3 การจำลองการทำงานและการทดสอบ

การจำลองการทำงานและการทดสอบ เป็นการจำลองการทำงานเพื่อทดสอบความถูกต้องของวงจรที่ออกแบบ โดยใช้โปรแกรมโมเดลซิม ในการแปลโปรแกรมและจำลองการทำงานเพื่อทดสอบ ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ว่าสามารถทำงานได้ถูกต้องเป็นไปตามข้อกำหนดหรือไม่ ถ้ายังไม่ถูกต้องตามข้อกำหนดให้กลับไปแก้ไขโปรแกรมต้นฉบับ แล้วทำการแปลโปรแกรมใหม่ เพื่อนำกลับมาจำลองการทำงาน จนกว่าจะถูกต้องเป็นไปตามข้อกำหนด

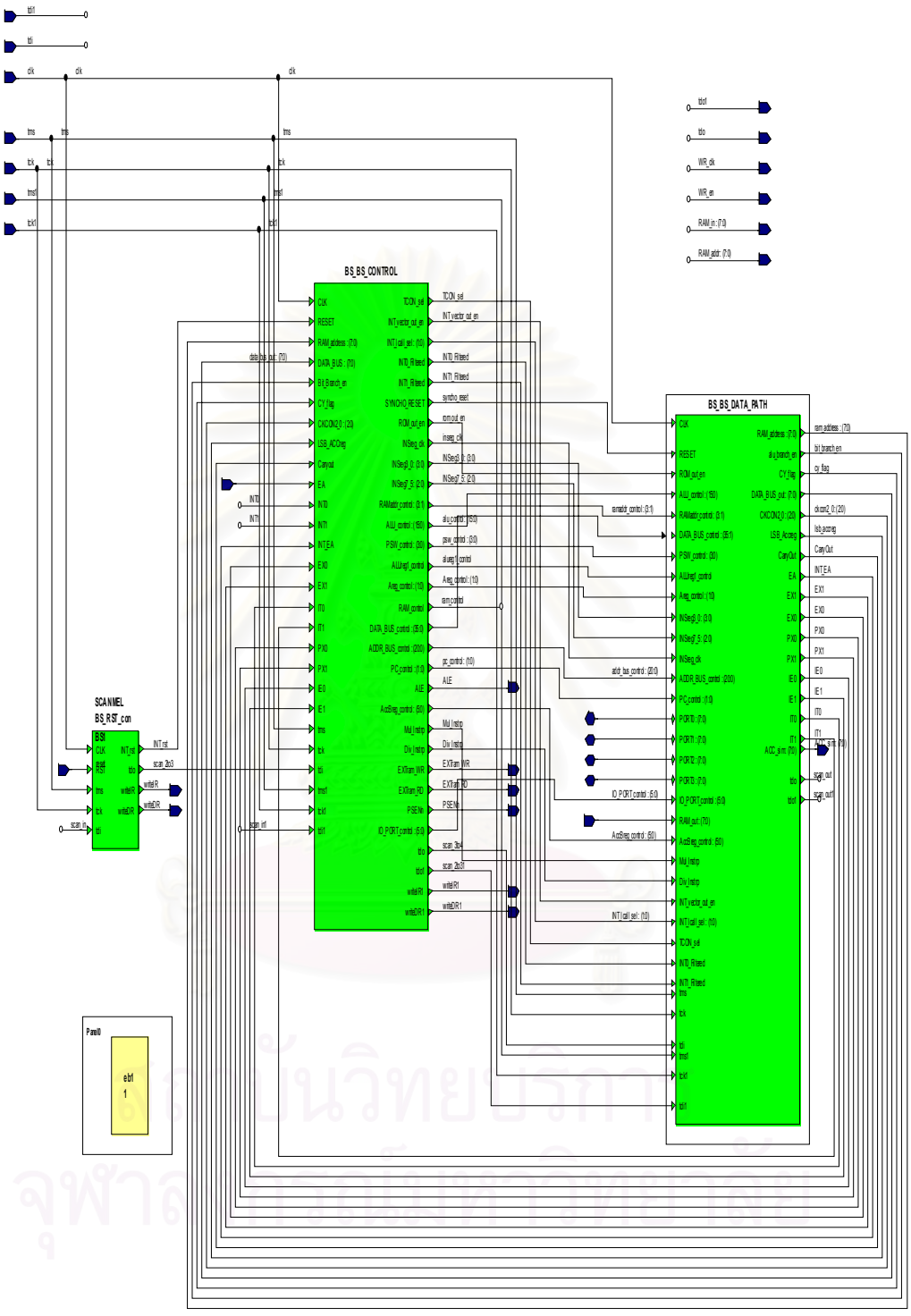
3.1.4 ส่วนจำเพาะ สำหรับการทดสอบที่สมบูรณ์

ส่วนจำเพาะที่ทดสอบจนมั่นใจว่า ส่วนจำเพาะที่ออกแบบทำงานถูกต้องตามข้อกำหนด ซึ่งจะได้ออกมาเป็นวงจรที่สมบูรณ์ และสามารถนำไปใช้ทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ต่อไป

ในการออกแบบนั้น จะทำการออกแบบและตรวจสอบความถูกต้องของ ตรรกะทดสอบ ในแต่ละชิ้นส่วนต่างๆก่อน แล้วจึงทำการทดสอบวงจรใหญ่ทั้งระบบอีก ทำการแก้ไขข้อผิดพลาดจนได้วงจรที่สมบูรณ์ดังรูปที่ 3.6 และรูปที่ 3.7



รูปที่ 3.6 โครงสร้างของส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน



รูปที่ 3.7 วงจรภายในของส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน

ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X จะประกอบด้วยส่วนประกอบต่างๆ ดังนี้

- บาวน์ดารี สแกนพาท

บาวน์ดารี สแกนพาท จะประกอบไปด้วยส่วนต่างๆดังตารางที่ 3.1

ตารางที่ 3.1 ตรรกะทดสอบที่ใช้สำหรับบาวน์ดารี สแกนพาท

Boundary scan register	Boundary scan cell			Instruction register	Instruction cell	TAP controller
	Input	Output	Total			
BS_RTS_con	2	1	3	1	2	1
BS_T_GEN	5	20	25	1	2	1
BS_PC_CON	16	7	23	1	2	1
BS_muldiv_cntrl	7	7	14	1	2	1
BS_INT_con	31	6	37	1	2	1
BS_ALU	35	12	47	1	2	1
BS_RAMaddr	19	8	27	1	2	1
BS_PC_BLOCK	36	48	84	1	2	1
รวม	151	109	260	8	16	8

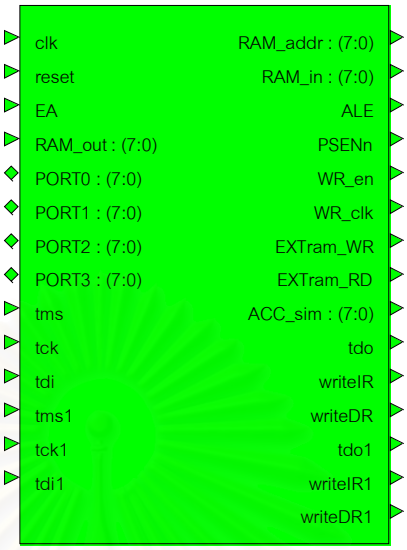
- บาวน์ดารี สแกนพาทหนึ่ง

บาวน์ดารี สแกนพาทหนึ่ง จะประกอบไปด้วยส่วนต่างๆดังตารางที่ 3.2

ตารางที่ 3.2 ตรรกะทดสอบที่ใช้สำหรับบาวน์ดารี สแกนพาทหนึ่ง

Boundary scan register	Boundary scan cell			Instruction register	Instruction cell	TAP controller
	Input	Output	Total			
BS_BS_control	37	120	157	1	2	1
BS_BS_data_path	155	40	195	1	2	1
รวม	192	160	352	8	16	8

ส่วนจำเพาะ ที่ออกแบบเมื่อประกอบเข้ากับ แกนไมโครคอนโทรลเลอร์ MEL 805X และจะได้วงจรที่สมบูรณ์ ดังรูปที่ 3.8

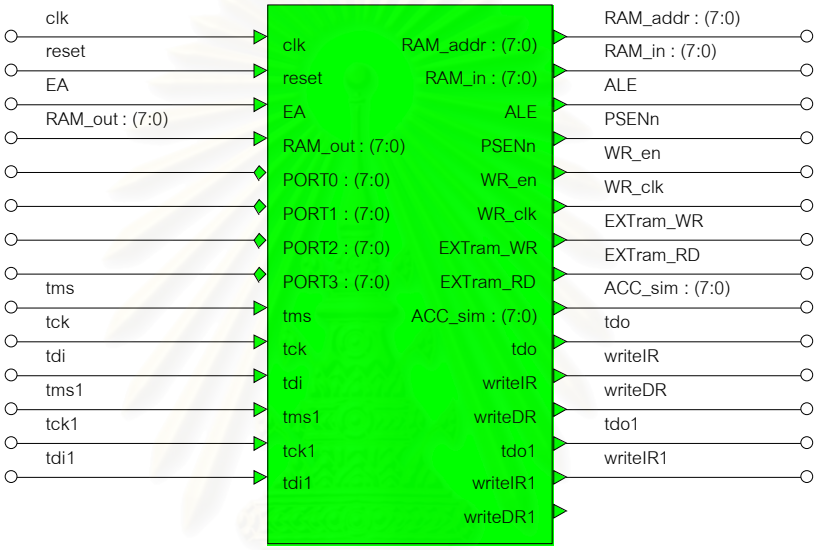


รูปที่ 3.8 ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน

3.2 การทำงานของส่วนจำเพาะ สำหรับการทดสอบ

การทำงานของส่วนจำเพาะ จะเป็นไปตามสถาปัตยกรรม บาวน์ดารี สแกน ซึ่งงานวิจัยนี้จะนำมาใช้ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยจะประกอบ ส่วนจำเพาะสำหรับการทดสอบ เข้ากับ แกนไมโครคอนโทรลเลอร์ MEL 805X เพื่อทดสอบการทำงาน ซึ่งลักษณะการทำงานของส่วนจำเพาะ จะทดสอบแกนไมโครคอนโทรลเลอร์ MEL 805X โดยการป้อนข้อมูลทดสอบเข้าไปแบบอนุกรม และอ่านผลลัพธ์ออกมาแบบอนุกรมเช่นเดียวกัน และการทำงานของส่วนจำเพาะนี้สามารถควบคุมได้จากสัญญาณนาฬิกาควบคุม สัญญาณเลือกวิธีทดสอบ สัญญาณนาฬิกาควบคุม (TCK1 : test clock 1) และสัญญาณเลือกวิธีทดสอบหนึ่ง (TMS1 : test mode select 1) ส่วนของข้อมูลทดสอบจะถูกป้อนเป็นสัญญาณอินพุตข้อมูลทดสอบ และสัญญาณอินพุตข้อมูลทดสอบหนึ่ง (TDI1 : test data input) อ่านผลโดยอ่านจากสัญญาณเอาท์พุตข้อมูลทดสอบ และ สัญญาณเอาท์พุตข้อมูลทดสอบหนึ่ง (TDO1 : test data output 1) โดยการควบคุมสัญญาณทั้งหมดจะควบคุมโดยวงจรเทสเบนช์ (test bench) ที่ออกแบบมาโดยเฉพาะ ซึ่งเป็นวงจรที่ออกแบบมาควบคุม การทำงานของส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน ดังรูปที่ 3.9

ซึ่งการทำงานของวงจรเทสเบนช์ สามารถควบคุมการทำงานของส่วนจำเพาะ โดยควบคุมการสร้างสัญญาณควบคุมต่างๆ การป้อนข้อมูลทดสอบซึ่งสามารถอ่านข้อมูลทดสอบจากแฟ้ม (file) และการอ่านผลการทดสอบ สามารถบันทึกลงแฟ้ม เพื่อนำไปวิเคราะห์ และสามารถจะแก้ไขเปลี่ยนแปลงได้ โดยการแก้ไขการอธิบายพฤติกรรมของวงจร เพื่อให้เหมาะสมกับการทดสอบในแบบต่างๆ

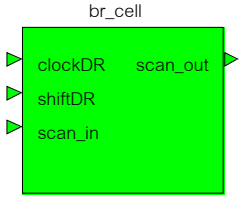


รูปที่ 3.9 วงจรเทสเบนช์ สำหรับทดสอบส่วนจำเพาะ

3.3 การทำงานแต่ละชิ้นส่วนของส่วนจำเพาะ สำหรับการทดสอบ

การทำงานในแต่ละชิ้นส่วนจะอธิบายรายละเอียดชิ้นส่วนต่างๆที่นำมาประกอบเป็น ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ดังนี้

3.3.1 เซลล์เรจิสเตอร์ผ่าน (Br_cell : bypass register cell)



รูปที่ 3.10 เซลล์เรจิสเตอร์ผ่าน

เซลล์เรจิสเตอร์ผ่านทำหน้าที่ เป็นผ่านสัญญาณจากขาอินพุทสแกนอิน (scan_in) ออกไปยังขาเอาท์พุท แสกนเอาท์ (scan_out) ดังรูปที่ 3.10 สามารถแบ่ง สัญญาณต่างๆได้ดังนี้

นาฬิกาดีอาร์ (clockDR) สัญญาณนาฬิกา

ชิพท์ดีอาร์ (shiftDR) สัญญาณควบคุมค่าที่ผ่านไปยังแสกนเอาท์

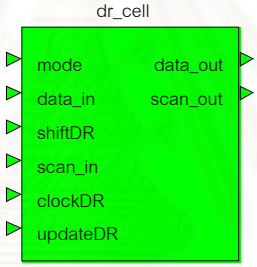
0 ค่าสัญญาณที่แสกนเอาท์เป็น 0

1 ค่าสัญญาณที่แสกนเอาท์เลื่อนมาจากสแกนอิน

สแกนอิน ทำหน้าที่รับสัญญาณ อินพุท

แสกนเอาท์ ค่าผลลัพธ์ที่ได้

3.3.2 เซลล์เรจิสเตอร์ข้อมูล (Dr_cell : data register cell)



รูปที่ 3.11 เซลล์เรจิสเตอร์ข้อมูล

เซลล์เรจิสเตอร์ข้อมูลทำหน้าที่ เป็น สแกน เซลล์ อยู่ตามขาอินพุทและเอาท์พุทของชิ้นส่วนต่างๆ ดังรูปที่ 3.11 สามารถแบ่ง สัญญาณต่างๆได้ดังนี้

วิธี (mode) สัญญาณควบคุมวิธีทำงาน ของเซลล์เรจิสเตอร์ข้อมูล

0 แบบปกติ

1 แบบทดสอบ พร้อมทำการทดสอบ

ข้อมูลเข้า (data in) ทำหน้าที่รับสัญญาณ อินพุท

ชิพท์ดีอาร์ สัญญาณควบคุมการเลื่อนค่า

0 จับค่าจากข้อมูลเข้า เก็บไว้ใน เรจิสเตอร์

1 เลื่อนค่าจากสแกนอินไปยังแสกนเอาท์

สแกนอิน ทำหน้าที่รับสัญญาณ อินพุทจากสแกนพาท

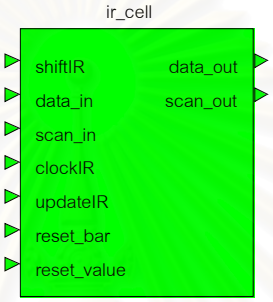
นาฬิกาดีอาร์ สัญญาณนาฬิกา

อัปเดตดีอาร์ สัญญาณควบคุมการขับสัญญาณ

ข้อมูลออก (data out) ค่าผลลัพธ์ที่ได้

สแกนเอาท์ ค่าผลลัพธ์ที่ได้ ต่อเป็นสแกนพาท

3.3.3 เซลล์เรจิสเตอร์คำสั่ง (Ir_cell : instruction register)



รูปที่ 3.12 เซลล์เรจิสเตอร์คำสั่ง

เซลล์เรจิสเตอร์คำสั่งทำหน้าที่ เป็น เซลล์อยู่ที่เรจิสเตอร์คำสั่ง ดังรูปที่ 3.12 สามารถแบ่ง สัญญาณต่างๆได้ดังนี้

ชิพท์ไออาร์ (shiftIR) สัญญาณควบคุมการเลื่อนค่า

0 จับค่าจากข้อมูลเข้า เก็บไว้ใน เรจิสเตอร์

1 เลื่อนค่าจากข้อมูลเข้า ไปยัง สแกนเอาท์

ข้อมูลเข้า ทำหน้าที่รับสัญญาณ อินพุท

สแกนอิน ทำหน้าที่รับสัญญาณ อินพุท

นาฬิกาไออาร์ สัญญาณนาฬิกา

อัปเดตไออาร์ สัญญาณควบคุมการขับสัญญาณ

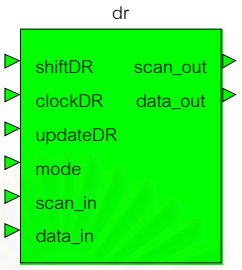
รีเซ็ตบาร์ (reset_bar) สัญญาณตั้งใหม่

ค่าตั้งใหม่ (reset value) ค่าสัญญาณตั้งใหม่

ข้อมูลออก ค่าผลลัพธ์ที่ได้

สแกนเอาท์ ค่าผลลัพธ์ที่ได้ ต่อเป็นสแกนพาท

3.3.4 เรจิสเตอร์ข้อมูล



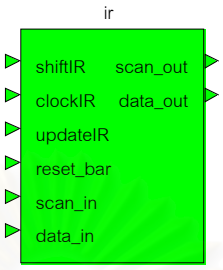
รูปที่ 3.13 เรจิสเตอร์ข้อมูล

เรจิสเตอร์ข้อมูลทำหน้าที่เป็นเรจิสเตอร์ข้อมูลอยู่ตามชิ้นส่วนต่างๆ ดังรูปที่ 3.13 และสามารถแบ่ง สัญญาณต่างๆได้ดังนี้

ชิพท์ดีอาร์	สัญญาณควบคุมการเลื่อนค่า
	0 จับค่าจากข้อมูลเข้า เก็บไว้ใน เรจิสเตอร์
	1 เลื่อนค่าจากสแกนอินไปยังแสกนเอาท์
นาฬิกาดีอาร์	สัญญาณนาฬิกา
อัปเดตดีอาร์	สัญญาณควบคุมการขับสัญญาณ
วิธี	สัญญาณควบคุมวิธีทำงาน ของเซลล์เรจิสเตอร์ข้อมูล
	0 แบบปกติ
	1 แบบทดสอบ พร้อมทำการทดสอบ
สแกนอิน	ทำหน้าที่รับสัญญาณ อินพุทจากสแกนพาท
แสกนเอาท์	ค่าผลลัพธ์ที่ได้ ต่อเป็นสแกนพาท
ข้อมูลเข้า	ทำหน้าที่รับ อินพุท ขนาดเท่ากับจำนวนเซลล์เรจิสเตอร์ข้อมูล
ข้อมูลออก	ค่าผลลัพธ์ที่ได้ ขนาดเท่ากับจำนวนเซลล์เรจิสเตอร์ข้อมูล

สัญญาณควบคุมทั้งหมดจะต่อไปยังทุกๆ เซลล์เรจิสเตอร์ข้อมูล และ สแกนพาทต่อ สแกนอินเข้ากับ แสกนเอาท์ของ เซลล์เรจิสเตอร์ข้อมูล ตัวแรกและ ต่อ แสกนเอาท์จาก แสกนเอาท์ของเซลล์เรจิสเตอร์ข้อมูลตัวสุดท้าย

3.3.5 เรจิสเตอร์คำสั่ง



รูปที่ 3.14 เรจิสเตอร์คำสั่ง

เรจิสเตอร์คำสั่ง ทำหน้าที่เป็นเรจิสเตอร์คำสั่งอยู่ตามชิ้นส่วนต่างๆ ตามรูปที่ 3.14 สามารถแบ่ง สัญญาณต่างๆได้ดังนี้

ชิพท์ไออาร์ (shiftIR) สัญญาณควบคุมการเลื่อนค่า
0 จับค่าจากข้อมูลเข้า เก็บไว้ใน เรจิสเตอร์
1 เลื่อนค่าจากข้อมูลเข้า ไปยัง สแกนเอาท์

นาฬิกาไออาร์ สัญญาณนาฬิกา

อัปเดตไออาร์ สัญญาณควบคุมการขับสัญญาณ

รีเซ็ตบาร์ สัญญาณตั้งใหม่

ข้อมูลออก ค่าผลลัพธ์ที่ได้

สแกนอิน ทำหน้าที่รับสัญญาณ อินพุทจากสแกนพาท

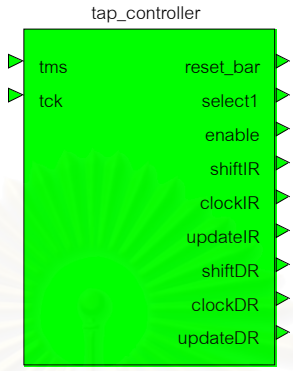
สแกนเอาท์ ค่าผลลัพธ์ที่ได้ ต่อเป็นสแกนพาท

ข้อมูลเข้า ทำหน้าที่รับ อินพุท ขนาดเท่ากับจำนวนเรจิสเตอร์คำสั่ง

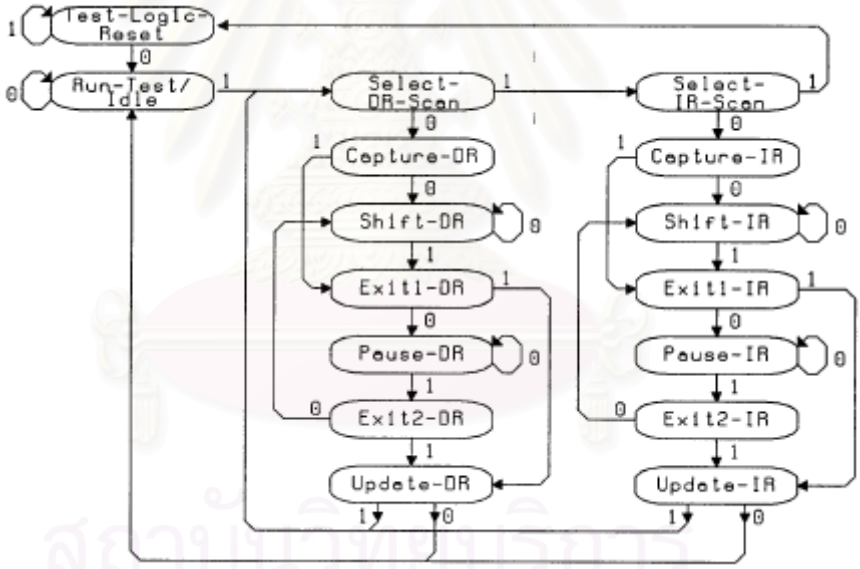
ข้อมูลออก ค่าผลลัพธ์ที่ได้ ขนาดเท่ากับจำนวนเรจิสเตอร์คำสั่ง

สัญญาณควบคุมทั้งหมดจะต่อไปยังทุกๆ เซลล์เรจิสเตอร์คำสั่ง และสแกนพาทจะต่อ สแกนอินเข้ากับ สแกนอินของ เซลล์เรจิสเตอร์คำสั่งตัวแรกและ ต่อ แสกนเอาท์จากแสกนเอาท์ ของ เซลล์เรจิสเตอร์คำสั่งตัวสุดท้าย

3.3.6 ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ



รูปที่ 3.15 ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ



รูปที่ 3.16 สถานะการทำงานของตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ[2]

ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ ทำหน้าที่สร้างสัญญาณนาฬิกาภายในและสัญญาณควบคุมต่างๆที่ใช้สำหรับวงจรทดสอบประกอบด้วย 16 สถานะ ตามรูปที่ 3.16 และการเปลี่ยนสถานะจะขึ้นอยู่กับค่าสัญญาณเลือกวิธีทดสอบ และเกิดขึ้นที่ขอบขาขึ้นของสัญญาณนาฬิกาทดสอบ เท่านั้น ดังรูปที่ 3.15 สามารถแบ่ง สัญญาณต่างๆได้ดังนี้

ข้อมูลออก ค่าผลลัพธ์ที่ได้ ขนาดเท่ากับจำนวนเซลล์เรจิสเตอร์คำสั่ง

เลือกวิธีทดสอบ ใช้ควบคุมการทำงาน ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ

