การตรวจสอบการทำงานของไมโครโพรเซสเซอร์โดยใช้เทคนิคการตรวจสอบสถานะนามธรรมและเทคนิคจำลองการทำงาน

FUNCTIONAL TESTING OF MICROPROCESSOR USING ABSTRACT STATE MACHINE AND SIMULATION-BASED TECHNIQUES

เจริญ วงศ์ชุ่มเย็น
CHAROEN WONGCHUMYEN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
จุฬาลงกรณ์มหาวิทยาลัย
พ.ศ. 2544

ISBN 974-648-347-1

การตรวจสอบการทำงานของไมโครโพรเซสเซอร์โดยใช้เทคนิคการตรวจสอบสถานะนามธรรมและเทคนิคจำลองการทำงาน

FUNCTIONAL TESTING OF MICROPROCESSOR USING ABSTRACT STATE MACHINE AND SIMULATION-BASED TECHNIQUES

เจริญ วงษ์ชุ่มเย็น

CHAROEN VONGCHUMYEN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2544

ISBN 974-648-347-1

FUNCTIONAL TESTING OF MICROPROCESSOR USING ABSTRACT STATE MACHINE AND SIMULATION-BASED TECHNIQUES

CHAROEN VONGCHUMYEN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2001

ISBN 974-648-347-1

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การตรวจสอบการทำงานของไมโครโพรเซสเซอร์โดยใช้เทคนิคการตรวจสอบสถานะนามธรรมและเทคนิคจำลองการทำงาน
FUNCTIONAL TESTING OF MICROPROCESSOR USING ABSTRACT STATE MACHINE AND SIMULATION-BASED TECHNIQUES

ชื่อนักศึกษา นายเจริญ วงษ์ชุ่มเย็น

รหัสประจำตัว 41061079

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.สมศักดิ์ มิตะถา

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ดร.วิศิษฏ์	หิรัญกิตติ	
รศ.ดร.บุญวัฒน์	อัตชู	
ผศ.ดร.เอื้อน	ปิ่นเงิน	
รศ.บรรจง	ปิยธำรง	
รศ.สมศักดิ์	มิตะถา	

วัน/เดือนปี ที่สอบ 1 พฤษภาคม 2544 **เวลา** 12.00-13.00 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-404)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.บุญวัฒน์ อัตชู)

คณบดีบัณฑิตวิทยาลัย

วันที่ 29 **เดือน** มิถุนายน **พ.ศ.** 2544

หัวข้อวิทยานิพนธ์	การตรวจสอบการทำงานของไมโครโพรเซสเซอร์โดยใช้เทคนิคการตรวจสอบสถานะนามธรรมและเทคนิคจำลองการทำงาน
นักศึกษา	นายเจริญ วงษ์ชุ่มเย็น
รหัสประจำตัว	41061079
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2544
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ. สมศักดิ์ มิตะถา

บทคัดย่อ

เนื้อหาของวิทยานิพนธ์เล่มนี้ ได้ประยุกต์ใช้การตรวจสอบเพื่อตรวจสอบการทำงานของไมโครโพรเซสเซอร์ที่ออกแบบโดยอ้างอิงการทำงานตามไมโครโพรเซสเซอร์ ARM7 ในส่วน User mode ด้วยภาษา VHDL ในงานวิจัยได้แบ่งการทดสอบออกเป็น 2 ระดับคือ การตรวจสอบสถานะการทำงานแบบนามธรรม (Abstract State Machine) เพื่อตรวจสอบความถูกต้องในระดับองค์ประกอบย่อย (Component) ซึ่งเป็นส่วนประกอบของโครงสร้างภายในไมโครโพรเซสเซอร์ และการจำลองการทำงาน (Simulation-based Technique) เพื่อทดสอบความถูกต้องของไมโครโพรเซสเซอร์ ซึ่งวิธีการที่ประยุกต์ใช้ในวิทยานิพนธ์นี้สามารถลดจำนวนรูปแบบข้อมูลที่ใช้ในการทดสอบให้น้อยลงได้ และจากการตรวจสอบความครอบคลุมการทำงานของโปรแกรม (Code Coverage) ผลปรากฏว่าสามารถทดสอบการทำงานได้ 91.9 เปอร์เซ็นต์ โดยวัดจากจำนวนบรรทัดของโปรแกรมที่มีผลกระทบจากชุดทดสอบทั้งหมดที่ป้อนเข้าไป 2,660 อินพุท และมีการทำงานที่ถูกต้องตามข้อกำหนดที่กำหนดไว้ (Specification)

Thesis Title	Functional Testing Of Microprocessor Using Abstract State Machine And Simulation-Based Techniques
Student	Mr. Charoen Vongchumyen
Student ID.	41061079
Degree	Master of Engineering
Programme	Electrical Engineering
Year	2001
Thesis Advisor	Assoc.Prof. Somsak Mitatha

ABSTRACT

The thesis applied the method of functional testing of the user mode of a designed microprocessor used ARM7 microporcessor as its model a VHDL langauge as its designing tool. In this work the tests are divided into 2 levels, i.e. component level and system level. At the component level me adopt an Abstract State Machine technique and the sytem level a Simulation-based technique. The abstract state machine technique is used to check for the correction of both design and function of each component of the microprocessor. The simulation-based technique is applied for testing the whole microprocessor. The benefit of this work is that it can reduce the number of test data. The test data generated by own approach can cover upto 91.9% computed form 2660 testing data samples input.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้เป็นอย่างดี ด้วยความช่วยเหลือของ รศ. สมศักดิ์ มิตะถา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งได้ให้คำแนะนำและคำปรึกษาในการทำวิทยานิพนธ์ฉบับนี้มาโดยตลอด รวมทั้งอาจารย์อีกหลายๆ ท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้คำปรึกษาและความรู้เพื่อใช้ในการทำวิทยานิพนธ์ฉบับนี้ ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์ของท่านและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณ นางสาวปัญจภา เรืองสินทรัพย์ที่ให้คำปรึกษาในการออกแบบไมโครโพรเซสเซอร์ และขอขอบคุณนางสาววรรณรัช สันติอมรทัต เป็นอย่างสูงที่ช่วยให้คำปรึกษาในหลายๆ เรื่อง ทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี

และสุดท้ายนี้ผู้วิจัยขอกราบขอบพระคุณ คุณแม่ผู้ให้กำเนิดและให้โอกาสทางการศึกษามาโดยตลอด รวมทั้งผู้ให้กำลังใจทุกๆ ท่านด้วย

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

เจริญ วงษ์ชุ่มเย็น

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

การออกแบบวงจรดิจิตอล (logic design) ถือเป็นวิวัฒนาการทางเทคโนโลยีที่เด่นที่สุดในรอบ 30 ปีหลังจากทรานซิสเตอร์ซึ่งปรากฏต่อชาวโลกเมื่อปี พ.ศ. 2491 ซอฟต์แวร์และเครื่องมือที่ช่วยในการออกแบบได้ถูกพัฒนาอย่างต่อเนื่อง จนสามารถใช้งานได้ง่าย สะดวกและมีประสิทธิภาพมากขึ้น ทำให้การออกแบบวงจรดิจิตอลขนาดใหญ่และซับซ้อนอย่างเช่น ไมโครโพรเซสเซอร์ หรือไมโครคอนโทรลเลอร์ สามารถทำได้บนเครื่องคอมพิวเตอร์ส่วนบุคคล ดังนั้นการออกแบบและพัฒนางานทางด้านนี้จึงเป็นไปได้อย่างรวดเร็ว

ไมโครโพรเซสเซอร์ได้ถูกนำมาใช้งานเพิ่มมากขึ้น เพื่อเพิ่มประสิทธิภาพในการทำงานของอุปกรณ์อิเล็กทรอนิกส์ ไมโครโพรเซสเซอร์ที่ชื่อว่า ARM 7 เป็นหนึ่งในไมโครโพรเซสเซอร์ที่ได้รับความนิยมอย่างมากซึ่งผลิตโดยบริษัท Advance RISC Machine Co., Ltd. เป็นไมโครโพรเซสเซอร์ ที่มีดาต้าพาร์ทขนาด 32 บิต โครงสร้างสถาปัตยกรรมแบบ RISC (Reduced Instruction Set Computer) ซึ่งแตกต่างจากสถาปัตยกรรมแบบ CISC (Complex Instruction Set Computer) ตรงที่การใช้ชุดคำสั่งที่ง่ายกว่า ทำให้จำนวนชุดคำสั่งที่ใช้ในการประมวลผลมีน้อยลง นอกจากนี้ไมโครโพรเซสเซอร์ ARM 7 มีโครงสร้างสถาปัตยกรรมของการทำโหลดและเก็บค่าที่เป็นการแลกเปลี่ยนข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์ได้มากกว่าหรือเท่ากับ 1 ชุดข้อมูลภายในคำสั่งเดียวกัน ไมโครโพรเซสเซอร์ ARM 7 มีขนาดเล็กแต่มีชุดคำสั่งที่มีประสิทธิภาพสูงทำให้ ARM 7 ถูกใช้ในงานอาทิเช่น โทรศัพท์มือถือ สมาร์ตการ์ด หรือ เครื่องรับจานดาวเทียม นอกจากนี้ฟังก์ชันการทำงานที่ประหยัดพลังงานทำให้ไมโครโพรเซสเซอร์ ARM 7 ถูกใช้ในเครื่องคอมพิวเตอร์ขนาดพกพาหรือเครื่องเซิฟเวอร์ ที่มีการใช้งานตลอดเวลา

1.1 ความสำคัญและความเป็นมาของปัญหา

ปัญหาสำคัญอย่างหนึ่งของการออกแบบและพัฒนาไมโครโพรเซสเซอร์คือ การตรวจสอบและความมั่นใจในประสิทธิภาพการทำงานของไมโครโพรเซสเซอร์ที่ออกแบบ ดังนั้นงานวิจัยทางด้านการทดสอบและการตรวจสอบจึงเป็นจุดสำคัญของงานทางด้านการออกแบบอีกส่วนหนึ่ง การจำลองการทำงานซึ่งเป็นที่นิยมใช้กันมากในการตรวจสอบความถูกต้องของระบบที่ออกแบบ โดยการสร้างชุดทดสอบเพื่อจำลองการทำงานและตรวจหาข้อผิดพลาดที่จะเกิดขึ้น แต่การจำลองการทำงานไม่สามารถรับประกันความถูกต้องทั้งหมดของระบบได้ ดังนั้นจึงมีการนำเสนอการตรวจสอบโดยใช้การตรวจสอบสถานะนามธรรม ซึ่งเป็นหนึ่งในหลายๆ วิธีการที่ถูกนำมาประยุกต์ใช้ในการตรวจสอบและยืนยันความถูกต้องของระบบที่ออกแบบ แต่สิ่งสำคัญของการประยุกต์ใช้งาน

คือการนำวิธีการดังกล่าวมาพัฒนาเพื่อให้ใช้งานได้จริงกับโครงสร้างและระบบที่มีอยู่เดิมได้อย่างไร ความเหมาะสมและเป้าหมายของการใช้งานวิธีการตรวจสอบสถานะ มีมากน้อยอย่างไร ปัจจัยเหล่านี้ต้องถูกนำมาพิจารณาในการเลือกและประยุกต์ใช้งานตรวจสอบดังกล่าว

การออกแบบวงจรระบบดิจิตอลด้วยภาษาในระดับสูง เช่น VHDL (Very High Speed Integreated Circuit Hardware Description Languages), Verilog, AHDL (Altera Hardware Description Language) เป็นที่นิยมและมีการใช้งานกันเป็นจำนวนมาก เนื่องจากสามารถที่จะพัฒนาระบบดิจิตอลที่ต้องการได้อย่างรวดเร็ว และซอฟต์แวร์ที่ช่วยสนับสนุนงานทางด้านนี้บนเครื่องคอมพิวเตอร์ส่วนบุคคลมีประสิทธิภาพมากขึ้น ซึ่งเป็นการช่วยให้สามารถพัฒนาระบบดิจิตอลเช่น ไมโครโพรเซสเซอร์ หรือไมโครคอนโทรลเลอร์ ได้ภายในเวลาอันสั้น

ดังนั้นงานวิจัยนี้จึงเป็นการนำเสนอการตรวจสอบไมโครโพรเซสเซอร์ โดยได้เลือกเอาไมโครโพรเซสเซอร์ที่อ้างอิงการทำงานตามไมโครโพรเซสเซอร์ ARM7 ในส่วนของ User mode มาเป็นตัวอย่างในการตรวจสอบ เพื่อเป็นแนวทางในการนำการตรวจสอบไมโครโพเซสเซอร์นี้ไปใช้พัฒนาในออกแบบไมโครโพเซสเซอร์ที่มีใช้ในอุปกรณ์อิเล็กทรอนิกส์ต่างๆ มากมายเช่น โทรศัพท์มือถือ, Smart Card, GPS เป็นต้น โดยเป็นการตรวจสอบไมโครโพรเซสเซอร์ที่ได้ออกแบบให้มีความถูกต้องสูงสุดโดยใช้ Simulation-based technique และ Abstract State Machine ในการตรวจสอบ

แต่เนื่องจากการตรวจสอบการทำงานเพื่อยืนยันและเพิ่มความน่าเชื่อถือในระบบที่ได้ทำการออกแบบ จะเป็นส่วนที่ทำให้เกิดค่าใช้จ่าย และระยะเวลา (time-to-market) เพิ่มสูงขึ้น งานวิจัยนี้จึงได้นำเสนอวิธีการประยุกต์การตรวจสอบเพื่อยืนยันความถูกต้องในการทำงานของไมโครโพรเซสเซอร์ที่ออกแบบใน 2 ลักษณะดังนี้

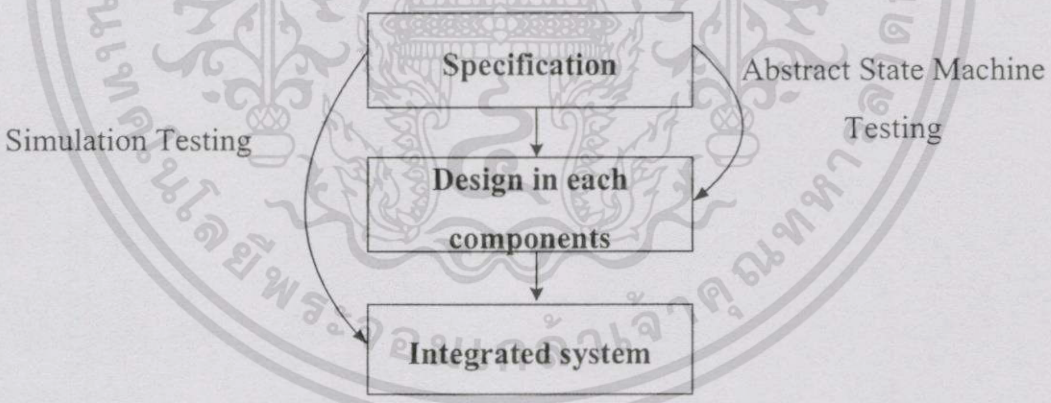
1.1.1 การตรวจสอบสถานะการทำงานนามธรรม (Abstract State Machine)

จะเป็นการนำหลักการของการตรวจสอบวงจรดิจิตอลมาประยุกต์ใช้งานในการยืนยันความถูกต้องของไมโครโพรเซสเซอร์ ซึ่งจะช่วยทำให้สามารถลดระยะเวลาและค่าใช้จ่ายในกระบวนการตรวจสอบและเป็นการตรวจสอบประสิทธิภาพของวงจรที่ออกแบบได้อีกด้วย งานวิจัยนี้จะตรวจสอบระบบโดยการตรวจสอบสถานะการทำงานแบบนามธรรม ซึ่งเป็นการตรวจสอบคุณสมบัติของการทำงาน โดยแทนวงจรที่ออกแบบให้อยู่รูปแบบที่เป็นแบบแผน เพื่อให้สามารถใช้ขั้นตอนวิธี (Algorithm) ในการตรวจสอบเฉพาะคุณสมบัติที่ต้องการตรวจสอบเท่านั้น ในงานวิจัยนี้เลือกใช้เทคนิคของ Abstract State Machine ในการตรวจสอบโครงสร้างไมโครโพรเซสเซอร์ แล้วตรวจสอบด้วยอัลกอลิทึมเพื่อเปรียบเทียบกับคุณลักษณะของข้อกำหนดที่มี

1.1.2 การตรวจสอบโดยการจำลองการทำงาน (Simulation-based Verification)

การตรวจสอบในลักษณะนี้ นักออกแบบจำเป็นต้องใช้การเขียนชุดทดสอบ (Test vectors) เพื่อใช้ในการตรวจสอบขึ้นมาเอง งานวิจัยนี้จะมีการเก็บคำสั่งที่ต้องการตรวจสอบอยู่ในลักษณะของหน่วยความจำ เพื่อให้ไมโครโพรเซสเซอร์สามารถเรียกคำสั่งเข้าไปทำงาน และแสดงผลลัพธ์ที่ได้ โดยการเขียนการตรวจสอบในลักษณะของชุดทดสอบการทำงาน (Test Bench) ด้วยภาษา VHDL เข้าไปทำงานในซอฟต์แวร์ที่ใช้ในการจำลองการทำงาน ซึ่งชุดทดสอบจะได้มาจากการวิเคราะห์โครงสร้างและการทำงานของไมโครโพรเซสเซอร์เพื่อให้ชุดทดสอบสามารถทำหน้าที่แทนชุดทดสอบที่สามารถเป็นไปได้ทั้งหมดของระบบ

ข้อแตกต่างของการใช้งานวิธีการอย่างมีแบบแผน และการจำลองการทำงาน คือ การใช้ วิธีการอย่างมีแบบแผนในการตรวจสอบวงจรจะสามารถตรวจสอบการทำงานได้ทั้งหมด ร่วมถึงประสิทธิภาพ ของวงจรที่ออกแบบได้ ส่วนการจำลองการทำงานจะเป็นการตรวจสอบได้เฉพาะผลลัพธ์เท่านั้น แต่เหมาะสำหรับการตรวจสอบการทำงานที่มีค่าของเวลาเข้ามาเกี่ยวข้อง (Timing Constant)



รูปที่ 1.1 การตรวจสอบอย่างมีแบบแผนและการจำลองการทำงานในกระบวนการออกแบบ

จากรูปที่ 1.1 หลังจากทำการออกแบบไมโครโพรเซสเซอร์ 32 บิต ARM 7 มีการนำวิธีการอย่างมีแบบแผนมาใช้เพื่อตรวจสอบความถูกต้องของวงจรในระดับ RTL (Register Transfer Language) ของแต่ละชิ้นส่วนย่อยเช่น ALU, shifter หรือ multiply เป็นต้น กับคุณลักษณะที่กำหนดเนื่องจากการใช้ วิธีการอย่างมีแบบแผนจะสามารถตรวจสอบทุกจุดของวงจร รวมทั้งประสิทธิภาพของวงจรที่ออกแบบ โดยวิธีการอย่างมีแบบแผนเป็นวิธีการตรวจสอบที่เหมาะสมกับวงจรที่มีขนาดไม่ใหญ่,

ไม่ซับซ้อน และไม่เกี่ยวกับเวลา (Timing) ดังนั้นจึงเลือกใช้วิธีการอย่างมีแบบแผนในการตรวจสอบระหว่างคุณลักษณะที่ต้องการกับแต่ละชิ้นส่วนของไมโครโพรเซสเซอร์ในระดับ RTL แต่เนื่องจากไมโครโพรเซสเซอร์เป็นระบบที่มีความซับซ้อนและมีขนาดใหญ่ และสังเคราะห์วงจรในระดับเกตซึ่งจะเกี่ยวข้องกับเวลาในการทำงาน ดังนั้นจึงใช้วิธีการจำลองการทำงานในการตรวจสอบความถูกต้องในการทำงานของระบบทั้งหมด

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

- เพื่อให้มีความรู้และความเข้าใจการออกแบบไมโครโพรเซสเซอร์ ด้วยภาษาระดับสูง
- เพื่อตรวจสอบความถูกต้องในระดับของแต่ละชิ้นส่วน ด้วยการนำหลักการของการตรวจสอบอย่างมีแบบแผนมาใช้งาน
- เพื่อยืนยันความถูกต้องของวงจรไมโครโพรเซสเซอร์ 32 บิต ARM7 ที่ออกแบบด้วยภาษา VHDL ในระดับของการตรวจสอบการทำงาน
- เพื่อเป็นแนวทางในการสร้างเครื่องมือในการตรวจสอบวงจรที่ออกแบบด้วยภาษาระดับสูง (HDL) โดยการใช้วิธีการตรวจสอบอย่างมีแบบแผนต่อไปในอนาคต

1.3 ขอบเขตของการวิจัย

- ตรวจสอบโดยใช้วิธีการอย่างมีแบบแผนกับไมโครโพรเซสเซอร์ 32 บิต ARM 7 ที่ออกแบบมา ในส่วนของแต่ละชิ้นส่วนด้วยวิธีการ Abstract State Machine ที่เป็นโครงสร้างของไมโครโพรเซสเซอร์
- ทดสอบไมโครโพรเซสเซอร์ 32 บิต ARM 7 ที่ออกแบบมา ให้มีความถูกต้องในการทำงานด้วยวิธีการจำลองการทำงานโดยใช้ชุดทดสอบการทำงาน (Test Bench)

1.4 ขั้นตอนการทำวิจัย

- นำโครงสร้างของไมโครโพรเซสเซอร์ในส่วนของแต่ละชิ้นส่วน โดยใช้หลักการของวิธีการอย่างมีแบบแผน มาตรวจสอบการทำงาน เพื่อเปรียบเทียบองค์ประกอบย่อยที่ได้จากการออกแบบในระดับ RTL กับคุณลักษณะที่ต้องการ (Specification) ที่แทนด้วย Abstract State Machine
- ใช้วิธีการจำลองการทำงานเพื่อทดสอบการทำงานในระดับเกตที่มีส่วนของค่าคงที่ของเวลา (Timing Constant) เข้ามาเกี่ยวข้อง ให้ตรงกับคุณลักษณะที่ต้องการ
- สร้างชุดทดสอบโดยใช้วิธีการวิเคราะห์โครงสร้างของไมโครโพรเซสเซอร์เพื่อเป็นตัวแทนการทำงานของไมโครโพรเซสเซอร์ในระดับของชุดคำสั่งการทำงานทั้งหมด

- นำชุดทดสอบผ่านโปรแกรมตรวจสอบจุดบกพร่อง (Debugger) ของ ARM 7 ซึ่งใช้แทนคุณลักษณะ (Specification) ของไมโครโพรเซสเซอร์ที่ออกแบบ
- นำชุดทดสอบชุดเดิมผ่านการจำลองการทำงานในไมโครโพรเซสเซอร์ที่ออกแบบมา และนำผลลัพธ์ที่ได้เปรียบเทียบผลจากโปรแกรมตรวจสอบจุดบกพร่อง

โดยรายละเอียดต่างๆ ภายในวิทยานิพนธ์ฉบับนี้ได้จัดแบ่งในส่วนของเนื้อหาทั้งหมดออกเป็น 5 บท ซึ่งแต่ละบทจะมีหัวข้อและเนื้อหาดังต่อไปนี้

บทที่ 1 บทนำ

อธิบายถึงความเป็นมาและความสำคัญของปัญหาตลอดจนวัตถุประสงค์ของขอบเขตวิทยานิพนธ์ และงานวิจัยอื่นๆ ที่เกี่ยวข้อง

บทที่ 2 หลักการตรวจสอบวงจรดิจิตอล

กล่าวถึงทฤษฎีของการทดสอบและการตรวจสอบวงจรดิจิตอลในเบื้องต้น รวมทั้งอธิบายวิธีการตรวจสอบโดยใช้เทคนิคของ Abstract State Machine และการตรวจสอบความถูกต้องของวงจรดิจิตอลด้วยการจำลองการทำงาน

บทที่ 3 การทำงานของไมโครโพรเซสเซอร์ ARM7

กล่าวถึงการทำงานพื้นฐานของไมโครโพรเซสเซอร์ ARM7 ชุดคำสั่ง โหมดการทำงาน และการออกแบบไมโครโพรเซสเซอร์ดังกล่าว โดยแยกเป็นองค์ประกอบย่อย รวมทั้งผลของการสังเคราะห์วงจรจากองค์ประกอบย่อยต่างๆ ที่ได้ทำการออกแบบ

บทที่ 4 การตรวจสอบด้วยเทคนิคของ Abstract State Machine และการจำลองการทำงาน

กล่าวถึงการตรวจสอบองค์ประกอบย่อยต่างๆ ของไมโครโพรเซสเซอร์ที่ได้ทำการออกแบบ โดยใช้เทคนิคของ Abstract State Machine รวมทั้งผลลัพธ์ที่ได้จากการตรวจสอบนั้น รวมถึงการตรวจสอบไมโครโพรเซสเซอร์ที่ได้ทำการออกแบบ โดยใช้วิธีการจำลองการทำงาน ด้วยการสร้างแบบทดสอบ (Test Case) แล้วจำลองการทำงานโดยใช้โปรแกรม V-System เพื่อตรวจสอบความถูกต้องของระบบเมื่อมีการรวมองค์ประกอบย่อยต่างๆ เข้าด้วยกัน รวมทั้งผลลัพธ์ที่ได้จากการจำลองการทำงานนั้น

บทที่ 5 สรุปผลการวิจัยและข้อเสนอแนะ

สรุปและวิจารณ์ผลที่ได้จากการทดลองพร้อมทั้งปัญหาที่เกิดขึ้นตลอดจนข้อเสนอแนะต่างๆ ในการทำวิจัยต่อไป

1.5 งานวิจัยที่เกี่ยวข้อง

James K. Huggins, David Van Campenhout (1998) ได้นำเสนอผลงานวิจัยในหัวข้อเรื่อง "Specification and Verification of Pipelining in the ARM2 RISC" โดยเป็นการเสนอวิธีการทวนสอบไมโครโปรเซสเซอร์ ARM 2 ในส่วนการทำงานของไปป์ไลน์ด้วยวิธีของ Abstract State Machine (ASM) โดยการสร้างองค์ประกอบย่อย 2 ชนิดคือส่วนของการทำงานแบบตามลำดับและส่วนของไปป์ไลน์แล้วเปรียบเทียบผลที่ได้จากทั้ง 2 องค์ประกอบย่อยที่สร้างขึ้นในการทวนสอบการทำงาน

H. AI Asaad, D. Van Campenhout, J.P. Hayes, T. Mudge and R.B. Brown (1997) ได้นำเสนอผลงานวิจัยในหัวข้อเรื่อง "High Level Design verification of microprocessor via error modeling." เป็นการนำเสนอการทวนสอบไมโครโพรเซสเซอร์ด้วยวิธีการจำลองการทำงาน ซึ่งใช้หลักการของการหาข้อผิดพลาดในไมโครโพรเซสเซอร์ โดยสร้างอินพุทที่มีผลทำให้การทำงานผิดพลาดเข้าสู่ไมโครโปรเซสเซอร์ให้จำลองการทำงาน เพื่อตรวจผลลัพธ์ที่ได้ โดยได้กล่าวไว้ว่า *"การจำลองการทำงานเหมาะที่สุดสำหรับระบบที่ซับซ้อนและมีขนาดใหญ่อย่างเช่นไมโครโปรเซสเซอร์"*

M. Kentrowitz, L.M. Noack (1996) ได้นำเสนอผลงานวิจัยที่ชื่อ "Verification Coverage Analysis and Correctness Checking of the DECChip 21164 Alpha microprocessor." ซึ่งเป็นการนำเสนอแนวการเลือกชุดทดสอบเพื่อจำลองการทำงานของไมโครโปรเซสเซอร์ในตระกูล Alpha รุ่น 21164 ซึ่งเป็นไมโครโพรเซสเซอร์ที่มีขนาดใหญ่ และซับซ้อนไม่เหมาะกับการทวนสอบโดยการพิสูจน์ด้วยวิธีการทางคณิตศาสตร์ โดยการหาตัวแทนของกลุ่มคำสั่งเพื่อเป็นอินพุทให้กับระบบในการจำลองการทำงานและตรวจผลลัพธ์ที่ได้

1.6 ผลงานที่ตีพิมพ์จากงานวิจัย

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์เป็นบทความทางวิชาการ และนำเสนอในที่ประชุมวิชาการต่างๆ ดังนี้

1.6.1 เจริญ วงษ์ชุ่มเย็น, สมศักดิ์ มิตะถา, วรรณรัช สันติอมรทัต, อาทิตย์ ทองทักษ์, ***"การทวนสอบไมโครโพรเซสเซอร์ ARM7 ที่ออกแบบด้วยภาษา VHDL."***, วารสารลาดกระบัง, มิถุนายน, 2542.

บทที่ 2
หลักการตรวจสอบวงจรดิจิตอล

การออกแบบวงจรรวมเพื่อให้ได้การทำงานตามที่ต้องการเป็นสิ่งที่ยากอย่างหนึ่ง การออกแบบระบบที่ซับซ้อนทั้งฮาร์ดแวร์ ซอฟต์แวร์ หรือทั้งสองอย่างเพื่อให้ระบบทำงานได้อย่างถูกต้องจึงเป็นสิ่งที่ยากยิ่งกว่า ดังนั้นงานหลักของนักออกแบบจึงมี 2 อย่างคือ การออกแบบวงจรตามขั้นตอนและการตรวจสอบความถูกต้อง โดยต้องแน่ใจว่าสิ่งที่ได้ออกแบบไปแล้วนั้นสามารถทำงานได้อย่างถูกต้อง การออกแบบและการวิเคราะห์จึงถูกใช้ควบคู่กันไปเกือบทุกขั้นตอนของการออกแบบ การจำลองการทำงานด้วยเหตุการณ์ที่กำหนดขึ้นเพื่อตรวจสอบวงจรให้มากที่สุดจึงเป็นสิ่งที่ต้องกระทำก่อนที่จะเข้าสู่กระบวนการผลิต เนื่องจากถ้ามีข้อผิดพลาดที่เกิดขึ้นในวงจรจะทำให้ค่าใช้จ่ายในเจือสาร (Fabrication) ซึ่งเป็นขั้นตอนการผลิตสูงมากขึ้น ในบทนี้เป็นการกล่าวถึง การกำหนดคุณลักษณะของวงจรดิจิตอลที่ต้องการออกแบบอย่างมีแบบแผน(Formal specification) การตรวจสอบด้วยวิธี Abstract State Machine และการตรวจสอบด้วยวิธีการจำลองการทำงาน (Simulation-Based Technique) เพื่อใช้ในการตรวจสอบและทดสอบการทำงานของไมโครโพรเซสเซอร์ที่ออกแบบให้มีความถูกต้องในการทำงานมากที่สุด

2.1 การกำหนดคุณลักษณะ (Specification)

ทฤษฎีขั้นพื้นฐานของการออกแบบวงจรดิจิตอลให้ถูกต้องนั้นคือความสามารถในการกำหนดคุณลักษณะและอธิบายพฤติกรรมของวงจรนั้นๆ ซึ่งภาษาที่ใช้ในการอธิบายวงจรขึ้นอยู่กับระดับของวงจรที่ต้องการออกแบบ โดยเทคนิคในการอธิบายการทำงานของวงจรสามารถมีได้หลากหลายวิธีเช่น การวาดวงจร (Schematics), ไดอะแกรมแสดงเวลาในการทำงาน (Timing diagram) และใช้ภาษาในการอธิบายการทำงาน (Description language) โดยภาษาที่ใช้ในการอธิบายการทำงานของวงจรจะต้องครบตามกฎของการกำหนดคุณลักษณะซึ่งมีอยู่ 3 ข้อได้แก่

- **สามารถสื่อถึงความต้องการในวงจรที่ต้องการออกแบบได้**

  ภาษาที่ใช้ในการอธิบายการทำงานของวงจรจะต้องสามารถสื่อถึงความต้องการของนักออกแบบในการออกแบบวงจรว่าต้องการให้วงจรที่ออกแบบมาทำหน้าที่อย่างไร มีการทำงานอย่างไรบ้าง รวมทั้งสามารถระบุข้อกำหนดต่างๆ ของวงจรนั้นๆ ได้

- **สามารถใช้ในการตรวจสอบวงจรในขั้นต้น**

  ภาษาที่ใช้ในการอธิบายการทำงานของวงจรนอกจากจะต้องสามารถอธิบายถึงการทำงานของวงจรได้แล้วยังจะต้องสามารถทำหน้าที่เป็นตัวกำหนดคุณลักษณะการทำงานขอ

งวงจรเพื่อใช้ในการตรวจสอบการทำงานของวงจรได้ด้วย เนื่องจากเมื่อนักออกแบบสามารถออกแบบวงจรนั้นขึ้นมาได้ จำเป็นอย่างยิ่งที่ต้องมีการตรวจสอบความถูกต้องของการทำงานของวงจรนั้นๆ โดยเปรียบเทียบกับคุณลักษณะที่ต้องการของวงจรนั้นๆ

- **สามารถเชื่อมต่อเข้ากับอุปกรณ์การออกแบบ**

  นอกจากนั้นภาษาที่ใช้ในการอธิบายการทำงานของวงจรยังจะต้องสามารถอธิบายถึงการเชื่อมต่อระหว่างองค์ประกอบย่อยแต่ละส่วนของการออกแบบนั้นๆ ได้เพื่อให้นักออกแบบสามารถออกแบบวงจรดังกล่าวโดยการแบ่งส่วนการออกแบบเป็นส่วนย่อยๆ เพื่อให้ง่ายต่อการทำความเข้าใจและการออกแบบ

โดยส่วนมากสื่อใช้ที่แสดงถึงคุณลักษณะทางโครงสร้าง (Structure) ของวงจรที่ต้องการออกแบบ ได้แก่ ไดอะแกรมทางเรขาคณิต และทฤษฎีกราฟ (Graph theory) ส่วนการอธิบายพฤติกรรมการทำงานจะใช้บูลีนอัลจีบรา (Boolean algebra) และภาษาในการเขียนโปรแกรม ส่วนสุดท้ายที่สามารถอธิบายข้อมูลเกี่ยวกับเวลาที่ใช้ในการทำงานจะใช้ไดอะแกรมแสดงเวลาที่เป็นรูปคลื่นสัญญาณ แต่ไม่ว่าจะเป็นการกำหนดคุณลักษณะแบบใดก็ตามนักออกแบบมักจะใช้ภาษาธรรมชาติ (ภาษาอังกฤษ) เป็นหลักในการกำหนดคุณลักษณะ และปัญหาหลักที่พบในการกำหนดคุณลักษณะคือความกำกวมของสื่อที่ใช้ หรือความไม่สมบูรณ์ของคุณลักษณะซึ่งในวิทยานิพนธ์นี้ไม่เกี่ยวข้องกับการตรวจสอบหรือทดสอบคุณลักษณะที่กำหนด แต่จะใช้ภาษาที่ใช้การกำหนดคุณลักษณะที่มีใช้อยู่แล้วมาเป็นตัวกำหนดคุณลักษณะของไมโครโพรเซสเซอร์ที่ต้องการออกแบบ ตัวอย่างของคุณลักษณะที่กำหนดไว้แสดงได้ดังรูปที่ 2.1

```
Rule : ExecuteALU
    If  ExecuteOK and ALUInstr(Instr)  then
        If  Satisfies(Status, CondCode(Instr))  then
            If  WriteResult(Instr)  then
                Contents(DesReg)  := ALU( ALUop(Instr), Aop, Bop, Carry(Status))
            Endif
            If  SetCondCode(Instr)  then
                Status := UpdateStatus( Status, ALUop(Instr), Aop, Bop, ShiftCarryOp)
            Endif
        Endif
Endif
```

Specification :

Rule a : ExecuteOK and ALUInstr(Instr)

Rule b : Satisfies(Status, CondCode(Instr))

Rule c : WriteResult(Instr)

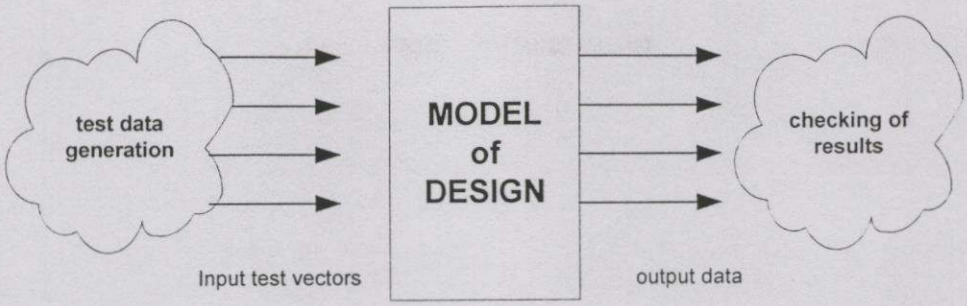
Rule d : SetCondCode(Instr)

รูปที่ 2.1 ตัวอย่างของคุณลักษณะที่กำหนด (Specification)

จากคุณลักษณะของข้อกำหนดในรูปที่ 2.1 แสดงให้เห็นถึงข้อกำหนดในการทำงานของส่วน ALU โดยในส่วนของ Specification จะเป็นกฎในการกำหนดการทำงานของระบบ ว่าเมื่อเงื่อนไขใดเป็นจริง จึงจะทำงานเช่น If ExecuteOK and ALUInstr(Instr) then หมายถึงหากสัญญาณหรือข้อกำหนดในการทำงานเป็นไปตาม Rule a : คือ มีสัญญาณ ExecuteOK และมีข้อมูลอยู่ใน ALUInstr จริงจะสามารถเข้าไปทำงานในบรรทัดของ If Satifies(Status, CondCode(Instr)) then ได้ และในทำนองเดียวกันกันหากเงื่อนไขในการทำงานเป็นไปตาม Rule b : คือ มีเงื่อนไขในการทำงานที่ถูกต้อง Satisfies(Status, CondConde(Instr)) จึงจะสามารถไปทำงานในส่วนถัดไปได้ คือ If WriteResult(Instr) then และหากข้อกำหนดการทำงานเป็นไปตาม Rule c : ระบบถึงจะสามารถทำการเขียนข้อมูลได้ Contents(DesReg) := ALU(ALUop(Instr), Aop, Bop, Carry(Status)) นั่นคือหากระบบจะสามารถเขียนข้อมูลลง Contecs(DesReg) ได้ จะต้องมีการทำงานตาม Rule a, b, c ตามลำดับก่อน และหากจะมีการแก้ไขค่า Status ก็ต้องมีการทำงานตาม Rule a, b, d ก่อนตามลำดับ

2.2 การตรวจสอบและการทดสอบการทำงานของไมโครโพรเซสเซอร์

ในการตรวจสอบและทดสอบความถูกต้องของการทำงานของไมโครโพรเซสเซอร์ในปัจจุบันส่วนใหญ่จะใช้วิธีการจำลองการทำงานเป็นหลัก เนื่องจากสามารถทำได้ง่ายและสามารถกระทำได้ในทุกขั้นตอนของการออกแบบไม่ว่าจะเป็นส่วนองค์ประกอบย่อยของวงจรหรือแม้กระทั่งเมื่อมีการรวมระบบทั้งหมดเข้าด้วยกัน แต่ปัญหาของการตรวจสอบโดยใช้วิธีการจำลองการทำงานแต่เพียงอย่างเดียวคือ ผู้ตรวจสอบไม่สามารถจะทำการสร้างชุดทดสอบทั้งหมดที่เป็นไปได้ของระบบเพื่อมาตรวจสอบวงจรได้เนื่องจากมีจำนวนรูปแบบของชุดทดสอบจำนวนมหาศาล เทคนิคนึงที่ช่วยแก้ปัญหานี้คือการวิเคราะห์ระบบและสร้างชุดทดสอบที่เป็นตัวแทนของชุดทดสอบทั้งหมดที่สามารถเป็นไปได้ วิธีการนี้สามารถที่จะตรวจสอบระบบได้อย่างมีประสิทธิภาพและประหยัดเวลาและค่าใช้จ่ายลง



รูปที่ 2.2 แนวความคิดของการตรวจสอบด้วยวิธีการจำลองการทำงาน

ซึ่งการจำลองการทำงานเป็นวิธีการที่นักออกแบบนิยมใช้มากในการตรวจสอบการทำงานของวงจรที่ได้ออกแบบก่อนที่จะมีการส่งเข้าสู่กระบวนการผลิตจริง โดยนักออกแบบต้องสร้างชุดทดสอบเพื่อป้อนให้กับวงจรที่ต้องการตรวจสอบ ผลลัพธ์ที่ได้จะถูกแสดงออกมาด้วยรูปแบบที่แตกต่างกัน ทั้งนี้ขึ้นกับซอฟต์แวร์ที่ใช้และการออกแบบวงจรในแต่ละระดับการจำลองการทำงานก็จะแตกต่างกัน ซึ่งแนวความคิดของการจำลองการทำงานสามารถแทนได้ด้วยแผนภาพดังรูปที่ 2.2 แต่การจำลองการทำงานก็มีข้อเสียที่ว่า หากวงจรที่ออกแบบมีขนาดใหญ่และซับซ้อนก็จะทำให้เสียเวลาในการตรวจสอบมาก

ปัญหาที่ตามมาคือหากผู้สร้างชุดทดสอบวิเคราะห์ระบบไม่ถูกต้องหรือไม่เข้าใจระบบอย่างแท้จริงจะไม่สามารถสร้างชุดทดสอบที่มีประสิทธิภาพได้เลย และถึงแม้ว่าจะสามารถสร้างชุดทดสอบที่มีประสิทธิภาพขึ้นมาได้ก็ไม่สามารถรับประกันได้ว่ารูปแบบอื่นที่ไม่ได้ตรวจสอบจะสามารถทำงานได้ถูกต้องจริง ดังนั้นงานวิทยานิพนธ์นี้ จึงได้ประยุกต์วิธีการตรวจสอบสองแบบคือ การตรวจสอบโดยใช้การตรวจสอบสถานะการทำงานนามธรรม (Abstract State machine) และการตรวจสอบโดยใช้การจำลองการทำงาน (Simulation-Based)

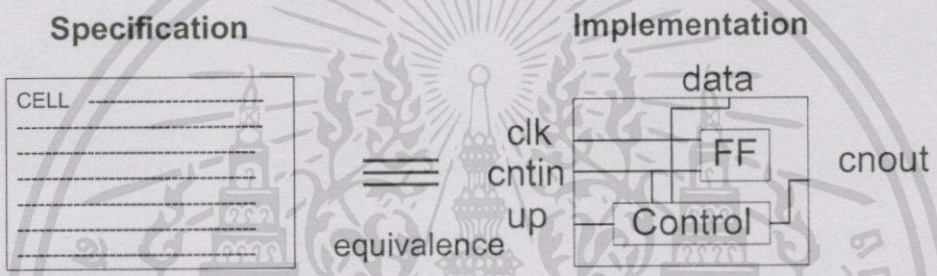
จากการผสมผสานแนวทางการตรวจสอบและการทดสอบเพื่อให้สามารถใช้งานได้จริงกับการออกแบบในระดับสูงด้วยภาษา VHDL ที่กำลังนิยมอยู่ในขณะนี้ ดังนั้นจึงแบ่งการตรวจสอบและการทดสอบออกเป็น 2 ระดับคือ

1. ใช้การตรวจสอบโดยใช้ Abstract State Machine เพื่อตรวจสอบแต่ละองค์ประกอบย่อยว่ามีการทำงานได้อย่างถูกต้อง นั่นหมายถึงตรวจสอบฟังก์ชันหรือหน้าที่หลักของแต่ละองค์ประกอบย่อย

2. ใช้การจำลองการทำงานเพื่อตรวจสอบระบบทั้งหมด ซึ่งจะเป็นตรวจสอบการทำงานของทุกองค์ประกอบย่อยให้เป็นไปอย่างถูกต้อง

2.2.1 การตรวจสอบสถานะการทำงานนามธรรม (Abstract State Machine)

การตรวจสอบสถานะการทำงานเป็นการใช้เทคนิคของการแปลงระบบไปสู่รูปแบบที่เหมาะสมในการตรวจสอบความถูกต้อง เช่น ไดอะแกรมของสถานะ แล้วทำการตรวจสอบไดอะแกรมดังกล่าวเปรียบเทียบกับคุณลักษณะที่ได้กำหนดไว้ ซึ่งส่วนใหญ่ใช้เพื่อให้นักออกแบบวงจรรวมสามารถพิสูจน์ได้ว่าวงจรที่ออกแบบตรงตามคุณลักษณะพฤติกรรมที่ต้องการ ส่วนวงจรที่ได้ผ่านการตรวจสอบแล้วจะหมายถึงทุกกรณีที่เป็นไปได้ของอินพุทของวงจรสามารถทำงานได้ โดยพิสูจน์ได้ด้วยเทคนิคทางคณิตศาสตร์ว่าทำงานได้ตรงตามคุณลักษณะที่กำหนด ไม่ใช่เพียงแค่การจำลองการทำงานของบางอินพุทที่สมมติขึ้น ดังนั้นความหมายของการตรวจสอบอย่างมีแบบแผนสามารถแสดงได้ดังรูปที่ 2.3



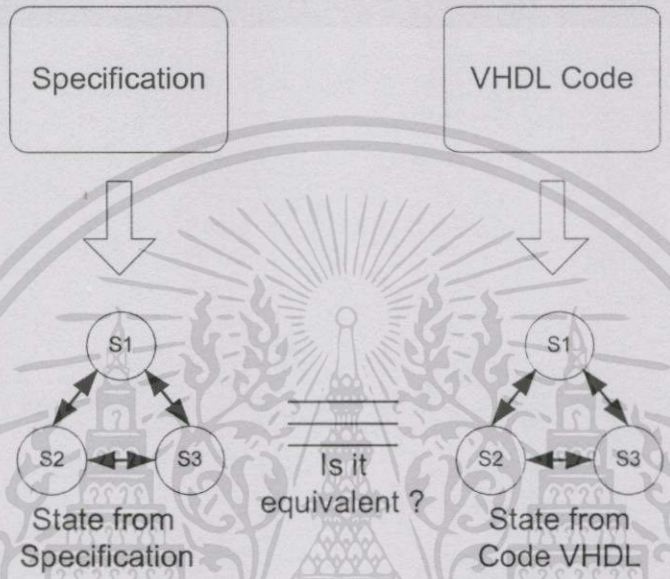
รูปที่ 2.3 การตรวจสอบอย่างมีแบบแผน

การตรวจสอบสามารถแบ่งออกได้เป็นงานหลัก 3 งานคือ

- กำหนดคุณลักษณะ
- สร้างวงจรตามคุณลักษณะที่กำหนด
- พิสูจน์ความถูกต้องของวงจร

เนื่องจากการกำหนดคุณลักษณะและการสร้างวงจรตามคุณลักษณะที่กำหนดได้ใช้ภาษาในการทำงานคนละภาษาดังนั้นจึงเป็นการยากที่จะนำสิ่งทั้งสองมาเปรียบเทียบกัน ดังนั้นวิทยานิพนธ์นี้จึงได้นำเสนอการตรวจสอบการทำงานอย่างมีแบบแผนโดยใช้การตรวจสอบสถานะการทำงานแบบนามธรรมเนื่องจากภาษาที่ใช้ในการกำหนดคุณลักษณะการทำงานของระบบกับภาษาที่ใช้ในการบรรยายการทำงานของวงจรเป็นคนละภาษากัน ทำให้ยากในการตรวจสอบความถูกต้องของวงจรที่ออกแบบเพื่อเปรียบเทียบกับคุณลักษณะที่กำหนดได้ ดังนั้นงานวิจัยนี้จึงได้นำเสนอรูปแบบที่เป็นแบบแผนในการเปรียบสิ่งทั้งสองเข้าด้วยกัน โดยใช้ Abstract State Machine และยังได้นำเสนอวิธีการแปลงภาษาที่ใช้ในการกำหนดคุณลักษณะและภาษาที่ใช้บรรยายการทำงานให้เป็นรูปแบบที่เป็นแบบแผนได้ ดังรูปที่ 2.4

หลังจากนั้นผู้ออกแบบหรือผู้ตรวจสอบความถูกต้องของระบบสามารถนำรูปแบบที่ได้ มาการเปรียบเทียบระหว่างคุณลักษณะที่กำหนดไว้ และระบบที่ได้ทำการออกแบบได้ โดยการป้อนอินพุทให้กับโมเดลทั้งสองแล้วทำการเปรียบเทียบการเปลี่ยนสถานะของโมเดลทั้งสองรวมทั้งสถานะสุดท้ายในการทำงานเนื่องจากอินพุทดังกล่าว



รูปที่ 2.4 การพิสูจน์ความถูกต้องของวงจรที่ออกแบบโดยใช้ Abstract State Machine

2.2.2 การจำลองการทำงาน (Simulation)

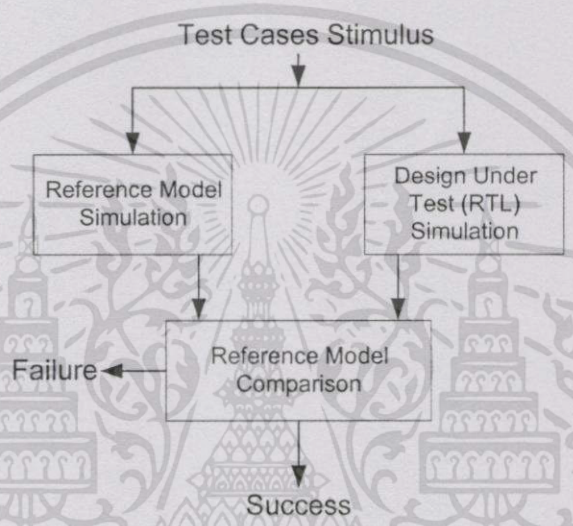
วิธีการตรวจสอบการทำงานของวงจรอีกวิธีหนึ่งที่ได้รับความนิยมและใช้งานกันอย่างแพร่หลายคือการจำลองการทำงาน (simulation) เนื่องจากการตรวจสอบแบบนี้ไม่มีความซับซ้อนเหมาะสำหรับการตรวจสอบวงจรที่มีขนาดใหญ่ และซับซ้อนอย่างไมโครโพรเซสเซอร์ เป็นต้น แต่หากใช้การจำลองการทำงานเป็นการตรวจสอบแล้วจำเป็นที่จะต้องสร้างชุดทดสอบเพื่อกำหนดเป็นอินพุทให้กับระบบที่ต้องการตรวจสอบ จากนั้นจึงตรวจผลลัพธ์ที่ได้ว่าตรงตามการทำงานของคุณลักษณะที่ต้องหรือไม่ ชุดทดสอบที่สร้างขึ้นเรียกว่า Test vectors ในที่นี้วงจรที่ออกแบบด้วยเครื่องมือ หรือภาษาใดชุดทดสอบก็จะถูกสร้างขึ้นให้สอดคล้องกับรูปแบบที่ออกแบบวงจร จึงทำให้ง่ายต่อการใช้งานของนักออกแบบทั่วไปดังหลักการจำลองการทำงานในรูปที่ 2.5

Test cases Stimulus : เป็นชุดทดสอบที่ถูกเลือกเป็นตัวแทนของคำสั่งการทำงานทั้งหมดของระบบที่ต้องการจำลองการทำงาน

Reference Model Simulation : แสดงถึงคุณลักษณะที่ต้องการ (Specification) เพื่อเป็นข้อมูลอ้างอิง ในการตรวจสอบการทำงาน

Design Under Test (RTL) : เป็นส่วนของวงจรที่ออกแบบ(Implementation) ในที่นี้อธิบายการทำงานของวงจรด้วยภาษา VHDL

Reference Model Comparison : เป็นส่วนของการเปรียบเทียบผลลัพธ์ที่ได้จากทั้งคุณลักษณะและการวงจรที่ออกแบบ เขียนเป็นชุดทดสอบได้ด้วยภาษา VHDL



รูปที่ 2.5 ขั้นตอนการจำลองการทำงาน

การจำลองการทำงานยังสามารถกระทำได้ทุกขั้นและทุกระดับของการออกแบบ เช่นการสร้างชุดสอบในระดับของ Instruction หรือการตรวจสอบในระดับของเกตที่รวมค่าหน่วงเวลาในการทำงานจริงของวงจรด้วย (Gate Delay) ในวิทยานิพนธ์ฉบับนี้ได้ทำการตรวจสอบด้วยการสร้างชุดทดสอบการทำงานด้วยภาษา VHDL ขึ้นเพื่อป้อนชุดคำสั่งหรืออินพุทให้กับระบบพร้อมกันนี้ชุดทดสอบการทำงานยังช่วยในการตรวจสอบข้อผิดพลาดที่เกิดขึ้นในระหว่างการจำลองการทำงานได้ด้วย โดยการแสดงข้อความจุดที่ผิดพลาดเตือนที่หน้าต่าง Command ของโปรแกรมจำลองการทำงานนั้นๆ

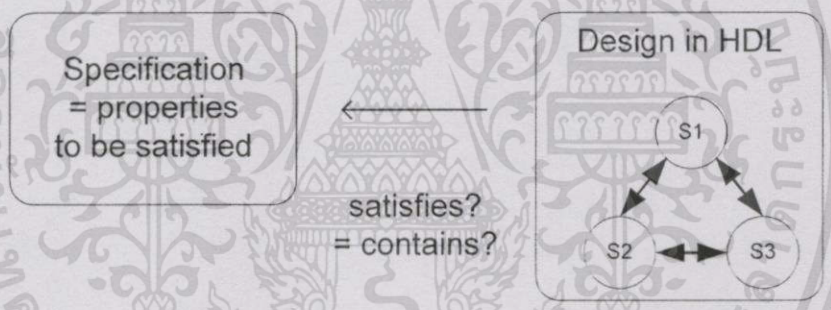
2.3 ข้อเสนอการตรวจสอบการทำงานนามธรรม

การตรวจสอบการทำงานแบบนามธรรมเป็นการตรวจสอบความถูกต้องของการทำงานของไมโครโพรเซสเซอร์ในระดับองค์ประกอบย่อยเพื่อให้นักออกแบบมั่นใจในความถูกต้องของระบบที่ได้ทำการออกแบบก่อนที่จะนำไปรวมเป็นระบบใหญ่ (Integration System) โดยมีพื้นฐาน

การตรวจสอบมาจาก State Machine แต่เนื่องจากการตรวจสอบโดยใช้ State Machine ในไมโครโพรเซสเซอร์จะทำให้ได้สถานะในการตรวจสอบมากจนเกินไป จึงจำเป็นต้องอาศัยเทคนิคในการลดจำนวนสถานะที่สนใจให้น้อยลงโดยใช้ Semantic Extraction ดังนี้

2.3.1 State Machine

การตรวจสอบแบบ State Machine เป็นเทคนิคหนึ่งในการตรวจสอบวงจรที่ออกแบบให้สอดคล้องตรงตามคุณลักษณะที่กำหนด เนื่องจากปัจจุบันวงจรมีขนาดใหญ่และซับซ้อนมากขึ้นดังนั้นการจำลองการทำงานจึงไม่มีประสิทธิภาพพอที่จะตรวจหาความถูกต้องของวงจร หากข้อผิดพลาดสามารถถูกตรวจสอบได้ในช่วงต้นของกระบวนการออกแบบก็จะทำให้สามารถประหยัดค่าใช้จ่ายได้มาก เนื่องจากราคาของการเจือสาร (Fabrication) เพื่อให้ได้ไอซีมีราคาสูงมาก รูปแบบของ State Machine เป็นไปตามหลักการดังรูปที่ 2.6 ซึ่งวงจรที่ออกแบบด้วย VHDL สามารถถูกตรวจสอบได้ทุกกรณีให้ตรงกับคุณลักษณะ



รูปที่ 2.6 หลักการตรวจสอบแบบ State Machine

State Machine เป็นการนำทฤษฎีกราฟ พื้นฐานมาเพื่อแสดงสถานะการทำงานของระบบโดยใช้ Vertex หรือ Node ในการแทนสถานะในการทำงาน และแทนการเปลี่ยนจากสถานะหนึ่งไปยังอีกสถานะหนึ่งโดยใช้ Arc โดยจะมีลักษณะของ Input ที่ต้องการในการเปลี่ยนสถานะระบุควบคู่ไปด้วย โดยพื้นฐานของ State Machine จะประกอบไปด้วย

$M = (I, O, S, \delta, \lambda, S_0)$

โดย I คือ อินพุทของระบบ

O คือ เอ้าท์พุทของระบบ

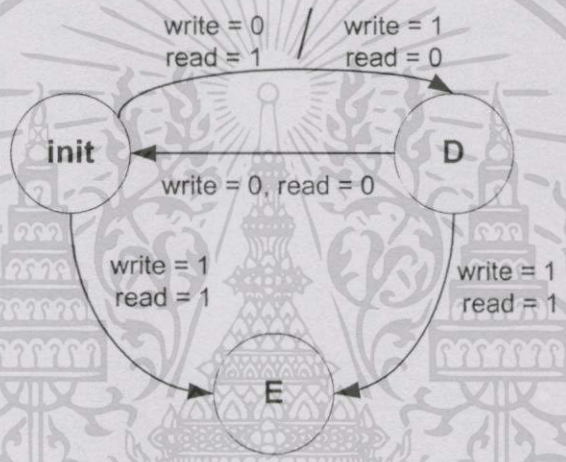
S คือ สถานะที่เป็นไปได้ทั้งหมดของระบบ

δ คือ ฟังก์ชั่นของสถานะถัดไป ($\delta : S \times I \rightarrow S$)

λ คือ ฟังก์ชั่นของเอ้าท์พุท ($\lambda : S \times I \rightarrow O$)

S_0 คือ สถานะเริ่มต้น

State Machine ใช้พื้นฐานของการแปลงระบบให้เป็นไดอะแกรมของสถานะการทำงาน โดยค้นหาทุกๆ สถานะที่จะเป็นไปได้ในวงจรที่ออกแบบโดยการตรวจสอบคุณลักษณะที่กำหนดไว้ให้ตรงกับพฤติกรรมของวงจรที่ได้ทำการออกแบบว่าตรงกันทั้งหมดหรือไม่ คุณลักษณะสำหรับ State Machine จะเป็นกลุ่มของคุณสมบัติที่วงจรนั้นๆควรจะเป็น ตัวอย่างเช่น หากต้องการระบุคุณสมบัติให้สัญญาณการ read/ write ไม่มีการ set ค่าเป็น 1 ขึ้นพร้อมกัน ดังนั้นสามารถแทนได้ด้วยไดอะแกรมดังรูปที่ 2.7



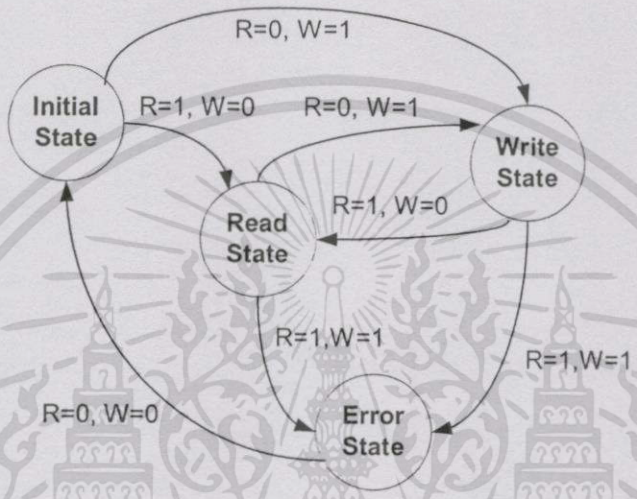
รูปที่ 2.7 ไดอะแกรมสถานะของคุณลักษณะที่กำหนด

จากรูปที่ 2.7 สถานะเริ่มต้นในการทำงานคือ Init และมีอีกสองสถานะในระบบคือ D (Doing) เป็นสถานะที่ระบบทำงานอย่างถูกต้อง E (Error) เป็นสถานที่เกิดขึ้นเมื่อการทำงานของระบบมีความผิดพลาดเกิดขั้น โดยมีสัญญาณ Read และ Write เป็นเงื่อนไขในการเปลี่ยนสถานะการทำงาน เช่น จากเดิมเมื่ออยู่ที่สถานะ Init หากสัญญาณ Read และ Write มีการเปลี่ยนแปลงจาก Read = 0 และ Write = 0 เป็น Read = 1 และ Write = 0 หรือ Read = 0 และ Write = 1ก็ตามจำให้ระบบเปลี่ยนสถานะไปยังสถานะ Doing แต่หากสัญญาณ Read = 1 และ Write = 1 พร้อมกันจะทำให้รบบเกิดความผิดพลาดขึ้นเนื่องจากสภาวะที่หน่วยความจำได้รับสัญญาณทั้ง Read และ Write เป็น 1 พร้อมกันเป็นสถานะที่ไม่ต้องการให้เกิดขึ้นในระบบ