นาฬิกาทดสอบ สัญญาณนาฬิกา

รีเซ็ตบาร์ สัญญาณตั้งใหม่

เลือกหนึ่ง (select1) สัญญาณ นี้ไม่ได้ใช้

เปิดทาง สัญญาณ นี้ไม่ได้ใช้

ชิพท์ไออาร์ สัญญาณควบคุมการเลื่อนค่า

นาฬิกาไออาร์ สัญญาณนาฬิกาสำหรับเรจิสเตอร์คำสั่ง

อัปเดตไออาร์ สัญญาณควบคุมการขับสัญญาณสำหรับเรจิสเตอร์คำสั่ง

ชิพท์ไออาร์ สัญญาณควบคุมการเลื่อนค่า

นาฬิกาดีอาร์ สัญญาณนาฬิกาสำหรับเรจิสเตอร์ข้อมูล

อัปเดตดีอาร์ สัญญาณควบคุมการขับสัญญาณสำหรับเรจิสเตอร์ข้อมูล

ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ จะมีการทำงานสำคัญสำหรับการทดสอบ 3 งานคือ กระตุ้น (stimulus) ,การกระทำการ (execution) และ จับผลตอบรับ (response capture) โดยจะมีสถานะที่เกี่ยวของและสำคัญดังนี้

Test-Logic-Reset

ตรรกะทดสอบ จะไม่ทำงานในสถานะนี้โดย ชิ้นส่วนต่างๆจะทำงานปกติ และจะกลับมายังสถานะนี้เมื่อค่าสัญญาณเลือกวิธีทดสอบเป็น 1 ติดต่อกัน 5 สัญญาณนาฬิกาทดสอบ เมื่อเข้าสู่สถานะนี้แล้วเอาท์พุตเรจิสเตอร์คำสั่ง จะให้ค่าเป็น เรจิสเตอร์ผ่าน

Run-Test/Idle

เมื่อคำสั่ง ปรากฏในเรจิสเตอร์คำสั่ง สถานะนี้จะทำงานเพื่อจะกระทำการ คำสั่งนั้นๆและถ้าคำสั่งนั้นไม่มีการ ฟังก์ชันกระทำการ ก็จะไม่เปลี่ยนค่าในเรจิสเตอร์ข้อมูล

Capture-DR / Capture-IR

ข้อมูลจะถูกบรรจุแบบขนานไว้ในเรจิสเตอร์ข้อมูลหรือ เรจิสเตอร์คำสั่งโดยจะถูกเลือกโดยคำสั่งนั้นๆ

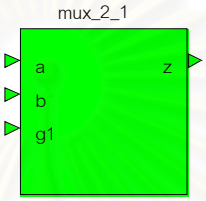
Shift-DR / Shift-IR

ข้อมูลจะถูก เลื่อนผ่านเรจิสเตอร์ที่ต่อระหว่าง สัญญาณอินพุตข้อมูลทดสอบกับสัญญาณเอาท์พุตข้อมูลทดสอบโดยจะเลื่อนหนึ่งครั้งต่อขอบขาขึ้นของสัญญาณนาฬิกาทดสอบ

Update-DR / Update-IR

ข้อมูลจะถูกแลตช์ ไว้ในเอาท์พุตแบบขนาน ของเรจิสเตอร์ข้อมูลหรือเรจิสเตอร์คำสั่ง เพื่อป้องกันการเปลี่ยนแปลงของข้อมูล

3.3.7 อุปกรณ์รวมส่งสัญญาณแบบสองอินพุตต่อหนึ่งเอาท์พุต

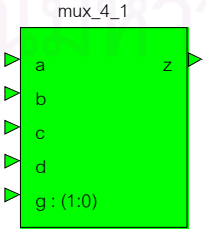


รูปที่ 3.17 อุปกรณ์รวมส่งสัญญาณแบบสองอินพุตต่อหนึ่งเอาท์พุต

อุปกรณ์รวมส่งสัญญาณแบบสองอินพุตต่อหนึ่งเอาท์พุต ทำหน้าที่เลือกสัญญาณระหว่าง สัญญาณจาก เอาท์พุตของ อุปกรณ์รวมส่งสัญญาณแบบสี่อินพุตต่อหนึ่งเอาท์พุตต่อเข้าขา a (pin a) กับสัญญาณจากเรจิสเตอร์คำสั่งต่อเข้าขาบี (pin b) โดยขึ้นอยู่กับสัญญาณชิพท์ไออาร์ต่อเข้าขา จีหนึ่ง (g1) ดังรูปที่ 3.17 สามารถแบ่ง สัญญาณๆได้ดังนี้

เอ และ บี	สัญญาณอินพุต
จีหนึ่ง	สัญญาณควบคุมการเลือกค่า
z (แซด)	สัญญาณเอาท์พุต

3.3.8 อุปกรณ์รวมส่งสัญญาณแบบสี่อินพุตต่อหนึ่งเอาท์พุต



รูปที่ 3.18 อุปกรณ์รวมส่งสัญญาณแบบสี่อินพุตต่อหนึ่งเอาท์พุต

อุปกรณ์รวมส่งสัญญาณแบบสี่อินพุตต่อหนึ่งเอาท์พุต ทำหน้าที่เลือกสัญญาณระหว่าง สัญญาณจาก เรจิสเตอร์ผ่าน โดยต่อเข้าขาเอและขาบี กับสัญญาณจาก เรจิสเตอร์ข้อมูลต่อเข้าขาซี (pin c) และขาดี (pin d) โดยขึ้นอยู่กับ เอาท์พุตของเรจิสเตอร์คำสั่ง ต่อเข้าขาจี (pin g) ตามรูปที่ 3.18 สามารถแบ่ง สัญญาณต่างๆ ได้ดังนี้

เอ ,บี ,ซี และ ดี	สัญญาณอินพุต
จี	สัญญาณควบคุมการเลือกค่า ขนาด 2 บิต
แซด	สัญญาณเอาท์พุต

บทที่ 4

การสร้างชุดข้อมูลทดสอบสำหรับทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X

4.1 แบบจำลองความผิดพร่อง (fault model)

สำหรับการทดสอบแกนไมโครคอนโทรลเลอร์จะใช้ แบบจำลองความผิดพร่อง[5,6]เกี่ยวกับโครงสร้าง (structural fault) และความผิดพร่องเกี่ยวกับหน้าที่ (functional fault) โดยใช้ความผิดพร่องเกี่ยวกับโครงสร้าง สำหรับการทดสอบการเชื่อมต่อ และใช้ความผิดพร่องเกี่ยวกับหน้าที่สำหรับการทดสอบฟังก์ชันการทำงานของหน่วยคำนวณและตรรกะ และ ตัวถอดรหัสในคอนโทรลพาท ซึ่งอธิบายได้ดังนี้

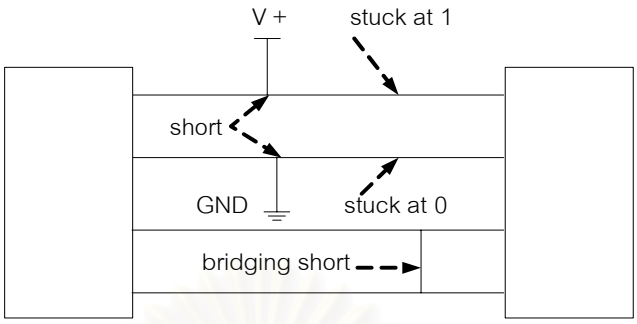
4.1.1 แบบจำลองความผิดพร่องเกี่ยวกับโครงสร้าง (structural fault model)

แบบจำลองความผิดพร่องเกี่ยวกับโครงสร้างโดยปกติจะสมมุติให้ ส่วนประกอบต่างๆปราศจากข้อผิดพร่อง ยกเว้นการเชื่อมต่อ ที่จะมีผลกระทบ ปกติผลกระทบนั้นจะเป็น การลัดวงจร (short) และการเปิด (open) รวมทั้งส่วนประกอบซึ่งใส่ไว้ไม่เหมาะสม (improperly inserted component)[9,10]

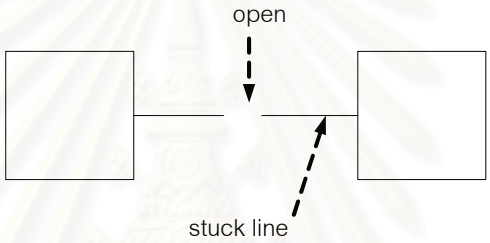
การลัดวงจร[6,9] คือการเกิดจุดเชื่อมต่อที่ไม่ต้องการให้เกิดการเชื่อมต่อขึ้น เช่นเทคโนโลยีส่วนใหญ่ จะเกิดการลัดวงจร เกิดขึ้นระหว่าง กราวด์ (ground) หรือ พาวเวอร์ (power) กับสายสัญญาณทำให้สัญญาณมีแรงดันไฟฟ้าคงที่เกิด สตัคแอต (stuck at) ค่าตรรกะ v $(v \in \{0,1\})$ ดังรูปที่ 4.1 หรือการลัดวงจรระหว่างสายสัญญาณทำให้เกิดความผิดพร่องประสาน (bridging falut)

การเปิด [6,9]คือการเกิดจุดซึ่งทำให้การเชื่อมต่อขาดออกจากกัน เช่นเทคโนโลยีส่วนใหญ่จะเกิดการเปิด กับสายสัญญาณแบบทิศทางเดียว ทำให้สัญญาณอินพุตไม่สามารถรับได้เนื่องจากไม่สามารถเชื่อมต่อถึงกัน โดยสันนิษฐานว่าค่าตรรกะจะคงที่ซึ่งทำให้เกิดสตัคแอต ดังรูป 4.2 ทำให้สายสัญญาณ มีค่าแรงดันไฟฟ้าคงที่ สายสัญญาณ สตัคแอต ค่าตรรกะ a $(a \in \{0,1\})$

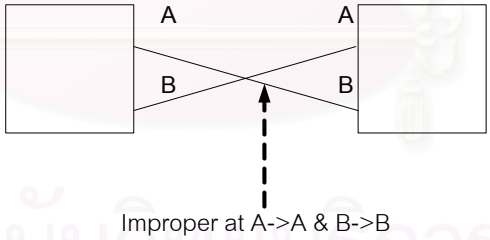
ส่วนประกอบซึ่งใส่ไว้ไม่เหมาะสม คือการเชื่อมต่อภายในที่ไม่ถูกต้องตามที่ได้กำหนดไว้ทำให้เกิดความผิดพร่อง



รูปที่ 4.1 การลัดวงจร[6]



รูปที่ 4.2 การเปิด[6]



รูปที่ 4.3 การเชื่อมต่อที่ไม่เหมาะสม[6]

4.1.2 แบบจำลองความผิดพร่องเกี่ยวกับหน้าที่ (functional fault model)[6]

แบบจำลองความผิดพร่องเกี่ยวกับหน้าที่ จะพยายามแสดงข้อผิดพร่องที่เกิดขึ้นจากการทำงานตามหน้าที่ของระบบ สามารถพิจารณาประสิทธิภาพของแบบจำลองความผิดพร่องเกี่ยวกับหน้าที่ ว่าดีหรือไม่ ได้จากการครอบคลุมการค้นหาความผิดพร่อง ซึ่งยากที่จะวัดในเชิงคุณภาพ เพราะ ไม่สามารถที่จะรู้การทดสอบความรู้ทั้งหมดของ แบบจำลองความผิดพร่องเกี่ยวกับหน้าที่

แบบจำลองความผิดพร่องเกี่ยวกับหน้าที่เป็นทั้ง แบบจำลองชัดแจ้ง (explicit model) หรือแบบจำลองเป็นนัย (implicit model) ซึ่งแบบจำลองชัดแจ้งจะแสดงความผิดพร่องแยกเป็นแต่ละประเภท โดยจะแยกกันทดสอบ ทำให้สามารถนิยามขอบเขตความผิดพร่องได้เล็กตามความเหมาะสม ทำให้การคำนวณในกระบวนการทดสอบมีความเป็นไปได้ แบบจำลองเป็นนัยจะแสดง กลุ่มของความผิดพร่องที่มีคุณสมบัติคล้ายกันรวมไว้ด้วยกัน ซึ่งความผิดพร่องทั้งหมดที่อยู่ในกลุ่มเดียวกันสามารถที่จะค้นหาได้โดยใช้กระบวนคำสั่ง (procedure) ที่คล้ายกัน

ในงานวิจัยได้เลือกทดสอบการลัดวงจรกับพาวเวอร์หรือกราวด์ ทดสอบการเปิดและส่วนประกอบซึ่งใส่ไว้ไม่เมาะสม สำหรับแบบจำลองความผิดพร่องเกี่ยวกับโครงสร้าง เพื่อทดสอบการเชื่อมต่อ สำหรับการลัดวงจรระหว่างสายสัญญาณ จะถูกตรวจสอบโดยอัตโนมัติในขั้นตอนการแปลภาษา จึงไม่ได้นำมาใช้ทดสอบ และสำหรับ แบบจำลองความผิดพร่องเกี่ยวกับหน้าที่ ได้เลือกแบบจำลองชัดแจ้ง เพื่อทดสอบเพราะเหมาะสำหรับการทดสอบ หน่วยคำนวณและตรรกะและตัวถอดรหัสใน คอนโทรลพาท

4.2 หลักการสร้างชุดข้อมูลทดสอบ (test data set)[6,9]

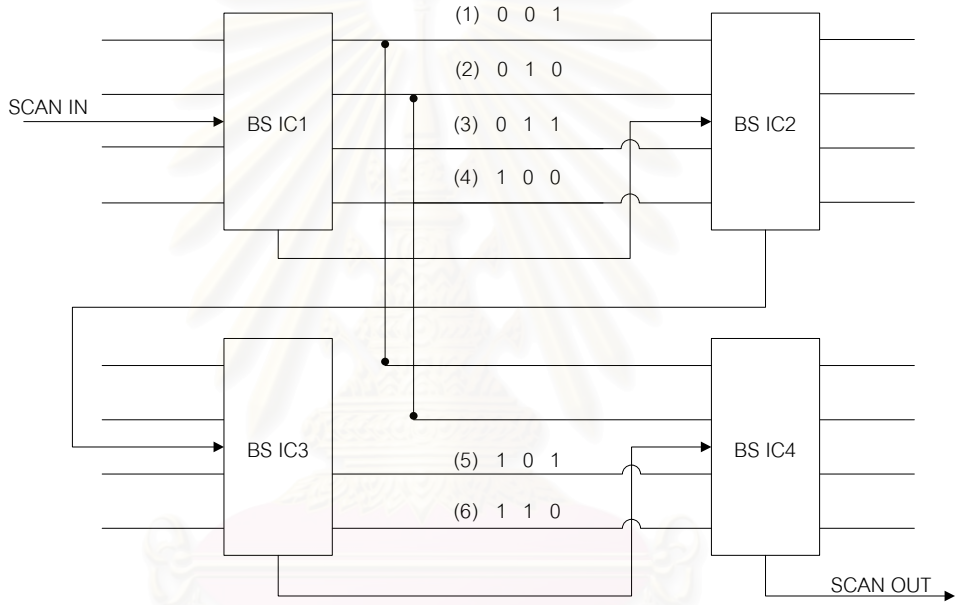
การออกแบบชุดข้อมูลทดสอบ จะนำข้อมูลของ แกนไมโครคอนโทรลเลอร์ MEL 805X มาใช้ในการออกแบบชุดข้อมูลทดสอบ โดยชุดข้อมูลทดสอบจะแบ่ง ชุดข้อมูลทดสอบตามลักษณะการทดสอบดังนี้

4.2.1 ชุดข้อมูลทดสอบการเชื่อมต่อ

การออกแบบชุดข้อมูลทดสอบการเชื่อมต่อ จะออกแบบเพื่อทดสอบการเชื่อมต่อสำหรับแกนไมโครคอนโทรลเลอร์ MEL 805X โดยออกแบบจากข้อมูลการเชื่อมต่อซึ่งเป็นข้อมูลที่เก็บรายละเอียดของการเชื่อมต่อของสัญญาณต่างๆภายในแกนไมโครคอนโทรลเลอร์ MEL 805X ในลักษณะตารางซึ่งจะเก็บข้อมูลของขาสัญญาณเอาท์พุตของแต่ละชิ้นส่วนในลักษณะแถว (row) และเก็บข้อมูลของขาสัญญาณอินพุตของแต่ละชิ้นส่วนในลักษณะ คอลัมน์ (column) ซึ่งขาสัญญาณเอาท์พุตที่มีการเชื่อมต่อไปยังขาสัญญาณอินพุตใดจะให้สัญลักษณ์ ‘1’ เพื่อแสดงถึงการเชื่อมต่อของสัญญาณระหว่างชิ้นส่วนหากจุดใดไม่มีการเชื่อมต่อจะให้สัญลักษณ์ ‘0’ ดังตัวอย่าง ตารางที่ 4.1 ซึ่งจะเห็นว่าขาสัญญาณเอาท์พุต RTS_CON_INT_rst เชื่อมต่อกับขาสัญญาณอินพุต T_GEN_RESET ถ้าพิจารณา การเชื่อมต่อระหว่างชิ้นส่วนต่างๆ สามารถนำมาหาความสัมพันธ์เพื่อสร้างชุดข้อมูลทดสอบการเชื่อมต่อ อธิบายได้จากตัวอย่างดังรูปที่ 4.3

ตารางที่ 4.1 ตัวอย่างข้อมูลการเชื่อมต่อ

O/I	1RTS_CON_CLK#1	1RTS_CON_RST#2	2_T_GEN_CLK#4	2T_GEN_RESET#5
1RTS_CON_INT_rst#3	0	0	0	1
2T_GEN_T_1#9	0	0	0	0
2T_GEN_T_2#10	0	0	0	0
2T_GEN_T_3#11	0	0	0	0



รูปที่ 4.4 ตัวอย่างการเชื่อมต่อระหว่างชิ้นส่วน BS IC1 ,BS IC2 ,BS IC3 และ BS IC4[9]

ตารางที่ 4.2 แสดงเลขจุดต่อ (node) และค่าเลขฐานสอง (binary)

เลขจุดต่อ	บิตที่ 3	บิตที่ 2	บิตที่ 1
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0

จากตัวอย่างการเชื่อมต่อระหว่างชิ้นส่วนต่างๆ สามารถใช้ความสามารถของสถาปัตยกรรม บาวน์ดารี สแกน ในการสร้างชุดข้อมูลทดสอบโดยกำหนดจุดต่อตามฝั่งตัวขับสัญญาณซึ่งจะแทนด้วย หมายเลขจุดต่อซึ่งจะไม่ซ้ำกันเพื่อใช้สำหรับการทดสอบการเชื่อมต่อที่ไม่เหมาะสม และค่าเลขฐานสองที่แสดงค่าตามเลขจุดต่อ ซึ่งจำนวนบิตของค่าเลขฐานสอง เท่ากับ $\log_2(n+2)$ เมื่อ n แทนจำนวน จุดต่อทั้งหมด ที่ต้องบวกสองเพราะว่าค่าเลขฐานสองเป็น '0' ทุกบิตใช้ทดสอบการลัดวงจรกับกราวด์ และค่าเลขฐานสองเป็น '1' ทุกบิตใช้ทดสอบการลัดวงจรกับพาวเวอร์ ตามตัวอย่าง ดังตารางที่ 4.2 ซึ่งหลักการทดสอบนั้นคือ ให้ BS IC1 และ BS IC3 ขับสัญญาณ ตามชุดข้อมูลทดสอบทีละบิตและให้ BS IC2 และ BS IC4 อ่านสัญญาณกลับมาถ้าการเชื่อมต่อไม่มีความผิดพลาด ก็จะสามารถอ่านผลทดสอบกลับมาได้อย่างถูกต้อง โดยตามตัวอย่างข้อมูลทดสอบจะมีทั้งหมด 3 เวกเตอร์ โดยแต่ละเวกเตอร์ที่สร้างจะได้มาจากแต่ละบิต เช่นเวกเตอร์ 1 จะมีค่าเป็น "101010" เวกเตอร์2 จะมีค่าเป็น "011001" และเวกเตอร์3 จะมีค่าเป็น "000111" ซึ่งจะนำเวกเตอร์ที่ได้นี้มาจัดรูปแบบใหม่เพื่อให้สามารถใส่เข้าไปในส่วนจำเพาะได้ โดยจัดตำแหน่งข้อมูลแต่ละบิตให้ตรงกับตำแหน่งในบาวน์ดารี สแกนพาท

ซึ่งการสร้างชุดข้อมูลทดสอบจะสร้างตามหลักการเดียวกันนี้ คือตรวจสอบการเชื่อมต่อทั้งหมด กำหนดหมายเลขจุดต่อและค่าเลขฐานสอง สร้างเป็นชุดข้อมูลทดสอบ โดยออกแบบชุดข้อมูลทดสอบให้ครอบคลุมการทดสอบการเชื่อมต่อ จะออกแบบชุดข้อมูลทดสอบสำหรับ บาวน์ดารี สแกนพาท และบาวน์ดารี สแกนพาทหนึ่ง โดยที่บาวน์ดารี สแกนพาทจะมีทั้งหมด 25 จุดต่อแทนด้วยเลขฐานสองจำนวน 5 บิต ข้อมูลทดสอบ 5 เวกเตอร์ และบาวน์ดารี สแกนพาทหนึ่ง จะมีทั้งหมด 146 จุดต่อแทนด้วยเลขฐานสองจำนวน 8 บิต ข้อมูลทดสอบ 8 เวกเตอร์รวมทั้งสอง สแกนพาทมี ข้อมูลทดสอบ 13 ข้อมูล

4.2.2 ชุดข้อมูลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ[1,6,11]

การออกแบบชุดข้อมูลทดสอบ จะออกแบบเพื่อทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะใน ดาตาพาท สำหรับแกนไมโครคอนโทรลเลอร์ MEL 805X ซึ่งจะออกแบบชุดข้อมูลทดสอบแบ่งเป็นกรณีทดสอบ (test case) เพื่อใช้ทดสอบการทำงานหน่วยคำนวณและตรรกะที่มีอินพุต เป็นตัวดำเนินการหนึ่ง (operand1) และตัวดำเนินการสอง (operand2) หรือ ตัวดำเนินการหนึ่งอย่างเดียว แบ่งได้เป็น 8 ชุดการดำเนินการ

1. บวกลบ (ADD_SUB)
   ตัวดำเนินการหนึ่งบวกหรือลบกับตัวดำเนินการสอง
2. หมุนซ้าย (RL : rotate left)
   นำตัวดำเนินการหนึ่งมาทำการหมุนรอบไปทางซ้าย(บิตต่ำเลื่อนไปบิตสูง)
3. หมุนขวา (RR : rotate right)
   เหมือน RL แต่หมุนรอบ ไปทางขวา
4. แลกเปลี่ยนบิตล่าง (XCHD : exchange low order digit)
   นำเอา 4 บิตล่างตัวดำเนินการหนึ่งและตัวดำเนินการสองมาแลกเปลี่ยนกัน
5. สลับ (SWAP)
   นำเอา 4 บิตบนของตัวดำเนินการหนึ่ง สลับตำแหน่งกับ 4 บิตล่าง
6. ออร์เฉพาะ (XOR : exclusive or)
   ตัวดำเนินการหนึ่ง ออร์เฉพาะกับตัวดำเนินการสอง
7. แอนด์ (AND)
   ตัวดำเนินการหนึ่ง แอนด์กับตัวดำเนินการสอง
8. ออร์ (OR)
   ตัวดำเนินการหนึ่ง ออร์กับตัวดำเนินการสอง

จากนั้นคำตอบทั้งหมดจะผ่านอุปกรณ์รวมส่งสัญญาณ เพื่อเลือกประเภทของการดำเนินการ แล้วได้คำตอบผ่านสัญญาณ เอแอลยูเอาท์ (ALU_out) ขนาด 8 บิตซึ่งการออกแบบชุดข้อมูลทดสอบจะออกแบบตามคำสั่งต่างๆ โดยการใส่อินพุตตามกรณีทดสอบ ซึ่งถ้าจะทำการทดสอบทั้งหมดทุกอินพุตจะมีจำนวนอินพุตมากมายเช่น คำสั่ง ออร์มีอินพุตขนาด 8 บิต 2 ชุดทำให้จำนวนอินพุตทั้งหมดเป็น 2^{16} ซึ่งเป็นจำนวนที่มากเกินไปสำหรับการทดสอบ

จากจำนวนอินพุตที่มาก จึงออกแบบชุดข้อมูลทดสอบใหม่โดย บางคำสั่งสามารถแบ่งการทดสอบออกเป็นแต่ละบิตอิสระต่อกัน[6] ทำให้จำนวนอินพุตที่จะใช้ทดสอบมีจำนวนลดลงมาก โดยคำสั่งที่จะใช้การทดสอบแบบนี้ได้แก่ คำสั่ง ออร์ ,แอนด์ ,ออร์เฉพาะ ,สลับ และคำสั่ง แลกเปลี่ยนบิตล่าง โดยในส่วนของคำสั่ง หมุนขวา ,หมุนซ้าย และบวกลบ จะออกแบบชุดข้อมูลทดสอบโดยแบ่งเป็นกรณีทดสอบในแบบต่างๆ เพื่อให้ครอบคลุมตามการทำงานของแต่ละคำสั่ง และกรณีทดสอบคำสั่งต่างๆที่ใช้สร้างชุดข้อมูลทดสอบมีดังนี้