โดยในสถานะเริ่มต้น (Init) จะเป็นการเริ่มต้นการทำงาน เมื่อมีสัญญาณ Read/Write เกิดขึ้นก็จะเปลี่ยนสถานะไปยังสถานะอื่น โดยหากสัญญาณ Read/Write เป็น 1 เพียงอันเดียวจะทำให้สามารถทำงานได้อย่างถูกต้อง คือเปลี่ยนไปยังสถานะทำงาน (D = Doing) และจะเปลี่ยนกลับไปยัง

สถานะเริ่มต้น (Init) เมื่อสัญญาณ Read/Write กลับไปเป็น 0 ทั้งคู่ แต่หากเมื่อใดก็ตามที่สัญญาณ Read/Write เป็น 1 ทั้งคู่จะทำให้เกิดการทำงานที่ไม่ถูกต้อง นั่นคือจะเปลี่ยนไปยังสถานะที่ยอมรับไม่ได้ (E = Error)

วงจรที่ออกแบบและสามารถตรวจสอบได้แสดงดังรูปที่ 2.8 ซึ่งสามารถทำการ Trace state transition ไปจนกระทั่งครบทั้งวงจรแล้วบอกผลที่ได้ว่าตรงตามคุณสมบัติที่ตั้งไว้หรือไม่



รูปที่ 2.8 วงจรที่ออกแบบการทำงานของสัญญาณการ read/write

ในทางเดียวกันนั้น State Machine ก็มีข้อเสียคือถ้าสถานะของการทำงานมีมากเกินไปการ trace เข้าไปแต่ละ transition จนครบทุกสถานะก็จะทำได้ยากเช่น ถ้ามี 100 สถานะก็จะต้อง trace ทั้งหมด 2^{100} กรณี ซึ่งจะใช้เวลานานมากดังนั้นการไม่ใส่รายละเอียดลงไปในการตรวจสอบจะเป็นการช่วยลดขนาดของสถานะไม่ให้ใหญ่มากจนเกินไป ดังนั้นการแทนรายละเอียดให้เป็น Abstract ซึ่งแทนฟังก์ชันการทำงานหลักเท่านั้นจึงเป็นวิธีการที่เหมาะสมในการนำเสนอการตรวจสอบแบบ Abstract State Machine สำหรับไมโครโพรเซสเซอร์ที่มีขนาดใหญ่และซับซ้อน

เครื่องมือที่ใช้ในการทำ State Machine สามารถแบ่งออกได้เป็น temporal logic และการตรวจพฤติกรรมให้ตรงกับคุณลักษณะ โดยเครื่องมือตรวจสอบแบบ temporal logic ได้แก่ EMC[3], CASESAR[4], SMV[5], SPIN[6], Murφ[7], UV[8], SVE[9], CV[10] และ Kronos[11] เป็นต้น ส่วนเครื่องมือตรวจสอบพฤติกรรมในรูปแบบของไดอะแกรมให้ตรงกับคุณลักษณะที่กำหนดไว้ เช่น Cospan/Formal Check[10] และ FDR[11] แต่เครื่องมือทั้งหมดนี้จำเป็นต้องใช้ภาษาเพื่อการออกแบบที่เฉพาะกับเครื่องมือนั้นๆ ดังนั้นจึงไม่เหมาะสมที่จะนำไปใช้งานจริง เนื่องจากการออกแบบในปัจจุบันเป็นการออกแบบในระดับสูงที่ใช้ภาษา VHDL หรือ Verilog

State Machine ได้ถูกนำเสนอขึ้นในช่วงต้นระหว่างปี ค.ศ. 1980-1989 แม้ว่า State Machine จะสามารถตรวจสอบวงจรขนาดซึ่งใช้ Flip/Flop ประมาณ 200 – 300 ตัวได้อย่างสบาย แต่ในทางปฏิบัติวงจรที่ออกแบบจะมีขนาดของ Flip/Flop อยู่ที่ไม่ต่ำกว่า 1,000 ตัว ยิ่งไปกว่านั้นไม่เพียงแต่จำนวนสถานะของ Data path ที่ออกแบบแต่จะมีสถานะของส่วนการเชื่อมต่อกับอุปกรณ์ภายนอกผ่านทาง I/O ซึ่งจำเป็นต้องมีการตรวจสอบทั้งหมดด้วย ดังนั้นวงจรของโพรเซสเซอร์ที่ใช้งานจริงจึงมีขนาดใหญ่มาก จึงจำเป็นต้องใช้วิธีการทำให้เป็น Abstract เพื่อลดสถานะและทำให้ไม่ซับซ้อนแล้วเหมาะที่จะใช้กับการตรวจสอบ State Machine ปัญหาที่พบในเครื่องมือการตรวจสอบแบบ State Machine ที่มีอยู่ในปัจจุบันคือ ไม่สามารถทำให้เป็น Abstract (ลดจำนวนสถานะ) หรือทำให้การตรวจสอบไมโครโพรเซสเซอร์ซับซ้อนน้อยลงได้โดยอัตโนมัติ จึงจำเป็นต้องจัดการทั้งหมดขั้นตอนวิธีหรืออัลกอริทึ่มแทน อย่างไรก็ตามในวิทยานิพนธ์ฉบับนี้ได้เสนอแนวคิดการสร้าง Abstract State จากวงจรที่ออกแบบด้วยภาษา VHDL โดยการใช้ Semantic Extraction เพื่อให้ง่ายต่อการนำไปใช้งานจริงกับนักออกแบบในระดับบนที่ออกแบบด้วยภาษา VHDL

เนื่องจากวิธีการของการสังเคราะห์วงจรจากระดับสูงให้เป็นวงจรในระดับเกต (Synthesis) ได้ทำการลดขนาดวงจร (Optimization) ไปพร้อมๆ กันด้วย ดังนั้นรีจิสเตอร์หรือสัญญาณอาจถูกรวมเข้าด้วยกัน การลดขนาดวงจรเช่นนี้เป็นต้นเหตุให้การทำ Semantic Extraction และการสร้างสถานะแบบนามธรรมยากมากยิ่งขึ้น ยิ่งไปกว่านั้นผลลัพธ์ที่ได้จากการสังเคราะห์วงจรอาจซ่อน Semantic ของแต่ละ Abstract โมดูลไว้ จึงต้องทำการ Extract Semantic ในโมดูลออก เพื่อให้การตรวจสอบทำได้รวดเร็วมากยิ่งขึ้น โดยแนวความคิดนี้ จำเป็นต้องสร้างเครื่องมือในการทำ Semantic Extract ภาษา VHDL ที่ใช้ออกแบบเพื่อวิเคราะห์โมดูลวงจรที่ออกแบบด้วยภาษา VHDL ทั้งแบบพร้อมกัน (Concurrent) และแบบต่อเนื่อง (Sequence) ซึ่งเครื่องมือนี้จะสร้างอยู่บนพื้นฐานของเทคนิคการวิเคราะห์ Data Flow

2.3.2 Abstract State Machine

การสร้าง Abstract State Machine มาจากพื้นฐานของการทำ Semantic Extraction ในโมเดลนั้นๆ ที่ได้มาจากการอธิบายพฤติกรรมของภาษา VHDL เทคนิคของ Semantic Extraction มี 3 ข้อหลักคือ 1. การ Extract key value 2. การแบ่งย่อยโมดูล (Model Partitioning) และ 3. Abstract ในส่วนของเนื้อหาเท่านั้น ซึ่งในข้อ 1 และ 3 จะช่วยลด state-space(จำนวนของการใช้สถานะ) ได้โดยการกำหนดให้สัญญาณและตัวแปรแทนด้วยสัญลักษณ์หรือ abstract data type ในแต่ละ abstract จะระบุให้เป็นแบบ non-deterministic finite automata (NFA) ซึ่งถูกสร้างโดยให้ 1 สถานะแทน abstract การทำงานหนึ่งๆ ส่วนการแบ่งย่อยโมดูลนั้นจะช่วยลดจำนวนการใช้สถานะด้วยการเอาบางส่วนที่ไม่เกี่ยวข้องกับคุณลักษณะที่ต้องการทวนสอบออกไป NFA จะต้องถูกสร้างขึ้นและทำการตรวจสอบให้สามารถครอบคลุมทุกสถานะได้ทั้งหมดซึ่งเรียกว่าการ accept

- Key Value Extraction

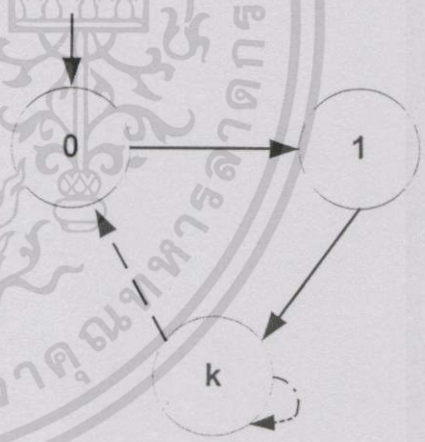
ในการควบคุมระบบหลักนั้นคุณสมบัติที่จะใช้ตรวจสอบจะถูกนำเข้ามาหาความสัมพันธ์ว่าระบบนั้นทำงานตรงตามคุณลักษณะในเงื่อนไขการทำงานแบบปกติหรือไม่ ดังนั้นจึงทวนสอบระบบได้โดยการตรวจว่าเงื่อนไขการทำงานถูกหรือผิด เช่นถ้าต้องการตรวจสอบว่าระบบมีการเปลี่ยนสถานะจาก β ไปเป็น β' ภายใต้เงื่อนไขที่ว่าค่าของ a ต้องเท่ากับ b ดังนั้นระบบต้องตรวจสอบ ในกรณีที่ $a = b$ และ $a \neq b$ ซึ่งสามารถกระจายออกได้เป็น $a = b$, $a > b$ และ $a < b$ โดยที่กรณีทั้งหมดที่สามารถเกิดขึ้นได้มาจากเซ็ตของ $\{ a = b, a \neq b, a > b, a < b, a \geq b, a \leq b\}$ abstract ของตัวแปร a ให้แทนด้วย $a^{\wedge}$ จึงสามารถมี key values ได้เป็น (b, k1, k2) โดยที่ k1 เป็นค่าสำหรับกรณีของ $a < b$ ส่วน k2 เป็นค่าสำหรับกรณีที่ $a > b$ จาก key values จึงนำไปสร้างเป็นความสัมพันธ์ $\Re$ ได้ดังนี้

$$a^{\wedge} = \begin{cases} b & \text{if } a = b \\ k1 & \text{if } a < b \\ k2 & \text{if } a > b \end{cases}$$

สมมติให้วงจรที่ออกแบบเป็นวงจรนับดังโมดูลที่ออกแบบด้วยภาษา VHDL ดังรูปที่ 2.9

```
process(clk, reset)
begin
        if reset = '1' then
                cnt <= "0000";
        elsif rising_edage(clk) then
                cnt <=  cnt + 1;
        end if;
end process;
```

รูปที่ 2.9 โมดูลวงจรนับที่ออกแบบด้วยภาษา VHDL



รูปที่ 2.10 สถานะทั้งหมดในวงจรนับ

จากรูปสถานะที่ 0 และ 1 เป็นแบบ deterministic state ในขณะที่สถานะ k เป็นแบบ non-deterministic การเปลี่ยนสถานะจากสถานะ 0 ไปยังสถานะ 1 และการเปลี่ยนสถานะจากสถานะ 1 ไปยัง สถานะ k จะเป็นการเคลื่อนสถานะแบบ deterministic ซึ่งแทนด้วยลูกศรเส้นทึบดำ แต่การเปลี่ยนสถานะจาก k กลับไปยัง k เองและไปยังสถานะ 0 นั้นเป็นการเคลื่อนสถานะแบบ non-deterministic ซึ่งใช้ลูกศรเส้นประในการแทน ทั้งนี้จะใช้แทนสถานะตั้งแต่ช่วง 2 – 16 ด้วย ดังนั้น

ค่าของ key values จึงช่วยลดจำนวนของสถานะที่ใช้และให้ symbolic แทนตัวแปรหรือสัญญาณในช่วงที่เล็กกว่า

- Model Partitioning

การทำการแบ่งย่อยโมเดลออกเป็นส่วนๆนั้นมีประโยชน์ทั้งในด้านการออกแบบและการตรวจสอบ เนื่องจากโมเดลที่แบ่งย่อยออกมามีความซับซ้อนน้อยกว่าทั้งระบบทำให้ง่ายต่อการออกแบบและการตรวจสอบ ในส่วนของการตรวจสอบนั้นโมเดลที่ไม่เกี่ยวข้องจะถูกตัดออกไปจากการแบ่งย่อยโมดูล ดังนั้นในแต่ละโมดูลจะมีขนาดเหมาะสมและครอบคลุมส่วนที่ต้องการตรวจสอบเท่านั้น ซึ่งจะใช้ node graph แทนในแต่ละโมดูลที่ต้องการทำการแบ่งย่อย โดยให้แต่ละ node แทนการทำงาน (operations) ส่วนการเคลื่อนที่ในแต่ละ node ซึ่งจะแทนสัญญาณหรือตัวแปรของโมดูลที่ออกแบบ เมื่อการเคลื่อน node ใดจำเป็นต้องขึ้นกับสัญญาณใดสัญญาณนั้นจะถูกกำหนดเป็นชื่ออยู่บนเส้นทางนั้นๆ ดังนั้นเมื่อแทนเป็น node graph แล้วทั้งระบบจะได้เส้นทางการทำงานที่ซับซ้อน หลากหลาย ดังนั้นการเขียนในลักษณะของ กราฟต้นไม้ (Tree) จะช่วยในการแบ่งย่อยโมดูลได้ง่ายขึ้น หมายถึงในแต่ละกิ่งหรือสาขาหนึ่งจะสามารถตัดแบ่งออกเป็นโมดูลย่อยได้โดยไม่ซ้อนทับหรือคาบเกี่ยวการทำงานกับโมดูลอื่น แต่จะได้ลักษณะของโมดูลออกมาเป็นลำดับชั้นอย่างชัดเจน

- Data Abstraction

ในส่วนของการทำ data abstraction นั้นจะช่วยให้การลดขนาดของสถานะมีประสิทธิภาพมากขึ้น เพื่อช่วยเสริมประสิทธิภาพของการทำ key values และการแบ่งย่อยโมดูล แต่การทำ data abstraction จะมีความยุ่งยากมากกว่า ในบางระบบค่าของตัวแปรหรือสัญญาณไม่ได้มีความสำคัญมากนัก แต่ความสัมพันธ์ระหว่างตัวแปรจะมีความสำคัญมากกว่า ตัวอย่างเช่น จากเงื่อนไขความสัมพันธ์ของ v1 < v2 จะไม่มี key value ที่สามารถถูก extract ออกมาได้เลย อย่างไรก็ตามถ้าค่าของสองตัวแปรไม่มีความสำคัญ การทำ abstraction สามารถสร้างได้จากความสัมพันธ์ของทั้งสองตัวแปรแทน ดังนั้น 2 ตัวแปรสามารถสร้างเป็น abstract ได้ดังนี้

$$v1^{\wedge} = \begin{cases} k1 & \text{if } v1 < v2 \\ k2 & \text{if } v1 = v2 \\ k3 & \text{if } v1 > v2 \end{cases}$$

$$v2^{\wedge} = \begin{cases} k3 & \text{if } v1 > v2 \\ k2 & \text{if } v1 = v2 \\ k1 & \text{if } v1 < v2 \end{cases}$$

โดยที่ k1, k2 และ k3 เป็นค่าคงที่แทนเซตของค่าของ (v1,v2) เช่น (v1 < v2), (v1 = v2) และ (v1 > v2) ตามลำดับ

2.3.3 การแทนระบบด้วย Abstract State Machine

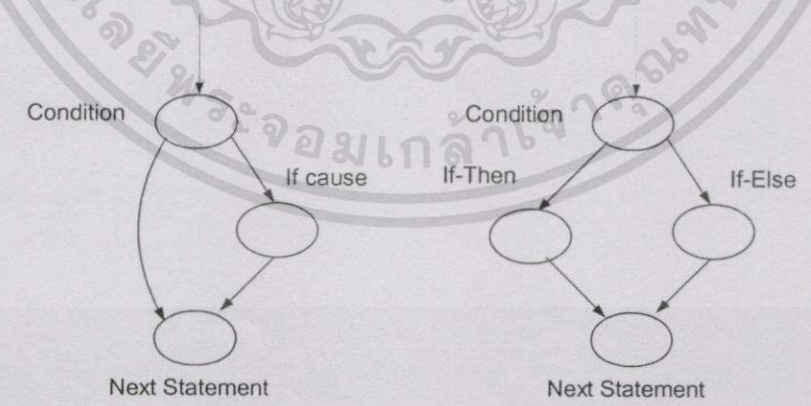
ในงานวิจัยนี้ได้ประยุกต์วิธีการตรวจสอบองค์ประกอบย่อยของไมโครโพรเซสเซอร์ด้วยเทคนิคของ Abstract State Machine โดยอาศัยการแปลงไมโครโพรเซสเซอร์ที่ออกแบบด้วยภาษา VHDL ให้อยู่ในรูปไดอะแกรมของ State Machine และเพื่อเป็นการลดสถานะของไดอะแกรมจึงได้มีการทำให้เป็น Abstract โดยใช้เทคนิคของ Semantic Extraction อย่างไรก็ตามเครื่องมือที่ช่วยในการแปลงระบบที่ออกแบบให้อยู่ในรูปของไดอะแกรมที่ต้องการในปัจจุบันยังไม่สามารถทำได้ดีเท่าที่ควร งานวิจัยนี้จะจึงได้นำเสนอวิธีคิด (Algorithm) ในการแปลงระบบให้อยู่ในรูปแบบของไดอะแกรมที่ต้องการ โดยมีวิธีดังนี้

1. Abstract ระบบให้เหลือแต่ข้อมูลหรือฟังก์ชั่นการทำงานหลักที่สนใจโดยใช้ Semantic Extraction
2. ใช้กฎหรือวิธีคิดดังต่อไปนี้เพื่อแปลงระบบให้กลายเป็นไดอะแกรมที่ต้องการดังรูปที่ 2.11- 2.13

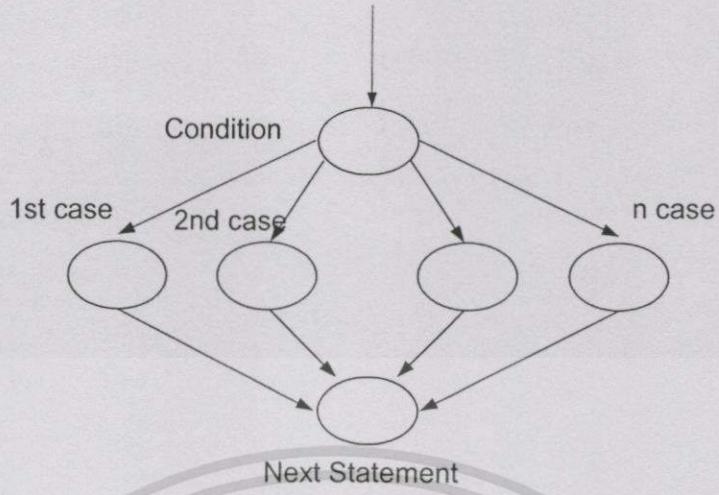
Rule 1: If-Then-Else statement will generate two states of each condition.
If there is not an Else statement, the state of else must be disappeared.

Rule 2: Case-Of statement will generate states according to the number of conditions.

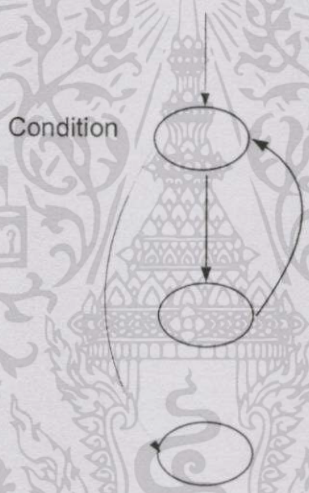
Rule 3: While-Loop statement will generate state that has link pointed back to its own state.



รูปที่ 2.11 สถานะของรูปประโยค If-Then และ If-Then-Else



รูปที่ 2.12 สถานะของรูปประโยค Case-Of



รูปที่ 2.13 สถานะของรูปประโยค While-Loop

จากกฎและวิธีดังกล่าวสามารถแปลงข้อกำหนดคุณลักษณะในข้อ 2.1 ให้เป็นไดอะแกรมที่ต้องการได้ดังรูปที่ 2.14

Rule : ExecuteALU

```
If   ExecuteOK and ALUInstr(Instr)  then
    If Satisfies(Status, CondCode(Instr))  then
        If  WriteResult(Instr)   then
          Contents(DesReg)  := ALU( ALUop(Instr), Aop, Bop, Carry(Status))
        Endif
```

```
            If SetCondCode(Instr) then
               Status := UpdateStatus( Status, ALUop(Instr), Aop, Bop, ShiftCarryOp)
            Endif
         Endif
      Endif
```

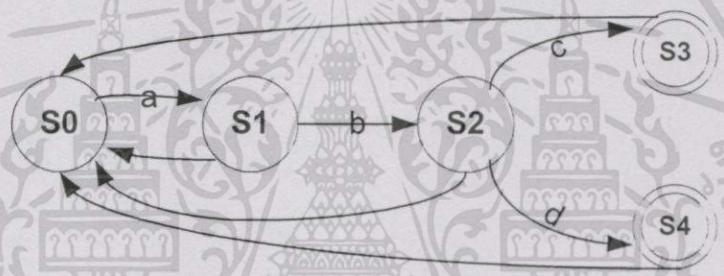
Specification :

Rule a : ExecuteOK and ALUInstr(Instr)

Rule b : Satisfies(Status, CondCode(Instr))

Rule c : WriteResult(Instr)

Rule d : SetCondCode(Instr)



รูปที่ 2.14 ไดอะแกรมของข้อกำหนดคุณลักษณะ

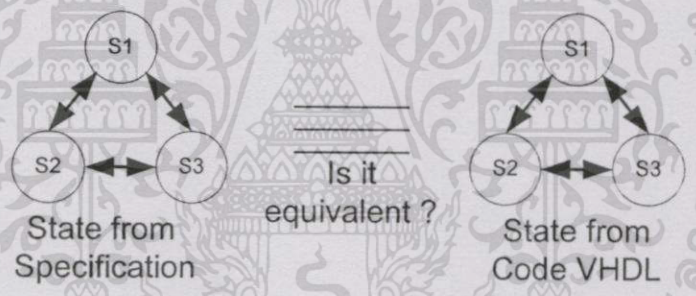
โดยที่ S0 เป็นสถานะเริ่มต้นในการทำงาน และหากระบบได้รับการทำงานที่เป็นไปตาม Rule a คือ ExecuteOK and ALUInstr(Instr) จะทำให้ระบบเปลี่ยนสถานะจาก S0 ไปยัง S1 ตาม Arc a ในทำนองเดียวกันระบบจะเปลี่ยนจากสถานะ S1 ไปเป็น S2 ได้ก็ต่อเมื่อระบบได้รับการทำงานตาม Rule b และหากระบบได้รับการทำงานที่เป็นไปตาม Rule c จะทำให้ระบบไปหยุดอยู่ที่สถานะ S3 ซึ่งเป็นสถานะที่ยอมรับว่ามีการทำงานที่ถูกต้อง โดยในทุกๆ ครั้งที่มีการทำงานที่ไม่เป็นไปตาม Rule a, b, c และ d จะทำให้ระบบกลับไปสู่สถานะแรกคือ S0 เสมอ

2.4 การตรวจสอบการทำงานของ Abstract State Machine

เมื่อทำการสร้างไดอะแกรมจากข้อกำหนดคุณลักษณะ และไดอะแกรมจากโค๊ดโปรแกรม โดยใช้กฎหรือวิธีคิดดังกล่าวจะทำให้ทั้งข้อกำหนดคุณลักษณะและโค๊ดโปรแกรมถูกแปลงให้อยู่ในรูปแบบที่เป็นมาตรฐานเดียวกัน และสามารถนำไดอะแกรมดังกล่าวมาตรวจสอบความถูกต้องของระบบได้ ดังรูปที่ 2.15 โดยทำการ Trace การทำงานของไดอะแกรมทั้งสองเพื่อดูผลการเปลี่ยน

สถานะการทำงานและสถานะสุดท้ายหลังจากป้อนอินพุทเข้าไป โดยสามารถแยกการตรวจสอบได้สองกรณีคือ

1. กรณีที่ไดอะแกรมที่ได้จากทั้งข้อกำหนดคุณลักษณะและจากโค๊ดโปรแกรมมีลักษณะเหมือนกันหรือไม่ต่างกันมากนัก การตรวจสอบสอบความถูกต้องสามารถตรวจสอบได้ทั้งการเปลี่ยนสถานะระหว่างการป้อนอินพุทและสถานะสุดท้ายหลังจากการป้อนอินพุทว่าเหมือนหรือแตกต่างกันหรือไม่อย่างไร หากไม่มีความแตกต่างกันแสดงว่าโค๊ดโปรแกรมที่เขียนขึ้นมาสามารถทำงานได้ตามข้อกำหนดคุณลักษณะดังกล่าวทุกประการ
2. กรณีที่ไดอะแกรมที่ได้จากทั้งข้อกำหนดคุณลักษณะและจากโค๊ดโปรแกรมมีลักษณะแตกต่างกันออกไป การตรวจสอบสอบความถูกต้องสามารถตรวจสอบได้เฉพาะสถานะสุดท้ายหลังจากการป้อนอินพุทว่าอยู่ในสถานะเดียวกันหรือไม่ หากสถานะสุดท้ายของทั้งสองไดอะแกรมหลังจากป้อนอินพุทอยู่ในสถานะเดียวกัน แสดงว่าโค๊ดโปรแกรมที่เขียนขึ้นมาสามารถทำงานได้ตามข้อกำหนดคุณลักษณะดังกล่าว

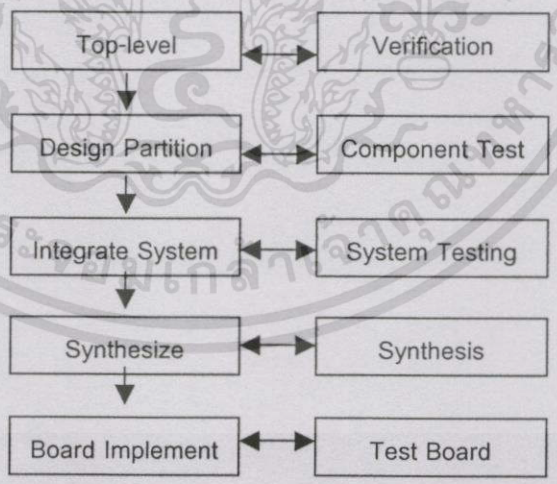


รูปที่ 2.15 การตรวจสอบความถูกต้องของไดอะแกรมจากข้อกำหนดคุณลักษณะและจากโค๊ดโปรแกรม

บทที่ 3
การทำงานของไมโครโพรเซสเซอร์ ARM7

ARM 7 เป็นไมโครโพรเซสเซอร์ในตระกูล 32 บิตของบริษัท Advanced RISC Machines(ARM) ที่ได้รับความนิยมเนื่องจากมีประสิทธิภาพในการทำงานสูง ประหยัดพลังงาน และราคาถูก โครงสร้างสถาปัตยกรรมจะเป็นแบบ Reduced Instruction Set Computer (RISC) พร้อมชุดคำสั่งที่มีประสิทธิภาพสูง การทำงานของไปป์ไลน์ทำให้มีความต่อเนื่องของการทำงานและสถานะการทำงาน 3 สถานะ คือการนำเอาคำสั่งเข้ามาทำงาน (Fetch) การถอดรหัส (Decode) และเริ่มทำงานจริง (Execute)

จากการศึกษาคู่มือโครงสร้างของ ARM 7 และกระบวนการออกแบบไมโครโพรเซสเซอร์ สามารถสรุปและเสนอขั้นตอนของการออกแบบให้เหมาะสมกับ ARM 7 ได้ดังรูปที่ 3.1 ซึ่งแสดงขั้นตอนต่างๆ ที่ใช้ในการออกแบบโดยจะมีการทำงานไปพร้อมกันของส่วนการออกแบบและการตรวจสอบ เมื่อพบข้อผิดพลาดจะต้องทำการแก้ไขจนได้รับความถูกต้อง เพื่อให้ได้การทำงานของวงจรถูกต้องมากขึ้น เนื่องจาก ไมโครโพรเซสเซอร์ ARM 7 มีโครงสร้างการทำงานที่ซับซ้อน ดังนั้นการตรวจสอบในทุกระดับของการออกแบบจะทำให้สามารถตรวจพบข้อผิดพลาดได้ง่ายกว่าการตรวจสอบระบบรวมทั้งหมดแต่เพียงอย่างเดียว



รูปที่ 3.1 ขั้นตอนการออกแบบ

3.1 การทำงานของไมโครโพรเซสเซอร์ ARM 7

ARM 7 มีโครงสร้างสถาปัตยกรรมแบบ RISC ซึ่งมีรูปแบบโดยทั่วไปดังนี้

- ทุกคำสั่งในหมวดคำสั่ง Data-Processing จะผ่านการทำงานทั้งในส่วนของหน่วยคำนวณทางคณิตศาสตร์และตรรกะ พร้อมด้วยตัวชิฟเตอร์เสมอ
- มีการเพิ่มและลดค่าของการอ้างอิงหน่วยความจำเพื่อลดวงวน (loop) ที่เกิดขึ้นในโปรแกรม
- สามารถเก็บและบรรจุ (Load/Store) ข้อมูลได้ทีละจำนวนมากได้ภายในคำสั่งเดียว
- มีเงื่อนไขการประมวลผลของทุกๆ คำสั่ง
- มีไปป์ไลน์ในการทำงาน 3 สถานะ คือการนำคำสั่งเข้า(Fetch) การถอดรหัส(Decode) และการทำงาน (Execute)

3.1.1 รีจิสเตอร์ใน ARM 7

ARM 7 มีรีจิสเตอร์ขนาด 32 บิตจำนวน 31 ตัวแต่ผู้ใช้สามารถมองเห็นและใช้งานได้จำนวน 16 รีจิสเตอร์ในโหมดการทำงานปกติ ซึ่งทำหน้าที่เป็นรีจิสเตอร์ทั่วไป ในส่วนของโหมดการทำงานปกติ ในขณะเดียวกันรีจิสเตอร์ตัวที่ 15 สามารถใช้งานเป็น PC โดยค่า PC จะเพิ่มขึ้นครั้งละ 4 เนื่องจากชุดคำสั่งมีขนาด 32 บิต ดังนั้น PC จะมีค่า 30 บิตที่แปลงเปลี่ยน ส่วน 2 บิตล่างจะเป็น 0 เสมอ ส่วนรีจิสเตอร์ตัวที่ 14 จะถูกใช้เก็บเป็นรีจิสเตอร์เชื่อมโยง (Index Register) เมื่อมีการทำงานคำสั่งกระโดดข้ามแบบมีเงื่อนไข (BL) และรีจิสเตอร์ที่ 13 จะถูกใช้โดยซอฟต์แวร์ให้เป็น Stack Pointer (SP)