การดำเนินการออร์

ตารางที่ 4.3 กรณีการดำเนินการออร์

อินพุต		เอาท์พุต
ตัวดำเนินการหนึ่ง	ตัวดำเนินการสอง	เอแอลยูเอาท์
00000000	00000000	00000000
11111111	00000000	11111111
00000000	11111111	11111111
11111111	11111111	11111111

การดำเนินการแอนด์

ตารางที่ 4.4 กรณีทดสอบการดำเนินการแอนด์

อินพุต		เอาท์พุต
ตัวดำเนินการหนึ่ง	ตัวดำเนินการสอง	เอแอลยูเอาท์
00000000	00000000	00000000
11111111	00000000	00000000
00000000	11111111	00000000
11111111	11111111	11111111

การดำเนินการออร์เฉพาะ

ตารางที่ 4.5 กรณีทดสอบการดำเนินการออร์เฉพาะ

อินพุต		เอาท์พุต
ตัวดำเนินการหนึ่ง	ตัวดำเนินการสอง	เอแอลยูเอาท์
00000000	00000000	00000000
11111111	00000000	11111111
00000000	11111111	11111111
11111111	11111111	00000000

การดำเนินการสลับ

ตารางที่ 4.6 กรณีทดสอบการดำเนินการสลับ

อินพุต		เอาท์พุต
ตัวดำเนินการหนึ่ง บิตที่7ถึง4	ตัวดำเนินการสองบิตที่3ถึง0	เอแอลยูเอาท์
0000	0000	00000000
1111	0000	00001111
0000	1111	11110000
1111	1111	11111111

การดำเนินการแลกเปลี่ยนบิตล่าง

ตารางที่ 4.7 กรณีทดสอบการดำเนินการแลกเปลี่ยนบิตล่าง

อินพุต		เอาท์พุต
ตัวดำเนินการสอง บิตที่7ถึง4	ตัวดำเนินการหนึ่งบิตที่3ถึง0	เอแอลยูเอาท์
0000	0000	00000000
1111	0000	11110000
0000	1111	00001111
1111	1111	11111111

การดำเนินการหมุนขวา

ตารางที่ 4.8 กรณีทดสอบการดำเนินการหมุนขวา

อินพุต	เอาท์พุต	กรณีการทดสอบ
ตัวดำเนินการหนึ่ง	เอแอลยูเอาท์	
00000000	00000000	หมุนอินพุต 0 ทุกบิต
11111111	11111111	หมุนอินพุต 1 ทุกบิต
10000000	01000000	หมุนครบ 1 รอบ
01000000	00100000	
00100000	00010000	
00010000	00001000	
00001000	00000100	

อินพุต	เอาท์พุต	กรณีการทดสอบ
ตัวดำเนินการหนึ่ง	เอแอลยูเอาท์	
00000100	00000010	หมุนครบ 1 รอบ
00000010	00000001	
00000001	10000000	
00000011	10000001	
10101010	01010101	หมุนอินพุต สลับบิต

การดำเนินการหมุนซ้าย

ตารางที่ 4.9 กรณีทดสอบการดำเนินการหมุนซ้าย

อินพุต	เอาท์พุต	กรณีการทดสอบ
ตัวดำเนินการหนึ่ง	เอแอลยูเอาท์	
00000000	00000000	หมุนอินพุต 0 ทุกบิต
11111111	11111111	หมุนอินพุต 1 ทุกบิต
00000001	00000010	หมุนครบ 1 รอบ
00000010	00000100	
00000100	00001000	
00001000	00010000	
00010000	00100000	
00100000	01000000	
01000000	10000000	
10000000	00000001	
11000000	10000001	หมุนอินพุต 1 สองบิตติดกัน
10101010	01010101	หมุนอินพุต สลับบิต

การดำเนินการบวก

ตารางที่ 4.10 กรณีทดสอบการดำเนินการบวก

อินพุต		เอาท์พุต	กรณีการทดสอบ
ตัวดำเนินการหนึ่ง	ตัวดำเนินการสอง	เอแอลยูเอาท์	
00000000	00000000	00000000	ตัวตั้ง บวกด้วยศูนย์
11111111	00000000	11111111	$A + 0 = A$
00000000	10101010	10101010	ตัวตั้งเป็นศูนย์
00000000	11111111	11111111	$0 + B = B$
00000001	00000001	00000010	บวกกันแบบปกติ ค่าผลลัพธ์ ที่ได้น้อยกว่าหรือเท่ากับ 255
00000010	00000010	00000100	
00000100	00000100	00001000	
00001000	00001000	00010000	$A + B \leq 255$
00010000	00010000	00100000	
00100000	00100000	01000000	
01000000	01000000	10000000	
10000000	01111111	11111111	
10000000	10000000	00000000	บวกกันแล้ว ค่าผลลัพธ์เท่ากับ 256 เกิดล้น
11111111	00000001	00000000	
00000001	11111111	00000000	$A + B = 256$
11111111	00000010	00000001	บวกกันแล้ว ค่าผลลัพธ์มากกว่า 256 เกิดล้น
10000000	10000001	00000001	
01111111	11111111	01111110	$A + B > 256$
11111111	11111111	11111110	

การดำเนินการลบ

ตารางที่ 4.11 กรณีทดสอบการดำเนินการลบ

อินพุต		เอาท์พุต	กรณีการทดสอบ
ตัวดำเนินการหนึ่ง	ตัวดำเนินการสอง	เอแอลยูเอาท์	
00000000	00000000	00000000	ตัวลบ เป็น 0
11111111	00000000	11111111	$B - 0 = B$
00000001	00000000	00000001	ตัวตั้งมากกว่าตัวลบ ค่าผล
00000010	00000001	00000001	ลัพธ์เป็นค่าบวก
00000100	00000010	00000010	$A - B > 0; A > B$
00001000	00000100	00000100	
00010000	00001000	00001000	
00100000	00010000	00010000	
01000000	00100000	00100000	
10000000	01000000	01000000	
10000000	01111111	00000001	
11111111	00001111	11110000	
10000000	10000000	00000000	ตัวตั้งเท่ากับตัวลบ ค่าผล
10101010	10101010	00000000	ลัพธ์เป็น ศูนย์
11111111	11111111	00000000	$A - B = 0; A = B$
00000000	00000001	11111111	ตัวตั้งน้อยกว่าตัวลบ ค่าผล
00000000	11111111	00000001	ลัพธ์เป็นค่าลบ
00000001	00001111	11110010	$A - B < 0; A < B$
00001111	11110000	00001111	
01111111	11111111	10000000	

กรณีทดสอบทั้งหมดจะเก็บไว้ในแฟ้มข้อมูลเอแอลยู แล้วนำมาสร้างเป็นชุดข้อมูลทดสอบ โดยนำอินพุตมาสร้างเป็นชุดข้อมูลทดสอบ และเก็บค่าเอาท์พุตไว้เปรียบเทียบกับผลการทดสอบ ได้จำนวนข้อมูลทดสอบทั้งหมด 83 ข้อมูลทดสอบ

4.2.3 ชุดข้อมูลทดสอบฟังก์ชันการทำงานตัวถอดรหัส[1,6,11]

การออกแบบชุดข้อมูลทดสอบ ออกแบบเพื่อทดสอบฟังก์ชันการทำงาน ตัวถอดรหัสใน คอนโทรลพาทสำหรับแกนไมโครคอนโทรลเลอร์ MEL 805X โดยจะทดสอบการทำงานในส่วนของการถอดรหัสคำสั่ง ซึ่งจะทดสอบการถอดรหัสคำสั่งที่ส่งสัญญาณควบคุมการทำงานของหน่วยคำนวณและตรรกะว่าสามารถถอดรหัสได้ถูกต้องหรือไม่ โดยตรวจสอบที่สัญญาณควบคุมหน่วยคำนวณและตรรกะ เอแอลยู คอนโทรล (ALU control) โดยการออกแบบชุดข้อมูลทดสอบฟังก์ชันการทำงานของ ตัวถอดรหัส จะให้อินพุตเป็นชุดคำสั่งเพื่อให้ถอดรหัสคำสั่งออกมาตรวจสอบการทำงาน ซึ่งชุดคำสั่งที่นำมาทดสอบดังตารางที่ 4.12

ตารางที่ 4.12 ชุดคำสั่งที่ใช้ทดสอบฟังก์ชันการทำงานตัวถอดรหัส

Instruction	Opcode	ALU control
ADD A,Rn	00101000,00101001,00101010,00101011, 00101100,00101101,00101110,00101111	0000011
ADDC A,Rn	00111000,00111001,00111010,00111011, 00111100,00111101,00111110,00111111	1000011
SUBB A,Rn	10011000,10011001,10011010,10011011, 10011100,10011101,10011110,10011111	1100011
ANL A,Rn	01011000,01011001,01011010,01011011, 01011100,01011101,01011110,01011111	0011011
ORL A,Rn	01001000,01001001,01001010,01001011, 01001100,01001101,01001110,01001111	00111111
XRL A,Rn	01101000,01101001,01101010,01101011, 01101100,01101101,01101110,01101111	0010111
MOV A,Rn	11101000,11101001,11101010,11101011, 11101100,11101101,11101110,11101111	0011001
XCH A,Rn	11001000,11001001,11001010,11001011, 11001100,11001101,11001110,11001111	0011001
MOV Rn,A	11111000,11111001,11111010,11111011, 11111100,11111101,11111110,11111111	0011001

Instruction	Opcode	ALU control
DEC Rn	00011000,00011001,00011010,00011011, 00011100,00011101,00011110,00011111	0100010
INC Rn	00001000,00001001,00001010,00001011, 00001100,00001101,00001110,00001111	0000010
ADD A,@Ri	00100110,00100111	0000011
ADDC A,@Ri	00110110,00110111	1000011
SUBB A,@Ri	10010110,10010111	1100011
ANL A,@Ri	01010110,01010111	0011011
ORL A,@Ri	01000110,01000111	0011111
XRL A,@Ri	01100110,01100111	0010111
XCH A,@Ri	11000110,11000111	0011001
XCHD A,@Ri	11010110,11010111	0001111
MOV A,@Ri	11100110,11100111	0011001
MOV @Ri,A	11110110,11110111	0011001
INC @Ri	00000110,00000111	0000010
DEC @Ri	00010110,00010111	0100010
INC A	00000100	0000010
DEC A	00010100	0100010
DA A	11010100	0000000
CLR A	11100100	0011000
CPL A	11110100	0010101
RL A	00100011	0000100
RLC A	00110011	1000100
RR A	00000011	0001000
RRC A	00010011	1001000
SWAP A	11000100	0010000
CLR C	11000011	0011000

Instruction	Opcode	ALU control
SETB C	11010011	0011101
CPL C	10110011	0011101
ADD A, direct	00100101	0000011
ADDC A, direct	00110101	1000011
SUBB A, direct	10010101	1100011
ANL A, direct	01010101	0011011
ORL A, direct	01000101	0011111
XRL A, direct	01100101	0010111
MOV A, direct	11100101	0011001
XCH A, direct	11000101	0011001
ANL direct,A	01010010	0011011
ORL direct,A	01000010	0011111
XRL direct,A	01100010	0000011
MOV direct,@Ri	10000110,10000111	0011001
MOV Rn,direct	10101000,10101001,10101010,10101011, 10101100,10101101,10101110,10101111	0011001
MOV @Ri, direct	10100110,10100111	0011001
MOV direct,A	11110101	0011001
INC direct	00000101	0000010
DEC direct	00010101	0100010
ADD A,#data	00100100	0000011
ADDC A,#data	00110100	1000011
SUBB A,#data	10010100	1100011
ANL A,#data	01010100	0011011
ORL A,#data	01000100	0011111
XRL A,#data	01100100	0010111
MOV A,#data	01110100	0011001

Instruction	Opcode	ALU control
MOV Rn,#data	01111000,01111001,01111010,01111011, 01111100,01111101,01111110,01111111	0011001
MOV @Ri#data	01110110,01110111	0011001
MOV direct,Rn	10001000,10001001,10001010,10001011, 10001100,10001101,10001110,10001111	0011001
ORL C,bit	01110010	0011010
ORL C,/bit	10100000	0011010
ANL C,bit	10000010	0011010
ANL C,/bit	10110000	0011010
MOV C,bit	10100010	0011010
CPL bit	10110010	0010110
CLR bit	11000010	0011010
SET bit	11010010	0011110
MOV bit,C	10010010	0011010
MOV direct,#data	01110101	0011001
ANL direct,#data	01010011	0011011
ORL direct,#data	01000011	0011111
XRL direct,#data	01100011	0010111
MOV direct,direct	10000101	0011001
MOV DPTR,#dat16	10010000	0011001
AJMP addr11	00000001,00100001,01000001,01100001, 10000001,10100001,11000001,11100001	00000000
ACALL addr11	00010001,00110001,01010001,01110001, 10010001,10110001,11010001,11110001	0000010
LJMP addr16	00000010	0000000

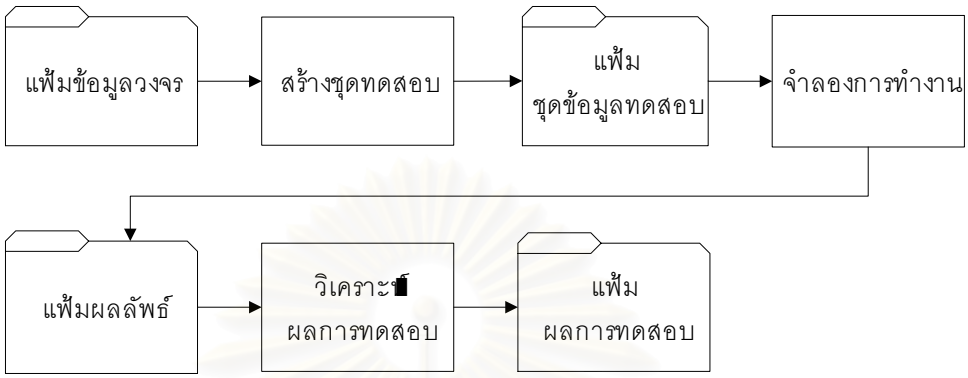
Instruction	Opcode	ALU conrol
LCALL addr16	00010010	0000010
RET or RETI	00100010,00110010	0100010
MOVX A,@DPTR	11100000	0000000
MOVX @DPTR,A	11110000	0011001
MOVX A,@Ri	11100011,11100010	0000000
MOVX @Ri,A	11110011,11110010	0011001

จะเก็บชุดคำสั่งที่ใช้ทดสอบฟังก์ชันการทำงานตัวถอดรหัสไว้ในแฟ้มข้อมูลคอนโทรล แล้วนำมาสร้างเป็นชุดข้อมูลทดสอบโดยนำ รหัสดำเนินการมาสร้างเป็นชุดข้อมูลทดสอบ และเก็บค่าเอาท์พุต เออแลยูคอนโทรลไว้เปรียบเทียบกับผลการทดสอบ

ซึ่งในการสร้างชุดข้อมูลทดสอบพบว่าไม่สามารถใส่รหัสคำสั่งเพื่อนำไปถอดรหัสได้โดยตรงเนื่องจาก ไม่สามารถควบคุมสัญญาณเพื่อให้คอนโทรลพาทรับรหัสดำเนินการจาก บัสข้อมูลจึงจำเป็นต้องแก้ไข แกนไมโครคอนโทรลเลอร์ MEL 805X ให้คอนโทรลพาทสามารถรับรหัสดำเนินการจาก บัสข้อมูลเมื่ออยู่ในสถานะตั้งใหม่ (reset state) เนื่องจากจะไม่ทำให้กระทบกับการทำงานปกติของแกนไมโครคอนโทรลเลอร์

คำสั่งทั้งหมดที่ใช้ทดสอบได้มาจากฟังก์ชันการถอดรหัสคำสั่งของ คอนโทรลพาท 218 Opcode จาก ทั้งหมด 253 Opcode โดยมีบางคำสั่งที่ไม่สามารถทดสอบได้จำนวน 37 Opcode เนื่องจากไม่สามารถควบคุมสัญญาณภายในที่ใช้ ร่วมกับการถอดรหัส จำนวน ข้อมูลทดสอบทั้งหมด 218 ข้อมูลทดสอบ ครอบคลุมการทดสอบการถอดรหัสทั้งหมด 86%

4.3 กระบวนการทดสอบ



รูปที่ 4.5 กระบวนการทดสอบ

จากรูปที่ 4.5 กระบวนการทดสอบ สามารถแบ่งได้สามกระบวนการหลักดังนี้

1. กระบวนการสร้างชุดข้อมูลทดสอบ

เริ่มจากนำข้อมูลวงจร สร้างเป็นชุดข้อมูลทดสอบเก็บไว้ในแฟ้มชุดข้อมูลทดสอบ

2. กระบวนการจำลองการทำงาน

เมื่อทำการจำลองการทำงาน จะอ่านชุดข้อมูลทดสอบจากแฟ้มแล้วทำการทดสอบ บันทึกผลลัพธ์ที่ได้เก็บไว้ในแฟ้มผลลัพธ์

3. กระบวนการวิเคราะห์ผล

วิเคราะห์ผลการทดสอบจากแฟ้มผลลัพธ์ และทำการบันทึกในแฟ้มผลการทดสอบ

กระบวนการสร้างชุดข้อมูลทดสอบและกระบวนการวิเคราะห์ผล จะใช้โปรแกรมที่พัฒนาขึ้นมาเฉพาะ โดยจะทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ทั้งที่ถูกและแกนไมโครคอนโทรลเลอร์ MEL 805X ที่ดัดแปลงให้มีความผิดพร่อง เพื่อทดสอบส่วนจำเพาะ สำหรับการทดสอบ ว่าสามารถทำงานได้อย่างถูกต้องหรือไม่ รวมทั้งโปรแกรมการสร้างและวิเคราะห์ชุดข้อมูลทดสอบมีประสิทธิภาพหรือไม่โดยจะอธิบายกระบวนการทดสอบแต่ละแบบ ดังนี้

4.3.1 กระบวนการทดสอบการเชื่อมต่อ

ในส่วนของกระบวนการทดสอบการเชื่อมต่อเริ่มจากนำแฟ้มข้อมูลวงจรมาสร้างชุดข้อมูลทดสอบตามหลักการสร้างชุดข้อมูลทดสอบในหัวข้อ 4.2.1 โดยการสร้างนั้นโปแกรมสร้างชุดทดสอบจะสร้างชุดทดสอบแล้วเขียนลงในแฟ้มชุดข้อมูลทดสอบ โดยในกระบวนการสร้างชุดข้อมูลทดสอบนี้จะอ่านข้อมูลวงจรจากแฟ้มขึ้นมาเพื่อเก็บข้อมูลการเชื่อมต่อของวงจร โดยการสร้างชุดข้อมูลทดสอบจะใช้แฟ้มข้อมูลวงจร 2 แฟ้มคือแฟ้ม interconnectBS_info และแฟ้ม interconnectBS1_info เนื่องจากในการทดสอบการเชื่อมต่อให้ครอบคลุมการทดสอบ จำเป็นต้องใช้ทั้ง 2 บาวน์ดารี สแกน พาท แต่เวลาสร้างเป็นแฟ้มข้อมูลทดสอบจะสร้างออกมาเพียงแฟ้มเดียว โดยข้อมูลในแฟ้มนั้นจะมีข้อมูลทดสอบทั้ง สองบาวน์ดารี สแกน พาทซึ่งประกอบด้วยคำสั่งเอกซ์เทสเพื่อใช้สำหรับสั่งส่วนจำเพาะ และข้อมูลทดสอบทั้งหมด 13 ข้อมูล โดยทำการจัดเรียงรูปแบบให้เข้ากับ บาวน์ดารี สแกน พาท ทั้งสอง แล้วเขียนลงแฟ้มชุดข้อมูลทดสอบ interconnect_test.vec

กระบวนการจำลองการทำงานจะใช้โปรแกรม โมเดลซิมเพื่อจำลองการทำงานโดยจะเขียนวงจรเทสเบนช์ MEL805X_intercon_tester เพื่อใช้ในการทดสอบโดยการทำงานจะอ่านข้อมูลจากแฟ้มชุดข้อมูลทดสอบทีละบรรทัดเพื่อตรวจสอบว่าเป็นคำสั่งสำหรับส่วนจำเพาะหรือเป็นข้อมูลทดสอบและดูว่าข้อมูลนั้นใช้กับ บาวน์ดารีสแกนพาทใด โดยจะตรวจสอบจากขนาดความยาวของบิตข้อมูล ถ้าพบว่าคำสั่งก็จะ แสกนข้อมูลคำสั่งนั้นให้กับส่วนจำเพาะ แต่ถ้าเป็นข้อมูลทดสอบก็จะสแกนเข้ามูลทดสอบเข้าไปเพื่อทดสอบ และทำการเขียนผลลัพธ์ที่ได้จากสแกนเอาท์ เขียนลงในแฟ้มผลลัพธ์ interconnect_result.vec เพื่อนำไปใช้ในการวิเคราะห์ผลต่อไป

การวิเคราะห์ผลการทดสอบจะใช้โปรแกรมวิเคราะห์ผลการทดสอบ โดยการอ่านแฟ้มผลลัพธ์มาทำการวิเคราะห์ผลโดยใช้ข้อมูลวงจรที่ได้จากการสร้างชุดข้อมูลทดสอบมาช่วยในการวิเคราะห์โดยจะทำการตรวจสอบหาข้อผิดพลาดของการเชื่อมต่อตามหัวข้อ 4.4.1 และแสดงผลการวิเคราะห์ที่โปรแกรมและเขียนผลการวิเคราะห์เก็บไว้ในแฟ้มผลลัพธ์ interconnection_log

4.3.2 กระบวนการทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ

กระบวนการทดสอบการเชื่อมต่อเริ่มจากนำแฟ้มข้อมูลเอแอลยู alu_info ซึ่งเป็นแฟ้มข้อมูลที่เก็บชุดคำสั่งสำหรับทดสอบ หน่วยคำนวณและตรรกะ รวมทั้งอินพุต และเอาท์พุตที่ถูกต้อง โดยมีข้อมูลเหมือนกรณีทดสอบในหลักการสร้างชุดข้อมูลทดสอบในหัวข้อ 4.2.2 นำมาสร้างชุดข้อมูลทดสอบ การทดสอบจะใช้เพียง บาวน์ดารี สแกน พาท เท่านั้นโดยการสร้างนั้นโปรแกรม

สร้างชุดทดสอบจะอ่านอินพุต และคำสั่งควบคุมการดำเนินการ ซึ่งจะทดสอบทั้งหมด 8 การดำเนินการรวมแล้วทั้งหมดทุกกรณีทดสอบจะมี 83 ข้อมูลทดสอบ ซึ่งชุดข้อมูลทดสอบประกอบด้วยคำสั่งเอกซ์เทส เพื่อใช้สำหรับสั่งส่วนจำเพาะ และข้อมูลทดสอบทั้งหมด 83 ข้อมูลมาจัดเรียงรูปแบบให้เข้ากับ บาวน์ดารี สแกน พาท และในส่วนของ เอาท์พุตจะเก็บไว้เพื่อใช้ในการวิเคราะห์ผล เมื่อสร้างชุดทดสอบแล้วเขียนลงในแฟ้มชุดข้อมูลทดสอบ alu_test.vec

กระบวนการจำลองการทำงานจะเหมือนกับกระบวนการจำลองการทำงานในหัวข้อ 4.3.1 เพียงแต่ใช้วงจรเทสเบนช์ MEL805X_alu_tester และทดสอบเพียง บาวน์ดารี สแกน พาท และผลลัพธ์ที่ได้จากสแกนเอาท์ เขียนลงในแฟ้มผลลัพธ์ alu_result.vec เพื่อนำไปใช้ในการวิเคราะห์ผลต่อไป

การวิเคราะห์ผลการทดสอบจะใช้โปรแกรมวิเคราะห์ผลการทดสอบ โดยการอ่านแฟ้มผลลัพธ์มาทำการวิเคราะห์ผลโดยใช้ข้อมูลเอาท์พุตที่ได้จากการสร้างชุดข้อมูลทดสอบมาช่วยในการวิเคราะห์โดยจะทำการตรวจสอบหาข้อผิดพลาดฟังก์ชันการทำงานหน่วยคำนวณและตรรกะตามหัวข้อ 4.4.2 และแสดงผลการวิเคราะห์ที่โปรแกรมและเขียนผลการวิเคราะห์เก็บไว้ในแฟ้มผลลัพธ์ alu_log