3.1.2 คำสั่งในไมโครโพรเซสเซอร์ ARM 7

ชุดคำสั่งสามารถแบ่งออกได้เป็น

- กลุ่มคำสั่ง Branch คือการกระโดดข้ามการทำงานทั้งเป็นแบบมีเงื่อนไขและไม่มีเงื่อนไขในการกระโดดข้ามการทำงาน
- Data-Processing คือ กลุ่มการทำงานที่เกี่ยวข้องกับหน่วยคำนวณทางคณิตศาสตร์, ตรรกะ และการชิฟข้อมูล
- Load and Store เป็นกลุ่มคำสั่งของการเคลื่อนย้าย แลกเปลี่ยนข้อมูลระหว่างหน่วยความจำกับรีจิสเตอร์ทั้งเป็นแบบ 1 ต่อ 1 หรือการแลกเปลี่ยนข้อมูลพร้อมกันครั้งละหลายๆ รีจิสเตอร์
- Coprocessor *

หมายเหตุ ในส่วน coprocessor ไม่มีการออกแบบ

ในทุกๆ คำสั่งใน ARM 7 จะมีการตรวจเงื่อนไขก่อนมีการทำงาน โดยเงื่อนไขในการทำงานจะเป็น 4 บิตบนของทุกคำสั่ง (บิตที่ 31-28)

Branch : คำสั่งกระโดดข้ามจะมีผลต่อลำดับการทำงานของโปรแกรม โดยการเปลี่ยนแปลงค่าใน PC โดยคำสั่งกระโดดข้ามมีการใช้ 24 บิตเพื่อเป็นค่าออฟเซต ทำให้สามารถกระโดดข้ามได้ทั้งไปหน้าและกลับหลังถึง 32 Mbytes (2^{24} = 16 M ตำแหน่ง แบ่งเป็นไปหน้า 8 M Address และกลับหลัง 8 M Address โดยแต่ละ Address มี 4 ไบต์)

Data-Processing : มีด้วยกัน 3 ชนิดคือ

- Data-processing ทั่วไป
- คำสั่งการคูณ
- คำสั่งบอกสถานะของรีจิสเตอร์

คำสั่งของหน่วยคำนวณและตรรกะ จะใช้ตัวถูกดำเนินการ (Operand) 2 ตัวและเก็บผลลัพธ์ใส่ รีจิสเตอร์โดยมี บางคำสั่งของ data-processing ที่ไม่มีการเก็บค่าลงรีจิสเตอร์ เพราะเป็นการเปรียบเทียบค่าเท่านั้น จึงเปลี่ยนแปลงเฉพาะค่าบ่งชี้ ตัวถูกดำเนินการจะได้มาจาก 2 แหล่งคือ มาจากรีจิสเตอร์ทั้งคู่ และมากจากตัวอื่นได้แก่ ค่าจำนวนเต็ม, ค่าจากรีจิสเตอร์ที่สามารถมีการชิฟค่าข้อมูลได้ ถ้าหากเป็นการชิฟข้อมูลในรีจิสเตอร์ จำนวนครั้งในการชิฟจะมาจากจำนวนเต็ม หรือค่าจากรีจิสเตอร์ ดังนั้นทุกคำสั่งใน Dataprocessing สามารถทำการชิฟข้อมูลได้ ARM 7 จึงไม่มีคำสั่งการชิฟ

คำสั่งการคูณสามารถแบ่งออกได้เป็น 2 แบบ โดยทั่วไปใช้รีจิสเตอร์ 32 บิต 2 ค่าในการเก็บ

(ปกติ) 32 บิต ผลลัพธ์เก็บในรีจิสเตอร์เดียว

(ยาว) 64 บิตผลลัพธ์เก็บในรีจิสเตอร์ 2 ตัวแยกกัน

* ซึ่งทั้ง 2 แบบสามารถทำการคำนวณการบวกพร้อมกันได้ด้วย

Load/Store : คำสั่งการเก็บและบรรจุ แบ่งออกได้เป็น 3 ชนิด

- เก็บและบรรจุ ด้วยการใช้รีจิสเตอร์ 1 ตัว
- เก็บและบรรจุ ด้วยการใช้รีจิสเตอร์หลายตัว
- แลกเปลี่ยนข้อมูลระหว่างรีจิสเตอร์และหน่วยความจำ

การทำงานของคำสั่งเก็บและบรรจุจะมีการอ้างอิงหน่วยความจำได้ 3 แบบคือ offset, pre-index และ post-index นอกจากนี้แล้วยังมีคำสั่งในการแลกเปลี่ยนข้อมูลระหว่างรีจิสเตอร์และหน่วยความจำ ซึ่งคำสั่งทั้งหมดสามารถส่งและรับข้อมูลได้ทีละ 8 บิต, 16 บิต และ 32 บิต

ตารางที่ 3.1 สรุปคำสั่งที่สามารถทำงานได้ในไมโครโพรเซสเซอร์ที่ออกแบบ

คำสั่ง	Source 1	Source 2	Shift	Execute	Write-back
B	-	-	-	-	#Jump>Pc
BL	-	-	-	-	#Jump>Pc
AND	W	W	OP2	SHIFT,AND	W
EOR	W	W	OP2	SHIFT,XOR	W
SUB	W	W	OP2	SHIFT,SUB	W
RSB	W	W	OP2	SHIFT,RSB	W
ADD	W	W	OP2	SHIFT,ADD	W
ADC	W	W	OP2	SHIFT,ADC	W
SBC	W	W	OP2	SHIFT,SBC	W
RSC	W	W	OP2	SHIFT,RSC	W
TST	-	-	OP2	SHIFT,TST	-
TEQ	-	-	OP2	SHIFT,TEQ	-
CMP	-	-	OP2	SHIFT,CMP	-
CMN	-	-	OP2	SHIFT,CMN	-
ORR	W	W	OP2	SHIFT,ORR	W
MOV	-	W	OP2	SHIFT,MOV	W
BIC	W	W	OP2	SHIFT,OP2 AND NOT OP1	W
MVN	W	W	OP2	SHIFT,NOT OP2	W
MUL	W	W	-	MUL	W
MLA	W	W	-	MUL,ADD	W
LDR	W,B	W,B	OP2	SHIFT,LOAD	W,B
STR	W,B	W,B	OP2	SHIFT,STORE	W,B
SWP	W,B	W,B	OP2	LOAD,STORE	W,B

3.2 การทำงานในแต่ละชิ้นส่วนย่อยภายในไมโครโพรเซสเซอร์ ARM 7

3.2.1 หน่วยคำนวณและตรรกะ

ประกอบด้วย 3 ส่วนด้วยกันคือ

1. *ส่วนที่ทำการประมวลผลตามฟังก์ชันการทำงานที่ได้รับมาจากหน่วยควบคุม (Control Unit)* มีฟังก์ชัน (Function) การทำงาน 16 ฟังก์ชัน โดยแบ่งเป็น ตัวถูกดำเนินการลอจิก 8 ฟังก์ชัน และ ตัวถูกดำเนินการทางคณิตศาสตร์ 8 ฟังก์ชัน ดังนี้

0000 = AND	OUT_ALU = (OP1 AND OP2)
0001 = XOR (EOR)	OUT_ALU = (OP1 XOR OP2)
0010 = SUB	OUT_ALU = (OP1 - OP2)
0011 = RSB	OUT_ALU = (OP2 - OP1)
0100 = ADD	OUT_ALU = (OP1 + OP2)
0101 = ADC	OUT_ALU = (OP1 + OP2 + CARRY)
0110 = SBC	OUT_ALU = (OP1 - OP2 + CARRY - 1)
0111 = RSC	OUT_ALU = (OP2 - OP1 + CARRY - 1)
1000 = TST	SAME AND BUT NOT WRITE RESULT
1001 = TEQ	SAME EOR BUT NOT WRITE RESULT
1010 = CMP	SAME SUB BUT NOT WRITE RESULT
1011 = CMN	SAME ADD BUT NOT WRITE RESULT
1100 = ORR	OUT_ALU = (OP1 OR OP2)
111010 = BIC	OUT_ALU = OP1 AND (NOT OP2)

2. *ส่วนที่ทำการกำหนดค่า บ่งชี้ ต่างๆ* จะทำการกำหนดค่าบ่งชี้ ได้แก่ ตัวบ่งชี้การทด (Carry Flag, C), ตัวบ่งชี้การล้น (Over Flag, V), ตัวบ่งชี้ค่าศูนย์ (Zero Flag, Z), และ ตัวบ่งชี้ค่าลบ (Negative Flag, N) ก็ต่อเมื่อค่า Set_fg มีค่าเป็น 1 คือ เป็นการประมวลผลที่กำหนดให้มีการเปลี่ยนแปลงค่า บ่งชี้ด้วย และจะทำการเปลี่ยนแปลงทุกๆ ขอบขาลงของสัญญาณนาฬิกา

3. *ส่วนที่ทำการกำหนดค่าเงื่อนไขการบ่งชี้ เพื่อส่งไปให้ตัวควบคุม* จะนำค่าตัวบ่งชี้ที่เปลี่ยนแปลงแล้วมาทำการตรวจสอบว่าเป็นไปตามเงื่อนไข (Condition) ของเงื่อนไขนั้นๆ หรือไม่ ซึ่งจะกำหนดเป็น 1 เมื่อเป็นไปตามเงื่อนไข โดยมีเงื่อนไข ดังนี้

0000 = EQ - Z set (equal)

0001 = NE - Z clear (not equal)

0010 = CS – C set (unsigned higher or same)

0011 = CC – C clear (unsigned lower)

0100 = MI – N set (negative)

0101 = PL – N clear (positive or zero)

0110 = VS – O set (overflow)

0111 = VC – O clear (no overflow)

1000 = HI – C set and Z clear (unsigned higher)

1001 = LS – C clear and Z set (unsigned lower or same)

1010 = GE – N set and O set , or N clear and O clear (greater or equal)

1011 = LT – N set and V clear, or N clear and V set (less than)

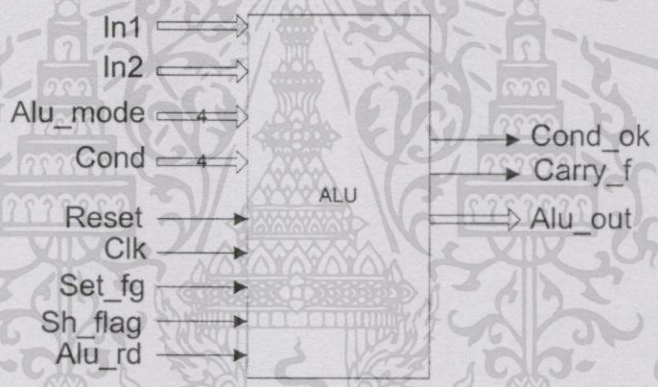
1100 = GT – Z clear, and either N set and V set, or N clear and V clear (greater than)

1101 = LE – Z set, or N set and V clear, or N clear and V set (less than equal)

1110 = AL – always

1111 = NV – never

โดยค่าตัวถูกดำเนินการที่ 2 ของหน่วยคำนวณและตรรกะ นั้นเป็นค่าที่มาจากตัวชิฟข้อมูลหรือตัวคูณ



รูปที่ 3.2 ส่วนของหน่วยคำนวณและตรรกะ

จากการศึกษารูปแบบการทำงานทั้งหมดของหน่วยคำนวณและตรรกะสามารถทำการออกแบบได้ดังรูปที่ 3.2 โดยการแบ่งเป็นส่วนของอินพุทที่รับข้อมูล 2 พอร์ทคือ In1 และ In2 โดยผลลัพธ์ที่ได้จากการทำงานในส่วนนี้จะถูกส่งออกทางพอร์ท Alu_out ทั้งนี้การทำงานจะขึ้นอยู่กับสัญญาณนาฬิกา โดยจะมีรายละเอียดของหน้าที่ในแต่ละขาสัญญาณดังนี้

Alu_mode	บอกฟังก์ชั่นการทำงานของหน่วยคำนวณและตรรกะ
Cond	บอกเงื่อนไขของการทำงาน
Set_fg	ควบคุมคำนวณค่าลงตัวบ่งชี้
Alu_rd	ทำหน้าที่บอกเมื่อมีข้อมูลรอที่พอร์ทอินพุทเรียบร้อย
Cond_ok	ส่งกลับไปยังหน่วยควบคุมเมื่อเงื่อนไขการทำงานถูกต้อง

จากการศึกษาจึงได้ทำการออกแบบส่วนของหน่วยคำนวณและตรรกะให้มีการทำงาน โดยมี 4 ส่วนย่อยที่ทำงานพร้อมกันดังนี้

- ส่วนของการรับค่า จะต้องทำหน้าที่ในการรับข้อมูลที่พอร์ทอินพุท ซึ่งจะทำการตรวจสอบค่าจาก Alu_rd และทำงานสอดคล้องกับสัญญาณนาฬิกาขาขึ้น (0-1)
- ส่วนของการตรวจสอบเงื่อนไขการทำงาน โดยจะนำค่าที่ได้จาก Cond ซึ่งมีขนาด 4 บิตมาทำงานตรวจสอบ หากถูกต้องจะทำการกำหนดค่าสัญญาณให้กับ Cond_ok เพื่อส่งให้ส่วนควบคุมทราบว่าเงื่อนไขในการเข้าทำงานถูกต้อง
- ส่วนของการประมวลผลทำหน้าในการตรวจฟังก์ชั่นการทำงานจาก Alu_mode แล้วประมวลผลเพื่อให้ได้ผลลัพธ์ส่งให้กับพอร์ทเอาท์พุท
- ส่วนของการตรวจค่าตัวบ่งชี้ ทำการตรวจสัญญาณ Set_fg เมื่อไมโครโพรเซสเซอร์ต้องการให้เปลี่ยนแปลงค่าในตัวบ่งชี้ ซึ่งถ้าต้องการให้ทำการเปลี่ยนแปลงค่า จะนำค่าตัวบ่งชี้เข้ามาประมวลผลด้วย และเมื่อมีผลต่อตัวบ่งชี้ carry ก็จะสร้างสัญญาณให้กับหน่วยควบคุมทางขา carry_f

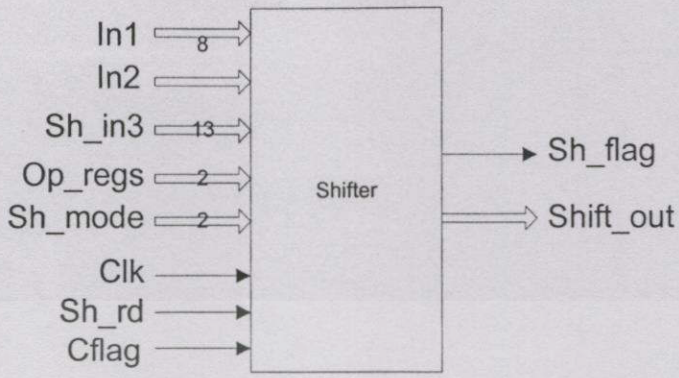
วงจรส่วนคำนวณทางคณิตศาสตร์และตรรกะ (ALU)

พบว่าส่วนคำนวณและตรรกะที่ได้ทำการออกแบบใช้พื้นที่ในส่วนของเอฟพีจีเอ (XC4062XLA)ดังนี้

ส่วนประกอบในเอฟพีจีเอ	ใช้งาน
ขาอินพุทเอาท์พุท	116 ขา
ตัวสร้างฟังก์ชั่น FG	257 ตัว
ตัวสร้างพังก์ชั่น H	105 ตัว
ฟลิปฟล็อป	5 ตัว
สามารถทำงานได้ที่ความถี่	69.8 MHz

<u>หมายเหตุ</u> ฟังก์ชั่น FG และ H ต่างเป็นส่วนของวงจรเชิงผสม (Combination circuit) ต่างกันที่ FG มีได้ 4 อินพุท ส่วน H มี 3 อินพุท

3.2.2 SHIFTER



รูปที่ 3.3 ส่วนของชิฟเตอร์1

มี 4 ฟังก์ชันการทำงาน ได้แก่

1. Logical Shift Left (LSL) ทำการชิฟข้อมูลไปทางซ้าย โดยถ้า N เป็นค่าจำนวนครั้งแล้ว

   N < 32 ได้ผลลัพธ์ตามค่าที่ชิฟ และตัวบ่งชี้ มีค่าตามบิตที่ 32-N

   N = 32 ผลลัพธ์เป็น 0 และตัวบ่งชี้มีค่าตามบิตที่ 0

   N > 32 ผลลัพธ์เป็น 0 และ ตัวบ่งชี้มีค่าเป็น 0

2. Logical Shift Right (LSR) ทำการชิฟข้อมูลไปทางขวา โดยถ้า N มีค่าเป็น

   N < 32 ได้ผลลัพธ์ตามค่าที่ชิฟา และตัวบ่งชี้มีค่าตามบิตที่ N-1

   N = 32 ผลลัพธ์เป็น 0 และ ตัวบ่งชี้ มีค่าตามบิตที่ 31

   N > 32 ผลลัพธ์เป็น 0 และ ตัวบ่งชี้ มีค่าเป็น 0

3. Arithmetic Shift Right (ASR) ทำการชิฟข้อมูลไปทางขวา แล้วเติมบิตทางซ้ายด้วยค่าของบิตที่ 31 (Sign Bit) ของค่าตัวถูกดำเนินการที่ถูกชิฟไป โดยถ้า N มีค่าเป็น

   N < 32 ได้ผลลัพธ์ตามค่าที่ชิฟ และตัวบ่งชี้ มีค่าตามบิตที่ N-1

   N >= 32 ผลลัพธ์เป็นค่า บิตเครื่องหมายจำนวน 32 บิต) และ ตัวบ่งชี้มีค่าตามบิตที่ 31

4. Rotate Right (ROR) ทำการหมุนค่าไปทางขวา โดยถ้า N มีค่าเป็น

   N < 32 ได้ผลลัพธ์ตามค่าที่ หมุนค่า และตัวบ่งชี้ มีค่าตามบิตที่ N-1

   N = 32 ผลลัพธ์เป็นค่าเดิม และ ตัวบ่งชี้ มีค่าตามบิตที่ 31

   N > 32 ผลลัพธ์ และ ตัวบ่งชี้ มีค่าเหมือนการ หมุนค่า ด้วยค่าจำนวนครั้งที่อยู่ระหว่าง 1-32 (ค่าเศษที่ได้จากการหารด้วย 32 นั่นเอง)

เมื่อทำการศึกษาและออกแบบส่วนชิฟเตอร์ได้ตามรูปที่ 3.3 โดยสามารถแบ่งออกเป็นส่วนของพอร์ทรับข้อมูล พอร์ทผลลัพธ์ และสัญญาณต่างๆ โดยมีหน้าที่ต่างๆดังนี้

In1	มีขนาด 8 บิตทำหน้าที่เป็นอินพุทที่เป็นจำนวนครั้งในการชิฟข้อมูลที่ได้จากค่าในรีจิสเตอร์ส่วนไบต์ล่าง (บิตที่ 0-7)
In2	มีขนาด 32 บิตทำหน้าที่เป็นอินพุทรับข้อมูลที่ต้องการชิฟ
Sh_In3	มีขนาด 13 บิตทำหน้าที่เป็นอินพุทรับจำนวนครั้งในการชิฟหรือค่าที่ต้องการชิฟซึ่งเป็นจำนวนเต็มที่ได้จากหน่วยควบคุม
Op_reg	ขนาด 2 บิตทำหน้าที่กำหนดรูปแบบของการชิฟดังนี้ 00 ได้จำนวนครั้งจาก In1 โดยทำการชิฟใน In2 01 ได้จำนวนครั้งจาก Sh_In3 และชิฟใน In2 10 ได้จำนวนครั้งและค่าที่ต้องการชิฟจาก Sh_In3
Sh_mode	มีขนาด 2 บิตใช้ในการกำหนดชนิดของการชิฟดังนี้ 00 ชิฟไปทางซ้ายแบบลอจิก (Logical shift left) 01 ชิฟไปทางขวาแบบลอจิก (Logical shift right) 10 ชิฟไปทางขวาแบบคิดบิตทด (Arithmetic shift right) 11 ชิฟไปทางซ้ายแบบคิดบิตทด (Arithmetic shift left)
Sh_rd	ใช้ในการอ่านค่าจากพอร์ทอินพุท
Cflag	บิตทดเพื่อใช้ในการชิฟแบบคิดบิตทด
Shift_out	ค่าผลลัพธ์ที่ได้จากการชิฟ
Sh_flag	ค่าตัวบ่งชี้ที่เปลี่ยนแปลงเมื่อมีการชิฟข้อมูล

ในหน่วยของชิฟเตอร์สามารถแบ่งออกเป็นส่วนย่อยในการออกแบบการทำงานภายในซึ่งทำงานพร้อมๆกันได้ดังต่อไปนี้

- ส่วนของการรับข้อมูลโดยจะทำงานตามสัญญาณนาฬิกา
- ส่วนของการชิฟโดยจะทำการตรวจสอบจากฟังก์ชั่นที่ได้จากสัญญาณ Sh_mode และได้ผลลัพธ์ใส่ในพอร์ทของ Shift_out
- ส่วนของการคำนวณค่าตัวบ่งชี้ โดยค่าที่ได้จากการเปลี่ยนแปลงจะถูกส่งออกไปทางขา sh_flag ให้กับหน่วยควบคุม

วงจรส่วนชิฟเตอร์

ส่วนประกอบในเอฟพีจีเอ	ใช้งาน
ขาอินพุทเอาท์พุท	93 ขา
ตัวสร้างฟังก์ชั่น FG	636 ตัว
ตัวสร้างพังก์ชั่น H	249 ตัว
ฟลิปฟล็อป	0 ตัว
สามารถทำงานได้ที่ความถี่	69.8 MHz

3.2.3 ตัวคูณ (MULTIPLIER)

รูปที่ 3.4 Multiplier Component

ในงานวิจัยนี้ได้ออกแบบวงจรให้ทำการคูณแบบ Booth's Algorithm เป็นรูปแบบของการคูณที่ใช้การสร้าง Booth Digit จากตัวคูณ ในที่นี้เป็น Booth 2 Encoding โดยวงรอบการคูณที่มากที่สุดจะเท่ากับจำนวนบิตของตัวคูณ หาร 2 ซึ่งใน ARM 7 เป็น 32 บิต ดังนั้นวงรอบการคูณที่มากที่สุดจะเท่ากับ 16 จำนวนวงรอบการคูณแปรผันตามค่าของตัวคูณ ดังตารางที่ 3.2

การสร้าง Booth Digit จะใช้การตัดตัวคูณทีละ 3 บิตโดยในครั้งแรกจะมีการเพิ่มบิตพิเศษเข้าไปต่หลังบิตต่ำสุดและให้ค่าเป็นศูนย์ 3 บิตถัดไปจะตัดเอาบิตบนของการจับครั้งแรกมาเป็นบิตล่างและจะทำไปเรื่อยๆ จนกว่าบิตที่เหลือเป็นศูนย์ทั้งหมดหรือตัดครบ 16 ครั้ง โดยในแต่ละ 3 บิตที่ตัดมาได้จะมีความหมายนี้คือ

000 = Result เดิม +0

001 = Result เดิม + Operand1

010 = Result เดิม + Operand1

011 = Result เดิม + (2*Operand1)

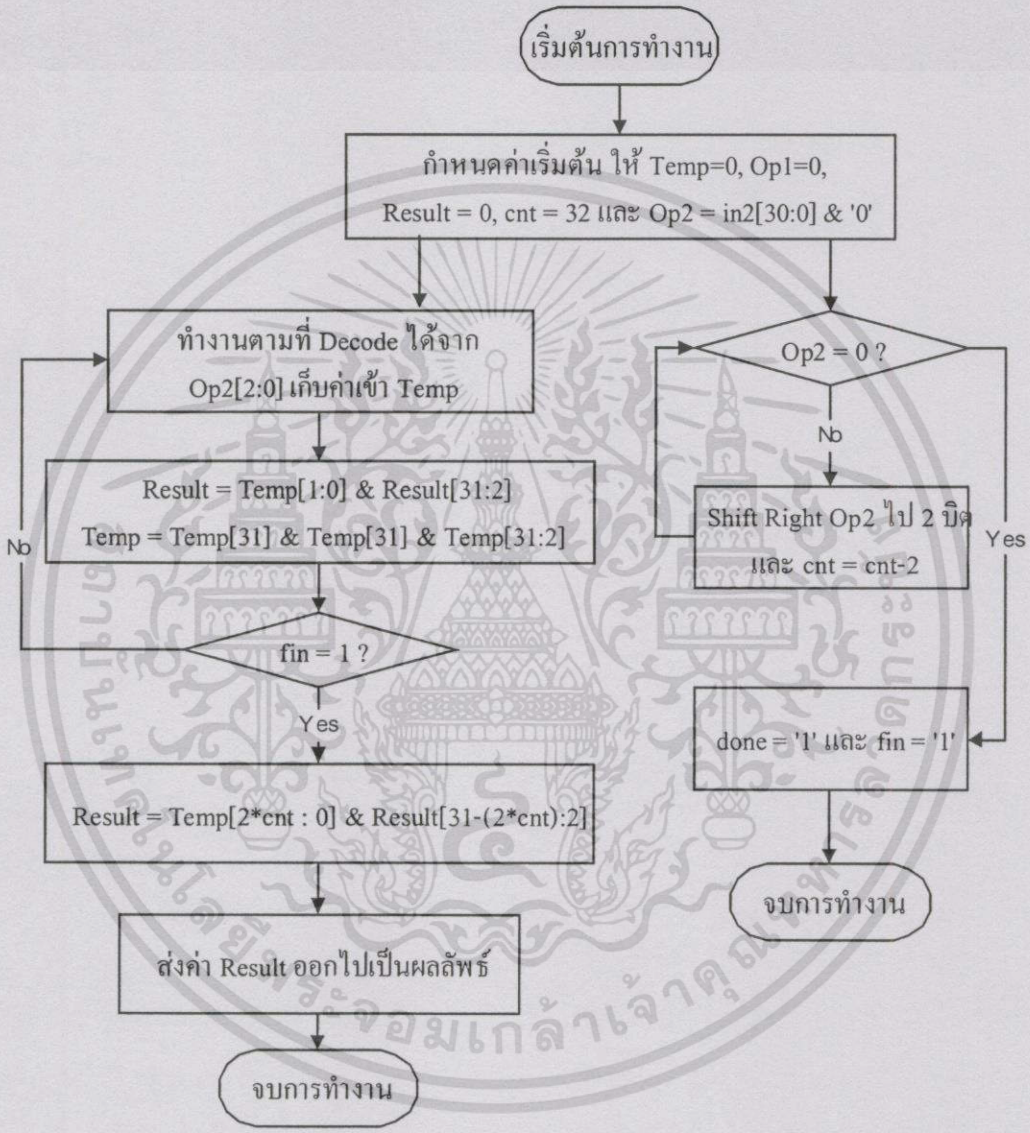
100 = Result เดิม – (2*Operand1)

101 = Result เดิม – Operand1

110 = Result เดิม – Operand1

111 = Result เดิม - 0

Booth's Algorithm ที่ได้ทำการออกแบบมาใช้ในตัวคูณ นั้นมีการทำงานดังผังงานด้านล่างนี้

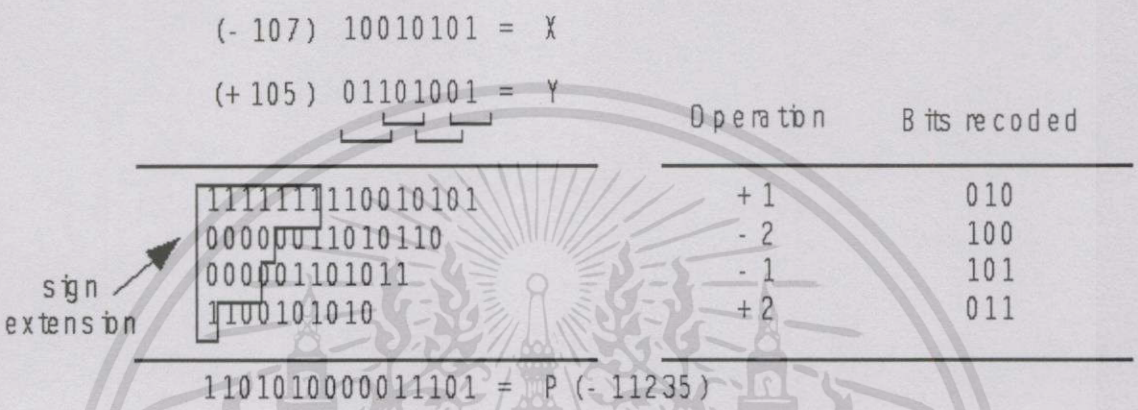


รูปที่ 3.5 Flow Chart of Booth's Multiplier

ในที่นี่จะยกตัวอย่างของการคูณโดยใช้ค่าเพียง 8 บิตเท่านั้น โดยที่การทำงานกับ 32 บิตก็มีลักษณะเหมือนกัน โดยให้ตัวตั้งมีค่าเป็น 10010101 หรือ –107 ตัวคูณมีค่าเป็น 01101001 หรือ + 105 ซึ่ง 10010101 * 01101001 (in1 * in2) โดยที่ In1 และ In2 เป็นพอร์ทที่ใช้รับข้อมูลขนาด 32 บิต และสัญญาณ Start จะทำหน้าที่ควบคุมการรับข้อมูล เมื่อทำงานเสร็จเรียบร้อยผลลัพธ์จะส่งออกทาง

พอร์ท Mul_out พร้อมกับให้สัญญาณ Done เพื่อบอกให้หน่วยควบคุมทราบเมื่อทำงานเสร็จ เนื่องจากการคูณจะเป็นต้องใช้เวลาในการทำงานมากกว่าปกติ ซึ่งอาจทำให้เกิดปัญหาของไปป์ไลน์ได้ จึงจำเป็นต้องมีการขยายช่วงเวลาของการทำงานจนกระทั่งการคูณจะเสร็จสิ้น