4.3.3 กระบวนการทดสอบฟังก์ชันการทำงานตัวถอดรหัส

กระบวนการทดสอบการเชื่อมต่อเริ่มจากนำแฟ้มข้อมูลคอนโทรล control_info ซึ่งเป็นแฟ้มข้อมูลเก็บชุดคำสั่งสำหรับทดสอบ รหัสดำเนิการ และเอาท์พุตของการถอดรหัสที่ถูกต้อง โดยมีข้อมูลเหมือนชุดข้อมูลทดสอบฟังก์ชันการทำงานตัวถอดรหัสในหลักการสร้างชุดข้อมูลทดสอบในหัวข้อ 4.2.3 นำมาสร้างชุดข้อมูลทดสอบ การทดสอบจะใช้เพียง บาวน์ดารี สแกน พาทหนึ่ง เท่านั้น โดยรหัสดำเนินการจำถูกใส่ไว้ในดาตาบัสเพื่อใช้เป็นอินพุตในการถอดรหัสคำสั่ง การสร้างนั้นโปรแกรมสร้างชุดทดสอบจะอ่านรหัสดำเนินการ และเอาท์พุตของการถอดรหัส ซึ่งจะทดสอบทั้งหมด 218 รหัสดำเนินการ ซึ่งชุดข้อมูลทดสอบประกอบด้วยคำสั่งเอกซ์เทส เพื่อใช้สำหรับสั่งส่วนจำเพาะ และข้อมูลทดสอบทั้งหมด 218 ข้อมูลมาจัดเรียงรูปแบบให้เข้ากับ บาวน์ดารี สแกน พาทหนึ่ง และในส่วนของ เอาท์พุตของการถอดรหัสจะเก็บไว้เพื่อใช้ในการวิเคราะห์ผล เมื่อสร้างชุดทดสอบแล้วเขียนลงในแฟ้มชุดข้อมูลทดสอบ control_test.vec

กระบวนการจำลองการทำงานจะเหมือนกับกระบวนการจำลองการทำงานในหัวข้อ 4.3.1 เพียงแต่ใช้วงจรเทสเบนช์ MEL805X_control_tester และทดสอบเพียง บาวน์ดารี สแกน พาทหนึ่ง และผลลัพธ์ที่ได้จากสแกนเอาท์ เขียนลงในแฟ้มผลลัพธ์ control_result.vec เพื่อนำไปใช้ในการวิเคราะห์ผลต่อไป

การวิเคราะห์ผลการทดสอบจะใช้โปรแกรมวิเคราะห์ผลการทดสอบ โดยการอ่านแฟ้มผลลัพธ์มาทำการวิเคราะห์ผลโดยใช้ข้อมูลเอาท์พุตที่ได้จากการสร้างชุดข้อมูลทดสอบมาช่วยในการวิเคราะห์โดยจะทำการตรวจสอบหาข้อผิดพลาดของฟังก์ชันการถอรหัสคำสั่ง ตามหัวข้อ 4.4.3 และแสดงผลการวิเคราะห์ที่โปรแกรมและเขียนผลการวิเคราะห์เก็บไว้ในแฟ้มผลลัพธ์ control_log

4.4 การตรวจสอบหาความผิดพร่อง

การตรวจสอบหาความผิดพร่องจะจะแบ่งการตรวจสอบตามลักษณะการทดสอบดังนี้

4.4.1 การตรวจสอบการเชื่อมต่อ

ในส่วนของการตรวจสอบจะตรวจสอบตามแบบจำลองความผิดพร่อง คือจะทดสอบส่วนประกอบซึ่งใส่ไว้ไม่เหมาะสม การลัดวงจรและการเปิด ซึ่งโปรแกรมสามารถวิเคราะห์หาความผิดพร่องในการเชื่อมต่อ และแสดงผลการทดสอบ และบันทึกลงแฟ้มผลการทดสอบ และในส่วนของความผิดพร่องในการลัดวงจรระหว่างสายสัญญาณทำให้เกิดความผิดพร่องประสานโปรแกรมโมเดลซิม จะทำการตรวจสอบในขั้นตอนแปลภาษา

การตรวจสอบส่วนประกอบซึ่งใส่ไว้ไม่เหมาะสม จะตรวจสอบการเชื่อมต่อของบาวน์ดารี สแกน เรจิสเตอร์ว่าเชื่อมต่อถูกต้องหรือไม่ในแต่ละ บาวน์ดารี สแกน พาท โดยสามารถตรวจสอบบาวน์ดารี สแกน พาทขาดหรือไม่เหมาะสมจากการสแกน คำสั่งในเรจิสเตอร์คำสั่งออกมาตรวจสอบถ้าเกิดการเชื่อมต่อของ บาวน์ดารี สแกน พาทที่ไม่เหมาะสมหรือบาวน์ดารี สแกน พาทเกิดการขาด ผลลัพธ์ที่สแกนเอาท์ออกมาจะได้ค่าที่ผิดพลาดเช่นได้ค่า ‘U’ออกมาแทนที่จะเป็นค่า ‘0’หรือ‘1’ และยังทราบว่าเกิดความผิดพลาดที่ บาวน์ดารี สแกน เรจิสเตอร์ใด

และการตรวจสอบส่วนประกอบซึ่งใส่ไว้ไม่เหมาะสม ในการเชื่อมต่อจะตรวจสอบข้อมูลที่ขาสัญญาณอินพุตในแต่ละชิ้นส่วนได้รับมา เพื่อเปรียบเทียบกับข้อมูลที่มีใน ข้อมูลการเชื่อมต่อตรงกันหรือไม่ ถ้าการเชื่อมต่อถูกต้อง ค่าตรรกะที่ขาสัญญาณอินพุตนั้นจะต้องได้รับค่า เลขฐานสองเหมือนกับที่ขาเอาท์พุตที่เชื่อมต่อกันที่จุดต่อเดียวกันขับออกมา ถ้าได้รับข้อมูลไม่ตรงกัน แสดงว่าเกิดข้อผิดพร่องขึ้น และสามารถบอกได้ว่าเกิดข้อผิดพร่องขึ้นที่ขาสัญญาณใด

การตรวจสอบความผิดพร่องการลัดวงจร จะตรวจสอบการลัดวงจรกับกราวด์โดยการตรวจสอบข้อมูลที่ขาสัญญาณอินพุตในแต่ละชิ้นส่วนได้รับมา ถ้ามีค่าเลขฐานสองเป็น ‘0’ หมดแสดงว่าเกิด สตัคเอต 0 เกิดการลัดวงจรกับกราวด์ และตรวจสอบการลัดวงจรกับพาวเวอร์โดยการตรวจสอบข้อมูลที่ขาสัญญาณอินพุตในแต่ละชิ้นส่วนได้รับมา ถ้ามีค่าเลขฐานสองเป็น ‘1’ หมดแสดงว่าเกิด สตัคเอต 1 เกิดการลดวงจรกับพาวเวอร์

การตรวจสอบความผิดพร่องการเปิด โดยจะตรวจสอบการเชื่อมต่อภายในจากการเปรียบเทียบข้อมูลทดสอบโดยสามารถตรวจสอบหาข้อผิดพร่องที่เกิดขึ้นได้จาก ขั้นตอนวิธีการเปิด (open algorithm)[9] ดังนี้

1. ถ้ามีตัวรับมากกว่าหรือเท่ากับ 2 ตัวและตัวรับทั้งหมดเกิด สตัค แล้วเกิดการเปิดที่ตัวขับ
2. ถ้ามีตัวรับมากกว่าหรือเท่ากับ 2 ตัวและตัวรับอย่างน้อย 1 ตัวได้รับข้อมูลถูกต้องแล้ว เกิดการเปิดที่ตัวรับที่ได้รับข้อมูลทดสอบที่ผิดพลาด
3. ถ้ามีเพียงตัวขับและตัวรับและตัวรับได้รับข้อมูลทดสอบที่ผิดพลาดแล้ว อาจเกิดการเปิดที่ ตัวขับหรือตัวรับ

จากขั้นตอนวิธีการเปิดทำให้สามารถตรวจสอบหา ความผิดพร่องที่เกิดขึ้นได้จากการเปิดได้รวมทั้งบอกตำแหน่งของความผิดพร่องการเปิดที่เกิดขึ้น

4.4.2 การตรวจสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ[1,6]

การตรวจสอบจะทำการตรวจสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะในส่วน ดาตาพาท ซึ่งจะใช้ บาวน์ดารี สแกน พาทในการใส่อินพุต และคำสั่งควบคุมการดำเนินการเพื่อทดสอบฟังก์ชันการทำงานของหน่วยคำนวณและตรรกะ ตามแบบจำลองความผิดพร่องเกี่ยวกับหน้าที่โดยตรวจสอบเอาท์พุตของเอแอลยูเอาท์ ซึ่งเป็นเอาท์พุตที่ได้จากการดำเนินการ โดยนำมาเปรียบเทียบกับเอาท์พุตในกรณีทดสอบ ที่ออกแบบไว้ในชุดข้อมูลทดสอบในหัวข้อ 4.2.2 เปรียบ

เทียบผลลัพธ์ที่ได้ตรงกันหรือไม่ในแต่ละกรณี โดยมีสมมุติฐานตามนิยามการทดสอบหัวข้อ 2.1.5 คือรู้อินพุตและฟังก์ชันการทำงานของวงจร ถ้าผลลัพธ์ออกมาไม่ตรงกัน แสดงว่าเกิดความผิดพร่อง ฟังก์ชันการทำงานไม่ถูกต้อง และทำให้สามารถทราบคำสั่งที่ผิดพลาดได้ ซึ่งกรณีทดสอบทั้งหมดจะทำการทดสอบครอบคลุมทั้ง 8 การดำเนินการ

4.4.3 การตรวจสอบฟังก์ชันการทำงานตัวถอดหัส[1,6]

การตรวจสอบจะตรวจสอบฟังก์ชันการทำงานตัวถอดหัสในคอนโทรลพาท ซึ่งจะใช้บาวน์ดารี สแกน พาทหนึ่ง ในการใส่ รหัสดำเนินการ เพื่อทดสอบฟังก์ชันการทำงานตัวถอดหัสคำสั่ง ตามแบบจำลองความผิดพร่องเกี่ยวกับหน้าที่โดยตรวจสอบเอาท์พุตของการถอดรหัส ซึ่งเป็นเอาท์พุตที่ได้จากการถอดหัสคำสั่ง โดยนำมาเปรียบเทียบกับเอาท์พุตของการถอดรหัสที่ถูกต้องตามที่ออกแบบไว้ในชุดข้อมูลทดสอบในหัวข้อ 4.2.3 เปรียบเทียบผลลัพธ์ที่ได้ตรงกันหรือไม่ในแต่ละกรณี โดยมีสมมุติฐานตามนิยามการทดสอบหัวข้อ 2.1.5 คือรู้อินพุตและฟังก์ชันการทำงานของวงจร ถ้าผลลัพธ์ออกมาไม่ตรงกัน แสดงว่าเกิดความผิดพร่อง ฟังก์ชันการทำงานไม่ถูกต้อง และทำให้สามารถทราบคำสั่งที่ผิดพลาดได้ ซึ่งจะทดสอบการถอดรหัสทั้งหมด 218 รหัสดำเนินการ

บทที่ 5

ผลการทดสอบและวิเคราะห์ ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X

การทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน เป็นงานที่สำคัญ เพื่อเป็นการทดสอบความสามารถของส่วนจำเพาะว่าสามารถทดสอบแกนไมโครคอนโทรลเลอร์ MEL 805X ได้หรือไม่ โดยจะทดสอบโดยวิธีการจำลองการทำงาน ในการทดสอบจำเป็นต้องเขียนชุดข้อมูลทดสอบ เพื่อใช้ทดสอบโดยในงานวิจัยนี้จะใช้โปรแกรมสร้างชุดข้อมูลทดสอบในการสร้างชุดข้อมูลทดสอบ ตามหัวข้อ 4.2โดยจะสร้างชุดข้อมูลทดสอบทั้งหมด สามชุดข้อมูลทดสอบเพื่อการทดสอบแต่ละประเภทเก็บไว้เป็นแฟ้มชุดข้อมูลทดสอบ และในการทดสอบด้วยวิธีจำลองการทำงาน จะเขียนการทดสอบแบบเทสเบนช์ [3,4] ด้วยภาษาวีเอชดีแอล และจำลองการทำงานในโปรแกรมโมเดลซิม

ผลการทดสอบส่วนส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน ได้มาจากการจำลองการทำงาน ซึ่งได้แฟ้มผลลัพธ์และนำมาวิเคราะห์ผลด้วยโปรแกรมวิเคราะห์ผล โดยการวิเคราะห์จะใช้ข้อมูลจากชุดข้อมูลทดสอบตามข้อที่ 5.1 เพื่อวิเคราะห์ผลการทดสอบส่วนจำเพาะ ตามการทดสอบทั้งสามแบบ ตามหัวข้อ 5.2 โดยผลการวิเคราะห์จะแสดงผลในโปรแกรมรวมทั้งเก็บเป็นแฟ้มบันทึกเข้าออก (log file) จะแบ่งเป็นการวิเคราะห์การทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ที่ถูกต้องและแกนที่ดัดแปลงให้ผิดพลาดโดยผลการทดสอบทั้งหมดสามารถอ่านได้ที่ ภาคผนวก ก.

5.1 ชุดข้อมูลทดสอบส่วนจำเพาะ

สำหรับชุดข้อมูลทดสอบที่จะนำมาใช้ทดสอบนั้น ดูจากหัวข้อ 4.2 โดยได้จากการสร้างขึ้นด้วยโปรแกรมสร้างชุดข้อมูลทดสอบซึ่งจะได้แฟ้มชุดข้อมูลทดสอบเพื่อนำไปใช้ทดสอบส่วนจำเพาะทั้งหมด 3 แฟ้มแยกแต่ละประเภทการทดสอบ

5.1.1 แฟ้มชุดข้อมูลทดสอบการเชื่อมต่อ

แฟ้มข้อมูลทดสอบสร้างโดยใช้ข้อมูลจากแฟ้ม interconnectBS_info และแฟ้ม interconnectBS1_info ซึ่งเป็นแฟ้มข้อมูลที่อธิบายการเชื่อมต่อของแกนไมโครคอนโทรลเลอร์ MEL 805X

5.1.2 แฟ้มชุดข้อมูลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ

แฟ้มข้อมูลทดสอบสร้างโดยใช้ข้อมูลจากแฟ้ม alu_info ซึ่งเป็นแฟ้มข้อมูลที่เก็บชุดคำสั่งสำหรับทดสอบ หน่วยคำนวณและตรรกะ รวมทั้งอินพุต และเอาท์พุตที่ถูกต้อง

5.1.3 แฟ้มชุดข้อมูลทดสอบฟังก์ชันการทำงานตัวถอดรหัส

แฟ้มข้อมูลทดสอบสร้างโดยใช้ข้อมูลจากแฟ้ม control_info ซึ่งเป็นแฟ้มข้อมูลที่เก็บชุดคำสั่งสำหรับทดสอบ การถอดรหัสคำสั่ง ของตัวถอดรหัส และเอาท์พุตของการถอดรหัสที่ถูกต้อง

5.2 การทดสอบโดยใช้การจำลองการทำงาน

การทดสอบโดยใช้การจำลองการทำงาน[3,8] โดยจะทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X เพื่อ

5.2.1 ทดสอบการเชื่อมต่อ

ใช้วงจรเทสเบนช์ MEL805X_intercon_tester ในการทดสอบและเขียนผลการทดสอบลงแฟ้มผลลัพธ์ interconnect_result.vec

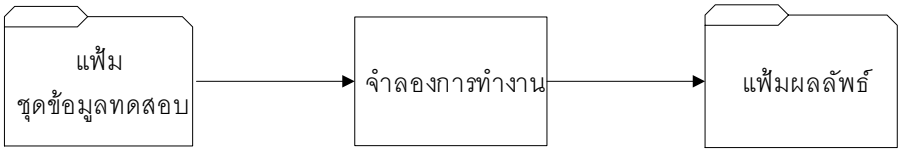
5.2.2 ทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ

ใช้วงจรเทสเบนช์ MEL805X_alu_tester ในการทดสอบและเขียนผลการทดสอบลงแฟ้มผลลัพธ์ alu_result.vec

5.2.3 ทดสอบฟังก์ชันการทำงานตัวถอดรหัส

ใช้วงจรเทสเบนช์ MEL805X_control_tester ในการทดสอบและเขียนผลการทดสอบลงแฟ้มผลลัพธ์ control_result.vec

ซึ่งการจำลองการทำงานเพื่อทดสอบแต่ละแบบนั้น จะจำลองการทำงานแยกกันตามวงจรที่ออกแบบมาเฉพาะ ลักษณะการทำงานจะคล้ายกัน คืออ่านชุดข้อมูลทดสอบจากแฟ้มชุดข้อมูลทดสอบแล้วป้อนข้อมูลทดสอบให้ส่วนจำเพาะ แบบอนุกรมเพื่อทดสอบและผลลัพธ์ที่ได้ออกมาเขียนลงแฟ้มเพื่อนำไปใช้วิเคราะห์ต่อไปดังรูปที่ 5.1



รูปที่ 5.1 การจำลองการทำงาน

โดยการทดสอบจะทดสอบทั้ง แกนไมโครคอนโทรลเลอร์ MEL 805X ที่ถูกต้องและแกนที่ดัดแปลงให้ผิดพลาด เพื่อทดสอบความสามารถของ ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X

5.3 ผลการวิเคราะห์การทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X

ผลการวิเคราะห์การทดสอบส่วนจำเพาะ ดังนี้

5.3.1 ผลการวิเคราะห์ผลทดสอบการเชื่อมต่อ

ผลการวิเคราะห์ผลทดสอบการเชื่อมต่อผ่านการทดสอบทั้งหมดตามหัวข้อ 4.4.1 ไม่พบความผิดพร่อง โดยการวิเคราะห์ เริ่มจากการตรวจสอบการเชื่อมต่อของ บาวน์ดารี สแกน เรจิสเตอร์ ซึ่งถ้า บาวน์ดารี สแกนพาท ขาด หรือเชื่อมต่อผิด[9] ซึ่งจะวิเคราะห์ได้จากเรจิสเตอร์คำสั่ง ซึ่งถ้าเกิดความผิดพร่อง ค่าที่ได้จะผิด และสามารถหาตำแหน่งที่ผิดได้ว่าเกิดการขาดที่ บาวน์ดารี สแกน เรจิสเตอร์ใด จากนั้นทำการวิเคราะห์ส่วนประกอบซึ่งใส่ไว้ไม่ถูกต้อง และการวิเคราะห์ความผิดพร่องการเปิด และการลัดวงจรซึ่งการวิเคราะห์ทั้งหมด จะวิเคราะห์ทั้ง บาวน์ดารี สแกนพาทและบาวน์ดารี สแกนพาทหนึ่ง เพื่อให้ครอบคลุมทดสอบการเชื่อมต่อทั้งหมด

5.3.2 ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ

ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะตามหัวข้อ 4.4.2 ผ่านการทดสอบทั้งหมดไม่พบข้อผิดพร่อง โดยการวิเคราะห์ฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ ตามชุดข้อมูลทดสอบ ซึ่งการวิเคราะห์ทั้งหมด 8 การดำเนินการตามกรณีทดสอบตามหัวข้อ 4.2.2

5.3.3 ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานตัวถอดรหัส [1]

ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานตัวถอดรหัส ตามหัวข้อ 4.4.3 ผ่านการทดสอบทั้งหมดไม่พบข้อผิดพร่อง โดยการวิเคราะห์ฟังก์ชันการทำงานตัวถอดรหัส ตามชุดข้อมูลทดสอบ ตามหัวข้อ 4.2.3

5.4 ผลการวิเคราะห์การทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ที่ดัดแปลงให้ผิดพลาด

ผลการวิเคราะห์การทดสอบส่วนจำเพาะ จะต้องสามารถวิเคราะห์ผลหาความผิดพร่องออกมาได้โดยการดัดแปลงแกนไมโครคอนโทรลเลอร์ MEL 805X ให้ผิดพลาดในแบบต่างๆดังนี้

5.4.1 ผลการวิเคราะห์ผลทดสอบการเชื่อมต่อ

การดัดแปลงแกนไมโครคอนโทรลเลอร์ MEL 805X ให้ผิดพลาดแล้ววิเคราะห์ดังนี้

1. ดัดแปลงให้บาวน์ดารี สแกนพาท ขาดที่บาวน์ดารี สแกน เรจิสเตอร์ BS_INT_con

2. ดัดแปลงให้ บาวน์ดารี สแกนพาทหนึ่ง ขาดที่บาวน์ดารี สแกน เรจิสเตอร์ BS_control

3. ดัดแปลงให้การเชื่อมต่อผิดพลาดโดยการเปิดขาสัญญาณ INT_CON_T_7 ที่ตัวขับสัญญาณ และเปิดขาสัญญาณ INT_CON_T_19 ที่ตัวรับสัญญาณที่บาวน์ดารีสแกน เรจิสเตอร์ BS_INT_con

4. ดัดแปลงให้การเชื่อมต่อผิดพลาดที่ขาสัญญาณ CONTROL_INT0_Filtered ต่อเข้ากับขาสัญญาณ DATA_PATH _INT1_Filtered

5. ดัดแปลงให้การลัดวงจรกับกราวด์ที่ขาสัญญาณ BS_INT_CON_T_20 และ ลัดวงจรกับพาวเวอร์ที่ขาสัญญาณ DATA_PATH _RESET

ผลการวิเคราะห์ผลทดสอบการเชื่อมต่อ ของการดัดแปลงแกนไมโครคอนโทรลเลอร์ MEL 805X ให้เชื่อมต่อผิดพลาด สามารถวิเคราะห์การลัดวงจร การขาดหรือการเชื่อมต่อที่ไม่เหมาะสม โดยทดสอบการเชื่อมต่อของสแกนพาททั้งสองเส้น รวมทั้งบอกตำแหน่งของ บาวน์ดารี สแกน เรจิสเตอร์ที่ผิดพร่องโดยผลการวิเคราะห์จะเปรียบเทียบจากข้อมูลในเรจิสเตอร์คำสั่งที่เลื่อนออกมาเพื่อนำไปตรวจสอบว่าเกิดความผิดพร่องที่บาวน์ดารี สแกน เรจิสเตอร์ใด

ในส่วนของจุดที่เชื่อมต่อที่ไม่เหมาะสม ก็สามารถที่จะวิเคราะห์ผลออกมาเพื่อบอก จุดที่ผิดพร่องได้โดยบอกวาเกิดการเชื่อมต่อที่ไม่เหมาะสมที่จุดเชื่อมต่อใดและขาสัญญาณใดบ้าง

จุดที่เกิดการเปิดก็สามารถที่จะวิเคราะห์ผลออกมาเพื่อบอก มีการเปิดที่จุดเชื่อมต่อใดและขาสัญญาณใดบ้าง และสามารถวิเคราะห์ได้ว่าเกิดการเปิดจากตัวขับหรือเกิดการเปิดที่ตัวรับ

วิเคราะห์การลัดวงจรกับกราวด์ หรือลัดวงจรกับพาวเวอร์เกิดขึ้นที่จุดเชื่อมต่อใดและขาสัญญาณใดบ้างซึ่งสามารถหาความผิดพร่อง ซึ่งการวิเคราะห์ทั้งหมดจะวิเคราะห์ตามหัวข้อ 4.4.1 ได้ครอบคลุมตามข้อกำหนดการเชื่อมต่อของวงจร

5.4.2 ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ

การดัดแปลงแกนไมโครคอนโทรลเลอร์ MEL 805X ให้ฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ ทำงานผิดพลาดดังนี้

1. การทำงานการดำเนินการออร์ โดยให้ บิตที่ 4 ถึงบิตที่ 7 ไม่ทำการดำเนินการออร์
2. การทำงานการดำเนินการแอนด์ โดยให้ ค่าผลลัพธ์ออกมาเป็น 11111111 ตลอด
3. การทำงานการดำเนินการออร์เฉพาะ โดยให้การทำงานเป็นการดำเนินการออร์แทน
4. การทำงานการดำเนินการสลับ โดยไม่ดำเนินการสลับ ค่าในตัวดำเนินการหนึ่ง
5. การทำงานการดำเนินการแลกเปลี่ยนบิตล่าง โดยให้เป็นค่าตัวดำเนินการสองแทน
6. การทำงานการดำเนินการหมุนขวา โดยให้การทำงานเป็นการดำเนินการหมุนซ้ายแทน
7. การทำงานการดำเนินการหมุนซ้ายโดยให้การทำงานเป็นการดำเนินการหมุนขวาแทน
8. การทำงานการดำเนินการบวกโดยให้การทำงานในส่วนของวงจรบวก ผิดพลาด
9. การทำงานการดำเนินการลบโดยแก้ไขวงจรส่วนเติมเต็มหนึ่ง (one's complement)

ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะที่ถูกดัดแปลง ซึ่งผลการวิเคราะห์ สามารถหาความผิดพร่อง ว่าเกิดขึ้นที่การดำเนินการใด ซึ่งการวิเคราะห์ทั้งหมดจะวิเคราะห์ตามหัวข้อ 4.4.2 ซึ่งสามารถหาความผิดพร่องได้ครอบคลุมตามกรณีทดสอบ

5.4.3 ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานตัวถอดรหัส

การดัดแปลงให้ แกนไมโครคอนโทรลเลอร์ MEL 805X ถอดรหัสคำสั่งให้ผิดพลาด โดยการแบ่งการทดสอบออกเป็นชุดของรหัสดำเนินการ ที่ทำงานลักษณะเดียวกันทั้งหมด 5 ชุด[11]โดย แบ่งการทดสอบออกเป็น 5 ชุดทดสอบ โดยจะแก้ไขฟังก์ชันการถอดรหัส ของแต่ละชุดรหัสดำเนินการให้ทำการถอดรหัสผิดพลาดทุกรหัสดำเนินการแล้วทดสอบทีละชุด ดังนี้