โดยการทำงานจะเริ่มจาการตัดบิตตัวคูณทีละ 3 บิตดังรูปที่ 3.6 โดยจะทำการเติมบิต 0 เพิ่มต่อท้ายเข้าไปหนึ่งตัว ดังนั้นจะตัดบิตตัวคูณได้ 4 ชุดคือ 010 100 101 และ 011 ซึ่งแต่ละชุดจะได้การทำงานดังรูป



รูปที่ 3.6 ตัวอย่างการทำงานของการคูณแบบ Booth's Algorithm

ผลคูณที่ได้จะเป็น 8 บิตล่างเท่านั้น (การทำงานจริงก็คือ 32 บิตล่าง) ซึ่งก็คือค่า 00011101 ตัวคูณประกอบด้วย 3 ส่วนด้วยกันคือ

3.2.3.1 ส่วนที่ทำการคำนวณ

จะทำงานที่ สัญญาณนาฬิกา ขาขึ้น ซึ่งจะทำการ ถอดรหัส ค่าของ ตัวถูกดำเนินการ ที่เป็นตัวคูณ 3 บิตหลัง แล้วทำการคำนวณตามการทำงานที่ ถอดรหัส ได้ และทำการ ชิฟค่าผลลัพธ์ที่ได้ในแต่ละครั้งไปทางขวา 2 บิต โดยนำ 2 บิตที่ ชิฟ ออกไปนั้นไปใส่ไว้ในตัวเก็บค่าผลลัพธ์ (ซึ่งก็จะถูก ชิฟไปทางขวา 2 บิตเช่นกัน แล้วเติมด้วย 2 บิตท้ายของค่าผลลัพธ์ที่ได้จากการคำนวณในแต่ละครั้งนั่นเอง)

3.2.3.2 ส่วนควบคุมซึ่งทำการกำหนดค่าสัญญาณการเสร็จสิ้นการทำงาน

จะทำงานที่ สัญญาณนาฬิกา ขาลง ซึ่งจะทำการพิจารณาค่า Op2 (ซึ่งเป็นค่า ตัวถูกดำเนินการ 31 บิต ที่ได้ มาจากการเติมค่าบิตที่ 30 ด้วยค่า 0 และบิตที่ 29-0 นั้นได้มาจากค่าบิตที่ 30-1 ของค่า ตัวถูกดำเนินการที่เป็นตัวคูณ) ว่าเป็นศูนย์หรือยัง ถ้าใช่ก็จะทำการส่งค่าสัญญาณ done และ fin ไปให้หน่วยควบคุมและส่วนที่ทำการคำนวณตามลำดับ ถ้าไม่ก็จะทำการลดค่า Count ลงไป 2 (เริ่มต้นการทำงานจะมีค่าเป็น 32) และทำการชิฟค่า Op2 ไปทางขวา 2 บิต

3.2.3.3 ส่วนที่ทำการส่งค่าผลลัพธ์

จะทำงานที่ สัญญาณนาฬิกา ขาขึ้น และเมื่อค่าสัญญาณ fin มีค่าเป็น 1 โดยจะทำการส่งค่า Result = Temp[cnt : 0] & Result[31-cnt:2] เมื่อ Temp คือ ตัวเก็บค่าผลลัพธ์ที่ได้จากการประมวลผลตามที่ ถอดรหัสได้ในแต่ละครั้ง โดยที่ Result คือ ตัวเก็บค่า 2 บิตหลังของ Temp ซึ่งจะถูก ชิฟไปทางขวา 2 บิตเพื่อรับค่าใหม่ และ Cnt คือ ตัวเก็บจำนวนครั้งในการ ชิฟก่อนที่จะส่งผลลัพธ์ออกไปนั่นเอง

ผลลัพธ์ที่ได้ออกมาจะมีค่าเพียง 32 บิตล่างโดยเมื่อทำการคูณเสร็จ จะทำการส่งสัญญาณ Done ไปบอก หน่วยควบคุม เพราะว่าระยะเวลา (Cycle Time) ในการคูณนั้นจะขึ้นตามขนาดของตัวถูกดำเนินการ ที่เป็นตัวคูณดังนี้

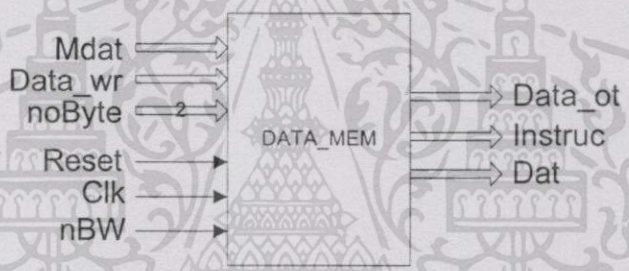
ตารางที่ 3.2 ค่าตัวถูกดำเนินการของการคูณ

ค่าข้อมูล	ระยะเวลาในการทำงาน
0 – 1	1
2 – 7	2
8 – 31	3
32 – 127	4
128 – 511	5
512 – 2047	6
2048 – 8191	7
8192 – 32767	8
32768 – 131071	9
131072 – 524287	10
524288 – 2097151	11
2097152 – 8388607	12
8388608 – 33554431	13
33554432 – 134217727	14
134217728 – 536870911	15
536870912 - 2147483648	16

วงจรส่วนการคูณ

ส่วนประกอบในเอฟพีจีเอ	ใช้งาน
ขาอินพุทเอาท์พุท	100 ขา
ตัวสร้างฟังก์ชั่น FG	558 ตัว
ตัวสร้างพังก์ชั่น H	178 ตัว
ฟลิปฟล็อป	267 ตัว
สามารถทำงานได้ที่ความถี่	26.7 MHz

3.2.4 ส่วนติดต่อกับหน่วยความจำ (Data Memory)



รูปที่ 3.7 ส่วนติดต่อหน่วยความจำ

เป็นส่วนที่จัดการกับข้อมูลที่อ่านมาหรือ ที่จะส่งไปให้หน่วยความจำ ให้สามารถนำไปใช้งานได้ เมื่อเป็นไบต์หรือเวิร์ดโดยมีการจัดการดังนี้

1. การโหลดค่าข้อมูลมาจากหน่วยความจำ (Load)

กรณีเป็นการโหลดแบบไบต์ (Byte) จะนำข้อมูลที่อ่านมาได้จากขาสัญญาณ Mdat ในไบต์ที่ต้องการไปไว้ในสัญญาณ Data_ot [7:0] และใส่บิตที่เหลือด้วยค่าศูนย์ ซึ่งตำแหน่งไบต์ที่ต้องการนั้นพิจารณาจากค่าAddress[1:0] แล้วส่งค่านั้นออกไปเพื่อเก็บค่าลง รีจิสเตอร์ กรณีเป็นการโหลดแบบ เวิร์ด (Word) จะนำค่าข้อมูลที่อ่านมาได้ส่งออกไปให้ รีจิสเตอร์ เลย

2. การเก็บค่าข้อมูลเข้าหน่วยความจำ (Store)

กรณีเป็นการเก็บค่าเข้าหน่วยความจำแบบไบต์ (Byte) จะนำค่า 8 บิตล่างของ Data_wr (ซึ่งก็คือค่าของรีจิสเตอร์) ใส่ลงไปทั้ง 4 ไบต์ของ บัสข้อมูลแล้วส่งค่าไปยังหน่วยความจำ กรณีเป็นการเก็บค่าเข้าหน่วยความจำแบบเวิร์ด (Word) จะนำค่าทั้ง 32 บิตของ รีจิสเตอร์ ใส่ลงไปในบัสข้อมูล

จากการศึกษาได้ทำการออกแบบให้ส่วนนี้มีขาสัญญาณเพื่อให้สามารถใช้งานดังรูปที่ 3.7 และมีรายละเอียดของสัญญาณดังนี้

Mdat	ขนาด 32 บิต เป็นบัสรับข้อมูล
Data_wr	ขนาด 32 บิต เป็นข้อมูลที่ต้องการเก็บในหน่วยความจำ
NoByte	ขนาด 2 บิต เพื่อเลือกว่าต้องการใช้ข้อมูลไบต์ไหน
NBW	ใช้เลือกว่าต้องการใช้ทีละไบต์หรือเวิร์ด
Data_ot	ขนาด 32 บิต เป็นข้อมูลที่ได้จัดเรียงแล้ว หลังจากเลือกไบต์หรือเวิร์ด
Instruc	ขนาด 32 บิต รับคำสั่งที่ถูกนำเอาเข้ามาทำงานในไมโครโพรเซสเซอร์
Dat	ขนาด 32 บิตใช้ส่งข้อมูลที่ต้องเก็บสู่หน่วยความจำ

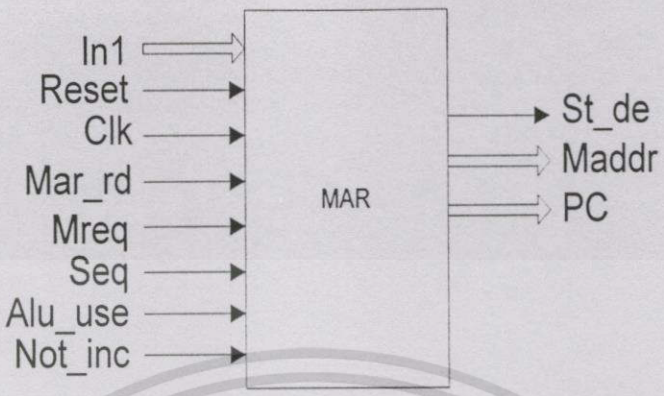
ส่วนของการติดต่อหน่วยความจำได้ทำการออกแบบโดยแบ่งเป็นส่วนย่อยภายในออกเป็นดังนี้

- ส่วนของการรับและส่งข้อมูล
- ส่วนของการเก็บข้อมูลเพื่อจัดเก็บลงสู่หน่วยความจำ
- ส่วนของการจัดเรียงข้อมูลเมื่อมีการเลือกไบต์หรือเวิร์ด

วงจรส่วนการติดต่อหน่วยความจำ

ส่วนประกอบในเอฟพีจีเอ	ใช้งาน
ขาอินพุทเอาท์พุท	166 ขา
ตัวสร้างฟังก์ชั่น FG	88 ตัว
ตัวสร้างพังก์ชั่น H	8 ตัว
ฟลิปฟล็อป	0 ตัว
สามารถทำงานได้ที่ความถี่	69.8 MHz

3.2.5 ส่วนจัดการและเก็บเลขที่อยู่



รูปที่ 3.8 ส่วนจัดการและเก็บเลขที่อยู่

เป็นส่วนที่จัดการส่งค่าตำแหน่งที่ต้องการจะอ้างอิงหน่วยความจำออกไป โดยมีโหมดการทำงานด้วยกันดังนี้

1. อ้างอิงตำแหน่งโดยใช้ค่าจาก พีซี เพื่อเป็นการนำคำสั่งถัดไปเข้ามาเตรียมไว้

จะทำการส่งค่า Program Counter (PC) ซึ่งได้ทำการนับเพิ่มไป 4 ตำแหน่งไว้แล้ว ส่งออกไปที่บัสเลขที่อยู่ซึ่งเป็นการอ้างอิงหน่วยความจำ เพื่อนำคำสั่งถัดไปเข้ามาเตรียมไว้ จากนั้นจะทำการเพิ่มค่า พีซี ไป 4 ตำแหน่ง

2. อ้างอิงตำแหน่งโดยใช้ค่าจาก หน่วยคำนวณและตรรกะ โดยไม่มีการเพิ่มค่า พีซี

จะทำการนำค่าที่ได้มาจาก หน่วยคำนวณและตรรกะ ส่งออกไปที่บัสเลขที่อยู่โดยที่จะไม่ทำการเพิ่มค่าของพีซีเพื่อทำการ Load, Swap, Store แบบ Pre-Index รวมทั้ง Multiple Load และ Multiple Store ค่าแรกจากหรือลงหน่วยความจำ

3. อ้างอิงตำแหน่งโดยใช้ค่าจาก TO_MAR

จะใช้ค่าจาก TO_MAR ซึ่งได้ทำการเก็บค่าไว้ใน วัฏจักร ที่แล้ว โดยเป็นค่าที่อ่านมาจาก รีจิสเตอร์ เพื่อใช้เป็นตำแหน่งอ้างอิงเพื่อการStore แบบ Post-Index ส่งออกไปที่บัสเลขที่อยู่โดยที่ไม่ทำการเพิ่มค่าของ พีซี

4. อ้างอิงตำแหน่งโดยใช้ค่าจาก หน่วยคำนวณและตรรกะ และมีการเพิ่มค่า พีซี

จะใช้ค่าจาก หน่วยคำนวณและตรรกะ ส่งออกไปที่บัสเลขที่อยู่ซึ่งเป็นการอ้างอิงหน่วยความจำ เพื่อกระโดด (Branch)ไปตำแหน่งที่ต้องการ และเพิ่มค่า พีซี โดยใช้ค่าของ ALU+4 เพื่อที่จะไว้อ้างอิงตำแหน่ง ในการนำเอาคำสั่งมาในรอบถัดไป

5. อ้างอิงตำแหน่งจากตำแหน่งเดิมไปอีก 4

จะใช้ค่าตำแหน่งเดิมเพิ่มไปอีก 4 ส่งออกไปที่บัสเลขที่อยู่เพื่อที่จะอ่านค่าจากหรือเก็บค่าเข้าหน่วยความจำลงไปในตำแหน่งถัดจากเดิม ใช้สำหรับในการทำคำสั่ง Multiple Load และ Multiple Store ซึ่งไม่ใช่การโหลดหรือเก็บค่าแรกลงหน่วยความจำ

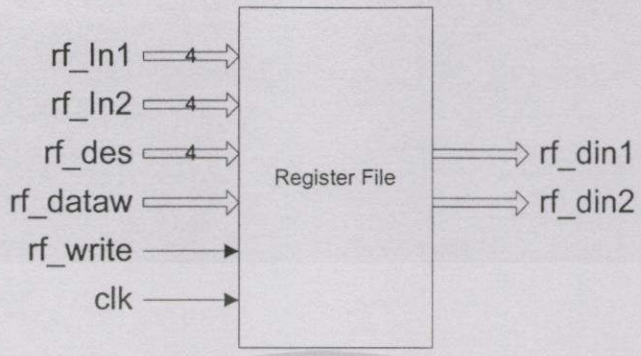
ส่วนของการจัดค่าตำแหน่งเลขที่อยู่ได้ทำการออกแบบโดยให้มีสัญญาณการใช้งานดังต่อไปนี้

In1	ขนาด 32 บิตรับค่าเลขที่อยู่
MAR_rd	สัญญาณเพื่อให้รับค่าในอินพุทเข้ามา
Mreq	ตรวจว่ามีการเรียกใช้หน่วยความจำหรือไม่
Seq	สัญญาณเพื่อบอกถึงลำดับการทำงานว่าต่อเนื่องหรือไม่
Alu_use	มีขนาด 2 บิตเพื่อเลือกรูปแบบของการเปลี่ยนค่าเลขที่อยู่ ดังนี้
	00 mar <= pc , pc <= pc + "100"
	01 mar <= to_mar (store in post index)
	10 mar <= alu_res , pc <= alu_res + "100"
	11 mar <= alu_res
Not_inc	สัญญาณบอกเมื่อไม่ต้องการเพิ่มค่าเลขที่อยู่
St_de	สัญญาณที่สร้างขึ้นให้สอดคล้องกับสัญญาณนาฬิกาเพื่อนำไปใช้ในการตรวจสอบ
Maddr	ขนาด 32 บิตค่าเลขที่อยู่ที่ได้ทำการเปลี่ยนแปลงแล้ว
PC	ขนาด 32 บิตค่าเลขที่อยู่ที่เก็บค่าตำแหน่งที่ทำงาน ณ. ปัจจุบัน

วงจรส่วนการติดต่อหน่วยความจำ

ส่วนประกอบในเอฟพีจีเอ	ใช้งาน
ขาอินพุทเอาท์พุท	102 ขา
ตัวสร้างฟังก์ชั่น FG	155 ตัว
ตัวสร้างพังก์ชั่น H	38 ตัว
ฟลิปฟล็อป	97 ตัว
สามารถทำงานได้ที่ความถี่	11.5 MHz

3.2.6 รีจิสเตอร์ (REGISTER FILE)



รูปที่ 3.9 ส่วนของรีจิสเตอร์

ประกอบด้วยรีจิสเตอร์ 32 บิตจำนวน 16 รีจิสเตอร์ คือ ตั้งแต่ R0-R15 ในบางคำสั่งจะไม่ควรใช้ค่าของ R15 เนื่องจากอาจทำให้เกิดความผิดพลาดได้ ซึ่ง R15 นั้นคือพีซี และมี R14 เป็นรีจิส-เตอร์ที่จะใช้เก็บค่าตำแหน่งเดิมที่ต้องการกระโดดกลับมา (ใช้ในคำสั่ง Branch with Link ซึ่งจะเป็นตำแหน่งของคำสั่งถัดจากคำสั่ง Branch) รีจิสเตอร์ นั้นจะมี 2 ช่องทางการอ่าน ทำให้สามารถทำการอ่านค่าของรีจิสเตอร์ ได้พร้อมกัน 2 ตัว จากรูปที่ 3.9 ได้ทำการออกแบบส่วนนี้และสามารถแจงรายละเอียดการทำงานของแต่ละขาสัญญาณได้ดังนี้

Rf_in1	ขนาด 4 บิตใช้ระบุรีจิสเตอร์ที่ใช้เป็นตัวตั้ง
Rf_in2	ขนาด 4 บิตใช้ระบุรีจิสเตอร์ที่ใช้เป็นตัวกระทำ
Rf_des	ขนาด 4 บิตใช้ระบุรีจิสเตอร์ที่เก็บผลลัพธ์
Rf_dataw	ขนาด 32 บิตเป็นพอร์ทการเขียนข้อมูลลงสู่รีจิสเตอร์
Rf_write	สัญญาณเลือกระหว่างการเขียนหรืออ่านค่าจากรีจิสเตอร์
Rf_din1	ขนาด 32 บิตเป็นข้อมูลที่ส่งออกสู่บัสข้อมูลที่ 1
Rf_din2	ขนาด 32 บิตเป็นข้อมูลที่ส่งออกสู่บัสข้อมูลที่ 2

การออกแบบในส่วนนี้ความง่ายโดยการประกาศจองเนื้อที่ว่างเป็นลักษณะของตารางของแถวลำดับ(Array) เพื่อเก็บค่าของรีจิสเตอร์ 16 ตัว โดยมีการเข้าถึงแต่ละแถวและข้อมูลได้จากสัญญาณ rf_in1, rf_in2 และ rf_des นอกจากนี้จะทำการตรวจสอบสถานะว่าเป็นการอ่านหรือเขียนข้อมูลจากสัญญาณ Rf_write เพื่อส่งข้อมูลออกทางพอร์ทที่ต้องการ

วงจรส่วนรีจิสเตอร์

ส่วนประกอบในเอฟพีจีเอ	ใช้งาน
ขาอินพุทเอาท์พุท	110 ขา
ตัวสร้างฟังก์ชั่น FG	628 ตัว
ตัวสร้างพังก์ชั่น H	192 ตัว
ฟลิปฟล็อป	512 ตัว
สามารถทำงานได้ที่ความถี่	33.8 MHz

พบว่าผลจากการสังเคราะห์วงจรมีส่วนของฟลิปฟล็อป ที่ใช้งานจำนวนมากเนื่องจากใช้ในการค่าในแต่ละบิตและแถวของรีจิสเตอร์ นั้นคือ รีจิสเตอร์16 ตัวแต่ละตัวมี 32 บิต จึงต้องใช้งานถึง 512 ฟลิปฟล็อป

3.2.7 ส่วนควบคุม (Control Unit)

ประกอบด้วยกัน 2 ส่วนคือ

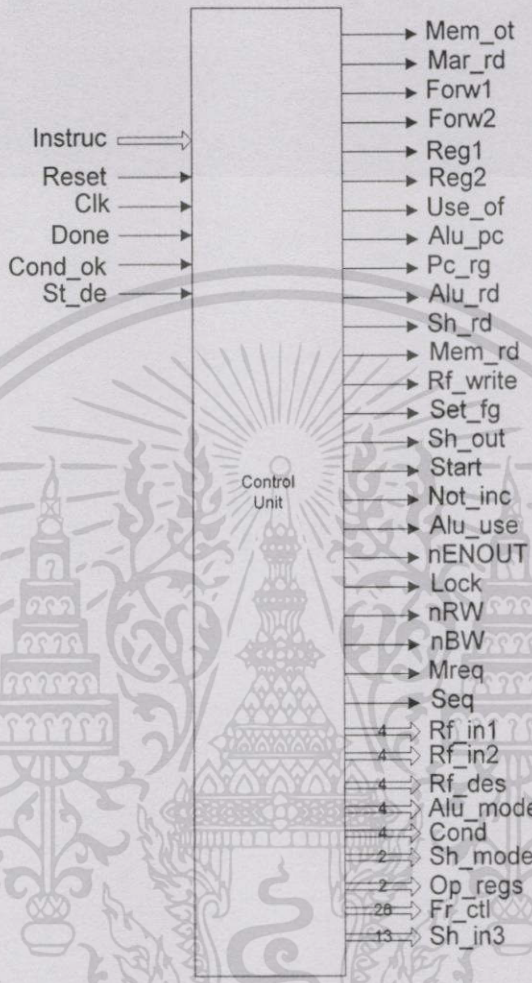
1. ส่วนที่ทำการเอาค่ามาและถอดรหัส คำสั่ง

ในการนำเอาคำสั่งมาจะรับคำสั่งมาจาก ส่วนติดต่อกับหน่วยความจำ โดยหากคำสั่งที่กำลังทำการ การประมวลผล อยู่นั้นเป็นคำสั่งที่มีการทำงานกับหน่วยความจำ จะมีการรำเอาคำสั่งมาเก็บไว้ในที่พักข้อมูลก่อน แทนที่จะทำการเก็บลงรีจิสเตอร์คำสั่ง

ในการ ถอดรหัส จะทำการถอดรหัสคำสั่งที่อยู่ในรีจิสเตอร์คำสั่งว่าคำสั่งใด เพื่อกำหนดสัญญาณควบคุมต่างๆ ไปยังแต่ละชิ้นส่วน โดยที่ในหนึ่งคำสั่งอาจจะมีการถอดรหัส มากกว่า 1 คาบสัญญาณนาฬิกา ส่วน DataProcessing แบบที่มีการชิฟที่มี ตัวถูกดำเนินการ เป็นค่าของรีจิสเตอร์ และจำนวนครั้งในการชิฟมาจากค่า 8 บิตล่างของรีจิสเตอร์ในวัฏจักรแรกจะทำการ ถอดรหัส เพื่อให้ได้ค่าของรีจิสเตอร์ที่เป็นจำนวนครั้งในการชิฟ ส่งไปให้ค่าที่จะทำการชิฟ ส่วนในรอบที่สองของการทำงาน จะทำการถอดรหัสเพื่อให้ได้ค่ารีจิสเตอร์มาทำการประมวลผลในหน่วยคำนวณและตรรกะ

2. *ส่วนที่ควบคุมการ การประมวลผล ของคำสั่งในแต่ละรอบการทำงาน*

จะทำการกำหนดสัญญาณต่างๆ ไปยังแต่ละ ชิ้นส่วน ให้ทำงาน โดยที่แต่ละคำสั่งนั้นจะมีคาบในการประมวลผล ไม่เท่ากัน



รูปที่ 3.10 ส่วนควบคุม

จากการออกแบบได้ขาสัญญาณตามรูปที่ 3.10 โดยจะเป็นส่วนของการส่งสัญญาณไปควบคุมชิ้นส่วนต่างๆที่ทำงานให้เป็นระบบสอดคล้องกัน และส่วนของการรับคำสั่งจากหน่วยความจำเพื่อทำการถอดรหัสและสร้างสัญญาณเพื่อควบคุมการทำงาน พบว่าการออกแบบในส่วนนี้จะมีขนาดใหญ่ และซับซ้อนกว่าส่วนอื่นๆ ดังนั้นการออกแบบที่ดีควรวางการไหลหรือการทำงานทั้งหมดของหน่วยนี้ก่อนเริ่มลงมือเขียนโค้ดการออกแบบ

วงจรส่วนรีจิสเตอร์

ส่วนประกอบในเอฟพีจีเอ	ใช้งาน
ขาอินพุทเอาท์พุท	127 ขา
ตัวสร้างฟังก์ชั่น FG	631 ตัว
ตัวสร้างพังก์ชั่น H	240 ตัว
ฟลิปฟล็อป	228 ตัว
สามารถทำงานได้ที่ความถี่	9.6 MHz

พบว่าจากการสังเคราะห์วงจรได้ส่วนของหน่วยควบคุมดังผลข้างต้น และเป็นส่วนที่ใช้ความถี่ของสัญญาณนาฬิกาที่น้อยที่สุด เนื่องจากค่าความหน่วงของเวลาเกิดขึ้นในส่วนนี้มากที่สุด

3.2.8 ไมโครโพรเซสเซอร์ ARM 7

เมื่อเชื่อมส่วนต่างๆเข้าด้วยกันทั้งหมด จึงนำไปสังเคราะห์วงจรรวมเพื่อให้ได้ไมโครโปรเซสเซอร์ ARM 7 ตามที่ต้องการ โดยผลจากการสังเคราะห์วงจรมีดังนี้

ส่วนประกอบในเอฟพีจีเอ	ใช้งาน
ขาอินพุทเอาท์พุท	109 ขา
ตัวสร้างฟังก์ชั่น FG	3109 ตัว
ตัวสร้างพังก์ชั่น H	1015 ตัว
ฟลิปฟล็อป	1375 ตัว
สามารถทำงานได้ที่ความถี่	9 MHz

โดยส่วนของรีจิสเตอร์จะเป็นส่วนที่ใช้ฟลิปฟล็อปในการทำงานสูงที่สุดถึง 512 ตัวและส่วนที่ใช้วงจรเชิงผสมมากที่สุดคือตัวชิฟเตอร์ โดยใช้ทั้งหมด 858 ตัว ส่วนหน่วยที่ทำงานเร็วที่สุดคือชิฟเตอร์หน่วยติดต่อหน่วยความจำ และหน่วยคำนวณและตรรกะ

บทที่ 4

การตรวจสอบการทำงานด้วยเทคนิคของ Abstract State Machine และจำลองการทำงาน

4.1 Abstract State Machine

เนื่องจากภาษาที่ใช้กำหนดคุณลักษณะของวงจรและภาษาที่ใช้ในการบรรยายการทำงานเป็นคนละภาษากันทำให้ไม่สามารถนำมาตรวจสอบความถูกต้องกันได้โดยตรง ดังนั้นจึงจำเป็นต้องมีการเปลี่ยนรูปแบบของภาษาทั้งสองให้อยู่ในรูปแบบเดียวกันเพื่อจะได้นำมาตรวจสอบกันได้โดยตรง ซึ่งในวิทยานิพนธ์นี้ได้เลือกใช้เทคนิคของ Abstract State Machine โดยมีวิธีการแปลงในบทที่ 2

วิทยานิพนธ์นี้ได้ทำการเปลี่ยนรูปแบบของภาษาที่ใช้ในการบรรยายการทำงานในองค์ประกอบย่อยต่างๆ ที่ได้ออกแบบเพื่อแบบเทียบกับข้อกำหนดคุณลักษณะที่มี แล้วทำการตรวจสอบความถูกต้องโดยการป้อนอินพุทแล้วตรวจดูการเปลี่ยนแปลงสถานะและสถานะสุดท้ายในการทำงานที่เกิดจากการป้อนอินพุท

4.1.1 การตรวจสอบการทำงานของรีจิสเตอร์

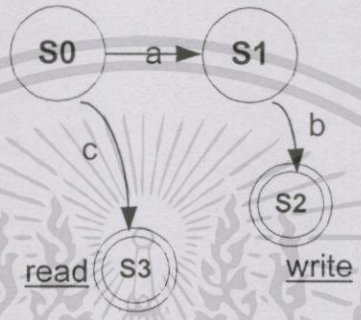
การทำงานของหน่วยรีจิสเตอร์จะมี 2 ส่วนที่สำคัญคือการเขียนข้อมูลลงรีจิสเตอร์และการอ่านข้อมูลจากรีจิสเตอร์โดยจะมีการตรวจสอบสัญญาณควาบคุมที่จะใช้ในการอ่านหรือเขียนข้อมูล คือ rf_write และ rf_read

```
Rule : Register_file
        If rf_write and Content(DesReg) then
                Content(Desop) = Content(DesReg)
        elsif rf_read then
                Aop = Content(AopReg)
                Bop = Content(BopReg)
        End if;
Specification :
        Rule a : rf_write
        Rule b : Content(DesReg)
```

Rule c : rf_read

รูปที่ 4.1 ข้อกำหนดคุณลักษณะการทำงานของรีจิสเตอร์

จากข้อกำหนดคุณลักษณะการทำงานของรีจิสเตอร์ดังรูปที่ 4.1 สามารถแปลงให้อยู่ในรูปที่เป็นแบบแผนพื่อทำการตรวจสอบได้ดังรูปที่ 4.2



รูปที่ 4.2 ไดอะแกรมสถานะของคุณลักษณะที่ต้องการของรีจิสเตอร์

ในข้อมูลอินพุทของข้อกำหนดคุณลักษณะสามารถทำการทดสอบการเปลี่ยนแปลงสถานะจากอินพุทที่ป้อนเข้าไปได้ดังตารางที่ 4.1

ตารางที่ 4.1 ผลจากการแทนอินพุทเข้าสู่องค์ประกอบย่อยของหน่วยรีจิสเตอร์

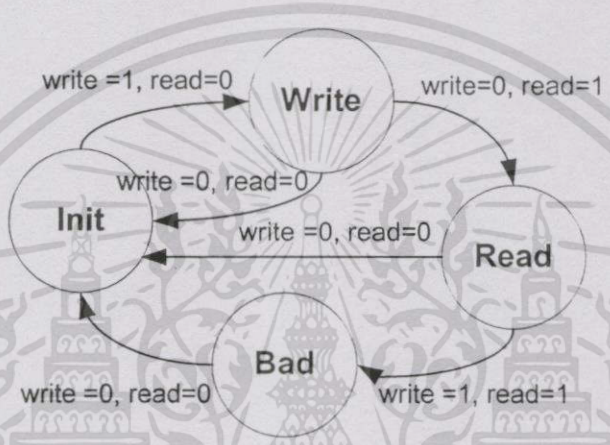
สายอักขระ	การทำงาน	ผลที่ได้
Ab	Rf_write and Content(DesReg)	Accept
C	Rf_read	Accept