ชุดที่ 1 รหัสดำเนินการที่เกี่ยวกับ การดำเนินการทางคณิตศาสตร์ (arithmetic operation)

คำสั่ง	รหัสดำเนินการ	เอแอลยู คอนโทรล
ADD A,Rn	00101000	0000011
ADD A,Rn	00101001	0000011
ADD A,Rn	00101010	0000011
ADD A,Rn	00101011	0000011
ADD A,Rn	00101100	0000011
ADD A,Rn	00101101	0000011
ADD A,Rn	00101110	0000011
ADD A,Rn	00101111	0000011
ADD A,@Ri	00100110	0000011
ADD A,@Ri	00100111	0000011
ADD A,di	00100101	0000011
ADD A,#data	00100100	0000011
ADDC A,Rn	00111000	1000011
ADDC A,Rn	00111001	1000011
ADDC A,Rn	00111010	1000011
ADDC A,Rn	00111011	1000011
ADDC A,Rn	00111100	1000011
ADDC A,Rn	00111101	1000011
ADDC A,Rn	00111110	1000011
ADDC A,Rn	00111111	1000011
ADDC A,@Ri	00110110	1000011
ADDC A,@Ri	00110111	1000011

ADDC A,di	00110101	1000011
ADDC A,#data	00110100	1000011
SUBB A,Rn	10011000	1100011
SUBB A,Rn	10011001	1100011
SUBB A,Rn	10011010	1100011
SUBB A,Rn	10011011	1100011
SUBB A,Rn	10011100	1100011
SUBB A,Rn	10011101	1100011
SUBB A,Rn	10011110	1100011
SUBB A,Rn	10011111	1100011
SUBB A,@Ri	10010110	1100011
SUBB A,@Ri	10010111	1100011
SUBB A,di	10010101	1100011
SUBB A,#data	10010100	1100011
INC A	00000100	0000010
INC Rn	00001000	0000010
INC Rn	00001001	0000010
INC Rn	00001010	0000010
INC Rn	00001011	0000010
INC Rn	00001100	0000010
INC Rn	00001101	0000010
INC Rn	00001110	0000010
INC Rn	00001111	0000010
INC @Ri	00000110	0000010
INC @Ri	00000111	0000010
INC di	00000101	0000010
DEC Rn	00011000	0100010
DEC Rn	00011001	0100010
DEC Rn	00011010	0100010
DEC Rn	00011011	0100010

DEC Rn	00011100	0100010
DEC Rn	00011101	0100010
DEC Rn	00011110	0100010
DEC Rn	00011111	0100010
DEC @Ri	00010110	0100010
DEC @Ri	00010111	0100010
DEC A	00010100	0100010
DEC di	00010101	0100010
DA A	11010100	0000000

ชุดที่ 2 รหัสดำเนินการที่เกี่ยวกับ การดำเนินการตรรกะ (logical operation)

คำสั่ง	รหัสดำเนินการ	เอแอลยู คอนโทรล
ORL A,Rn	01001000	0011111
ORL A,Rn	01001001	0011111
ORL A,Rn	01001010	0011111
ORL A,Rn	01001010	0011111
ORL A,Rn	01001011	0011111
ORL A,Rn	01001100	0011111
ORL A,Rn	01001101	0011111
ORL A,Rn	01001110	0011111
ORL A,Rn	01001111	0011111
ORL A,@Ri	01000110	0011111
ORL A,@Ri	01000111	0011111
ORL A,di	01000101	0011111
ORL di,A	01000010	0011111
ORL A,#data	01000100	0011111
ORL C,bit	01110010	0011010
ORL C,/bit	10100000	0011010
ORL di,#data	01000011	0011111
ANL A,Rn	01011000	0011011

ANL A,Rn	01011001	0011011
ANL A,Rn	01011010	0011011
ANL A,Rn	01011011	0011011
ANL A,Rn	01011100	0011011
ANL A,Rn	01011101	0011011
ANL A,Rn	01011110	0011011
ANL A,Rn	01011111	0011011
ANL A,@Ri	01010110	0011011
ANL A,@Ri	01010111	0011011
ANL A,di	01010101	0011011
ANL A,#data	01010100	0011011
ANL di,#data	01010011	0011011
ANL di,A	01010010	0011011
XRL A,Rn	01101000	0010111
XRL A,Rn	01101001	0010111
XRL A,Rn	01101010	0010111
XRL A,Rn	01101011	0010111
XRL A,Rn	01101100	0010111
XRL A,Rn	01101101	0010111
XRL A,Rn	01101110	0010111
XRL A,Rn	01101111	0010111
XRL A,@Ri	01100110	0010111
XRL A,@Ri	01100111	0010111
XRL A,di	01100101	0010111
XRL di,A	01100010	0000011
XRL A,#data	01100100	0010111
XRL di,#data	01100011	0010111
CLR A	11100100	0011000
CPL A	11110100	0010101
RL A	00100011	0000100

RLC A	00110011	1000100
RR A	00000011	0001000
RRC A	00010011	1001000
SWAP A	11000100	0010000

ชุดที่ 3 รหัสดำเนินการที่เกี่ยวกับ การถ่ายโอนข้อมูล (data transfer)

คำสั่ง	รหัสดำเนินการ	เอแอลยู คอนโทรล
MOV A,Rn	11101000	0011001
MOV A,Rn	11101001	0011001
MOV A,Rn	11101010	0011001
MOV A,Rn	11101011	0011001
MOV A,Rn	11101100	0011001
MOV A,Rn	11101101	0011001
MOV A,Rn	11101110	0011001
MOV A,Rn	11101111	0011001
MOV Rn,A	11111000	0011001
MOV Rn,A	11111001	0011001
MOV Rn,A	11111010	0011001
MOV Rn,A	11111011	0011001
MOV Rn,A	11111100	0011001
MOV Rn,A	11111101	0011001
MOV Rn,A	11111110	0011001
MOV Rn,A	11111111	0011001
MOV A,@Ri	11100110	0011001
MOV A,@Ri	11100111	0011001
MOV @Ri,A	11110110	0011001
MOV @Ri,A	11110111	0011001
MOV A,di	11100101	0011001
MOV di,@Ri	10000110	0011001
MOV di,@Ri	10000111	0011001

MOV Rn,di	10101000	0011001
MOV Rn,di	10101001	0011001
MOV Rn,di	10101010	0011001
MOV Rn,di	10101011	0011001
MOV Rn,di	10101100	0011001
MOV Rn,di	10101101	0011001
MOV Rn,di	10101110	0011001
MOV Rn,di	10101111	0011001
MOV @Ri,di	10100110	0011001
MOV @Ri,di	10100111	0011001
MOV di,A	11110101	0011001
MOV A,#data	01110100	0011001
MOV Rn,#data	01111000	0011001
MOV Rn,#data	01111001	0011001
MOV Rn,#data	01111010	0011001
MOV Rn,#data	01111011	0011001
MOV Rn,#data	01111100	0011001
MOV Rn,#data	01111101	0011001
MOV Rn,#data	01111110	0011001
MOV Rn,#data	01111111	0011001
MOV @Ri#data	01110110	0011001
MOV @Ri#data	01110111	0011001
MOV di,Rn	10001000	0011001
MOV di,Rn	10001001	0011001
MOV di,Rn	10001010	0011001
MOV di,Rn	10001011	0011001
MOV di,Rn	10001100	0011001
MOV di,Rn	10001101	0011001
MOV di,Rn	10001110	0011001
MOV di,Rn	10001111	0011001

MOV di,di	10000101	0011001
MOV di,#data	01110101	0011001
MOV DPTR,#dat16	10010000	0011001
MOVX A,@DPTR	11100000	0000000
MOVX @DPTR,A	11110000	0011001
MOVX A,@Ri	11100011	0000000
MOVX A,@Ri	11100010	0000000
MOVX @Ri,A	11110011	0011001
MOVX @Ri,A	11110010	0011001
XCH A,Rn	11001000	0011001
XCH A,Rn	11001001	0011001
XCH A,Rn	11001010	0011001
XCH A,Rn	11001011	0011001
XCH A,Rn	11001100	0011001
XCH A,Rn	11001101	0011001
XCH A,Rn	11001110	0011001
XCH A,Rn	11001111	0011001
XCH A,di	11000101	0011001
XCH A,@Ri	11000110	0011001
XCH A,@Ri	11000111	0011001
XCHD A,@Ri	11010110	0001111
XCHD A,@Ri	11010111	0001111

ชุดที่ 4 รหัสดำเนินการที่เกี่ยวกับ การจัดดำเนินการตัวแปรบูล (boolean variable manipulation)

คำสั่ง	รหัสดำเนินการ	เอแอลยู คอนโทรล
CLR C	11000011	0011000
SETB C	11010011	0011101
CPL C	10110011	0011101
ANL C,bit	10000010	0011010
ANL C,/bit	10110000	0011010

MOV C,bit	10100010	0011010
CPL bit	10110010	0010110
CLR bit	11000010	0011010
SET bit	11010010	0011110
MOV bit,C	10010010	0011010

ชุดที่ 5 รหัสดำเนินการที่เกี่ยวกับ โปแกรมกิ่ง (program branching)

คำสั่ง	รหัสดำเนินการ	เอแอลยู คอนโทรล
AJMP addr11	00000001	0000000
AJMP addr11	00100001	0000000
AJMP addr11	01000001	0000000
AJMP addr11	01100001	0000000
AJMP addr11	10000001	0000000
AJMP addr11	10100001	0000000
AJMP addr11	11000001	0000000
AJMP addr11	11100001	0000000
ACALL addr11	00010001	0000010
ACALL addr11	00110001	0000010
ACALL addr11	01010001	0000010
ACALL addr11	01110001	0000010
ACALL addr11	10010001	0000010
ACALL addr11	10110001	0000010
ACALL addr11	11010001	0000010
ACALL addr11	11110001	0000010
LJMP addr16	00000010	0000000
LCALL addr16	00010010	0000010
RET | RETI	00100010	0100010
RET | RETI	00110010	0100010

ผลการวิเคราะห์สามารถตรวจพบคำสั่งที่ถอดรหัสผิดพลาดทั้งหมดในแต่ละชุดรหัสดำเนินการ ทั้ง 5ชุด ตามที่ได้ดัดแปลงแกนไมโครคอนโทรลเลอร์ MEL 805X ให้ถอดรหัสคำสั่งผิดตามข้างต้นและฟังก์ชันการถอดรหัสในชุดรหัสดำเนินการที่ไม่ได้แก้ไขให้ผิดพลาดก็ยังตรวจสอบพบว่าสามารถถอดรหัสได้ถูกต้องเนื่องจากการถอดรหัสของแกนไมโครคอนโทรลเลอร์ MEL 805X นั้นได้เขียนอธิบายฟังก์ชันการถอดรหัสในแต่ละรหัสดำเนินการแยกกันอิสระ จึงทำให้การแก้ไขฟังก์ชันการถอดรหัสให้ผิดพลาดไม่กระทบกับรหัสดำเนินการที่ถูกต้อง ซึ่งการแบ่งเป็นชุดที่มีลักษณะการดำเนินการคล้ายๆกันเพื่อลดเวลาในการทดสอบ จึงพอที่จะทราบว่ารหัสดำเนินการในชุดนั้นสามารถที่จะตรวจสอบฟังก์ชันการถอดรหัสได้ครอบคลุม ซึ่งสามารถหาความผิดพร่องตามหัวข้อ 4.4.3 ได้ทุกคำสั่งที่มีในชุดข้อมูลทดสอบทั้ง 218 รหัสดำเนินการ แต่ไม่สามารถครอบคลุมการถอดรหัสคำสั่งทั้งหมด ดูได้จากหัวข้อ 4.2.3

บทที่ 6

สรุปผลการวิจัยและข้อเสนอแนะ

งานวิจัยนี้มีวัตถุประสงค์เพื่อ หาวิธีการออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน ด้วยภาษาวีเอชดีแอล เพื่อทดสอบการเชื่อมต่อ และทดสอบฟังก์ชันการทำงานของหน่วยคำนวณและตรรกะใน ดาตาพาท และทดสอบฟังก์ชันการทำงานตัวถอดรหัสในคอนโทรลพาท ด้วยวิธีการจำลองการทำงานรวมทั้งเพื่อเพิ่มความสามารถ และสนับสนุนงานวิจัยทางด้านการออกแบบ และพัฒนาวงจรทดสอบในประเทศไทย โดยงานวิจัยได้ผลดังนี้

1. ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน
2. ชุดข้อมูลทดสอบ สำหรับการทดสอบ 3 ชุด
3. โปรแกรมสร้างและวิเคราะห์ผลการทดสอบ

6.1 สรุปผลการวิจัย

6.1.1 ผลที่ได้จากการออกแบบ

ผลที่ได้จากการออกแบบ คือได้ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน และทำการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ได้ด้วยวิธีการจำลองการทำงาน ด้วยชุดข้อมูลทดสอบที่สร้างขึ้นจำนวน 3 ชุดมีจำนวนข้อมูลทดสอบทั้งหมดทั้งหมด 314 ข้อมูลโดยมี ชุดข้อมูลทดสอบการเชื่อมต่อจำนวน 13 ข้อมูล และชุดข้อมูลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะจำนวน 83 ข้อมูล และชุดข้อมูลทดสอบฟังก์ชันการทำงานตัวถอดรหัสจำนวน 218 ข้อมูล ซึ่งส่วนจำเพาะสามารถทำงานได้ตามข้อกำหนดของสถาปัตยกรรม บาวน์ดารี สแกนโดยใช้เวลาในการจำลองการทำงานทั้งหมด 0.011 วินาที ทดสอบที่สัญญาณนาฬิกาทดสอบ 10 Mhz

การออกแบบส่วนจำเพาะได้นำตรรกะทดสอบของนายปีเตอร์ เอ็ม แคมป์เบลล์และคณะวิจัยที่ได้พัฒนา ตามมาตรฐาน IEEE149.1 โดยการใช้ภาษาวีเอชดีแอล ปรับปรุงบางส่วนรวมทั้งปรับให้เข้ากับแกนไมโครคอนโทรลเลอร์ MEL 805X ประกอบเป็น ส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน

การออกแบบส่วนจำเพาะได้ออกแบบให้มีบาวน์ดารี สแกนพาททั้งหมด 2 เส้นทางเพื่อให้สามารถประกอบ บาวน์ดารี สแกน เรจิสเตอร์ได้ครอบคลุมทุกชิ้นส่วนให้ได้มากที่สุด เนื่องจากแกนไมโครคอนโทรลเลอร์ MEL 805X ออกแบบมาเป็นส่วนดาตาพาท และส่วนคอนโทรลพาท ซึ่งภายในทั้งสองส่วน ประกอบด้วยชิ้นส่วนย่อย จึงจำเป็นต้องออกแบบให้มี 2 บาวน์ดารี สแกนพาท เพื่อให้สามารถครอบคลุมการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ให้มากที่สุด บาวน์ดารี แสกนพาท มีขนาด 260 เซลล์ และบาวน์ดารี แสกนพาทหนึ่ง มีขนาด 352 เซลล์ รวมทั้งหมดมีบาวน์ดารี สแกน เซลล์ จำนวน 622 เซลล์ หรือมีความยาวเท่ากับ 622 บิต

การทดสอบได้ทำการแก้ไข การทำงานแกนไมโครคอนโทรลเลอร์ MEL 805X ในส่วนของคอนโทรลพาท เพื่อให้สามารถทดสอบฟังก์ชันการทำงานตัวถอดรหัส โดยจะไม่กระทบกับการทำงานปกติของแกนไมโครคอนโทรลเลอร์ MEL 805X เนื่องจากส่วนที่แก้ไขจะทำงานที่สถานะตั้งใหม่เท่านั้น

6.1.2 ผลที่ได้จากการทดสอบ

การทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน จะเริ่มจากการสร้างชุดข้อมูลทดสอบด้วยโปรแกรมสร้างชุดข้อมูลทดสอบ โดยการสร้างชุดข้อมูลทดสอบนั้นจะอาศัยข้อมูลของแกนไมโครคอนโทรลเลอร์ MEL 805X มาใช้สร้างชุดข้อมูลทดสอบ ทั้งหมด 3 ชุด จากนั้นทำการทดสอบด้วยวิธีจำลองการทำงานจนได้ผลการทดสอบแล้วนำไปวิเคราะห์ผลการทดสอบ ด้วยโปรแกรมวิเคราะห์ผลการทดสอบเพื่อทำการวิเคราะห์ แกนไมโครคอนโทรลเลอร์ MEL 805X หาความผิดพร่อง รวมทั้งวิเคราะห์หาจุดที่ทำให้เกิดความผิดพร่อง โดยผลการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 850X ที่ถูกต้อง ไม่พบความผิดพร่องใดๆ และเมื่อทำการดัดแปลง แกนไมโครคอนโทรลเลอร์ MEL 850X ให้ผิดพลาด ก็สามารถที่จะวิเคราะห์หาความผิดพร่องและรายงานผลออกมาได้ทั้งหมด

6.2 ข้อเสนอแนะ

การตรวจสอบและแก้ไขทุกครั้งจะต้องเก็บข้อมูลและบันทึกผลที่ได้ และการเก็บข้อมูลของส่วนจำเพาะและโปรแกรมสร้างชุทดสอบและวิเคราะห์ ผลในแต่ละเวอร์ชั่นทุกครั้งก่อนที่จะมีการเปลี่ยนแปลง และเมื่อออกแบบส่วนจำเพาะสมบูรณ์แล้ว ควรนำส่วนจำเพาะมาทำการทดสอบ ด้วยวิธีจำลองทำงานว่าสามารถทำงานได้ถูกต้องหรือไม่ก่อน จะทดสอบด้วยชุดข้อมูลทดสอบจริง

การออกแบบส่วนจำเพาะ มีข้อจำกัดหลายข้อด้วยกันอันเนื่องจาก การออกแบบส่วนจำเพาะเพิ่มเข้าไปหลังจากการออกแบบแกนไมโครคอนโทรลเลอร์ MEL 805X ดังนี้

1. การประกอบ บาวน์ดารี สแกน เรจิสเตอร์เข้ากับแกนไมโครคอนโทรลเลอร์ MEL 805X ไม่สามารถเข้าถึงได้ในบางชิ้นส่วน เนื่องจากบางชิ้นส่วนไม่ได้ออกแบบเป็น entity ทำให้ไม่สามารถประกอบ บาวน์ดารี สแกน เรจิสเตอร์ ได้

2. ข้อมูล แกนไมโครคอนโทรลเลอร์ MEL 805X มีไม่มากพอทำให้การออกแบบส่วนจำเพาะใช้เวลามาก

3. การออกแบบส่วนจำเพาะต้องระมัดระวังไม่ให้ ไปกระทบกับการทำงานของแกนไมโครคอนโทรลเลอร์ MEL 805X

ดังนั้นจึงควรจะออกแบบส่วนจำเพาะ สำหรับทดสอบไปพร้อมๆ กับการออกแบบ แกนไมโครคอนโทรลเลอร์ MEL 805X

การทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน มีข้อจำกัดคือ ไม่สามารถจะทดสอบบาวน์ดารี สแกน พาททั้งสองพร้อมๆกันได้ เนื่องจากการออกแบบมีบางสายสัญญาณที่บาวน์ดารี สแกน เซลล์ ของแต่ละบาวน์ดารี สแกน พาท จะทำงานพร้อมกับทำให้เกิดความผิดพลาด ดังนั้นจึงต้องทำการทดสอบทีละบาวน์ดารี สแกน พาท

การออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน สามารถที่จะนำไปสังเคราะห์เป็นวงจรรวมอยู่ในแกนไมโครคอนโทรลเลอร์ MEL 805X ได้แต่ต้องทำการดัดแปลงให้เหมาะสม และการทดสอบก็อาจจะใช้โปรแกรมบนเครื่องคอมพิวเตอร์ที่พัฒนาขึ้นเพื่อสร้างสัญญาณควบคุมและรับผลจากไมโครคอนโทรลเลอร์ MEL 805X ที่ใช้สถาปัตยกรรม บาวน์ดารี สแกน มาวิเคราะห์

งานวิจัยนี้ได้นำเสนอวิธีการออกแบบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรเลอร์ MEL 805X โดยใช้สถาปัตยกรรม บาวน์ดารี สแกน และการสร้างชุดข้อมูลทดสอบ รวมทั้งการวิเคราะห์ผลการทดสอบ ซึ่งสามารถนำไปประยุกต์ใช้ในการออกแบบส่วนจำเพาะ เพื่อทดสอบวงจรอื่นๆได้ ซึ่งสามารถพิจารณา ข้อดีและข้อเสียของการเพิ่มส่วนจำเพาะนี้เข้าไปดังตารางที่ 6.1 และเปรียบเทียบวิธีการออกแบบส่วนจำเพาะดังตารางที่ 6.2

ตารางที่ 6.1 ข้อดีและข้อเสียของการเพิ่มส่วนจำเพาะ สำหรับการทดสอบ รวมไว้ในการออกแบบ

ข้อดี	ข้อเสีย
การสร้างข้อมูลทดสอบทำได้ง่าย	เพิ่มความซับซ้อน
ง่ายต่อการวินิจฉัย และการแก้ไขจุดบกพร่อง	กระทบการใช้กำลังและจำนวนขาสัญญาณ
สามารถวัดคุณภาพเชิงกำหนดได้	กระทบความเร็ว และประสิทธิภาพ
ลดต้นทุนในการทดสอบ	เพิ่มพื้นที่บนซิลิคอน

ตารางที่ 6.2 ข้อแตกต่างระหว่างการออกแบบส่วนจำเพาะในระดับพฤติกรรมและระดับโครงสร้าง

ระดับพฤติกรรม	ระดับโครงสร้าง
ออกแบบส่วนจำเพาะออกแบบง่าย	ออกแบบส่วนจำเพาะได้ยาก
มีความซับซ้อนน้อยเข้าใจได้ง่าย	มีความซับซ้อนมากเข้าใจยาก
ครอบคลุมการทดสอบได้ไม่มาก	ครอบคลุมการทดสอบได้มาก
ประหยัดเวลาในการออกแบบ	เพิ่มขั้นตอนการออกแบบมากขึ้น
มีความยืดหยุ่นในการใช้งาน	ติดข้อจำกัดในด้านเทคโนโลยี

รายการอ้างอิง

1. เจนวิทย์ ศรีหารักษา, สุวิชา จิรายุเจริญศักดิ์, จันทิรา เจือกโว้น และ ชำนาญ ปัญญาใส. การพัฒนาชิพ 8051 High Speed Microcontroller. กรุงเทพมหานคร : ห้องปฏิบัติการไมโครอิเล็กทรอนิกส์ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ.(ม.ป.ป.).
2. IEEE Test Technology Technical Committee of the IEEE computer Society. IEEE Std 1149.1-1990 (Include IEEE Std 1149.1a-1993) IEEE Standard Test Access Port and Boundary-scan Architecture. New York : The institute of Electrical and Electronics Engineers, 1993.
3. Ashenden, P. J. The Designer's Guide to VHDL. San Francisco, CA : Morgan Kaufmann Publisher,1996.
4. ชาติชาย ดิษฐกุล. Thai VHDL Book. (เอกสารประกอบการเรียน). ขอนแก่น : มหาวิทยาลัยขอนแก่น.
5. Crouch, A. L. Design for Test. Upper Saddle River, New Jersey : Prentice-Hall, 1999.
6. Abramovici, M., Breuer, M. A. and Friedman, A. D. Digital systems testing & testable design. New York : Computer Science Press, 1990.
7. Whetsel, L. A Proposed Standard Test Bus and Boundary Scan Architecture. Proceedings International Conference on Computer Design (1988) : 330–333.
8. Campbell, P. M. and Navabi, Z. Implementation of IEEE Std 1149.1- 1990 in VHDL. Boston : Northeastern University, 1992.
9. Posse, K. Algorithm Diagnosis of multi chip module defects using the IEEE 1149.1 standard. Loveland, Corolado : Hewlett Packard, 1994.
10. Angelotti, F. W. Modeling for structured system interconnect test. Proceedings International Test Conference (1994) : 127-133
11. ไกรวุฒิ โรจน์ประเสริฐสุด. เข้าใจ/สร้าง/เล่น ไมโครโปรเซอร์ 2. กรุงเทพมหานคร : ซีเอ็ดยูเคชั่น 2539.

ภาคผนวก

ภาคผนวก ก.

ผลการวิเคราะห์การทดสอบส่วนจำเพาะ สำหรับการทดสอบ

ผลการวิเคราะห์การทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X

ผลการวิเคราะห์ผลทดสอบการเชื่อมต่อ

Analyze Improperly Inserted Component

Detect Instruction Registers of BS

BS IR is 0101010101010101

Analyze Improperly Inserted Component >> BS PASSED!