เมื่อได้ไดอะแกรมสถานะของการทำงานของหน่วยรีจิสเตอร์จากข้อกำหนดคุณลักษณะแล้ว ก็จะทำการเปลี่ยนรูปแบบของภาษาที่ใช้ในการบรรยายการทำงานของ หน่วยรีจีสเตอร์ให้เป็นรูปแบบเดียวกันดังรูปที่ 4.3

```
WRITE_REG:
process(clk)
begin
if rising_edge(clk) then
```

```
            if rf_write = '1' then
                regs(CONV_INTEGER(rf_des)) <= rf_dataw;
            end if;
        end if;
end process WRITE_REG;
```

รูปที่ 4.3 ภาษาที่ใช้บรรยายการทำงานหน่วยรีจิสเตอร์



รูปที่ 4.4 ไดอะแกรมสถานะของหน่วยรีจิสเตอร์จากภาษาที่ใช้บรรยายการทำงาน

เมื่อทำการป้อนอินพุทให้กับไดอะแกรมสถานะแล้วทำการตรวจสอบการทำงานของระบบจะเห็นว่าทั้งไดอะแกรมที่ได้จากข้อกำหนดคุณลักษณะและไดอะแกรมสถานะที่ได้จากภาษาบรรยายการทำงานมีการเปลี่ยนสถานะที่เหมือนกันและผลสุดท้ายจะไปอยู่ยังสถานะเดียวกันคือ Accept ทั้งคู่ดังตารางที่ 4.2

ตารางที่ 4.2 ผลการตรวจสอบวงจรส่วนรีจิสเตอร์

ชุดการ trace	สถานะ	ผลที่ได้
1. write = 1, read = 0	Write	Accept
2. write = 0, read = 1	Read	Accept
3. write = 1, read = 1	Bad	Accept
4. write = 0, read = 0	Init	Accept

สรุปผลที่ได้

พบว่าองค์ประกอบย่อยที่ออกแบบด้วยภาษา VHDL ของหน่วยรีจิสเตอร์ไม่มีข้อผิดพลาด ทำให้ได้อะแกรมสถานะที่เป็นคุณลักษณะที่กำหนดยอมรับการทำงานที่ป้อนให้ผ่านทางสายอักขระ ทั้งในส่วนของการอ่านและเขียนข้อมูลลงบนรีจิสเตอร์

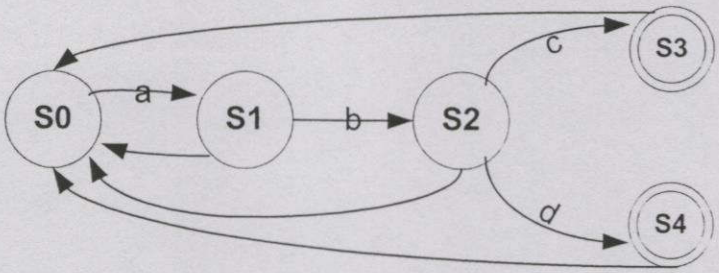
4.1.2 การตรวจสอบการทำงานของหน่วยคำนวณและตรรกะ

หน่วยคำนวณและตรรกะจะแบ่งการทำงานออกเป็นหลายส่วนเช่น ส่วนอ่านค่าข้อมูล ส่วนประมวลและส่วนการตั้งค่าสถานะ (Flag) ทำให้การบรรยายการทำงานแยกกันออกเป็น 3 ส่วน แต่ในภาษาที่ใช้กำหนดคุณลักษณะจะรวมเป็นส่วนเดียวกันดังรูปที่ 4.5 ดังนั้นในส่วนของการบรรยายการทำงานจะแยกไดอะแกรมออกเป็น 3 ไดอะแกรม

```
Rule : ExecuteALU
If  ExecuteOK and ALUInstr(Instr)  then
    If Satisfies(Status, CondCode(Instr))  then
        If  WriteResult(Instr)   then
            Contents(DesReg)  := ALU( ALUop(Instr), Aop, Bop, Carry(Status))
        Endif
        If  SetCondCode(Instr)  then
            Status := UpdateStatus( Status, ALUop(Instr), Aop, Bop, ShiftCarryOp)
        Endif
    Endif
Endif

Specification :
        Rule a : ExecuteOK and ALUInstr(Instr)
        Rule b : Satisfies(Status, CondCode(Instr))
        Rule c : WriteResult(Instr)
        Rule d : SetCondCode(Instr)
```

รูปที่ 4.5 ข้อกำหนดคุณลักษณะการทำงานของหน่วยประมวลผลและตรรกะ



รูปที่ 4.6 ไดอะแกรมสถานะของคุณลักษณะที่ต้องการของหน่วยคำนวณและตรรกะ

ในข้อมูลอินพุทของข้อกำหนดคุณลักษณะสามารถทำการทดสอบการเปลี่ยนแปลงสถานะจากอินพุทที่ป้อนเข้าไปได้ดังตารางที่ 4.3

ตารางที่ 4.3 ผลจากการแทนองค์ประกอบย่อย ของหน่วยคำนวณและตรรกะ

สายอักขระ	การทำงาน	ผลที่ได้
Ac	ExcuteOK and ALUInstr(Instr) และ WriteResult(Instr)	Not accept
Acd	ExcuteOK and ALUInstr(Instr) และ WriteResult(Instr) และ SetCondCode(Instr)	Not accept
Ad	ExcuteOK and ALUInstr(Instr) และ SetCondCode(Instr)	Not accept

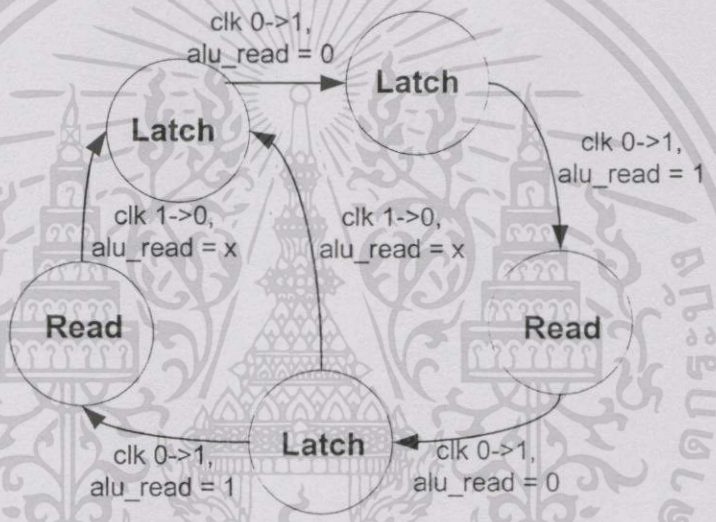
เมื่อได้ไดอะแกรมสถานะของการทำงานของหน่วยคำนวณและตรรกะจากข้อกำหนดคุณลักษณะแล้ว ก็จะทำการเปลี่ยนรูปแบบของภาษาที่ใช้ในการบรรยายการทำงานของ หน่วยคำนวณและตรรกะให้เป็นรูปแบบเดียวกัน

ALU_read_input เป็นส่วนที่ใช้ในการอ่านข้อมูลเข้าสู่หน่วยคำนวณทางคณิตศาสตร์และตรรกะซึ่งต้องทำการตรวจสอบตามคุณลักษณะที่ต้องการดังรูปที่ 4.8 ซึ่งการตรวจสอบกรณีที่สามารถเกิดขึ้นกับวงจรส่วนนี้ทั้งหมดแสดงดังตารางที่ 4.4 พร้อมทั้งผลของการตรวจสอบว่าเกิดการ Accept หรือไม่

```
READ_INPUT:
        Process (rclk)
        begin
                if rising_edge(rclk) then
```

```
                func <= alu_md;
                cin      <= carry;
                if alu_rd = '1' then
                        in1 <= alu_in1;
                end if;
        end if;
end process READ_INPUT;
```

รูปที่ 4.7 ภาษาที่ใช้บรรยายการทำงานชุดอ่านข้อมูลของหน่วยคำนวณและตรรกะ



รูปที่ 4.8 ไดอะแกรมสถานะของหน่วยคำนวณและตรรกะส่วนอ่านข้อมูล

ตารางที่ 4.4 ผลการตรวจสอบวงจรส่วนอ่านค่าเข้าสู่หน่วยคำนวณทางคณิตศาสตร์และตรรกะ

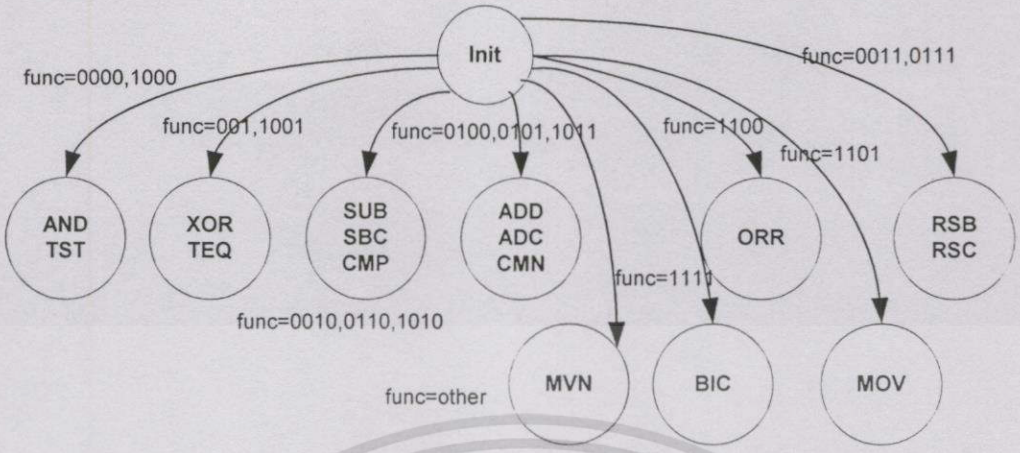
ชุดการ trace	สถานะ	ผลที่ได้
1. clk = 1 → 0, alu_read = '0'	Latch	Accept
2. clk = 1 → 0, alu_read = '1'	Latch	Accept
3. clk = 0 → 1, alu_read = '0'	Latch	Accept
4. clk = 0 → 1, alu_read = '1'	Read	Accept

ALU_Do เป็นการทำงานหลักของหน่วยคำนวณทางคณิตศาสตร์และตรรกะทั้ง 16 คำสั่งของกลุ่ม Data Processing ในไมโครโพรเซสเซอร์ ARM7 ดังรูปที่ 4.9 และกรณีของการตรวจสอบดังตารางที่ 4.10

```
DO_ALU:
        process (in1,alu_in2,func,cin)
                variable res      : std_logic_vector(16 downto 0);
                variable carry    : std_logic;
        begin
                cf <= '0';
                vf <= '0';
                case func is
                        when "0000" | "1000" =>
                        -- AND | TST
                          res(15 downto 0) := in1 and alu_in2;
                          extra <= '0';
                        when "0001" | "1001" =>
                        -- XOR (EOR) | TEQ
                          extra <= '0';
                          res(15 downto 0) := in1 xor alu_in2;
                        when "0010" | "0110" | "1010" =>
                        -- SUB | SBC | CMP
                          extra <= '1';
                          carry := cin or (not func(2));
                          res     := ('0' & in1) + ('1' & not alu_in2) + carry;
                          cf <= res(16);
                          vf <= in1(15) xor (alu_in2(15) and res(15));
                        when "0100" | "0101" | "1011" =>
                        -- ADD | ADC | CMN
                          extra <= '1';
                          carry := (func(2) and func(0)) and cin;
                          res     := ('0' & in1) + ('0' & alu_in2) + carry;
                         cf <= res(16);
                          vf <= (not (in1(15) xor alu_in2(15))) and (in1(15) xor res(15));
                        when "1100" =>
                        -- ORR
```

```
            extra <= '0';
            res(15 downto 0) := in1 or alu_in2;
          when "0011" | "0111" =>
          -- RSB | RSC
            extra <= '1';
            carry := cin or (not func(2));
            res := ('0' & alu_in2) + ('1' & not in1) + carry;
            cf <= res(16);
            vf <= alu_in2(15) xor (in1(15) and res(15));
          when "1101" =>
          -- MOV
            extra <= '0';
            res(15 downto 0) := alu_in2;
          when "1110" =>
            -- BIC
            extra <= '0';
            res(15 downto 0) := in1 and (not alu_in2);
          when others =>
            -- MVN
            extra <= '0';
            res(15 downto 0) := not alu_in2;
      end case;
      alu_out <= res(15 downto 0);
      result  <= res(15 downto 0);
end process DO_ALU;
```

รูปที่ 4.9 ภาษาที่ใช้บรรยายการทำงานของหน่วยคำนวณและตรรกะส่วนประมวลผล



รูปที่ 4.10 ไดอะแกรมสถานะของหน่วยคำนวณและตรรกะส่วนประมวลผล

ตารางที่ 4.5 ผลการตรวจสอบวงจรส่วนประมวลผลในหน่วยคำนวณทางคณิตศาสตร์และตรรกะ

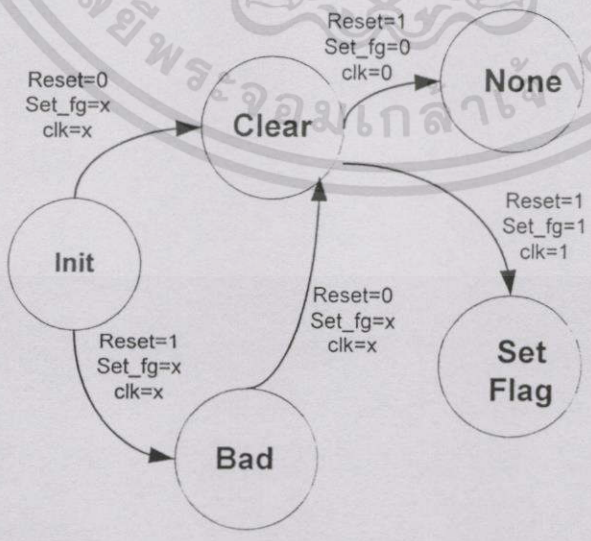
ชุดการ trace	สถานะ	ผลที่ได้
1. Func = 0000, 1000	AND, TST	Accept
2. Func = 0001, 1001	XOR, TEQ	Accept
3. Func = 0010, 0110, 1010	SUB, SBC, CMP	Accept
4. Func = 0100, 0101, 1011	ADD, ADC, CMN	Accept
5. Func = 1100	ORR	Accept
6. Func = 0011, 0111	RSB, RSC	Accept
7. Func = 1101	MOV	Accept
8. Func = 1110	BIC	Accept
9. Func = other	MVN	Accept

ALU_Setflag ใช้ในการตั้งค่า flag ของผลลัพธ์ที่ได้จากการประมวลผลในหน่วยคำนวณทางคณิตศาสตร์และตรรกะ

```
SET_FLAG:
        Process (reset,clk)
        begin
                if (reset = '0') then
                        neg     <= '0';
```

```
                carry    <= '0';
                cary     <= '0';
                over     <= '0';
                zero     <= '0';
        elsif rising_edge(clk) then
                if (set_fg = '1') then
                        neg<= result(15);
                        carry <= cflag;
                        cary  <= cflag;
                        over  <= vflag;
                        if result = ze then
                                zero <= '1';
                        else
                                zero <= '0';
                        end if;
                end if;
        end if;
end process SET_FLAG;
```

รูปที่ 4.11 ภาษาที่ใช้บรรยายการทำงานหน่วยคำนวณและตรรกะส่วนการตั้งค่า Flag



รูปที่ 4.12 ไดอะแกรมสถานะของหน่วยคำนวณและตรรกะส่วนตั้งค่า flag

ตารางที่ 4.6 ผลการตรวจสอบวงจรส่วนตั้งค่า flagในหน่วยคำนวณทางคณิตศาสตร์และตรรกะ

ชุดการ trace	สถานะ	ผลที่ได้
reset = 0, set_fg = 0 clk = 0 → 1 reset = 0, set_fg = 1 clk = 0 → 1 reset = 0, set_fg = 0 clk = 1 → 0 reset = 0, set_fg = 0 clk = 1 → 0	Clear	Accept
5. reset = 1 set_fg = 0 clk = 1 → 0	None	Accept
6. reset = 1 set_fg = 1 clk = 0 → 1	Set flag	Accept
init → (reset = 1 set_fg = 0 clk = 0 → 1) init → (reset = 1 set_fg = 1 clk = 0 → 1) init → (reset = 1 set_fg = 0 clk = 1 → 0) init → (reset = 1 set_fg = 1 clk = 1 → 0)	BAD	Accept

สรุปผลที่ได้

พบว่าจากการสร้างส่วนการทำงานของหน่วยคำนวณและตรรกะในส่วนขององค์ประกอบย่อยที่ออกแบบเองด้วย VHDL มีข้อผิดพลาดในส่วนที่ไม่ได้มีการตรวจสอบว่าเงื่อนไขค่าแฟลคก่อนที่จะเริ่มเข้าทำงาน ซึ่งเป็นส่วนของสายอักขระ b (Satisfies(Status,Condcode(Instr)) ที่ขาดหายไป ผลจากข้อผิดพลาด ณ. จุดนี้ไม่กระทบต่อการทำงานของหน่วยคำนวณและตรรกะ ทำให้การจำลองการทำงานไม่สามารถตรวจสอบข้อผิดพลาดจุดนี้ได้ และผลจากข้อผิดพลาดนี้ทำให้องค์ประกอบย่อยที่ออกแบบมีการทำงานตลอดเวลาจึงทำให้สิ้นเปลืองพลังงานที่ใช้ในการประมวลผลมากกว่าปกติ

นอกจากนี้บางองค์ประกอบของการทำงานที่มีการกำหนดคุณลักษณะที่เป็นภาษาที่แน่นอนแต่ในภาษาที่ได้บรรยายการทำงานไม่ได้สร้างให้ตรงตามคุณลักษณะที่กำหนดทุกประการ เนื่องจากติดปัญหาที่จำเป็นต้องมีการแยกเป็นองค์ประกอบย่อย รวมทั้งบางส่วนของการทำงานยังถูกฝังรวมอยู่กับองค์ประกอบย่อยอื่นๆ ในระบบทำให้ไม่สามารถที่จะนำเอาไดอะแกรมทั้งสองมาเปรียบเทียบกันได้ แต่อย่างไรก็ตามการแปลงข้อกำหนดคุณลักษณะที่มีให้เป็นไดอะแกรมที่เป็นแบบแผนจะช่วยให้ผู้ออกแบบสามารถเข้าใจการทำงานขององค์ประกอบย่อยนั้นๆ ดีขึ้นรวมทั้งสามารถทราบได้ถึงข้อบกพร่องที่อาจตกหล่นไปเนื่องจากไดอะแกรมสถานะสามารถให้รายละเอียดการทำงานได้ดีกว่า

4.1.3 การตรวจสอบการทำงานของหน่วยการเก็บข้อมูลจากหน่วยความจำลงสู่รีจิสเตอร์

Rule : Single Load

```
If SingleLoadInstr(ExecuteInstr)  then
   If ExecuteMode = first-step and Satisfies(Status, CondCode(executeInstr) then
         AddrReg :=  MemAddr
         ExecuteMode := load-read-memory
         Bop := offset
   elsif  ExecuteMode = load-read-memory then
         if ByteTransferInstr(ExecuteInstr)  then
               DataIn := PadWord(Memory(AddrReg))
         else  DataIn :=  MemoryWord(AddrReg)
         end if
         if WriteBack(ExecuteInstr)  then
               Contents(BaseOp(ExcuteInstr)) := ALU ("+",Aop,Bop,0)
         End if
         ExecuteMode := load-write-register
  elsif  ExcuteMode := load-write-register
         Contents(DesReg) := DataIn
         ExecuteMode := first-step
  End if
End if
```

Specification :

Rule a : SingleLoadInstr(ExecuteInstr)

Rule b : ExecuteMode = first-step and Satisfies(Status, CondCode(executeInstr))

Rule c : ExecuteMode = load-read-memory

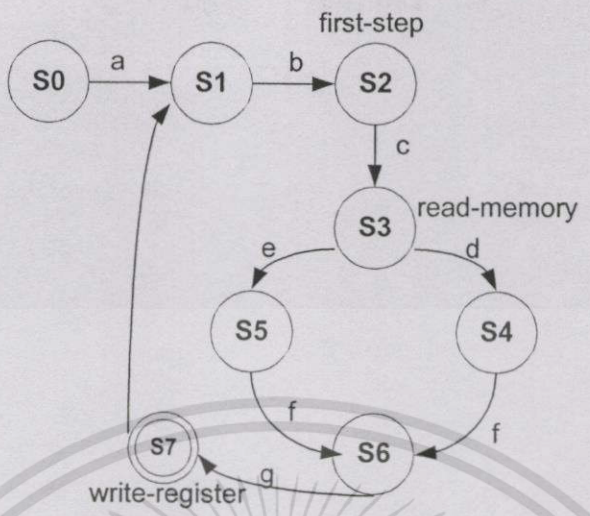
Rule d : ByteTransferInstr(ExecuteInstr)

Rule e : DataIn := MemoryWord(AddrReg)

Rule f : WriteBack(ExecuteInstr)

Rule g : ExcuteMode := load-write-register

รูปที่ 4.13 ภาษาที่ใช้ในการกำหนดคุณลักษณะของส่วนการอ่านข้อมูลจากรีจิสเตอร์



รูปที่ 4.14 ไดอะแกรมสถานะของคุณลักษณะที่ต้องการของหน่วยอ่านข้อมูลจากรีจิสเตอร์

ตารางที่ 4.7 ผลจากการแทนองค์ประกอบย่อยที่ออกแบบด้วย VHDL เข้าสู่ FSM ของหน่วยโหลด

สายอักขระ	การทำงาน	ผลที่ได้
abcdfg	SingleLoadInstr(ExecuteInstr) และ ExecuteMode=first-step and Satisfies(Status, CondCode(ExecuteInstr)) และ ExecuteMode=load-register-memory และ ByteTransferInstr(ExecuteInstr) และ WriteBack(ExecuteInstr) และ ExecuteMode=load-write-register	Accept
abcefg	SingleLoadInstr(ExecuteInstr) และ ExecuteMode=first-step and Satisfies(Status, CondCode(ExecuteInstr)) และ ExecuteMode=load-register-memory และ ByteTransferInstr(ExecuteInstr) และ DataIn := PadWord(Memory(AddrReg)) และ ExecuteMode=load-write-register	Accept

<u>สรุปผลที่ได้</u>

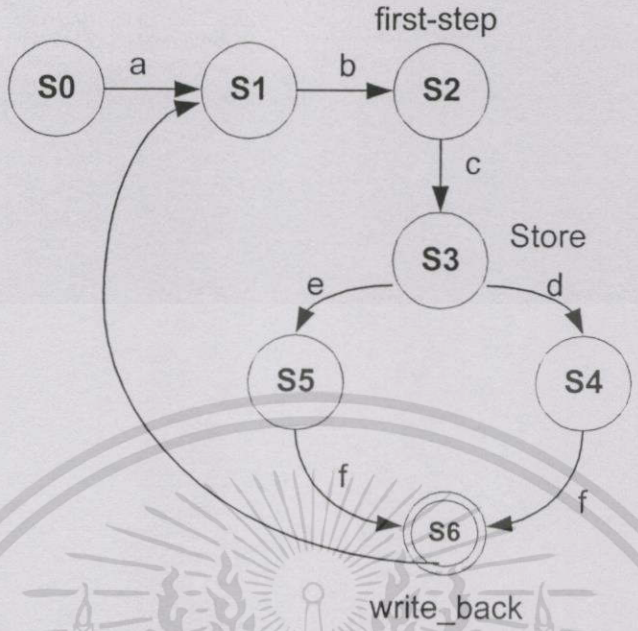
พบว่าองค์ประกอบย่อยที่ออกแบบด้วยภาษา VHDL ของหน่วยเก็บข้อมูลจากหน่วยความจำลงสู่รีจิสเตอร์ ไม่มีข้อผิดพลาด มีผลให้ Finite State Machine ที่เป็นคุณลักษณะที่กำหนด ยอมรับ

การทำงานที่ป้อนให้ผ่านทางสายอักขระ โดยการทดสอบในส่วนนี้การทำงานจะอยู่ในลักษณะที่เรียงตามลำดับขั้น หากเกิดข้อผิดพลาด การทำงานจะคงสถานะเดิมทันที

4.1.4 การตรวจสอบการทำงานของหน่วยการเก็บข้อมูลจากรีจิสเตอร์ลงสู่หน่วยความจำ

```
Rule : Single Store
If SingleInstr(ExecuteInstr) then
   If ExecuteMode = first-step and Satisfies(Status, CondCode(ExecuteInstr)) then
        MemAddr := AddrReg
        DataOut := Contents(DesReg)
        ExecuteMode := Store
        Bop := offset
  elsif ExecuteMode = store then
        if ByteTransferInstr(ExecuteInstr) then
             Memory(AddrReg) := DataOut
        else AssignWord(AddrReg,DataOut)
        endif
        if WriteBack(ExcuteInstr) then
             Contents(Baseop(ExecuteInstr)) := ALU ("+", AOP,BOP,0)
        end if
        ExecuteMode := first-step
   end if
 end if
Specification :
Rule a : SingleStoreInstr(ExecuteInstr)
Rule b : ExecuteMode = first-step and Satisfies(Status,CondCode(ExecuteInstr)))
Rule c : ExecuteMode = store
Rule d : ByteTransferInstr(ExecuteInstr)
Rule e : AssignWord(AddrReg,DataOut)
Rule f : WriteBack(ExecuteInstr)
```

รูปที่ 4.15 ภาษาที่ใช้กำหนดคุณลักษณะของการเก็บข้อมูลลงรีจิสเตอร์



รูปที่ 4.16 ไดอะแกรมของคุณลักษณะที่ต้องการของหน่วยเก็บข้อมูลลงหน่วยความจำ

ตารางที่ 4.8 ผลจากการแทนอิพุทขององค์ประกอบย่อยเข้าสู่ไดอะแกรมของหน่วยเก็บค่า

สายอักขระ	การทำงาน	ผลที่ได้
abcdf	SingleStoreInstr(ExecuteInstr) และ ExecuteMode = first-step and Satisfies(Status,CondCode(ExecuteInstr))) และ ExecuteMode = store และ ByteTransferInstr(ExecuteInstr) และ WriteBack(ExecuteInstr)	Accept
abcef	SingleStoreInstr(ExecuteInstr) และ ExecuteMode = first-step and Satisfies(Status,CondCode(ExecuteInstr))) และ ExecuteMode = store และ AssignWord(AddrReg,DataOut) WriteBack(ExecuteInstr)	Accept

<u>สรุปผลที่ได้</u>

พบว่าองค์ประกอบย่อยที่ออกแบบด้วยภาษา VHDL ของหน่วยเก็บข้อมูลจากรีจิสเตอร์ลงสู่หน่วยความจำ ไม่มีข้อผิดพลาด มีผลให้ Finite State Machine ที่เป็นคุณลักษณะที่กำหนด ยอมรับ

การทำงานที่ป้อนให้ผ่านทางสายอักขระ โดยการทดสอบในส่วนนี้การทำงานจะอยู่ในลักษณะที่เรียงตามลำดับขั้น หากเกิดข้อผิดพลาด การทำงานจะคงสถานะเดิมทันที

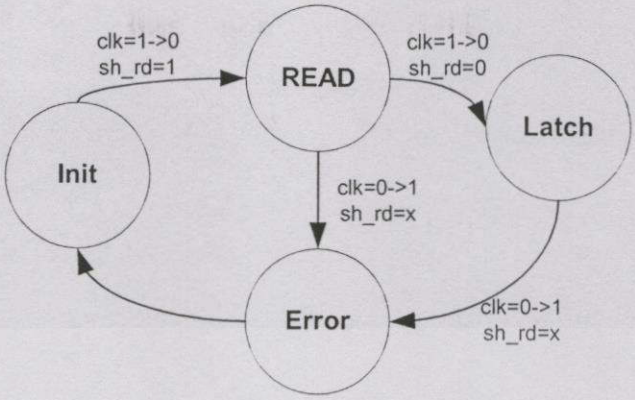
นอกจากนี้บางองค์ประกอบของการทำงานที่ไม่มีการกำหนดคุณลักษณะที่เป็นภาษาที่แน่นอนทำให้ไม่สามารถตรวจสอบการทำงานของประกอบย่อยส่วนนั้นกับคุณลักษณะที่เป็นแบบแผนได้ ซึ่งการออกแบบได้อาศัยข้อกำหนดที่อยู่ในรูปแบบของ Data Sheet และการทำงานที่เทียบได้จากโปรแกรม Debugger ของ ARM7 แต่อย่างไรก็ตามการแปลงภาษาที่ใช้ในการบรรยายการทำงานขององค์ประกอบที่ได้ออกแบบมาแล้วให้กลายเป็นไดอะแกรมสถานะนั้นก็ยังสามารถช่วยผู้ออกแบบสามารถเข้าใจการทำงานขององค์ประกอบย่อยนั้นๆ ดีขึ้นรวมทั้งสามารถทราบได้ถึงข้อบกพร่องที่อาจตกหล่นไปเนื่องจากไดอะแกรมสถานะสามารถให้รายละเอียดการทำงานได้ดีกว่า เช่น

4.1.5 การตรวจสอบการทำงานของชิฟเตอร์

Shift_read ส่วนของการรับข้อมูลเข้าสู่ชิฟเตอร์

```
INPUT_SHIFT:
        Process (rclk)
        begin
                if rising_edge(rclk) then
                        in2  <= sh_in2;
                        regs <= op_regs;
                        if (sh_rd = '1') then
                                in1 <= sh_in1;
                        end if;
                end if;
        end process INPUT_SHIFT
```

รูปที่ 4.17 ภาษาที่ใช้บรรยายการทำงานของหน่วยชิฟเตอร์ส่วนรับข้อมูลเข้า



รูปที่ 4.18 ไดอะแกรมสถานะของหน่วยชิฟเตอร์ส่วนรับข้อมูลเข้า

ตารางที่ 4.9 ผลการตรวจสอบวงจรส่วนรับข้อมูลเข้าสู่ชิฟเตอร์

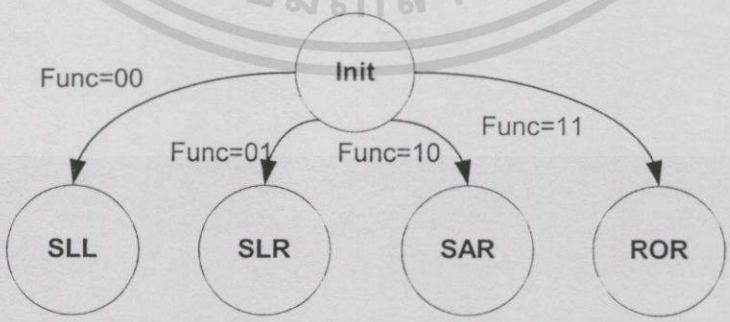
ชุดการ trace	สถานะ	ผลที่ได้
1. clk = 1 → 0, sh_rd = 1	Read	Accept
2. clk = 1 → 0, sh_rd = 0	Latch	Accept
3. clk = 0 → 1, sh_rd = 1	Error	Accept
4. clk = 0 → 1, sh_rd = 0	Error	Accept

Shift_do เป็นวงจรส่วนของการประมวลผลในชิฟเตอร์

```
SHIFT_PROC:
        process(idx,dshift,func,ovf)
        begin
                for i in 0 to 31 loop
                        case func is
                                when "00" => -- Logical shift left
                                if ovf = '0' then
                                        result(i) <= dshift(32-idx+i);
                                else
                                        result(i) <= '0';
                                end if;
                                when "01" => -- Logical shift right
```

```
                if ovf = '0' then
                        result(i) <= dshift(32+idx+i);
                else
                        result(i) <= '0';
                end if;
            when "10" => -- Arithmetic shift right
                if ovf = '1' then
                        result(i) <= dshift(63);
                else
                        if i < 32-idx then
                                result(i) <= dshift(32+idx+i);
                        else
                                result(i) <= dshift(63);
                        end if;
                end if;
            when others =>
                result(i) <= dshift(32+idx+i);
        end case;
    end loop;
end process SHIFT_PROC;
```

รูปที่ 4.19 ภาษาที่ใช้บรรยายการทำงานของหน่วยชิฟเตอร์ส่วนประมวลผล



รูปที่ 4.20 ไดอะแกรมสถานะของคุณลักษณะของส่วนประมูลผลข้อมูลในชิฟเตอร์

ตารางที่ 4.10 ผลการตรวจสอบวงจรส่วนประมวลผลข้อมูลในชิฟเตอร์

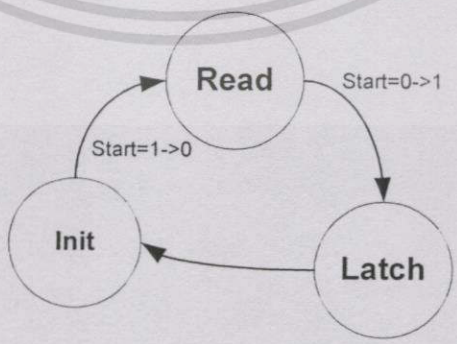
ชุดการ trace	สถานะ	ผลที่ได้
1. Func = 00	Sll	Accept
2. Func = 01	Slr	Accept
3. Func = 10	Sar	Accept
4. Func = 11	Ror	Accept

4.1.6 การตรวจสอบการทำงานของหน่วยการคูณ

Multiply_input ส่วนรับข้อมูลเข้าหน่วยการคูณ

```
DO_OUTPUT:
        Process (fin,rclk)
        begin
            if rising_edge(rclk) then
                if fin = '1' then
                    mul_out <= std_logic_vector(shr(unsigned(result),cnt)) or
                    std_logic_vector(shl(unsigned(t),"10000"-cnt));
                end if;
            end if;
        end process DO_OUTPUT;
```

รูปที่ 4.21 ภาษาที่ใช้บรรยายการทำงานหน่วยการคูณส่วนรับข้อมูลเข้า



รูปที่ 4.22 ไดอะแกรมสถานะของส่วนรับข้อมูลเข้าสู่หน่วยการคูณ

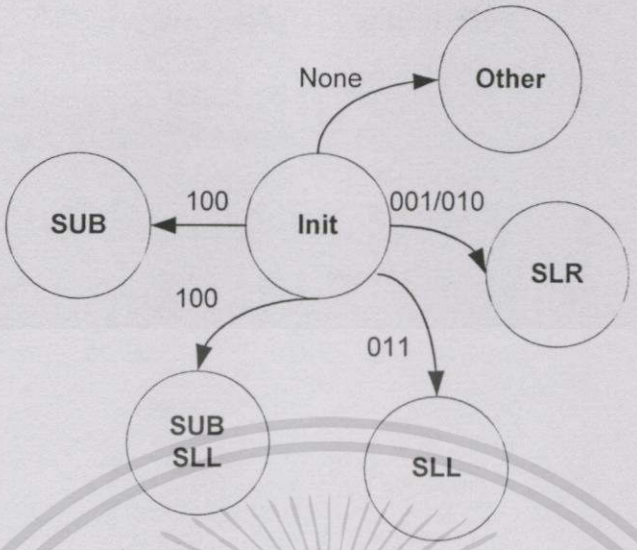
ตารางที่ 4.11 ผลการตรวจสอบสอบวงจรส่วนรับข้อมูลเข้าสู่หน่วยการคูณ

ชุดการ trace	สถานะ	ผลที่ได้
1. start = 1 → 0	Read	Accept
2. start = 0 → 1	Latch	Accept

Mul_do หน่วยประมวลผลการคูณ

```
DO_MUL:
        Process (rclk)
                variable in1 : std_logic_vector(31 downto 0);
        begin
                if rising_edge(rclk) then
                        temp <= t;
                        if st = '1' then
                                in1     := mul_in1;
                        end if;
                        case inp(2 downto 0) is
                                when "001" | "010" =>
                                        temp1 <= '0' & in1;
                                when "011" =>
                                        temp1 <= in1 & '0';
                                when "100" =>
                                        temp1 <= (not(in1) + "1" ) & '0';
                                when "101" | "110" =>
                                        temp1 <= ('1' & not in1) + "1";
                                when others =>
                                        temp1 <= (others => '0');
                        end case;
                        inp_buf <= "11" & inp(32 downto 2);
                end if;
        end process DO_MUL;
```

รูปที่ 4.23 ภาษาที่ใช้บรรยายการทำงานหน่วยการคูณส่วนประมวลผล



รูปที่ 4.24 ไดอะแกรมสถานะของส่วนประมวลผลหน่วยการคูณ

ตารางที่ 4.12 ผลการตรวจสอบวงจรส่วนประมวลผลหน่วยการคูณ

ชุดการ trace	สถานะ	ผลที่ได้
1. input2(2 down to 0) = other	None	Accept
2. input2(2 down to 0) = 001/010	SLR	Accept
3. input2(2 down to 0) = 011	SLL	Accept
4. input2(2 down to 0) = 100	SUB/SLL	Accept
5. input2(2 down to 0) = 101/110	SUB	Accept

4.2 การจำลองการทำงานของไมโครโพรเซสเซอร์ ARM 7

การสร้างชุดทดสอบเพื่อการตรวจสอบการทำงานของไมโครโพรเซสเซอร์นั้น จำเป็นต้องสร้างชุดทดสอบที่เหมาะสมกับการทำงานและโครงสร้างของไมโครโพรเซสเซอร์นั้นๆ เนื่องจากการวิเคราะห์โครงสร้างและการทำงานทำให้สามารถลดจำนวนอินพุทที่ใช้ตรวจสอบการทำงานได้ ยกตัวอย่างเช่นที่ M. Kentrowitz, L.M. Noack (1996) ได้นำเสนอผลงานวิจัยที่ชื่อ "Verification Coverage Analysis and Correctness Checking of the DECChip 21164 Alpha microprocessor." ซึ่งเป็นการนำเสนอแนวการเลือกชุดทดสอบเพื่อจำลองการทำงานของไมโครโปรเซสเซอร์ในตระกูล Alpha รุ่น 21164 ซึ่งเป็นไมโครโพรเซสเซอร์ที่มีขนาดใหญ่ และซับซ้อนไม่เหมาะกับการทวนสอบโดยการพิสูจน์ด้วยวิธีการทางคณิตศาสตร์ โดยการหาตัวแทนของกลุ่มคำสั่งเพื่อเป็นอินพุทให้กับระบบในการจำลองการทำงานและตรวจผลลัพธ์ที่ได้ ดังนั้นในงานวิจัยนี้จึงได้ใช้ลักษณะ

เดียวกันในการสร้างชุดทดสอบเพื่อตรวจสอบการทำงาน โดยการตรวจสอบจะสนใจเฉพาะการทำงานที่ทำให้ไมโครโพรเซสเซอร์ทำงานถูกต้องเท่านั้น ไม่รวมถึงการทำงานที่ผิดพลาดเนื่องจากคำสั่งที่ไม่ถูกต้องหรือไม่มีใช้งานในไมโครโพรเซสเซอร์ ARM7

การสร้างชุดทดสอบเพื่อการตรวจสอบการทำงานของไมโครโพรเซสเซอร์ ARM7 จะทำการคัดเลือกคำสั่งโดยการวิเคราะห์พฤติกรรมการส่งผ่านข้อมูลระหว่างองค์ประกอบย่อยในไมโครโพรเซสเซอร์ในระดับของคำสั่งการทำงานรวมทั้งโครงสร้างที่สำคัญขององค์ประกอบย่อยต่างๆ ที่พึงมีในไมโครโพรเซสเซอร์ทั่วๆ ไป โดยชุดทดสอบที่สร้างขึ้นมาทั้งหมดมี 2,660 คำสั่งจากคุณลักษณะที่กำหนดไว้ก่อนการออกแบบ โดยแบ่งออกเป็น 2 ชุดทดสอบดังนี้

4.2.1 ชุดทดสอบแรก

เป็นชุดทดสอบที่จัดแบ่งตามคำสั่งของ ARM 7 ทั้งหมด 26 คำสั่งดังตารางที่ 4.13 เนื่องจากคำสั่งของ ARM 7 สามารถกำหนดให้ไม่มีการคำนวณค่าตัวบ่งชี้ (flag) ดังนั้นชุดทดสอบนี้จึงไม่คำนึงผลของตัวบ่งชี้ที่เกิดขึ้นจากการทำงาน สามารถแบ่ง 26 คำสั่งการทำงานออกได้เป็น 16 คำสั่งใน หน่วยคำนวณและตรรกะ โดยแต่ละคำสั่งจะแบ่งออกได้ 4 กรณีที่แบ่งตามการใช้ตัวถูกดำเนินการดังนี้ มาจากรีจิสเตอร์ทั้งคู่, มาจากรีจิสเตอร์และค่าจำนวนเต็ม, ตัวถูกดำเนินการมาจากรีจิสเตอร์และอีกค่าทำการชิฟซึ่งได้ค่ามาจากรีจิสเตอร์, ตัวถูกดำเนินการ มาจากรีจิสเตอร์แต่อีกค่าที่สามารถทำการชิฟได้มาจากค่าจำนวนเต็ม ตัวอย่างเช่นการทดสอบการทำงานของคำสั่ง AND จะสามารถใช้ชุดทดสอบ 4 ชุดแทนการทำงานที่เกิดขึ้นทั้งหมดของ AND ได้ดังตารางที่ 4.14 การคำนึงถึงชนิดการใช้ตัวถูกดำเนินการเนื่องจากการเรียกใช้ตัวถูกดำเนินการที่แตกต่างกันจะมีการไหลของข้อมูลต่างกัน ซึ่งสามารถมีได้ถึง 4 แบบ ดังนั้นการเลือกชุดทดสอบที่ให้สามารถผ่านไปตามเส้นทางการไหลของข้อมูลและสามารถครอบคลุมการเรียกใช้สัญญาณควบคุมทั้งหมดอีกด้วย

คำสั่งการกระโดดข้ามการทำงาน (Branch) มีชุดทดสอบทั้งในกรณีที่มีการคืนค่ากลับตำแหน่งเดิมได้ (BL) และไม่สามารถกลับไปตำแหน่งเดิมได้ (B) โดดทั้ง 2 แบบจะมีชุดทดสอบการทำงานของไปป์ไลน์ซึ่งจะเกิดกรณีของการ Stalling ไปป์ไลน์ และค่าของ PC ให้มีการทำงานที่ถูกต้อง ดังนั้นเมื่อใช้ชุดทดสอบการทำงานที่มีคำสั่ง B จะทำให้สามารถครอบคลุมการทดสอบทั้งในส่วนของการอ้างอิงหน่วยความจำ, การทำงานไปป์ไลน์ และการทำงานของชุดคำสั่งทั้งหมด

ชุดทดสอบมีคำสั่งของการทำงานในแบบที่เป็นการคูณตามปกติ (MUL) และการคูณที่สามารถมีการคำนวณการบวกได้ (MLA) คำสั่งของการย้ายข้อมูลระหว่างหน่วยความกับรีจิสเตอร์มีชุดทดสอบที่คำนึงข้อมูลที่มีการแลกเปลี่ยน รวมทั้งค่า PC ที่เปลี่ยนไป คำสั่งการกระโดดข้ามการทำงาน การคูณและการย้ายข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์มีชุดทดสอบในการอ้างอิงหน่วยความจำทั้ง 3 แบบ Indirect, Index และ Base Index ดังตารางที่ 4.15

4.2.2 ชุดทดสอบที่สอง

เป็นชุดทดสอบที่คำนึงถึงการเปลี่ยนแปลงค่าตัวบ่งชี้ (N : Negative, Z : Zero, C : Carry, V : Over flow) ในการทำงานของหน่วยคำนวณและตรรกะ ดังนั้นชุดทดสอบจะคำนึงถึงกรณีที่สามารถทำให้ค่าตัวบ่งชี้เกิดการเปลี่ยนแปลงดังนี้ ค่า N จะมีการเปลี่ยนแปลงเมื่อบิตสุดท้ายของผลลัพธ์เป็น 1, Z จะเปลี่ยนแปลงเป็น 1 เมื่อผลลัพธ์ที่ได้มีค่าเป็นศูนย์ทั้ง 32 บิต, C จะเปลี่ยนแปลงเมื่อมีการบวก ลบหรือชิฟข้อมูลแล้วผลลัพธ์ที่ได้เกินขนาด 32 บิต, V จะเปลี่ยนแปลงเป็น 1 เมื่อมีการบวก ลบหรือการเปรียบเทียบแบบคิดเครื่องหมายแล้วผลลัพธ์เกินจาก 31 บิต

ตารางที่ 4.13 ชุดคำสั่งทดสอบ 26 ชุด

Instruction	Source 1	Source 2	Shift	Execute	Write-back
B	-	-	-	-	#Jump>Pc
BL	-	-	-	-	#Jump>Pc
AND	W	W	OP2	SHIFT,AND	W
EOR	W	W	OP2	SHIFT,XOR	W
SUB	W	W	OP2	SHIFT,SUB	W
RSB	W	W	OP2	SHIFT,RSB	W
ADD	W	W	OP2	SHIFT,ADD	W
ADC	W	W	OP2	SHIFT,ADC	W
SBC	W	W	OP2	SHIFT,SBC	W
RSC	W	W	OP2	SHIFT,RSC	W
TST	-	-	OP2	SHIFT,TST	-
TEQ	-	-	OP2	SHIFT,TEQ	-
CMP	-	-	OP2	SHIFT,CMP	-
CMN	-	-	OP2	SHIFT,CMN	-
ORR	W	W	OP2	SHIFT,ORR	W
MOV	-	W	OP2	SHIFT,MOV	W
BIC	W	W	OP2	SHIFT,OP2 AND NOT OP1	W
MVN	W	W	OP2	SHIFT,NOT OP2	W
MUL	W	W	-	MUL	W
MLA	W	W	-	MUL,ADD	W
LDR	W,B	W,B	OP2	SHIFT,LOAD	W,B
STR	W,B	W,B	OP2	SHIFT,STORE	W,B
LDM	W,B	W,B	OP2	Multiple Load	W,B
STM	W,B	W,B	OP2	Multiple Store	W,B
SWP	W,B	W,B	OP2	LOAD,STORE	W,B

ตารางที่ 4.14 แสดงชุดคำสั่งที่ใช้ในการทดสอบส่วน Data processing

Instruction Set	Instruction	Feature	Description
Data Processing without shift operator	AND	1. register and register 2. register and immediate value	Op1/reg AND op2/reg Op1/reg AND op2/immediate
	EOR	1. register and register 2. register and immediate value	Op1/reg EOR op2/reg Op1/reg EOR op2/immediate value
	SUB	1. register and register 2. register and immediate value 3. op1 little than op2	Op1/reg - op2/reg Op1/reg - op2/immediate Op1 - op2 but op1 little than op2
	RSB	1. register and register 2. register and immediate value 3. op1 larger than op2	Op2/reg – op1/reg Op2/reg – op1/immediate Op2 – op1 but op1 larger than op2
	ADD	register and register register and immediate value overflow add	Op1/reg + op2/reg Op1/reg + op2/reg Op1 + op2 but overflow
	ADC	register and register register and immediate value carry = 1	Op1/reg + op2/reg Op1/reg + op2/immediate value Op1 + op2 + carry
data Processing With shift operand	SBC	register and register add carry register and immediate value add carry op1 lower than op2	Op1/reg – op2/reg + C Op1/reg – op2/immediate + C Op1 – op2 + C but op1 lower than op2
	RSC	register and register register and immediate value op2 lower than op1	Op2/reg – op1/reg + Op2/reg – op1/immediate + C Op2 – op1 + C but op2 lower than op1
	TST	1. register 2. immediate value	Set condition code on op1 AND op2

ตารางที่ 4.14 (ต่อ)

Instruction Set	Instruction	Feature	Description
Data Processing	TEQ	1. register 2. immediate value	Set condition code op1 EOR op2
	CMP	1. register 2. immediate value	Set condition code op1 – op2
	CMN	1. register 2. immediate value	Set condition code op1 + op2
	ORR	1. register and register 2. register and immediate value	Op1/reg OR op2/reg Op1/reg OR op2/immediate
	MOV	1. register to register 2. immediate to register	Destination = op2/reg Destination = op2/immediate
	BIC	1. register and register 2. register and immediate	Op1/reg AND (not op2/reg) Op1/reg AND (not op2/immediate)
	MVN	1. register 2. immediate value	Not op2/reg Not op2/immediate
Branch	B		Jump no back track
	BL		Jump with return
Multiply	MUL	1. register and register 2. overflow	Multiply
	MLA	1. register and register 2. overflow	Multiply and accumulator R1= R2*R3+R4
Single Data Transfer	LDR	1. indirect addressing mode 2. index addressing mode 3. base index addressing mode 4. byte and word	Load Single Data from Memory to Register

ตารางที่ 4.14 (ต่อ)

Instruction Set	Instruction	Feature	Description
Single Data Transfer	STR	1. indirect addressing mode 2. index addressing mode 3. base index addressing mode 4. byte and word	Store Single Data from Register to Memory
Block Data Transfer	LDM	1. pre-increment load 2. post-increment load 3. post-decrement load 4. pre-decrement load	Load multiple Data from Memory to Register
	STM	1. pre-increment store 2. post-increment store 3. post-decrement store 4. pre-decrement store	Store multiple Data from Register to Memory
Data Swap	SWP	Byte and word	Swap data between Register & Memory

Access All 16 Register , R0-R13, R14(Stack), R15(PC)

ลักษณะของการป้อนคำสั่งเมื่อจำลองการทำงานจะเป็นแบบต่อเนื่องกันไปทั้งหมด ซึ่งมีการผสมโดยไม่ได้เรียงลำดับ หรือจัดหมวดหมู่ของแต่ละกลุ่มคำสั่ง ดังนั้นการทำงานจะเหมือนจริงและผสมกันไประหว่างแต่ละกลุ่มคำสั่งเพื่อตรวจสอบความต่อเนื่องในการทำงานของไมโครโปรเซสเซอร์ด้วย ข้อมูลที่ทำการสังเกตและตรวจสอบอย่างละเอียดคือค่าของข้อมูล ค่า PC ที่เก็บตำแหน่งเลขที่อยู่ ค่าของสัญญาณ รวมถึงคาบของเวลาในการใช้งาน ทั้งนี้การทำงานของไมโครโพรเซสเซอร์ ARM 7 เป็นแบบมีไปป์ไลน์ ดังนั้นการป้อนชุดคำสั่งอย่างต่อเนื่องจะทำให้สามารถตรวจสอบการทำงานของไปป์ไลน์ได้อีกทางหนึ่ง อย่างไรก็ตามการตรวจสอบโดยจำลองการทำงานนี้ถึงแม้จะทำการตรวจสอบโดยใช้ชุดทดสอบที่เป็นตัวแทนของระบบซึ่งมาจากการวิเคราะห์โครงสร้างและการทำงานก็ตาม การตรวจสอบยังอยู่เป็นสมมุติฐานที่ว่าจะทำการตรวจสอบเฉพาะกรณีของคำสั่งที่อยู่ User mode เท่านั้น ไม่รวมถึงคำสั่งในโหมดการทำงานอื่นๆ หรือคำสั่งที่ไม่ถูกต้องในการทำงานด้วย

4.2.3 ชุดทดสอบไมโครโพรเซสเซอร์ ARM และผลการจำลองการทำงาน

ตารางที่ 4.15 ผลการทดสอบด้วยชุดคำสั่งในกรณีต่างๆ

Instruction	Function	OPCODE	PSR (Flag)				Access	Result	B /	Number	Addressing Mode			
			N	V	Z	C	Register		W	Cycle	Direct	Immed.	Index	Base Index
AND (S=0)	Op1/Register AND Op2/Register	E0SXX00X	/	/	/	/	/	/	/	1, 3	/			
ANDS (S=1)	Op1/Register AND Op2/Immediate value	E2SXX0XX	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E0SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E0SXXXOX	/	/	/	/	/	/	/	2, 4	/			
EOR (S=2)	Op1/Register AND Op2/Register	E0SXX00X	/	/	/	/	/	/	/	1, 3	/			
EORS (S=3)	Op1/Register AND Op2/Immediate value	E2SXX0XX	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E0SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E0SXXXOX	/	/	/	/	/	/	/	2, 4	/			
SUB (S=4)	Op1/Register AND Op2/Register	E0SXX00X	/	/	/	/	/	/	/	1, 3	/			
SUBS (S=5)	Op1/Register AND Op2/Immediate value	E2SXX0XX	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E0SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E0SXXXOX	/	/	/	/	/	/	/	2, 4	/			
RSB (S=6)	Op1/Register AND Op2/Register	E0SXX00X	/	/	/	/	/	/	/	1, 3	/			
RSBS (S=7)	Op1/Register AND Op2/Immediate value	E2SXX0XX	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E0SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E0SXXXOX	/	/	/	/	/	/	/	2, 4	/			

ตารางที่ 4. 15 (ต่อ)

Instruction	Function	OPCODE	PSR (Flag)				Access	Result	B /	Number	Addressing Mode			
			N	V	Z	C	Register		W	Cycle	Direct	Immed.	Index	Base Index
ADD (S=8)	Op1/Register AND Op2/Register	E0SXX00X	/	/	/	/	/	/	/	1, 3	/			
ADDS (S=9)	Op1/Register AND Op2/Immediate value	E2SXX0XX	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E0SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E0SXXXOX	/	/	/	/	/	/	/	2, 4	/			
ADC (S=A)	Op1/Register AND Op2/Register	E0SXX00X	/	/	/	/	/	/	/	1, 3	/			
ADCS (S=B)	Op1/Register AND Op2/Immediate value	E2SXX0XX	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E0SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E0SXXXOX	/	/	/	/	/	/	/	2, 4	/			
SBC (S=C)	Op1/Register AND Op2/Register	E0SXX00X	/	/	/	/	/	/	/	1, 3	/			
SBCS (S=D)	Op1/Register AND Op2/Immediate value	E2SXX0XX	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E0SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E0SXXXOX	/	/	/	/	/	/	/	2, 4	/			
RSC (S=E)	Op1/Register AND Op2/Register	E0SXX00X	/	/	/	/	/	/	/	1, 3	/			
RSCS (S=F)	Op1/Register AND Op2/Immediate value	E2SXX0XX	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E0SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E0SXXXOX	/	/	/	/	/	/	/	2, 4	/			

ตารางที่ 4.15 (ต่อ)

Instruction	Function	OPCODE	PSR (Flag)				Access	Result	B /	Number	Addressing Mode			
			N	V	Z	C	Register		W	Cycle	Direct	Immed.	Index	Base Index
TST (S=1)	Op1/Register AND Op2/Register	E1SXX00X	/	/	/	/	/	/	/	1, 3	/			
V = Even	Op1/Register AND Op2/Immediate value	E3SXX00X	/	/	/	/	/	/	/	1, 3	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift immediate value	E1SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
	Op1/Register AND Op2/Register with Shift Register(Rs)	E1SXXXOX	/	/	/	/	/	/	/	2, 4	/			
TEQ (S=3)	Op1/Register AND Op2/Register	E1SXX00X	/	/	/	/	/	/	/	1, 3	/			
V = Even	Op1/Register AND Op2/Immediate value	E3SXX00X	/	/	/	/	/	/	/	1, 3	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift immediate value	E1SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
	Op1/Register AND Op2/Register with Shift Register(Rs)	E1SXXXOX	/	/	/	/	/	/	/	2, 4	/			
CMP (S=5)	Op1/Register AND Op2/Register	E1SXX00X	/	/	/	/	/	/	/	1, 3	/			
V = Even	Op1/Register AND Op2/Immediate value	E3SXX00X	/	/	/	/	/	/	/	1, 3	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift immediate value	E1SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
	Op1/Register AND Op2/Register with Shift Register(Rs)	E1SXXXOX	/	/	/	/	/	/	/	2, 4	/			
CMN (S=7)	Op1/Register AND Op2/Register	E1SXX00X	/	/	/	/	/	/	/	1, 3	/			
V = Even	Op1/Register AND Op2/Immediate value	E3SXX00X	/	/	/	/	/	/	/	1, 3	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift immediate value	E1SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
	Op1/Register AND Op2/Register with Shift Register(Rs)	E1SXXXOX	/	/	/	/	/	/	/	2, 4	/			

ตารางที่ 4.15 (ต่อ)

Instruction	Function	OPCODE	PSR (Flag)				Access	Result	B /	Number	Addressing Mode			
			N	V	Z	C	Register		W	Cycle	Direct	Immed.	Index	Base Index
ORR (S=8)	Op1/Register AND Op2/Register	E1SXX00X	/	/	/	/	/	/	/	1, 3	/			
ORRS (S=9)	Op1/Register AND Op2/Immediate value	E3SXX00X	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E1SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E1SXXXOX	/	/	/	/	/	/	/	2, 4	/			
MOV (S=A)	Op1/Register AND Op2/Register	E1SXX00X	/	/	/	/	/	/	/	1, 3	/			
MOVS (S=B)	Op1/Register AND Op2/Immediate value	E3SXX00X	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E1SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E1SXXXOX	/	/	/	/	/	/	/	2, 4	/			
BIC (S=C)	Op1/Register AND Op2/Register	E1SXX00X	/	/	/	/	/	/	/	1, 3	/			
BICS (S=D)	Op1/Register AND Op2/Immediate value	E3SXX00X	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E1SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E1SXXXOX	/	/	/	/	/	/	/	2, 4	/			
MVN (S=E)	Op1/Register AND Op2/Register	E1SXX00X	/	/	/	/	/	/	/	1, 3	/			
MVN (S=F)	Op1/Register AND Op2/Immediate value	E3SXX00X	/	/	/	/	/	/	/	1, 3	/	/		
V = Even	Op1/Register AND Op2/Register with Shift immediate value	E1SXXXVX	/	/	/	/	/	/	/	2, 4	/	/		
O = Odd	Op1/Register AND Op2/Register with Shift Register(Rs)	E1SXXXOX	/	/	/	/	/	/	/	2, 4	/			

ตารางที่ 4. 15 (ต่อ)

Instruction	Function	OPCODE	PSR (Flag)				Access	Result	B /	Number	Addressing Mode			
			N	V	Z	C	Register		W	Cycle	Direct	Immed.	Index	Base Index
B	Branch not have Condition	XAXXXXXX	-	-	-	-	/	/	/	3			/	
BL	Branch with Link (could return)	XBXXXXXX	-	-	-	-	/	/	/	3			/	
MUL	Multiply (Rd = Rs*Rm)	E0SX0X9X	/	/	/	/	/	/	/	>2	/		/	/
MLA	Multiply (Rd = Rs*Rm + Rn)	E0SXXX9X	/	/	/	/	/	/	/	>2	/		/	/
LDR	Add/Sub Offset Before/After From Register/Immediate Value with/without Shift and Write/not write Back		-	-	-	-	/	/	/	3, 5	/		/	/
STR	Add/Sub Offset Before/After From Register/Immediate Value with/without Shift and Write/not write Back		-	-	-	-	/	/	/	2	/		/	/
LDM	Add/Sub Offset Before/After with Write/not write Back		-	-	-	-	/	/	/	3, 5, >5	/		/	/
STM	Add/Sub Offset Before/After with Write/not write Back		-	-	-	-	/	/	/	2, >2	/		/	/
SWP	Swap Data between Register and Memory	E10XX09X	-	-	-	-	/	/	/	4	/		/	/

การวิเคราะห์ชุดทดสอบในกลุ่มของ Data Processing

จากตาราง 4.15 เป็นการสรุปชุดทดสอบที่สร้างขึ้นโดยแสดงให้เห็นว่าในแต่ละคำสั่งสามารถทดสอบส่วนใดได้บ้าง ตารางที่ 4.15 เป็นส่วนของชุดทดสอบคำสั่งในกลุ่มของ Data Processing โดยคำสั่งในกลุ่มนี้จะขึ้นจะมีค่ารหัสดำเนินการ (Op code) โดยให้ไบต์บนสุดเป็นไบต์ที่ 8 และล่างสุดเป็นไบต์ที่ 1 ตาม รายละเอียดดังนี้

ไบต์ที่ 8 : จะมีค่าเป็น E เสมอเนื่องจากเป็นส่วนของบิตเงื่อนไขถ้าต้องการให้มีการทำคำสั่งนั้นจะต้องให้ค่าเป็น E

ไบต์ที่ 7 : สามารถแบ่งออกเป็น 2 กลุ่มได้คือ

- กลุ่มของคำสั่งทางคณิตศาสตร์จะมีค่าเป็น 0 หรือ 2 เท่านั้นโดยที่จะเป็นค่า 2 ก็ต่อเมื่อมีการเรียกใช้ตัวถูกดำเนินการเป็นค่าจำนวนเต็มเท่านั้น
- กลุ่มของคำสั่งทางลอจิก จะมีค่าเป็น 1 หรือ 3 เท่านั้น โดยที่จะมีค่าเป็น 3 ก็ต่อเมื่อมีการเรียกใช้ตัวถูกดำเนินการเป็นค่าจำนวนเต็มเท่านั้น