Detect Instruction Registers of BS1

BS1 IR is 0101

Analyze Improperly Inserted Component >> BS1 PASSED!

Analyze BS Improper Interconnection

BS interconnect node 1-25 PASSED!

All BS interconnect Fault Test PASSED!

Analyze BS Open Fault

BS Detect Open Fault at node 1-25 PASSED!

All BS Open Fault Test PASSED!

Analyze BS Short Fault

All BS Short Fault Test PASS!

Analyze BS1 Improper Interconnection

BS1 interconnect node 1-146 PASSED!

All BS1 interconnect Fault Test PASSED!

Analyze BS1 Open Fault

BS1 Detect Open Fault at node 1 PASSED!

All BS1 Open Fault Test PASSED!

Analyze BS1 Short Fault

All BS1 Short Fault Test PASS!

Conclusion

All Interconnection Test PASSED!

ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ

Analyze ALU Functional

00000000 OR 00000000 = 00000000 >> PASSED!

11111111 OR 00000000 = 11111111 >> PASSED!

00000000 OR 11111111 = 11111111 >> PASSED!

11111111 OR 11111111 = 11111111 >> PASSED!

00000000 AND 00000000 = 00000000 >> PASSED!

11111111 AND 00000000 = 00000000 >> PASSED!

00000000 AND 11111111 = 00000000 >> PASSED!

11111111 AND 11111111 = 11111111 >> PASSED!

00000000 XOR 00000000 = 00000000 >> PASSED!

11111111 XOR 00000000 = 11111111 >> PASSED!

00000000 XOR 11111111 = 11111111 >> PASSED!

11111111 XOR 11111111 = 00000000 >> PASSED!

SWAP 00000000 = 00000000 >> PASSED!

SWAP 11110000 = 00001111 >> PASSED!

SWAP 00001111 = 11110000 >> PASSED!

SWAP 11111111 = 11111111 >> PASSED!

XXXX0000 XCHID 0000XXXX = 00000000 >> PASSED!

XXXX1111 XCHID 0000XXXX = 00001111 >> PASSED!

XXXX0000 XCHID 1111XXXX = 11110000 >> PASSED!

XXXX1111 XCHID 1111XXXX = 11111111 >> PASSED!

RR 00000000 = 00000000 >> PASSED!

RR 11111111 = 11111111 >> PASSED!

RR 10000000 = 01000000 >> PASSED!

RR 01000000 = 00100000 >> PASSED!

RR 00100000 = 00010000 >> PASSED!

RR 00010000 = 00001000 >> PASSED!

RR 00001000 = 00000100 >> PASSED!

RR 00000100 = 00000010 >> PASSED!

RR 00000010 = 00000001 >> PASSED!

RR 00000001 = 10000000 >> PASSED!

RR 00000011 = 10000001 >> PASSED!

RR 10101010 = 01010101 >> PASSED!

RL 00000000 = 00000000 >> PASSED!

RL 11111111 = 11111111 >> PASSED!

RL 00000001 = 00000010 >> PASSED!

RL 00000010 = 00000100 >> PASSED!

RL 00000100 = 00001000 >> PASSED!

RL 00001000 = 00010000 >> PASSED!

RL 00010000 = 00100000 >> PASSED!

RL 00100000 = 01000000 >> PASSED!

RL 01000000 = 10000000 >> PASSED!

RL 10000000 = 00000001 >> PASSED!

RL 11000000 = 10000001 >> PASSED!

RL 10101010 = 01010101 >> PASSED!

00000000 ADD 00000000 = 00000000 >> PASSED!

11111111 ADD 00000000 = 11111111 >> PASSED!

00000000 ADD 10101010 = 10101010 >> PASSED!

00000000 ADD 11111111 = 11111111 >> PASSED!

00000001 ADD 00000001 = 00000010 >> PASSED!

00000010 ADD 00000010 = 00000100 >> PASSED!

00000100 ADD 00000100 = 00001000 >> PASSED!

00001000 ADD 00001000 = 00010000 >> PASSED!

00010000 ADD 00010000 = 00100000 >> PASSED!

00100000 ADD 00100000 = 01000000 >> PASSED!

01000000 ADD 01000000 = 10000000 >> PASSED!

10000000 ADD 01111111 = 11111111 >> PASSED!

10000000 ADD 10000000 = 00000000 >> PASSED!

11111111 ADD 00000001 = 00000000 >> PASSED!

00000001 ADD 11111111 = 00000000 >> PASSED!

11111111 ADD 00000010 = 00000001 >> PASSED!

10000000 ADD 10000001 = 00000001 >> PASSED!

01111111 ADD 11111111 = 01111110 >> PASSED!

11111111 ADD 11111111 = 11111110 >> PASSED!

00000000 SUB 00000000 = 00000000 >> PASSED!

11111111 SUB 00000000 = 11111111 >> PASSED!

00000001 SUB 00000000 = 00000001 >> PASSED!

00000010 SUB 00000001 = 00000001 >> PASSED!

00000100 SUB 00000010 = 00000010 >> PASSED!

00001000 SUB 00000100 = 00000100 >> PASSED!

00010000 SUB 00001000 = 00001000 >> PASSED!

00100000 SUB 00010000 = 00010000 >> PASSED!

01000000 SUB 00100000 = 00100000 >> PASSED!

10000000 SUB 01000000 = 01000000 >> PASSED!

10000000 SUB 01111111 = 00000001 >> PASSED!

11111111 SUB 00001111 = 11110000 >> PASSED!

10000000 SUB 10000000 = 00000000 >> PASSED!

10101010 SUB 10101010 = 00000000 >> PASSED!

11111111 SUB 11111111 = 00000000 >> PASSED!

00000000 SUB 00000001 = 11111111 >> PASSED!

00000000 SUB 11111111 = 00000001 >> PASSED!

00000001 SUB 00001111 = 11110010 >> PASSED!

00001111 SUB 11110000 = 00011111 >> PASSED!

01111111 SUB 11111111 = 10000000 >> PASSED!

Conclusion

All ALU Functional test PASSED!

ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานตัวถอดรหัส

Analyze Control Functional

ADD A,Rn Opcode 00101000 Decode to 0000011 >> PASSED!

ADD A,Rn Opcode 00101001 Decode to 0000011 >> PASSED!

ADD A,Rn Opcode 00101010 Decode to 0000011 >> PASSED!

ADD A,Rn Opcode 00101011 Decode to 0000011 >> PASSED!

ADD A,Rn Opcode 00101100 Decode to 0000011 >> PASSED!

ADD A,Rn Opcode 00101101 Decode to 0000011 >> PASSED!

ADD A,Rn Opcode 00101110 Decode to 0000011 >> PASSED!

ADD A,Rn Opcode 00101111 Decode to 0000011 >> PASSED!

ADDC A,Rn Opcode 00111000 Decode to 1000011 >> PASSED!

ADDC A,Rn Opcode 00111001 Decode to 1000011 >> PASSED!

ADDC A,Rn Opcode 00111010 Decode to 1000011 >> PASSED!

ADDC A,Rn Opcode 00111011 Decode to 1000011 >> PASSED!

ADDC A,Rn Opcode 00111100 Decode to 1000011 >> PASSED!

ADDC A,Rn Opcode 00111101 Decode to 1000011 >> PASSED!

ADDC A,Rn Opcode 00111110 Decode to 1000011 >> PASSED!

ADDC A,Rn Opcode 00111111 Decode to 1000011 >> PASSED!

SUBB A,Rn Opcode 10011000 Decode to 1100011 >> PASSED!

SUBB A,Rn Opcode 10011001 Decode to 1100011 >> PASSED!

SUBB A,Rn Opcode 10011010 Decode to 1100011 >> PASSED!

SUBB A,Rn Opcode 10011011 Decode to 1100011 >> PASSED!

SUBB A,Rn Opcode 10011100 Decode to 1100011 >> PASSED!

SUBB A,Rn Opcode 10011101 Decode to 1100011 >> PASSED!

SUBB A,Rn Opcode 10011110 Decode to 1100011 >> PASSED!

SUBB A,Rn Opcode 10011111 Decode to 1100011 >> PASSED!

ANL A,Rn Opcode 01011000 Decode to 0011011 >> PASSED!

ANL A,Rn Opcode 01011001 Decode to 0011011 >> PASSED!

ANL A,Rn Opcode 01011010 Decode to 0011011 >> PASSED!

ANL A,Rn Opcode 01011011 Decode to 0011011 >> PASSED!

ANL A,Rn Opcode 01011100 Decode to 0011011 >> PASSED!

ANL A,Rn Opcode 01011101 Decode to 0011011 >> PASSED!

ANL A,Rn Opcode 01011110 Decode to 0011011 >> PASSED!

ANL A,Rn Opcode 01011111 Decode to 0011011 >> PASSED!

ORL A,Rn Opcode 01001000 Decode to 0011111 >> PASSED!

ORL A,Rn Opcode 01001001 Decode to 0011111 >> PASSED!

ORL A,Rn Opcode 01001010 Decode to 0011111 >> PASSED!

ORL A,Rn Opcode 01001010 Decode to 0011111 >> PASSED!

ORL A,Rn Opcode 01001011 Decode to 0011111 >> PASSED!

ORL A,Rn Opcode 01001100 Decode to 0011111 >> PASSED!

ORL A,Rn Opcode 01001101 Decode to 0011111 >> PASSED!

ORL A,Rn Opcode 01001110 Decode to 0011111 >> PASSED!

ORL A,Rn Opcode 01001111 Decode to 0011111 >> PASSED!

XRL A,Rn Opcode 01101000 Decode to 0010111 >> PASSED!

XRL A,Rn Opcode 01101001 Decode to 0010111 >> PASSED!

XRL A,Rn Opcode 01101010 Decode to 0010111 >> PASSED!

XRL A,Rn Opcode 01101011 Decode to 0010111 >> PASSED!

XRL A,Rn Opcode 01101100 Decode to 0010111 >> PASSED!

XRL A,Rn Opcode 01101101 Decode to 0010111 >> PASSED!

XRL A,Rn Opcode 01101110 Decode to 0010111 >> PASSED!

XRL A,Rn Opcode 01101111 Decode to 0010111 >> PASSED!

MOV A,Rn Opcode 11101000 Decode to 0011001 >> PASSED!

MOV A,Rn Opcode 11101001 Decode to 0011001 >> PASSED!

MOV A,Rn Opcode 11101010 Decode to 0011001 >> PASSED!

MOV A,Rn Opcode 11101011 Decode to 0011001 >> PASSED!

MOV A,Rn Opcode 11101100 Decode to 0011001 >> PASSED!

MOV A,Rn Opcode 11101101 Decode to 0011001 >> PASSED!

MOV A,Rn Opcode 11101110 Decode to 0011001 >> PASSED!

MOV A,Rn Opcode 11101111 Decode to 0011001 >> PASSED!

XCH A,Rn Opcode 11001000 Decode to 0011001 >> PASSED!

XCH A,Rn Opcode 11001001 Decode to 0011001 >> PASSED!

XCH A,Rn Opcode 11001010 Decode to 0011001 >> PASSED!

XCH A,Rn Opcode 11001011 Decode to 0011001 >> PASSED!

XCH A,Rn Opcode 11001100 Decode to 0011001 >> PASSED!

XCH A,Rn Opcode 11001101 Decode to 0011001 >> PASSED!

XCH A,Rn Opcode 11001110 Decode to 0011001 >> PASSED!

XCH A,Rn Opcode 11001111 Decode to 0011001 >> PASSED!

MOV Rn,A Opcode 11111000 Decode to 0011001 >> PASSED!

MOV Rn,A Opcode 11111001 Decode to 0011001 >> PASSED!

MOV Rn,A Opcode 11111010 Decode to 0011001 >> PASSED!

MOV Rn,A Opcode 11111011 Decode to 0011001 >> PASSED!

MOV Rn,A Opcode 11111100 Decode to 0011001 >> PASSED!

MOV Rn,A Opcode 11111101 Decode to 0011001 >> PASSED!

MOV Rn,A Opcode 11111110 Decode to 0011001 >> PASSED!

MOV Rn,A Opcode 11111111 Decode to 0011001 >> PASSED!

DEC Rn Opcode 00011000 Decode to 0100010 >> PASSED!

DEC Rn Opcode 00011001 Decode to 0100010 >> PASSED!

DEC Rn Opcode 00011010 Decode to 0100010 >> PASSED!

DEC Rn Opcode 00011011 Decode to 0100010 >> PASSED!

DEC Rn Opcode 00011100 Decode to 0100010 >> PASSED!

DEC Rn Opcode 00011101 Decode to 0100010 >> PASSED!

DEC Rn Opcode 00011110 Decode to 0100010 >> PASSED!

DEC Rn Opcode 00011111 Decode to 0100010 >> PASSED!

INC Rn Opcode 00001000 Decode to 0000010 >> PASSED!

INC Rn Opcode 00001001 Decode to 0000010 >> PASSED!

INC Rn Opcode 00001010 Decode to 0000010 >> PASSED!

INC Rn Opcode 00001011 Decode to 0000010 >> PASSED!

INC Rn Opcode 00001100 Decode to 0000010 >> PASSED!

INC Rn Opcode 00001101 Decode to 0000010 >> PASSED!

INC Rn Opcode 00001110 Decode to 0000010 >> PASSED!

INC Rn Opcode 00001111 Decode to 0000010 >> PASSED!

ADD A,@Ri Opcode 00100110 Decode to 0000011 >> PASSED!

ADD A,@Ri Opcode 00100111 Decode to 0000011 >> PASSED!

ADDC A,@Ri Opcode 00110110 Decode to 1000011 >> PASSED!

ADDC A,@Ri Opcode 00110111 Decode to 1000011 >> PASSED!

SUBB A,@Ri Opcode 10010110 Decode to 1100011 >> PASSED!

SUBB A,@Ri Opcode 10010111 Decode to 1100011 >> PASSED!

ANL A,@Ri Opcode 01010110 Decode to 0011011 >> PASSED!

ANL A,@Ri Opcode 01010111 Decode to 0011011 >> PASSED!

ORL A,@Ri Opcode 01000110 Decode to 0011111 >> PASSED!

ORL A,@Ri Opcode 01000111 Decode to 0011111 >> PASSED!

XRL A,@Ri Opcode 01100110 Decode to 0010111 >> PASSED!

XRL A,@Ri Opcode 01100111 Decode to 0010111 >> PASSED!

XCH A,@Ri Opcode 11000110 Decode to 0011001 >> PASSED!

XCH A,@Ri Opcode 11000111 Decode to 0011001 >> PASSED!

XCHD A,@Ri Opcode 11010110 Decode to 0001111 >> PASSED!

XCHD A,@Ri Opcode 11010111 Decode to 0001111 >> PASSED!

MOV A,@Ri Opcode 11100110 Decode to 0011001 >> PASSED!

MOV A,@Ri Opcode 11100111 Decode to 0011001 >> PASSED!

MOV @Ri,A Opcode 11110110 Decode to 0011001 >> PASSED!

MOV @Ri,A Opcode 11110111 Decode to 0011001 >> PASSED!

INC @Ri Opcode 00000110 Decode to 0000010 >> PASSED!

INC @Ri Opcode 00000111 Decode to 0000010 >> PASSED!

DEC @Ri Opcode 00010110 Decode to 0100010 >> PASSED!

DEC @Ri Opcode 00010111 Decode to 0100010 >> PASSED!

INC A Opcode 00000100 Decode to 0000010 >> PASSED!

DEC A Opcode 00010100 Decode to 0100010 >> PASSED!

DA A Opcode 11010100 Decode to 0000000 >> PASSED!

CLR A Opcode 11100100 Decode to 0011000 >> PASSED!

CPL A Opcode 11110100 Decode to 0010101 >> PASSED!

RL A Opcode 00100011 Decode to 0000100 >> PASSED!

RLC A Opcode 00110011 Decode to 1000100 >> PASSED!

RR A Opcode 00000011 Decode to 0001000 >> PASSED!

RRC A Opcode 00010011 Decode to 1001000 >> PASSED!

SWAP A Opcode 11000100 Decode to 0010000 >> PASSED!

CLR C Opcode 11000011 Decode to 0011000 >> PASSED!

SETB C Opcode 11010011 Decode to 0011101 >> PASSED!

CPL C Opcode 10110011 Decode to 0011101 >> PASSED!

ADD A,di Opcode 00100101 Decode to 0000011 >> PASSED!

ADDC A,di Opcode 00110101 Decode to 1000011 >> PASSED!

SUBB A,di Opcode 10010101 Decode to 1100011 >> PASSED!

ANL A,di Opcode 01010101 Decode to 0011011 >> PASSED!

ORL A,di Opcode 01000101 Decode to 0011111 >> PASSED!

XRL A,di Opcode 01100101 Decode to 0010111 >> PASSED!

MOV A,di Opcode 11100101 Decode to 0011001 >> PASSED!

XCH A,di Opcode 11000101 Decode to 0011001 >> PASSED!

ANL di,A Opcode 01010010 Decode to 0011011 >> PASSED!

ORL di,A Opcode 01000010 Decode to 0011111 >> PASSED!

XRL di,A Opcode 01100010 Decode to 0000011 >> PASSED!

MOV di,@Ri Opcode 10000110 Decode to 0011001 >> PASSED!

MOV di,@Ri Opcode 10000111 Decode to 0011001 >> PASSED!

MOV Rn,di Opcode 10101000 Decode to 0011001 >> PASSED!

MOV Rn,di Opcode 10101001 Decode to 0011001 >> PASSED!

MOV Rn,di Opcode 10101010 Decode to 0011001 >> PASSED!

MOV Rn,di Opcode 10101011 Decode to 0011001 >> PASSED!

MOV Rn,di Opcode 10101100 Decode to 0011001 >> PASSED!

MOV Rn,di Opcode 10101101 Decode to 0011001 >> PASSED!

MOV Rn,di Opcode 10101110 Decode to 0011001 >> PASSED!

MOV Rn,di Opcode 10101111 Decode to 0011001 >> PASSED!

MOV @Ri,di Opcode 10100110 Decode to 0011001 >> PASSED!

MOV @Ri,di Opcode 10100111 Decode to 0011001 >> PASSED!

MOV di,A Opcode 11110101 Decode to 0011001 >> PASSED!

INC di Opcode 00000101 Decode to 0000010 >> PASSED!

DEC di Opcode 00010101 Decode to 0100010 >> PASSED!

ADD A,#data Opcode 00100100 Decode to 0000011 >> PASSED!

ADDC A,#data Opcode 00110100 Decode to 1000011 >> PASSED!

SUBB A,#data Opcode 10010100 Decode to 1100011 >> PASSED!

ANL A,#data Opcode 01010100 Decode to 0011011 >> PASSED!

ORL A,#data Opcode 01000100 Decode to 0011111 >> PASSED!

XRL A,#data Opcode 01100100 Decode to 0010111 >> PASSED!

MOV A,#data Opcode 01110100 Decode to 0011001 >> PASSED!

MOV Rn,#data Opcode 01111000 Decode to 0011001 >> PASSED!

MOV Rn,#data Opcode 01111001 Decode to 0011001 >> PASSED!

MOV Rn,#data Opcode 01111010 Decode to 0011001 >> PASSED!

MOV Rn,#data Opcode 01111011 Decode to 0011001 >> PASSED!

MOV Rn,#data Opcode 01111100 Decode to 0011001 >> PASSED!

MOV Rn,#data Opcode 01111101 Decode to 0011001 >> PASSED!

MOV Rn,#data Opcode 01111110 Decode to 0011001 >> PASSED!

MOV Rn,#data Opcode 01111111 Decode to 0011001 >> PASSED!

MOV @Ri#data Opcode 01110110 Decode to 0011001 >> PASSED!

MOV @Ri#data Opcode 01110111 Decode to 0011001 >> PASSED!

MOV di,Rn Opcode 10001000 Decode to 0011001 >> PASSED!

MOV di,Rn Opcode 10001001 Decode to 0011001 >> PASSED!

MOV di,Rn Opcode 10001010 Decode to 0011001 >> PASSED!

MOV di,Rn Opcode 10001011 Decode to 0011001 >> PASSED!

MOV di,Rn Opcode 10001100 Decode to 0011001 >> PASSED!

MOV di,Rn Opcode 10001101 Decode to 0011001 >> PASSED!

MOV di,Rn Opcode 10001110 Decode to 0011001 >> PASSED!

MOV di,Rn Opcode 10001111 Decode to 0011001 >> PASSED!

ORL C,bit Opcode 01110010 Decode to 0011010 >> PASSED!

ORL C,/bit Opcode 10100000 Decode to 0011010 >> PASSED!

ANL C,bit Opcode 10000010 Decode to 0011010 >> PASSED!

ANL C,/bit Opcode 10110000 Decode to 0011010 >> PASSED!

MOV C,bit Opcode 10100010 Decode to 0011010 >> PASSED!

CPL bit Opcode 10110010 Decode to 0010110 >> PASSED!

CLR bit Opcode 11000010 Decode to 0011010 >> PASSED!

SET bit Opcode 11010010 Decode to 0011110 >> PASSED!

MOV bit,C Opcode 10010010 Decode to 0011010 >> PASSED!

MOV di,#data Opcode 01110101 Decode to 0011001 >> PASSED!

ANL di,#data Opcode 01010011 Decode to 0011011 >> PASSED!

ORL di,#data Opcode 01000011 Decode to 0011111 >> PASSED!

XRL di,#data Opcode 01100011 Decode to 0010111 >> PASSED!

MOV di,di Opcode 10000101 Decode to 0011001 >> PASSED!

MOV DPTR,#dat16 Opcode 10010000 Decode to 0011001 >> PASSED!

AJMP addr11 Opcode 00000001 Decode to 0000000 >> PASSED!

AJMP addr11 Opcode 00100001 Decode to 0000000 >> PASSED!

AJMP addr11 Opcode 01000001 Decode to 0000000 >> PASSED!

AJMP addr11 Opcode 01100001 Decode to 0000000 >> PASSED!

AJMP addr11 Opcode 10000001 Decode to 0000000 >> PASSED!

AJMP addr11 Opcode 10100001 Decode to 0000000 >> PASSED!

AJMP addr11 Opcode 11000001 Decode to 0000000 >> PASSED!

AJMP addr11 Opcode 11100001 Decode to 0000000 >> PASSED!

ACALL addr11 Opcode 00010001 Decode to 0000010 >> PASSED!

ACALL addr11 Opcode 00110001 Decode to 0000010 >> PASSED!

ACALL addr11 Opcode 01010001 Decode to 0000010 >> PASSED!

ACALL addr11 Opcode 01110001 Decode to 0000010 >> PASSED!

ACALL addr11 Opcode 10010001 Decode to 0000010 >> PASSED!

ACALL addr11 Opcode 10110001 Decode to 0000010 >> PASSED!

ACALL addr11 Opcode 11010001 Decode to 0000010 >> PASSED!

ACALL addr11 Opcode 11110001 Decode to 0000010 >> PASSED!

LJMP addr16 Opcode 00000010 Decode to 0000000 >> PASSED!

LCALL addr16 Opcode 00010010 Decode to 0000010 >> PASSED!

RET | RETI Opcode 00100010 Decode to 0100010 >> PASSED!

RET | RETI Opcode 00110010 Decode to 0100010 >> PASSED!

MOVX A,@DPTR Opcode 11100000 Decode to 0000000 >> PASSED!

MOVX @DPTR,A Opcode 11110000 Decode to 0011001 >> PASSED!

MOVX A,@Ri Opcode 11100011 Decode to 0000000 >> PASSED!

MOVX A,@Ri Opcode 11100010 Decode to 0000000 >> PASSED!

MOVX @Ri,A Opcode 11110011 Decode to 0011001 >> PASSED!

MOVX @Ri,A Opcode 11110010 Decode to 0011001 >> PASSED!

Conclusion

All Control Functional test PASSED!

ผลการวิเคราะห์การทดสอบส่วนจำเพาะ สำหรับการทดสอบ แกนไมโครคอนโทรลเลอร์ MEL 805X ที่ดัดแปลงให้ผิดพลาด

ผลการวิเคราะห์การทดสอบส่วนจำเพาะ ที่ดัดแปลงแกนไมโครคอนโทรลเลอร์ MEL 805X ให้ผิดพลาดในแบบต่างๆ

1. ผลการวิเคราะห์ผลทดสอบการเชื่อมต่อ

ดัดแปลงแกนไมโครคอนโทรลเลอร์ MEL 805X ให้ผิดพลาดแล้ววิเคราะห์ดังนี้

1. ผลการวิเคราะห์ผลทดสอบการเชื่อม ที่ดัดแปลงให้บาวน์ดารี สแกนพาท ขาดที่บาวน์ดารี สแกน เรจิสเตอร์ BS_INT_con

Analyze Improperly Inserted Component

Detect Instruction Registers of BS

BS IR is UUUUUUUUUU010101

Analyze Improperly Inserted Component >> BS FAILED!

BS link broken at >> BS_INT_con

2. ผลการวิเคราะห์ผลทดสอบการเชื่อม ที่ดัดแปลงให้ บาวน์ดารี สแกนพาทหนึ่ง ขาดที่บาวน์ดารี สแกน เรจิสเตอร์ BS_BS_control

Analyze Improperly Inserted Component

Detect Instruction Registers of BS1

BS1 IR is UU01

Analyze Improperly Inserted Component >> BS1 FAILED!

BS1 link broken at >> BS_BS_control

3. ผลการวิเคราะห์ผลทดสอบการเชื่อม ที่ดัดแปลงดัดแปลงให้การเชื่อมต่อผิดพลาดโดยการเปิดขาสัญญาณ INT_CON_T_7 ที่ตัวขับ สัญญาณ และเปิดขาสัญญาณ INT_CON_T_19 ที่ตัวรับสัญญาณที่บาวน์ดารีสแกน เรจิสเตอร์ BS_INT_con

Analyze BS Improper Interconnection

Fault at >> 5BS_INT_CON_T_7#73 >> 3BS_PC_CON_T_7#35

Fault at >> 5BS_INT_CON_T_19#82

Analyze BS Open Fault

BS Detect Open Fault at node 7 FAILED!

Fault open at driver >> 2BS_T_GEN_T_7#15

BS Detect Open Fault at node 16 FAILED!

Fault open at receiver >> 5BS_INT_CON_T_19#82

4. ผลการวิเคราะห์ผลทดสอบการเชื่อมต่อ ที่ดัดแปลงให้การเชื่อมต่อผิดพลาดที่ขาสัญญาณ CONTROL_INT0_Filtered ต่อเข้ากับขาสัญญาณ DATA_PATH _INT1_Filtered

Analyze BS1 Improper Interconnection

BS1 interconnect node 5 FAILED!

Fault at >> 2BS1_DATA_PATH_INT1_Filtered#335 >>

2BS1_DATA_PATH_INT0_Filtered#334

BS1 interconnect node 6 FAILED!

Fault at >> 2BS1_DATA_PATH_INT1_Filtered#335

Analyze BS1 Open Fault

BS1 Detect Open Fault at node 5 FAILED!

Fault may be open either >> 1BS1_CONTROL_INT0_Filtered#42 <-->

2BS1_DATA_PATH_INT0_Filtered#334

BS1 Detect Open Fault at node 6 FAILED!

Fault may be open either >> 1BS1_CONTROL_INT1_Filtered#43 <-->

2BS1_DATA_PATH_INT1_Filtered#335

5. ผลการวิเคราะห์ผลทดสอบการเชื่อมต่อ ที่ดัดแปลงให้การลัดวงจรกับกราวด์ที่ขาสัญญาณ 5BS_INT_CON_T_20 และลัดวงจรกับพาวเวอร์ที่ขาสัญญาณ DATA_PATH _RESET

Analyze BS Short Fault

Short fault to Ground at >> 5BS_INT_CON_T_20#83

Some BS Short Fault Test FAILED!

Analyze BS1 Short Fault

Short fault to Power at >> 2BS1_DATA_PATH_RESET#159

Some BS1 Short Fault Test FAILED!

2. ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ

ดัดแปลงให้ แกนไมโครคอนโทรลเลอร์ MEL 805X ให้ฟังก์ชันการทำงานหน่วยคำนวณและตรรกะ ทำงานผิดพลาดดังนี้

1. การทำงานการดำเนินการออร์ โดยให้ บิตที่ 4 ถึงบิตที่ 7 ไม่ทำดำเนินการออร์

ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

00000000 OR 00000000 = UUUU0000 >> FAILED!

11111111 OR 00000000 = UUUU1111 >> FAILED!

00000000 OR 11111111 = UUUU1111 >> FAILED!

11111111 OR 11111111 = UUUU1111 >> FAILED!

2. การทำงานการดำเนินการแอนด์ โดยให้ ค่าผลลัพธ์ออกมาเป็น 11111111 ตลอด

ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

00000000 AND 00000000 = 11111111 >> FAILED!

11111111 AND 00000000 = 11111111 >> FAILED!

00000000 AND 11111111 = 11111111 >> FAILED!

11111111 AND 11111111 = 11111111 >> PASSED!

3. การทำงานการดำเนินการออร์เฉพาะ โดยให้การทำงานเป็นการดำเนินการออร์แทน

ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

00000000 XOR 00000000 = 00000000 >> PASSED!

11111111 XOR 00000000 = 11111111 >> PASSED!

00000000 XOR 11111111 = 11111111 >> PASSED!

11111111 XOR 11111111 = 11111111 >> FAILED!

4. การทำงานการดำเนินการสลับ โดยไม่ดำเนินการสลับ ค่าในตัวดำเนินการหนึ่ง ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

SWAP 00000000 = 00000000 >> PASSED!

SWAP 11110000 = 11110000 >> FAILED!

SWAP 00001111 = 00001111 >> FAILED!

SWAP 11111111 = 11111111 >> PASSED!

5. การทำงานการดำเนินการแลกเปลี่ยนบิตล่าง โดยให้เป็นค่าตัวดำเนินการสองแทน ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

XXXX0000 XCHID 0000XXXX = 00000000 >> PASSED!

XXXX1111 XCHID 0000XXXX = 00000000 >> FAILED!

XXXX0000 XCHID 1111XXXX = 11110000 >> PASSED!

XXXX1111 XCHID 1111XXXX = 11110000 >> FAILED!

6. การทำงานการดำเนินการหมุนขวา โดยให้การทำงานเป็นการดำเนินการหมุนซ้ายแทน ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

RR 00000000 = 00000000 >> PASSED!

RR 11111111 = 11111111 >> PASSED!

RR 10000000 = 00000001 >> FAILED!

RR 01000000 = 10000000 >> FAILED!

RR 00100000 = 01000000 >> FAILED!

RR 00010000 = 00100000 >> FAILED!

RR 00001000 = 00010000 >> FAILED!

RR 00000100 = 00001000 >> FAILED!

RR 00000010 = 00000100 >> FAILED!

RR 00000001 = 00000010 >> FAILED!

RR 00000011 = 00000110 >> FAILED!

RR 10101010 = 01010101 >> PASSED!

7. การทำงานการดำเนินการหมุนซ้ายโดยให้การทำงานเป็นการดำเนินการหมุนขวาแทน ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

RL 00000000 = 00000000 >> PASSED!

RL 11111111 = 11111111 >> PASSED!

RL 00000001 = 10000000 >> FAILED!

RL 00000010 = 00000001 >> FAILED!

RL 00000100 = 00000010 >> FAILED!

RL 00001000 = 00000100 >> FAILED!

RL 00010000 = 00001000 >> FAILED!

RL 00100000 = 00010000 >> FAILED!

RL 01000000 = 00100000 >> FAILED!

RL 10000000 = 01000000 >> FAILED!

RL 11000000 = 01100000 >> FAILED!

RL 10101010 = 01010101 >> PASSED!

8. การทำงานการดำเนินการบวกโดยให้การทำงานในส่วนของวงจรบวก ผิดพลาด ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

00000000 ADD 00000000 = 00000000 >> PASSED!

11111111 ADD 00000000 = 11111111 >> PASSED!

00000000 ADD 10101010 = 10101010 >> PASSED!

00000000 ADD 11111111 = 11111111 >> PASSED!

00000001 ADD 00000001 = 00000010 >> PASSED!

00000010 ADD 00000010 = 00000100 >> PASSED!

00000100 ADD 00000100 = 00001000 >> PASSED!

00001000 ADD 00001000 = 00010000 >> PASSED!

00010000 ADD 00010000 = 00100000 >> PASSED!

00100000 ADD 00100000 = 01000000 >> PASSED!

01000000 ADD 01000000 = 10000000 >> PASSED!

10000000 ADD 01111111 = 11111111 >> PASSED!

10000000 ADD 10000000 = 00000000 >> PASSED!

11111111 ADD 00000001 = 00001110 >> FAILED!

00000001 ADD 11111111 = 00001110 >> FAILED!

11111111 ADD 00000010 = 00001101 >> FAILED!

10000000 ADD 10000001 = 00000001 >> PASSED!

01111111 ADD 11111111 = 01111110 >> PASSED!

11111111 ADD 11111111 = 11111110 >> PASSED!

9. การทำงานการดำเนินการลบโดยให้การทำงานในส่วนของวงจรส่วนเติมเต็มหนึ่ง ผิดพลาด

ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานหน่วยคำนวณและตรรกะดังนี้

00000000 SUB 00000000 = 00000000 >> PASSED!

11111111 SUB 00000000 = 11111111 >> PASSED!

00000001 SUB 00000000 = 00000001 >> PASSED!

00000010 SUB 00000001 = 00000010 >> FAILED!

00000100 SUB 00000010 = 00000100 >> FAILED!

00001000 SUB 00000100 = 00001000 >> FAILED!

00010000 SUB 00001000 = 00010000 >> FAILED!

00100000 SUB 00010000 = 00100000 >> FAILED!

01000000 SUB 00100000 = 01000000 >> FAILED!

10000000 SUB 01000000 = 10000000 >> FAILED!

10000000 SUB 01111111 = 10000000 >> FAILED!

11111111 SUB 00001111 = 11111111 >> FAILED!

10000000 SUB 10000000 = 10000000 >> FAILED!

10101010 SUB 10101010 = 10101010 >> FAILED!

11111111 SUB 11111111 = 11111111 >> FAILED!

00000000 SUB 00000001 = 00000000 >> FAILED!

00000000 SUB 11111111 = 00000000 >> FAILED!

00000001 SUB 00001111 = 00000001 >> FAILED!

00001111 SUB 11110000 = 00001111 >> FAILED!

01111111 SUB 11111111 = 01111111 >> FAILED!

3. ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานตัวถอดรหัส

ได้ผลการวิเคราะห์ผลทดสอบฟังก์ชันการทำงานตัวถอดรหัสดังนี้

ชุดที่ 1 รหัสดำเนินการที่เกี่ยวกับ การดำเนินการทางคณิตศาสตร์

ADD A,Rn Opcode 00101000 Decode to 0000001 >> FAILED!

ADD A,Rn Opcode 00101001 Decode to 0000001 >> FAILED!

ADD A,Rn Opcode 00101010 Decode to 0000001 >> FAILED!

ADD A,Rn Opcode 00101011 Decode to 0000001 >> FAILED!

ADD A,Rn Opcode 00101100 Decode to 0000001 >> FAILED!

ADD A,Rn Opcode 00101101 Decode to 0000001 >> FAILED!

ADD A,Rn Opcode 00101110 Decode to 0000001 >> FAILED!

ADD A,Rn Opcode 00101111 Decode to 0000001 >> FAILED!

ADD A,@Ri Opcode 00100110 Decode to 0000001 >> FAILED!

ADD A,@Ri Opcode 00100111 Decode to 0000001 >> FAILED!

ADD A,di Opcode 00100101 Decode to 0000001 >> FAILED!

ADD A,#data Opcode 00100100 Decode to 0000001 >> FAILED!

ADDC A,Rn Opcode 00111000 Decode to 1000001 >> FAILED!

ADDC A,Rn Opcode 00111001 Decode to 1000001 >> FAILED!

ADDC A,Rn Opcode 00111010 Decode to 1000001 >> FAILED!

ADDC A,Rn Opcode 00111011 Decode to 1000001 >> FAILED!

ADDC A,Rn Opcode 00111100 Decode to 1000001 >> FAILED!

ADDC A,Rn Opcode 00111101 Decode to 1000001 >> FAILED!

ADDC A,Rn Opcode 00111110 Decode to 1000001 >> FAILED!

ADDC A,Rn Opcode 00111111 Decode to 1000001 >> FAILED!

ADDC A,@Ri Opcode 00110110 Decode to 1000001 >> FAILED!

ADDC A,@Ri Opcode 00110111 Decode to 1000001 >> FAILED!

ADDC A,di Opcode 00110101 Decode to 1000001 >> FAILED!

ADDC A,#data Opcode 00110100 Decode to 1000001 >> FAILED!

SUBB A,Rn Opcode 10011000 Decode to 0000011 >> FAILED!

SUBB A,Rn Opcode 10011001 Decode to 0000011 >> FAILED!

SUBB A,Rn Opcode 10011010 Decode to 0000011 >> FAILED!

SUBB A,Rn Opcode 10011011 Decode to 0000011 >> FAILED!

SUBB A,Rn Opcode 10011100 Decode to 0000011 >> FAILED!

SUBB A,Rn Opcode 10011101 Decode to 0000011 >> FAILED!

SUBB A,Rn Opcode 10011110 Decode to 0000011 >> FAILED!

SUBB A,Rn Opcode 10011111 Decode to 0000011 >> FAILED!

SUBB A,@Ri Opcode 10010110 Decode to 0000011 >> FAILED!

SUBB A,@Ri Opcode 10010111 Decode to 0000011 >> FAILED!

SUBB A,di Opcode 10010101 Decode to 0000011 >> FAILED!

SUBB A,#data Opcode 10010100 Decode to 0000011 >> FAILED!

INC A Opcode 00000100 Decode to 0000110 >> FAILED!

INC Rn Opcode 00001000 Decode to 0000110 >> FAILED!

INC Rn Opcode 00001001 Decode to 0000110 >> FAILED!

INC Rn Opcode 00001010 Decode to 0000110 >> FAILED!

INC Rn Opcode 00001011 Decode to 0000110 >> FAILED!

INC Rn Opcode 00001100 Decode to 0000110 >> FAILED!

INC Rn Opcode 00001101 Decode to 0000110 >> FAILED!

INC Rn Opcode 00001110 Decode to 0000110 >> FAILED!

INC Rn Opcode 00001111 Decode to 0000110 >> FAILED!

INC @Ri Opcode 00000110 Decode to 0000110 >> FAILED!

INC @Ri Opcode 00000111 Decode to 0000110 >> FAILED!

INC di Opcode 00000101 Decode to 0000110 >> FAILED!

DEC Rn Opcode 00011000 Decode to 0000000 >> FAILED!

DEC Rn Opcode 00011001 Decode to 0000000 >> FAILED!

DEC Rn Opcode 00011010 Decode to 0000000 >> FAILED!

DEC Rn Opcode 00011011 Decode to 0000000 >> FAILED!

DEC Rn Opcode 00011100 Decode to 0000000 >> FAILED!

DEC Rn Opcode 00011101 Decode to 0000000 >> FAILED!

DEC Rn Opcode 00011110 Decode to 0000000 >> FAILED!

DEC Rn Opcode 00011111 Decode to 0000000 >> FAILED!

DEC @Ri Opcode 00010110 Decode to 0000000 >> FAILED!

DEC @Ri Opcode 00010111 Decode to 0000000 >> FAILED!

DEC A Opcode 00010100 Decode to 0000000 >> FAILED!

DEC di Opcode 00010101 Decode to 0000000 >> FAILED!

DA A Opcode 11010100 Decode to 0001111 >> FAILED!

Conclusion

Some Control Functional test FAILED!

ชุดที่ 2 รหัสดำเนินการที่เกี่ยวกับ การดำเนินการตรรกะ (logical operation)

ORL A,Rn Opcode 01001000 Decode to 1100111 >> FAILED!

ORL A,Rn Opcode 01001001 Decode to 1100111 >> FAILED!

ORL A,Rn Opcode 01001010 Decode to 1100111 >> FAILED!

ORL A,Rn Opcode 01001010 Decode to 1100111 >> FAILED!

ORL A,Rn Opcode 01001011 Decode to 1100111 >> FAILED!

ORL A,Rn Opcode 01001100 Decode to 1100111 >> FAILED!

ORL A,Rn Opcode 01001101 Decode to 1100111 >> FAILED!

ORL A,Rn Opcode 01001110 Decode to 1100111 >> FAILED!

ORL A,Rn Opcode 01001111 Decode to 1100111 >> FAILED!

ORL A,@Ri Opcode 01000110 Decode to 1100111 >> FAILED!

ORL A,@Ri Opcode 01000111 Decode to 1100111 >> FAILED!

ORL A,di Opcode 01000101 Decode to 1100111 >> FAILED!

ORL di,A Opcode 01000010 Decode to 1100111 >> FAILED!

ORL A,#data Opcode 01000100 Decode to 1100111 >> FAILED!

ORL C,bit Opcode 01110010 Decode to 0001010 >> FAILED!

ORL C,/bit Opcode 10100000 Decode to 0001010 >> FAILED!

ORL di,#data Opcode 01000011 Decode to 1100111 >> FAILED!

ANL A,Rn Opcode 01011000 Decode to 0000111 >> FAILED!

ANL A,Rn Opcode 01011001 Decode to 0000111 >> FAILED!

ANL A,Rn Opcode 01011010 Decode to 0000111 >> FAILED!

ANL A,Rn Opcode 01011011 Decode to 0000111 >> FAILED!

ANL A,Rn Opcode 01011100 Decode to 0000111 >> FAILED!

ANL A,Rn Opcode 01011101 Decode to 0000111 >> FAILED!

ANL A,Rn Opcode 01011110 Decode to 0000111 >> FAILED!

ANL A,Rn Opcode 01011111 Decode to 0000111 >> FAILED!

ANL A,@Ri Opcode 01010110 Decode to 0000111 >> FAILED!

ANL A,@Ri Opcode 01010111 Decode to 0000111 >> FAILED!

ANL A,di Opcode 01010101 Decode to 0000111 >> FAILED!

ANL A,#data Opcode 01010100 Decode to 0000111 >> FAILED!

ANL di,#data Opcode 01010011 Decode to 0000111 >> FAILED!

ANL di,A Opcode 01010010 Decode to 0000111 >> FAILED!

XRL A,Rn Opcode 01101000 Decode to 0010111 >> FAILED!

XRL A,Rn Opcode 01101001 Decode to 0010111 >> FAILED!

XRL A,Rn Opcode 01101010 Decode to 0010111 >> FAILED!

XRL A,Rn Opcode 01101011 Decode to 0010111 >> FAILED!

XRL A,Rn Opcode 01101100 Decode to 0010111 >> FAILED!

XRL A,Rn Opcode 01101101 Decode to 0010111 >> FAILED!

XRL A,Rn Opcode 01101110 Decode to 0010111 >> FAILED!

XRL A,Rn Opcode 01101111 Decode to 0010111 >> FAILED!

XRL A,@Ri Opcode 01100110 Decode to 0010111 >> FAILED!

XRL A,@Ri Opcode 01100111 Decode to 0010111 >> FAILED!

XRL A,di Opcode 01100101 Decode to 0010111 >> FAILED!

XRL di,A Opcode 01100010 Decode to 0000011 >> FAILED!

XRL A,#data Opcode 01100100 Decode to 0010111 >> FAILED!

XRL di,#data Opcode 01100011 Decode to 0010111 >> FAILED!

CLR A Opcode 11100100 Decode to 0000000 >> FAILED!

CPL A Opcode 11110100 Decode to 0011101 >> FAILED!

RL A Opcode 00100011 Decode to 0000111 >> FAILED!

RLC A Opcode 00110011 Decode to 1000110 >> FAILED!

RR A Opcode 00000011 Decode to 0001100 >> FAILED!

RRC A Opcode 00010011 Decode to 0001000 >> FAILED!

SWAP A Opcode 11000100 Decode to 0011000 >> FAILED!

Conclusion

Some Control Functional test FAILED!

ชุดที่ 3 รหัสดำเนินการที่เกี่ยวกับ การถ่ายโอนข้อมูล

MOV A,Rn Opcode 11101000 Decode to 0000111 >> FAILED!

MOV A,Rn Opcode 11101001 Decode to 0000111 >> FAILED!

MOV A,Rn Opcode 11101010 Decode to 0000111 >> FAILED!

MOV A,Rn Opcode 11101011 Decode to 0000111 >> FAILED!

MOV A,Rn Opcode 11101100 Decode to 0000111 >> FAILED!

MOV A,Rn Opcode 11101101 Decode to 0000111 >> FAILED!

MOV A,Rn Opcode 11101110 Decode to 0000111 >> FAILED!

MOV A,Rn Opcode 11101111 Decode to 0000111 >> FAILED!

MOV Rn,A Opcode 11111000 Decode to 0000111 >> FAILED!

MOV Rn,A Opcode 11111001 Decode to 0000111 >> FAILED!

MOV Rn,A Opcode 11111010 Decode to 0000111 >> FAILED!

MOV Rn,A Opcode 11111011 Decode to 0000111 >> FAILED!

MOV Rn,A Opcode 11111100 Decode to 0000111 >> FAILED!

MOV Rn,A Opcode 11111101 Decode to 0000111 >> FAILED!

MOV Rn,A Opcode 11111110 Decode to 0000111 >> FAILED!

MOV Rn,A Opcode 11111111 Decode to 0000111 >> FAILED!

MOV A,@Ri Opcode 11100110 Decode to 0000111 >> FAILED!

MOV A,@Ri Opcode 11100111 Decode to 0000111 >> FAILED!

MOV @Ri,A Opcode 11110110 Decode to 0000111 >> FAILED!

MOV @Ri,A Opcode 11110111 Decode to 0000111 >> FAILED!

MOV A,di Opcode 11100101 Decode to 0000111 >> FAILED!

MOV di,@Ri Opcode 10000110 Decode to 0000111 >> FAILED!

MOV di,@Ri Opcode 10000111 Decode to 0000111 >> FAILED!

MOV Rn,di Opcode 10101000 Decode to 0000111 >> FAILED!

MOV Rn,di Opcode 10101001 Decode to 0000111 >> FAILED!

MOV Rn,di Opcode 10101010 Decode to 0000111 >> FAILED!

MOV Rn,di Opcode 10101011 Decode to 0000111 >> FAILED!

MOV Rn,di Opcode 10101100 Decode to 0000111 >> FAILED!

MOV Rn,di Opcode 10101101 Decode to 0000111 >> FAILED!

MOV Rn,di Opcode 10101110 Decode to 0000111 >> FAILED!

MOV Rn,di Opcode 10101111 Decode to 0000111 >> FAILED!

MOV @Ri,di Opcode 10100110 Decode to 0000111 >> FAILED!

MOV @Ri,di Opcode 10100111 Decode to 0000111 >> FAILED!

MOV di,A Opcode 11110101 Decode to0000111 >> FAILED!

MOV A,#data Opcode 01110100 Decode to 0000111 >> FAILED!

MOV Rn,#data Opcode 01111000 Decode to 0000111 >> FAILED!

MOV Rn,#data Opcode 01111001 Decode to 0000111 >> FAILED!

MOV Rn,#data Opcode 01111010 Decode to 0000111 >> FAILED!

MOV Rn,#data Opcode 01111011 Decode to 0000111 >> FAILED!

MOV Rn,#data Opcode 01111100 Decode to 0000111 >> FAILED!

MOV Rn,#data Opcode 01111101 Decode to 0000111 >> FAILED!

MOV Rn,#data Opcode 01111110 Decode to 0000111 >> FAILED!

MOV Rn,#data Opcode 01111111 Decode to 0000111 >> FAILED!

MOV @Ri#data Opcode 01110110 Decode to 0000111 >> FAILED!

MOV @Ri#data Opcode 01110111 Decode to 0000111 >> FAILED!

MOV di,Rn Opcode 10001000 Decode to 0000111 >> FAILED!

MOV di,Rn Opcode 10001001 Decode to 0000111 >> FAILED!

MOV di,Rn Opcode 10001010 Decode to 0000111 >> FAILED!

MOV di,Rn Opcode 10001011 Decode to 0000111 >> FAILED!

MOV di,Rn Opcode 10001100 Decode to 0000111 >> FAILED!

MOV di,Rn Opcode 10001101 Decode to 0000111 >> FAILED!

MOV di,Rn Opcode 10001110 Decode to 0000111 >> FAILED!

MOV di,Rn Opcode 10001111 Decode to 0000111 >> FAILED!

MOV di,di Opcode 10000101 Decode to 0000111 >> FAILED!

MOV di,#data Opcode 01110101 0000111 >> FAILED!

MOV DPTR,#dat16 Opcode 10010000 Decode to 0000111 >> FAILED!

MOVX A,@DPTR Opcode 11100000 Decode to 0000011 >> FAILED!

MOVX @DPTR,A Opcode 11110000 Decode to 0000111 >> FAILED!

MOVX A,@Ri Opcode 11100011 Decode to 0000011 >> FAILED!

MOVX A,@Ri Opcode 11100010 Decode to 0000011 >> FAILED!

MOVX @Ri,A Opcode 11110011 Decode to 0000111 >> FAILED!

MOVX @Ri,A Opcode 11110010 Decode to 0000111 >> FAILED!

XCH A,Rn Opcode 11001000 Decode to 0000111 >> FAILED!

XCH A,Rn Opcode 11001001 Decode to 1100011 >> FAILED!

XCH A,Rn Opcode 11001010 Decode to 1100011 >> FAILED!

XCH A,Rn Opcode 11001011 Decode to 1100011 >> FAILED!

XCH A,Rn Opcode 11001100 Decode to 1100011 >> FAILED!

XCH A,Rn Opcode 11001101 Decode to 1100011 >> FAILED!

XCH A,Rn Opcode 11001110 Decode to 1100011 >> FAILED!

XCH A,Rn Opcode 11001111 Decode to 1100011 >> FAILED!

XCH A,di Opcode 11000101 Decode to 1100011 >> FAILED!

XCH A,@Ri Opcode 11000110 Decode to 1100011 >> FAILED!

XCH A,@Ri Opcode 11000111 Decode to 0011011 >> FAILED!

XCHD A,@Ri Opcode 1101011 Decode to 0011111 >> FAILED!

XCHD A,@Ri Opcode 1101011 Decode to 0011111 >> FAILED!

Conclusion

Some Control Functional test FAILED!

ชุดที่ 4 รหัสดำเนินการที่เกี่ยวกับ การจัดดำเนินการตัวแปรบูล

CLR C Opcode 11000011 Decode to 0011100 >> FAILED!

SETB COpcode 11010011 Decode to 0010101 >> FAILED!

CPL C Opcode 10110011 Decode to 0010101 >> FAILED!

ANL C,bit Opcode 10000010 Decode to 0011110 >> FAILED!

ANL C,/bit Opcode 10110000 Decode to 0011110 >> FAILED!

MOV C,bit Opcode 10100010 Decode to 0011110 >> FAILED!

CPL bit Opcode 10110010 Decode to 0010111 >> FAILED!

CLR bit Opcode 11000010 Decode to 0011011 >> FAILED!

SET bit Opcode 11010010 Decode to 0011111 >> FAILED!

MOV bit,C Opcode 10010010 Decode to 0011110 >> FAILED!

Conclusion

Some Control Functional test FAILED!

ชุดที่ 5 รหัสดำเนินการที่เกี่ยวกับ โปแกรมกิ่ง

AJMP addr11 Opcode 00000001 Decode to 1100000 >> FAILED!

AJMP addr11 Opcode 00100001 Decode to 1100000 >> FAILED!

AJMP addr11 Opcode 1000001 Decode to 1100000 >> FAILED!

AJMP addr11อ01100001 Decode to 1111111 >> FAILED!

AJMP addr11 Opcode 10000001 Decode to 1111111 >> FAILED!

AJMP addr11 Opcode 10100001 Decode to 1111111 >> FAILED!

AJMP addr11 Opcode 11000001 Decode to 1111111 >> FAILED!

AJMP addr11 Opcode 11100001 Decode to 1111111 >> FAILED!

ACALL addr11 Opcode 00010001 Decode to 0000011 >> FAILED!

ACALL addr11 Opcode 00110001 Decode to 0100000 >> FAILED!

ACALL addr11 Opcode 01010001 Decode to 0100000 >> FAILED!

ACALL addr11 Opcode 01110001 Decode to 0100000 >> FAILED!

ACALL addr11 Opcode 10010001 Decode to 0100000 >> FAILED!

ACALL addr11 Opcode 10110001 Decode to 0100000 >> FAILED!

ACALL addr11 Opcode 11010001 Decode to 0100000 >> FAILED!

ACALL addr11 Opcode 11110001 Decode to 0100000 >> FAILED!

LJMP addr16 Opcode 00000010 Decode to 1111111 >> FAILED!

LCALL addr16 Opcode 00010010 Decode to 0000011 >> FAILED!

RET | RETI Opcode 00100010 Decode to 0100011 >> FAILED!

RET | RETI Opcode 00110010 Decode to 0100011 >> FAILED!

Conclusion

Some Control Functional test FAILED!

คำสั่งที่เหลือสามารถถอดรหัสคำสั่งได้ถูกต้องจึงไม่ได้นำมาแสดงในแต่ละชุดการทดสอบ และโดยการทดสอบจะทดสอบทั้งหมด 5 ครั้งในแต่ละครั้งดัดแปลงฟังก์ชันการถอดรหัสให้ผิดพลาดเฉพาะชุดรหัสดำเนินการที่เหลือจะไม่ทำการดัดแปลงใดๆ

ภาคผนวก ข.

ตัวอย่างการออกแบบส่วนจำเพาะ สำหรับทดสอบ

ตัวอย่างโปรแกรมต้นฉบับตรรกะทดสอบ

เซลล์เรจิสเตอร์ผ่าน

```
------------------------------------------------------------------
-- Bypass Register Cell        P.M.Camille       Nov. 17, 1991
--
-- This is a bypass register cell.
--
-- The schematic for this cell is on page 9-1, fig. 9-1,
-- of IEEE Std 1149.1-1990.
------------------------------------------------------------------

ENTITY br_cell IS
  PORT (clockDR, shiftDR, scan_in : IN BIT;
        scan_out : OUT BIT);
END br_cell;

ARCHITECTURE behavioral OF br_cell IS
  SIGNAL temp : BIT;
BEGIN
  temp <= (shiftDR AND scan_in);
  dff : PROCESS (clockDR)
  BEGIN
    IF (clockDR = '1') THEN scan_out <= temp; END IF;
  END PROCESS dff;
END behavioral;
```

เซลล์เรจิสเตอร์ข้อมูล

```
------------------------------------------------------------------
-- Data Register Cell        P.M.Campbell        Nov. 17, 1991
--
-- This is a boundary-scan/data register cell.
```

```
--
--  The schematic for this cell is on page 10-18, fig. 10-16,
--  of IEEE Std 1149.1-1990.
-------------------------------------------------------------


ENTITY dr_cell IS
  PORT (mode, data_in, shiftDR, scan_in, clockDR, updateDR : IN BIT;
        data_out, scan_out : OUT BIT);
END dr_cell;

ARCHITECTURE behavioral OF dr_cell IS
  SIGNAL ff1, ff2 : BIT;
BEGIN
  dff1 : PROCESS (clockDR)
  BEGIN
    IF (clockDR = '1') THEN
      IF shiftDR = '0' THEN
        ff1 <= data_in;
      ELSE
        ff1 <= scan_in;
      END IF;
    END IF;
  END PROCESS dff1;
  dff2 : PROCESS (updateDR)
  BEGIN
    IF (updateDR = '1') THEN
      ff2 <= ff1;
    END IF;
  END PROCESS dff2;
  data_out <= data_in WHEN (mode = '0') ELSE ff2;
  scan_out <= ff1;
```

```
END behavioral;
```

เซลล์เรจิสเตอร์คำสั่ง

```
-------------------------------------------------------------------
-- Instruction Register Cell      P.M.Campbell     Dec. 6, 1991
--
-- This is an instruction register cell.
--
-- The schematic for this cell is on page 6-4, fig. 6-1,
-- of IEEE Std 1149.1-1990.
-------------------------------------------------------------------

ENTITY ir_cell IS
  PORT (shiftIR, data_in, scan_in, clockIR, updateIR, reset_bar,
        reset_value : IN BIT;
        data_out, scan_out : OUT BIT);
END ir_cell;

ARCHITECTURE behavioral OF ir_cell IS
  SIGNAL ff1 : BIT;
BEGIN
  dff : PROCESS (clockIR)
  BEGIN
    IF (clockIR = '1') THEN
      IF shiftIR = '0' THEN
        ff1 <= data_in;
      ELSE
        ff1 <= scan_in;
      END IF;
    END IF;
  END PROCESS dff;
```

```
 scan_out <= ff1;
 dffr : PROCESS (updateIR, reset_bar)
 BEGIN
  IF (reset_bar = '0') THEN
   data_out <= reset_value;
  ELSIF ((updateIR = '1') AND updateIR'EVENT) THEN
   data_out <= ff1;
  END IF;
 END PROCESS dffr;
END behavioral;
```

เรจิสเตอร์ข้อมูล

```
-------------------------------------------------------------
-- Data Register         P.M.Campbell          Nov. 21, 1991
--
-- This is a data register which is composed of a chain of
-- boundary-scan/data register cells.  The data register
-- is described generically to allow any size register to
-- be generated.
--
-- Note that the shift-register loads data into the data
-- register MSB and shifts data out of the LSB.  We assume
-- that the data register is defined as 'X DOWNTO Y'.
-------------------------------------------------------------

ENTITY dr IS
 PORT (shiftDR, clockDR, updateDR, mode, scan_in : IN BIT;
       scan_out : OUT BIT;
       data_in : IN BIT_VECTOR;
       data_out : OUT BIT_VECTOR);
END dr;
```

```
ARCHITECTURE structural OF dr IS
  COMPONENT dr_cell PORT (mode, data_in, shiftDR, scan_in, clockDR,
                          updateDR : IN BIT;
                          data_out, scan_out : OUT BIT);
  END COMPONENT;
  FOR ALL : dr_cell USE ENTITY WORK.dr_cell(behavioral);
  SIGNAL temp_scan : BIT_VECTOR (data_in'RANGE);
BEGIN
  a : FOR i IN data_in'LOW TO data_in'HIGH GENERATE
    b : IF (i = data_in'HIGH) GENERATE
      c : dr_cell PORT MAP (mode, data_in(i), shiftDR,
                            scan_in, clockDR, updateDR,
                            data_out(i), temp_scan(i));
    END GENERATE;
    d : IF ((i < data_in'HIGH) AND (i > 0)) GENERATE
      e : dr_cell PORT MAP (mode, data_in(i), shiftDR,
                            temp_scan(i+1), clockDR, updateDR,
                            data_out(i), temp_scan(i));
    END GENERATE;
    f : IF (i = 0) GENERATE
      g : dr_cell PORT MAP (mode, data_in(i), shiftDR,
                            temp_scan(i+1), clockDR, updateDR,
                            data_out(i), temp_scan(i));
    END GENERATE;
  END GENERATE;
  scan_out <= temp_scan(0);
END structural;
```

เรจิสเตอร์คำสั่ง

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

```
-- Instruction Register       P.M.Campbell       Nov. 21, 1991
--
-- This is an instruction register which is composed of a
-- chain of instruction register cells.  The instruction
-- register is described generically to allow any size register
-- to be generated.  Note that the register must contain at
-- least two stages and the last two stages must always load
-- a binary '01' value.  When the reset_bar signal is asserted,
-- each IR cell loads the value passed as the reset_value
-- parameter.  This allows the BYPASS instruction or IDCODE
-- value to be loaded when the Test-Logic-Reset state is
-- entered.
--
-- Note that the shift-register loads data into the instruction
-- register MSB and shifts data out of the LSB.  We assume
-- that the instruction register is defined as 'X DOWNTO Y'.
-------------------------------------------------------------------


ENTITY ir IS
  PORT (shiftIR, clockIR, updateIR, reset_bar, scan_in : IN BIT;
        scan_out : OUT BIT;
        data_in : IN BIT_VECTOR;
        data_out : OUT BIT_VECTOR);
END ir;

ARCHITECTURE structural OF ir IS
  COMPONENT ir_cell PORT (shiftIR, data_in, scan_in, clockIR,
                          updateIR, reset_bar, reset_value : IN BIT;
                          data_out, scan_out : OUT BIT);
  END COMPONENT;
  FOR ALL : ir_cell USE ENTITY WORK.ir_cell(behavioral);
```

```
  SIGNAL temp_scan : BIT_VECTOR (data_in'RANGE);
  SIGNAL vdd : BIT := '1';
  SIGNAL gnd : BIT := '0';
BEGIN
  a : FOR i IN data_in'LOW TO data_in'HIGH GENERATE
    b : IF (i = data_in'HIGH) GENERATE
      c : ir_cell PORT MAP (shiftIR, data_in(i), scan_in,
                    clockIR, updateIR, reset_bar, vdd,
                    data_out(i), temp_scan(i));
    END GENERATE;
    d : IF ((i < data_in'HIGH) AND (i > 1)) GENERATE
      e : ir_cell PORT MAP (shiftIR, data_in(i), temp_scan(i+1),
                    clockIR, updateIR, reset_bar, vdd,
                    data_out(i), temp_scan(i));
    END GENERATE;
    f : IF ((i = 1) AND (data_in'HIGH > 1)) GENERATE
      h : ir_cell PORT MAP (shiftIR, gnd, temp_scan(i+1),
                    clockIR, updateIR, reset_bar, vdd,
                    data_out(i), temp_scan(i));
    END GENERATE;
    j : IF (i = 0) GENERATE
      k : ir_cell PORT MAP (shiftIR, vdd, temp_scan(i+1),
                    clockIR, updateIR, reset_bar, vdd,
                    data_out(i), temp_scan(i));
    END GENERATE;
  END GENERATE ;
  scan_out <= temp_scan(0);
END structural;
```

ตัวควบคุมช่องทางเข้าถึงเพื่อทดสอบ

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

```
--
-- IEEE 1149.1 TAP Controller state machine
--
-- Peter M. Campbell, November 9, 1991
--
-- This is a state machine which implements the IEEE 1149.1 Test Access Port
-- (TAP) standard.  This is only the state machine for the TAP Controller.
-- This must be used with other devices to create the complete TAP.
--
-- One process is used to keep track of the current state of the FSM while
-- the other process is used to generate signals based upon the current state
-- of the FSM.
--
--------------------------------------------------------------------------------


ENTITY tap_controller IS
  PORT (tms, tck : IN BIT; reset_bar, select1, enable, shiftIR, clockIR,
        updateIR, shiftDR, clockDR, updateDR : OUT BIT);
END tap_controller;

ARCHITECTURE behavioral OF tap_controller IS
  TYPE state IS (test_logic_reset, run_test_idle, select_DR_scan,
                 capture_DR, shift_DR, exit1_DR, pause_DR, exit2_DR,
                 update_DR, select_IR_scan, capture_IR, shift_IR,
                 exit1_IR, pause_IR, exit2_IR, update_IR);
  TYPE state_hex IS ARRAY (state) OF BIT_VECTOR (3 DOWNTO 0);
  CONSTANT state_table : state_hex := ("1111","1100","0111","0110",
                                       "0010","0001","0011","0000",
                                       "0101","0100","1110","1010",
                                       "1001","1011","1000","1101");
```

```
  SIGNAL thestate : BIT_VECTOR (3 DOWNTO 0);
  SIGNAL current : state := test_logic_reset;
BEGIN
  PROCESS
  BEGIN
    CASE current IS
    WHEN test_logic_reset =>
      WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
      IF tms = '0' THEN current <= run_test_idle;
      ELSE current <= test_logic_reset;
      END IF;
    WHEN run_test_idle =>
      WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
      IF tms = '1' THEN current <= select_DR_scan;
      ELSE current <= run_test_idle;
      END IF;
    WHEN select_DR_scan =>
      WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
      IF tms = '0' THEN current <= capture_DR;
      ELSE current <= select_IR_scan;
      END IF;
    WHEN capture_DR =>
      WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
      IF tms = '0' THEN current <= shift_DR;
      ELSE current <= exit1_DR;
      END IF;
    WHEN shift_DR =>
      WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
      IF tms = '1' THEN current <= exit1_DR;
      ELSE current <= shift_DR;
      END IF;
```

```
WHEN exit1_DR =>
  WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
  IF tms = '0' THEN current <= pause_DR;
  ELSE current <= update_DR;
  END IF;
WHEN pause_DR =>
  WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
  IF tms = '1' THEN current <= exit2_DR;
  ELSE current <= pause_DR;
  END IF;
WHEN exit2_DR =>
  WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
  IF tms = '1' THEN current <= update_DR;
  ELSE current <= shift_DR;
  END IF;
WHEN update_DR =>
  WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
  IF tms = '0' THEN current <= run_test_idle;
  ELSE current <= select_DR_scan;
  END IF;
WHEN select_IR_scan =>
  WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
  IF tms = '0' THEN current <= capture_IR;
  ELSE current <= test_logic_reset;
  END IF;
WHEN capture_IR =>
  WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
  IF tms = '0' THEN current <= shift_IR;
  ELSE current <= exit1_IR;
  END IF;
WHEN shift_IR =>
```

```
    WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
    IF tms = '1' THEN current <= exit1_IR;
    ELSE current <= shift_IR;
    END IF;
  WHEN exit1_IR =>
    WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
    IF tms = '0' THEN current <= pause_IR;
    ELSE current <= update_IR;
    END IF;
  WHEN pause_IR =>
    WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
    IF tms = '1' THEN current <= exit2_IR;
    ELSE current <= pause_IR;
    END IF;
  WHEN exit2_IR =>
    WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
    IF tms = '1' THEN current <= update_IR;
    ELSE current <= shift_IR;
    END IF;
  WHEN update_IR =>
    WAIT UNTIL ((tck = '1') AND NOT tck'STABLE);
    IF tms = '0' THEN current <= run_test_idle;
    ELSE current <= select_DR_scan;
    END IF;
  END CASE;
  wait on current'TRANSACTION;
END PROCESS;

PROCESS (tck)
BEGIN
  IF ((tck = '0') AND tck'EVENT) THEN
```

```
    IF state_table(current) = "1111" THEN reset_bar <= '0';
    ELSE reset_bar <= '1'; END IF;
    IF state_table(current) = "1010" OR state_table(current) = "0010" THEN
      enable <= '1';
    ELSE
      enable <= '0';
    END IF;
    IF state_table(current) = "1010" THEN shiftIR <= '1';
    ELSE shiftIR <= '0'; END IF;
    IF state_table(current) = "0010" THEN shiftDR <= '1';
    ELSE shiftDR <= '0'; END IF;
  END IF;

  IF (tck = '0' AND (state_table(current) = "1110"
  OR state_table(current) = "1010")) THEN
    clockIR <= '0';
  ELSE
    clockIR <= '1';
  END IF;
  IF tck = '0' AND state_table(current) = "1101" THEN updateIR <= '1';
  ELSE updateIR <= '0'; END IF;
  IF (tck = '0' AND (state_table(current) = "0110"
  OR state_table(current) = "0010")) THEN
    clockDR <= '0';
  ELSE
    clockDR <= '1';
  END IF;
  IF tck = '0' AND state_table(current) = "0101" THEN updateDR <= '1';
  ELSE updateDR <= '0'; END IF;
END PROCESS;
```

```
    select1 <= state_table(current)(3);
    thestate <= state_table(current);

END behavioral;
```

อุปกรณ์รวมส่งสัญญาณแบบสองอินพุตต่อหนึ่งเอาท์พุต

```
ENTITY mux_2_1 IS
  PORT (a, b, g1 : IN BIT; z : OUT BIT);
END mux_2_1;

ARCHITECTURE behavioral OF mux_2_1 IS
BEGIN
  PROCESS (a, b, g1)
  BEGIN
    IF (g1 = '0') THEN z <= a;
    ELSIF (g1 = '1') THEN z <= b;
    END IF;
  END PROCESS;
END behavioral;
```

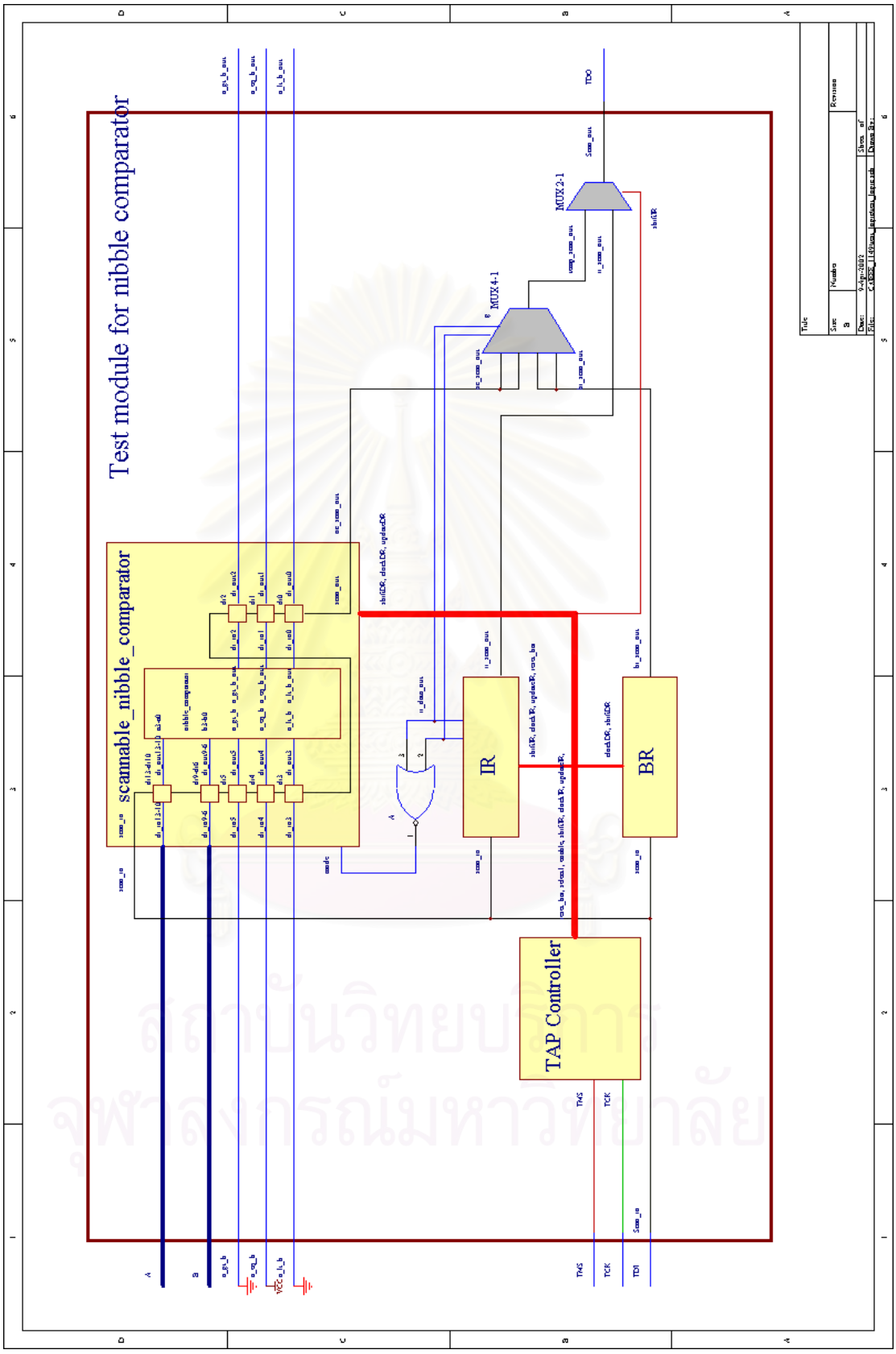
อุปกรณ์รวมส่งสัญญาณแบบสี่อินพุตต่อหนึ่งเอาท์พุต

```
        ENTITY mux_4_1 IS
          PORT (a, b, c, d : IN BIT;
              g : IN BIT_VECTOR (1 DOWNTO 0);
              z : OUT BIT);
        END mux_4_1;
```

```
ARCHITECTURE behavioral OF mux_4_1 IS
BEGIN
 PROCESS (a, b, c, d, g)
 BEGIN
  CASE g IS
   WHEN "00" => z <= a;
   WHEN "01" => z <= b;
   WHEN "10" => z <= c;
   WHEN "11" => z <= d;
  END CASE;
 END PROCESS;
END behavioral;
```

การประกอบตรรกะทดสอบเป็น บาวน์ดารี สแกน เรจิสเตอร์

จะยกตัวอย่างการประกอบตรรกะทดสอบเป็น บาวน์ดารี สแกน เรจิสเตอร์ ดังรูปที่ ก.1 ซึ่งสามารถนำไปประยุกต์ใช้สร้างบาวน์ดารี สแกน เรจิสเตอร์ ขึ้นมาโดยการประกอบสามารถเปลี่ยนแปลงได้เล็กน้อยขึ้นอยู่กับการออกแบบส่วนจำเพาะ สำหรับการทดสอบ โดยสามารถเพิ่มลดจำนวนขนาดของเรจิสเตอร์คำสั่ง ก็ขึ้นอยู่กับจำนวนคำสั่ง แต่ที่ยกตัวอย่างเป็น บาวน์ดารี สแกน เรจิสเตอร์ ที่สามารถทำงานได้ตามสถาปัตยกรรม บาวน์ดารี สแกน โดยใช้ตรรกะทดสอบจำนวนน้อยก็สามารถทำงานได้และขนาดของเรจิสเตอร์ข้อมูลก็ขึ้นอยู่กับจำนวนอินพุตเอาท์พุตของชิ้นส่วน



รูปที่ ข.1 ตัวอย่างการประกอบตรรกะทดสอบเป็น บาวน์ดารี สแกน เรจิสเตอร์

ประวัติผู้เขียนวิทยานิพนธ์

นายวิชชา เฟื่องจันทร์ เกิดเมื่อวันที่ 9 มกราคม พ.ศ. 2523 ที่จังหวัดขอนแก่น สำเร็จการศึกษาปริญญาวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น ในปีการศึกษา 2543 และศึกษาต่อในหลักสูตรวิทยาศาสตร์มหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2543

เคยได้รับรางวัลชนะเลิศการแข่งขันพัฒนาโปรแกรมคอมพิวเตอร์แห่งประเทศไทย ครั้งที่ 2