ไบต์ที่ 6 : เป็นค่าบอกถึงการคิดค่าตัวบ่งชี้หรือไม่ เนื่องจากไมโครโปรเซสเซอร์ ARM 7 สามารถให้ทำงานทั้งแบบที่คิดและไม่คิดค่าตัวบ่งชี้ ในที่นี้แทนด้วยตัวแปร S

ไบต์ที่ 4,5 : เป็นค่าใดก็ได้ทั้งนี้คือส่วนของการอ้างถึงรีจิสเตอร์หรือเป็นค่าจำนวนเต็ม

ไบต์ที่ 3 : เป็นการบอกว่ามีการชิฟ ในคำสั่งทางคณิตศาสตร์และลอจิกด้วยหรือไม่ โดยที่ค่า 0 หมายถึงไม่มีการชิฟ นอกจากนี้จะมีการชิฟซึ่งจะเป็นจำนวนที่ต้องการชิฟ

ไบต์ที่ 2 เป็นตัวบอกความแตกต่างของรูปแบบของแต่ละฟังก์ชั่นการทำงาน โดยแบ่งเป็น 2 กลุ่ม คือ

- กลุ่มที่ไม่มีการชิฟ (ดูจากไบต์ที่ 3) ถ้าค่าในไบต์ที่ 2 เป็น 0 หมายถึงการใช้รีจิสเตอร์เป็นทั้งตัวดำเนินการและตัวถูกดำเนินการ นอกจากนั้นเป็นการใช้ตัวถูกดำเนินการเป็นจำนวนเต็ม
- กลุ่มที่มีการชิฟ (ดูจากไบต์ที่ 3) ถ้าไบต์นี้มีค่าเป็นเลขคู่ (V = Even) หมายถึงการชิฟ โดยมีจำนวนครั้งจากจำนวนเต็ม แต่ถ้าเป็นเลขคี่ (O = Odd) หมายถึงการชิฟโดยใช้จำนวนครั้งในการชิฟจากรีจิสเตอร์ไบต์ล่าง

ไบต์ที่ 1 : เป็นค่าใดก็ได้

จากข้างต้นเป็นรูปแบบของการทำงานในกลุ่มของ Data Processing ที่สามารถเกิดขึ้นได้ทั้งหมด โดยการออกแบบส่วนของการถอดรหัสจะใช้ลักษณะเช่นนี้ในการแบ่งกลุ่มโดยจะเริ่มตรวจค่าจากไบต์บนลงสู่ไบต์ล่าง ในตารางที่ 4.15 จะแสดงส่วนต่างที่ได้ครอบคลุมการทดสอบซึ่งได้จากการสร้างชุดทดสอบตามลักษณะการแบ่งค่ารหัสดำเนินการเช่นนี้ ค่าที่สามารถทดสอบได้แก่ ค่าตัว

บ่งชี้ การทดสอบในส่วนของการเรียกใช้รีจิสเตอร์ การตรวจสอบผลของการทำงานในส่วนของหน่วยคำนวณและตรรกะ ชิฟเตอร์ การรับและส่งข้อมูลออกสู่บัสของข้อมูล การทำงานในลักษณะของไบต์และ เวิร์ด ตรวจสอบจำนวนของคาบที่ใช้ในการทำงาน และการอ้างอิงเลขที่อยู่โดยสามารถอ้างได้ 2 แบบคือส่วนของ Direct และ immediate เราสามารถสรุปส่วนต่างๆและสิ่งที่ได้ทำการตรวจสอบโดยการใช้ชุดทดสอบของกลุ่ม Data Processing ดังตารางที่ 4.16

ตารางที่ 4.16 ส่วนต่างๆที่ได้ทำการตรวจสอบด้วยชุดคำสั่ง Data Processing

หน่วยที่มีการเรียกใช้	สิ่งที่ได้ทำการตรวจสอบ	หมายเหตุ
หน่วยรีจิสเตอร์	การอ่าน / เขียน	ทุกคำสั่งที่มีการใช้ตัวถูกดำเนินการจาก รีจิสเตอร์และเก็บผลลัพธ์สู่รีจิสเตอร์
	สัญญาณอ่าน /เขียน	
	การเรียกใช้รีจิสเตอร์	มีการเรียกใช้รีจิสเตอร์ทุกตัวในชุดทดสอบกลุ่มของ Data Processing
บัสข้อมูลขนาด 32 บิต	การอ่านค่าจากบัส	
	การเขียนค่าลงสู่บัส	
หน่วยของการคูณ	การผ่านค่า	เนื่องจากไม่มีคำสั่งการคูณ
ชิฟเตอร์	การชิฟ ทั้ง 4 แบบ	
	การผ่านค่า	ในกรณีที่ทำคำสั่งที่ไม่มีการชิฟ
หน่วยควบคุม	สัญญาณที่ควบคุมแต่ละหน่วย	
	ข้อมูลที่เป็นจำนวนเต็ม	ในกรณีที่มีการเรียกใช้จำนวนเต็ม
	การทำงานของไปป์ไลน์	
หน่วยคำนวณและตรรกะ	การประมวลผล	
ปัญหาของไปป์ไลน์	ปัญหาจากข้อมูล	โดยชุดคำสั่งที่ทดสอบจะมีการเรียกใช้ รีจิสเตอร์ผลลัพธ์ของคำสั่งก่อนหน้ามาเป็นตัวถูกดำเนินการในคำสั่งถัดไป
ค่าอื่นๆ	ค่าตัวบ่งชี้	ในกรณีที่ใช้ชุดคำสั่งที่มีการเปลี่ยนค่าตัวบ่งชี้
	การเรียกใช้ไบต์ และเวิร์ด	

การวิเคราะห์ชุดทดสอบในกลุ่มของการกระโดดข้ามการทำงาน

ในส่วนของการกระโดดข้ามการทำงานจะทำการแบ่งออกเป็น 2 ชุดหลักคือ การกระโดดข้ามการทำงานแบบเก็บค่าเลขชี้ตำแหน่งเดิม และการกระโดดข้ามแบบไม่มีเงื่อนไข โดยมีความแตกต่างของรหัสดำเนินการ ณ. ไบต์ที่ 7 คือจะมีค่า B และ A ในเลขฐานสิบหกตามลำดับ การทำงานในส่วนนี้จะไม่มีผลต่อค่าตัวบ่งชี้ จึงมีการตรวจสอบเฉพาะส่วนของการเรียกใช้รีจิสเตอร์ ผลที่ได้หมายถึงค่าตำแหน่งของเลขที่อยู่ที่เปลี่ยนไปซึ่งรวมถึงในกรณีที่มีการเก็บค่าลงสู่รีจิสเตอร์ที่ 14 และ 15 นอกจากนี้มีการตรวจผลของการใช้คาบเวลาในการทำงาน รวมถึงการอ้างอิงเลขที่อยู่ซึ่งจะมีเฉพาะแบบเลขที่อยู่เชิงดรรชนี (Index addressing) การใช้ชุดทดสอบในกลุ่มการกระโดดข้ามการทำงานจะมีผลต่อการทำงานของไปป์ไลน์ ดังนั้นการทดสอบโดยป้อนชุดทดสอบอย่างต่อเนื่องจะช่วยเป็นการตรวจสอบการทำงานของไปป์ไลน์และการเกิดปัญหาชนิดควบคุมของไปป์ไลน์ได้อีกด้วย ชุดทดสอบสามารถครอบคลุมได้ดังตารางที่ 4.17

ตารางที่ 4.17 ส่วนต่างๆที่ได้ทำการตรวจสอบด้วยชุดคำสั่งกระโดดข้ามการทำงาน

หน่วยที่มีการเรียกใช้	สิ่งที่ได้ทำการตรวจสอบ	หมายเหตุ
หน่วยควบคุม	สัญญาณที่ควบคุมแต่ละหน่วย	
	ข้อมูลที่เป็นจำนวนเต็ม	ในกรณีที่มีการเรียกใช้จำนวนเต็ม
	การทำงานของไปป์ไลน์	
	ค่าเลขชี้ตำแหน่ง	เนื่องจากคำสั่งนี้มีผลกับค่าเลขชี้ตำแหน่งแต่ไม่มีผลต่อการทำงานในส่วนอื่น
ปัญหาของไปป์ไลน์	ปัญหาจากการควบคุม	โดยใช้วิธีการขยายคาบการทำงานและหยุดการทำงานอื่นๆจนกว่าจะมีการเอาคำสั่งใหม่เข้ามา
ค่าอื่นๆ	ค่าตัวบ่งชี้	ในกรณีที่ใช้ชุดคำสั่งที่มีการเปลี่ยนค่าตัวบ่งชี้

การวิเคราะห์ชุดทดสอบในกลุ่มของการคูณ

การคูณจะมี 2 แบบคือการคูณแบบธรรมดาและการคูณที่มีการบวกค่ารวมด้วย ซึ่งความแตกต่างจะอยู่ที่ไบต์ที่ 4 ของรหัสดำเนินการ โดยถ้ามีค่า 0 หมายถึงการคูณแบบธรรมดา นอกจากนั้นจะเป็นการคูณแบบบวกค่าด้วย การคูณทั้งสองแบบจะมีผลต่อค่าตัวบ่งชี้จึงมีการตรวจสอบใน

ส่วนนี้ด้วย รวมถึงค่าผลลัพธ์ที่ได้ การเรียกใช้รีจิสเตอร์ การใช้งานแบบไบต์และเวิร์ด โดยมีการอ้างอิงเลขที่อยู่แบบการเข้าถึงโดยตรง การอ้างอิงแบบเลขที่อยู่เชิงดรรชนี และแบบดรรชนีเลขฐาน (Base Index) การทำงานของการคูณนี้จะใช้เวลาในการทำงานมากกว่า 2 คาบทั้งนี้ขึ้นอยู่กับจำนวนของเลขที่ต้องการคูณ

ตารางที่ 4.18 ส่วนต่างๆที่ได้ทำการตรวจสอบด้วยชุดคำสั่งคูณ

หน่วยที่มีการเรียกใช้	สิ่งที่ได้ทำการตรวจสอบ	หมายเหตุ
หน่วยรีจิสเตอร์	การอ่าน / เขียน	เมื่อมีการอ่านและเขียนข้อมูลจากรีจิสเตอร์เพื่อมาทำการคูณ
	สัญญาณอ่าน /เขียน	
	การเรียกใช้รีจิสเตอร์	เมื่อต้องการเก็บและเรียกตัวถูกดำเนินการจากรีจิสเตอร์
บัสข้อมูลขนาด 32 บิต	การอ่านค่าจากบัส	
	การเขียนค่าลงสู่บัส	
หน่วยของการคูณ	การคูณค่า	ทั้ง 2 แบบ คือ MUL และ MLA
หน่วยควบคุม	สัญญาณที่ควบคุมแต่ละหน่วย	
	ข้อมูลที่เป็นจำนวนเต็ม	ในกรณีที่มีการเรียกใช้จำนวนเต็ม
	การทำงานของไปป์ไลน์	
หน่วยคำนวณและตรรกะ	การประมวลผล	
ปัญหาของไปป์ไลน์	ปัญหาจากข้อมูล	โดยชุดคำสั่งที่ทดสอบจะมีการเรียกใช้ รีจิสเตอร์ผลลัพธ์ของคำสั่งก่อนหน้ามาเป็นตัวถูกดำเนินการในคำสั่งถัดไป
ค่าอื่นๆ	ตัวบ่งชี้	ในกรณีที่ใช้ชุดคำสั่งที่มีการเปลี่ยนค่าตัวบ่งชี้
	การใช้ไบต์ และเวิร์ด	

การวิเคราะห์ชุดทดสอบในกลุ่มของการติดต่อหน่วยความจำ

1. คำสั่งในกลุ่มแรกนี้ไม่สามารถจัดแบ่งได้อย่างแน่นอน โดยมีคำสั่งที่สามารถทำงานได้ 4 แบบคือ การเก็บข้อมูลลงหน่วยความจำแบบเดี่ยว แบบหลายทาง การเอาข้อมูลจากหน่วยความจำแบบเดี่ยว แบบหลายทาง การทดสอบจะตรวจการเรียกใช้รีจิสเตอร์ ผลที่ได้จากกการทำงาน จำนวนของคาบเวลาที่ใช้งาน รวมถึงการอ้างเลขที่อยู่ทั้งแบบการเข้าถึงโดยตรง การอ้างอิงแบบเลขที่อยู่เชิงดรรชนี และแบบดรรชนีเลขฐาน (Base Index) การใช้งานแบบไบต์และเวิร์ด การทำงานปกติในแต่ละคำสั่งจะมีการใช้คาบเวลาไม่เท่ากัน โดยคำสั่งของการเอาข้อมูลจากหน่วยความจำแบบหลายทางจะใช้ถึง 3, 5 หรือมากกว่านั้น (>5) ทั้งนี้ขึ้นกับว่าจะมีการเอาข้อมูลจากหน่วยความจำลงสู่รีจิสเตอร์กี่ตัว ทำนองเดียวกันนี้จึงทำให้การเก็บข้อมูลลงหน่วยความจำมีการใช้คาบเวลา มากกว่าหรือเท่ากับ 2 คาบ

2. ในส่วนของการแลกเปลี่ยนข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์ จะมีค่ารหัสดำเนินการแน่นอน และคาบเวลาใช้งานคงที่คือ 4 มีการตรวจสอบในส่วนของการเรียกใช้รีจิสเตอร์การอ้าง ข้อมูลแบบไบต์และเวิร์ด รวมถึงการอ้างอิงเลขที่อยู่แบบการเข้าถึงโดยตรง การอ้างอิงแบบเลขที่อยู่เชิงดรรชนี และแบบดรรชนีเลขฐาน (Base Index)

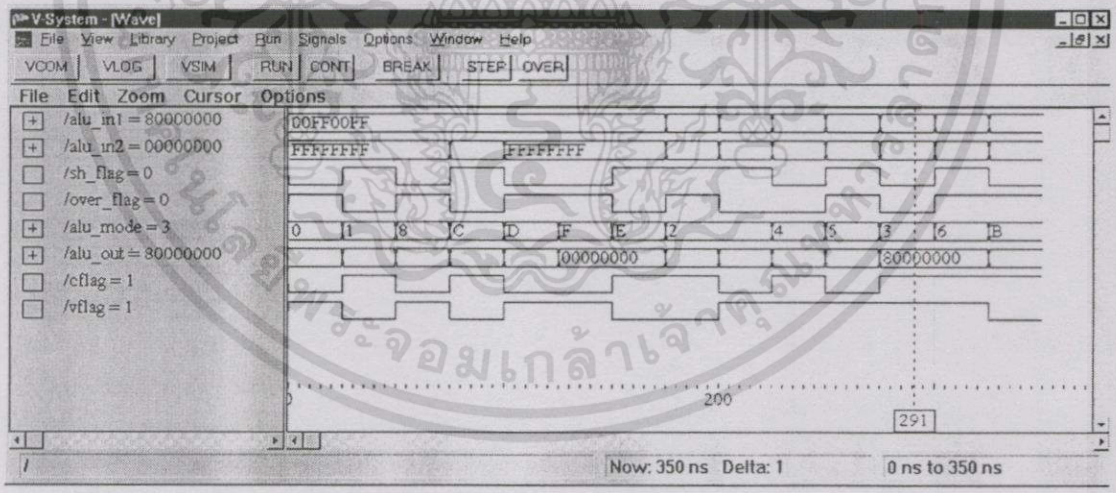
ตารางที่ 4.19 ส่วนต่างๆที่ได้ทำการตรวจสอบด้วยชุดคำสั่งการติดต่อหน่วยความจำ

หน่วยที่มีการเรียกใช้	สิ่งที่ได้ทำการตรวจสอบ	หมายเหตุ
หน่วยรีจิสเตอร์	การอ่าน / เขียน	ทุกคำสั่งที่มีการใช้ตัวถูกดำเนินการจาก รีจิสเตอร์และเก็บผลลัพธ์สู่รีจิสเตอร์
	สัญญาณอ่าน /เขียน	
	การเรียกใช้รีจิสเตอร์	เมื่อต้องการอ่านค่าหรือเก็บค่าลงสู่รีจิสเตอร์
บัสข้อมูลขนาด 32 บิต	การอ่านค่าจากบัส	
	การเขียนค่าลงสู่บัส	
หน่วยของการคูณ	การผ่านค่า	เนื่องจากไม่มีคำสั่งการคูณ
ชิฟเตอร์	การผ่านค่า	ในกรณีที่ทำคำสั่งที่ไม่มีการชิฟ
หน่วยควบคุม	สัญญาณที่ควบคุมแต่ละหน่วย	
	ข้อมูลที่เป็นจำนวนเต็ม	ในกรณีที่มีการเรียกใช้จำนวนเต็ม
	การทำงานของไปป์ไลน์	

	ค่าเลขชี้ตำแหน่ง	ในกรณีที่มีการเก็บค่าลงสู่รีจิสเตอร์ 15 ซึ่งจะเป็นรีจิสเตอร์เดียวกับ PC
หน่วยคำนวณและตรรกะ	การประมวลผล	ในกรณีที่มีการคำนวณเลขที่ตำแหน่งในหน่วยความจำ
ปัญหาของไปป์ไลน์	ปัญหาจากข้อมูล	โดยชุดคำสั่งที่ทดสอบจะมีการเรียกใช้ รีจิสเตอร์ผลลัพธ์ของคำสั่งก่อนหน้ามาเป็นตัวถูกดำเนินการในคำสั่งถัดไป
	ปัญหาจากการควบคุม	ในกรณีที่เก็บค่าลงสู่รีจิสเตอร์ที่ 15
ค่าอื่นๆ	ค่าตัวบ่งชี้	ในกรณีที่ใช้ชุดคำสั่งที่มีการเปลี่ยนค่าตัวบ่งชี้
	การเรียกใช้ไบต์ และเวิร์ด	

4.3 ผลจากการจำลองการทำงานด้วยชุดทดสอบที่กำหนด ตัวอย่างผลการทดลองของหน่วยต่างๆ

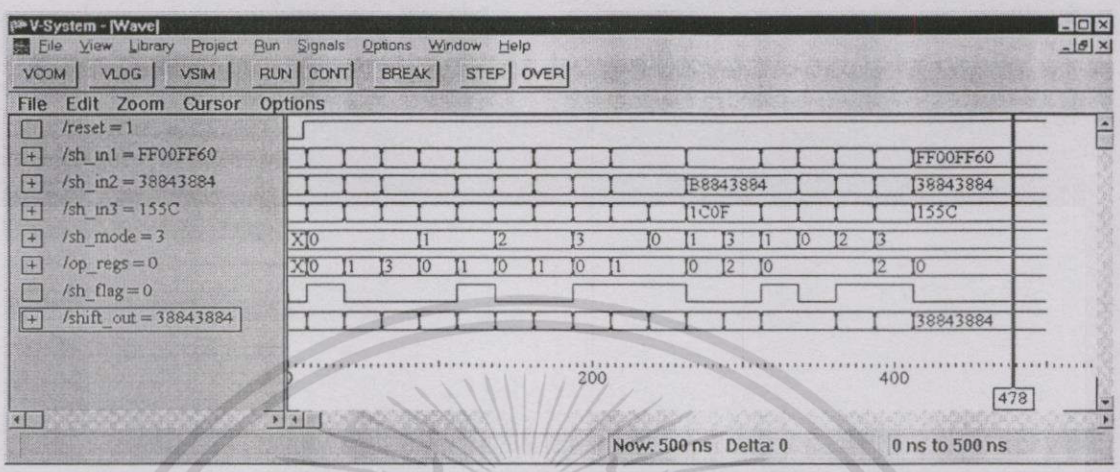
1. หน่วยคำนวณและตรรกะ



รูปที่ 4.25 Waveform ของหน่วยคำนวณและตรรกะ

จากรูปที่ 4.25 เป็นผลของการทำตรวจสอบการลบค่าตามคำสั่ง SUB r1,r2,r3 โดยที่ค่าใน r2 คือค่าใน alu_in1 r2 คือค่า alu_in2 และผลลัพธ์เก็บใน r3 ในรูปคือค่า alu_out ในที่นี้คือการบวกค่า 80000000 กับค่า 0 ผลที่ได้คือค่า 80000000 โดยไม่คำนึงถึงค่าตัวบ่งชี้จึงไม่ได้นำมาแสดงในรูป

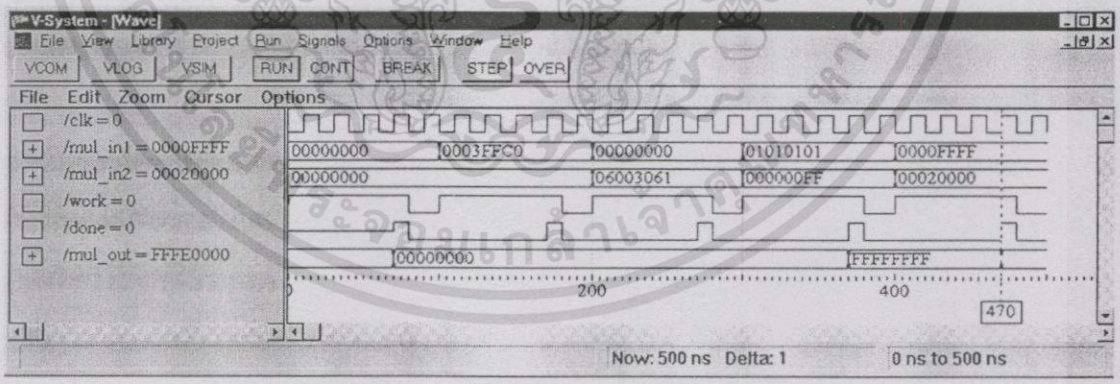
2. ชิฟเตอร์



รูปที่ 4.26 waveform ของชิฟเตอร์

จากรูปที่ 4.26 เป็นการแสดงผลของการผ่านค่าในหน่วยชิฟ โดยที่ผ่านเข้าไปทาง sh_in2 จะต้องออกสู่ผลลัพธ์จะเหมือนเดิม ดังในค่าของ shift_out ตามรูป

3. หน่วยของการคูณ



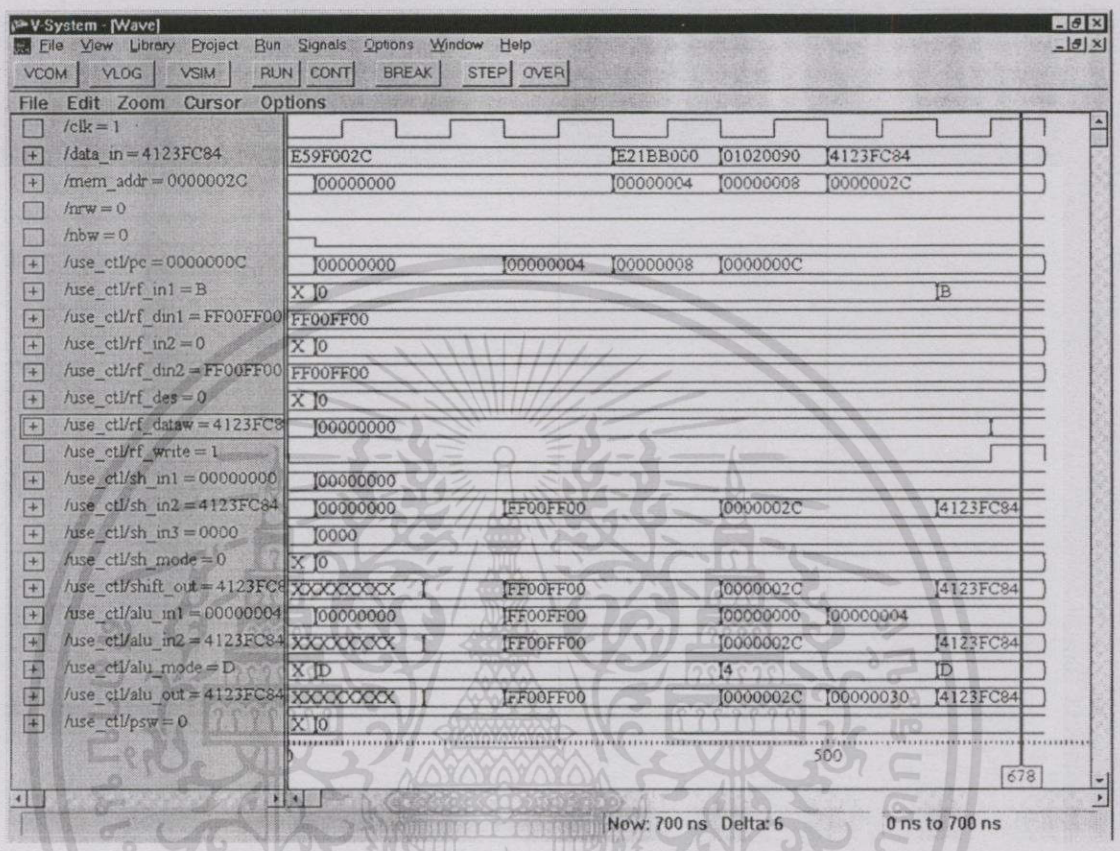
รูปที่ 4.27 Waveform ของหน่วยการคูณ

จากรูปที่ 4.27 เป็นการแสดงการคูณค่า FFFF กับ 20000 โดยผลลัพธ์ที่ได้จะเป็นค่า FFFE0000 ดังแสดงในค่าของ mul_out

4. การทดสอบคำสั่งอย่างต่อเนื่อง

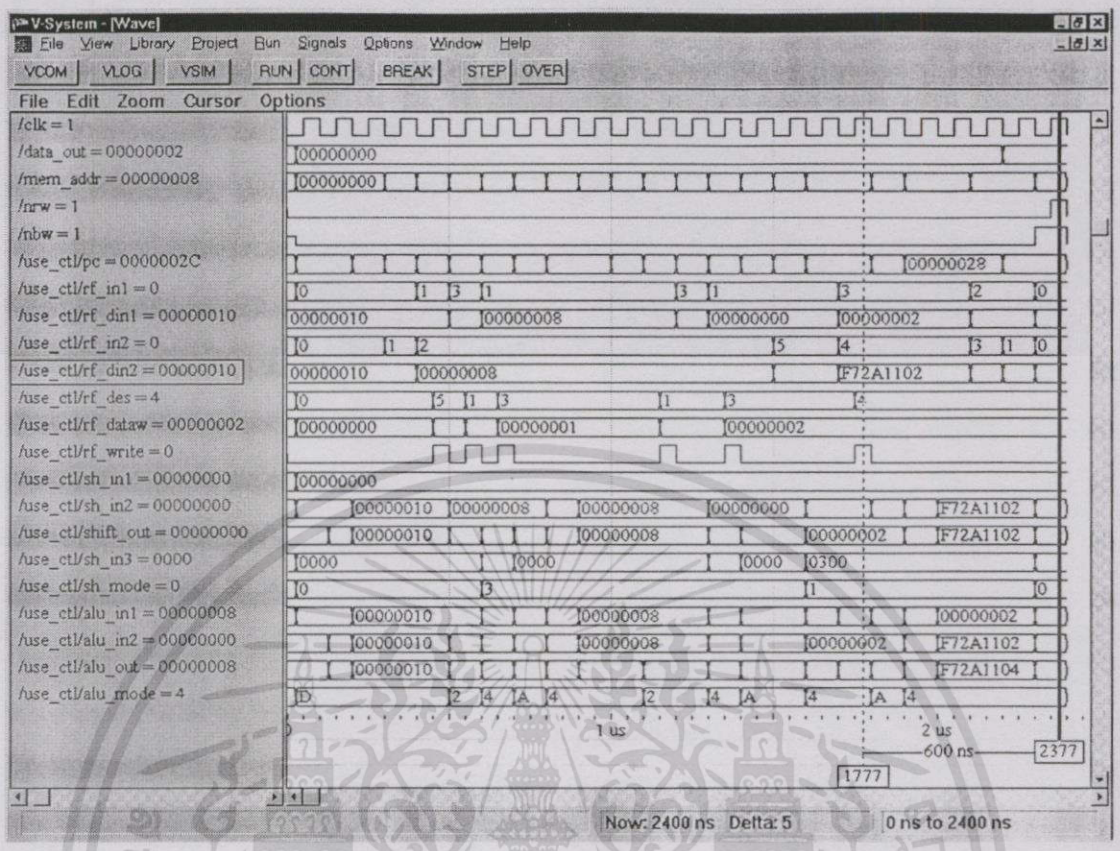
เพื่อตรวจสอบความถูกต้องของ ARM 7 ที่ได้ออกแบบนั้นมีการทำงานที่ถูกต้องหรือไม่ ดังนี้

รูปที่ 4.28 Waveform ของการทำงานคำสั่งโหลด



รูปที่ 4.28 Waveform ของการทำงานคำสั่งโหลด

จากรูปที่ 4.28 เป็นการตรวจสอบการทำงานของการทำงานของคำสั่งนำข้อมูลจากหน่วยความจำมาเก็บไว้ในรีจิสเตอร์ตัวที่ 11 ในที่นี้ข้อมูลที่นำเอาเข้ามาคือ 4123FC84 สังเกตที่สัญญาณ data_in ในที่นี้ต้องการนำเข้าไปในรีจิสเตอร์ที่ 11 คือ B สังเกตได้ค่า rf_in1 จะมีค่าเป็น B ตรงตามที่ต้องการและพอร์ทของการเขียนข้อมูลลงสู่รีจิสเตอร์ (rf_dataw) ก็มีค่าเช่นเดียวกับค่าใน data_in คือ 4123FC84 พร้อมทั้งสัญญาณของการเขียนค่าก็ถูกเซตตามสัญญาณของ rf_wโดยการทำงานในส่วนนี้จะมีการเอาค่าลงสู่บัสข้อมูลแล้วผ่านค่าทางชิฟเตอร์โดยจะสังเกตพบว่าทำงานได้ถูกต้อง จากค่าที่ออกจากหน่วยชิฟคือ shift_out



รูปที่ 4.29 Waveform ของการทำโปรแกรมการหาร

จากรูปที่ 4.29 เป็นผลจากการเขียนโปรแกรมการหารโดยการลดค่าทีละ 8 $(00000008)_{16}$ จากค่า 16 $(00000010)_{16}$ นั้นหมายถึงการเอา 8 ไปหาร 16 นั้นเองผลลัพธ์ที่ได้จะต้องเป็นค่า 2 โดยจะเขียนค่าลงใน รีจิสเตอร์ที่ 4 โดยดูจากค่าใน rf_des ผลลัพธ์ที่ได้สามารถสังเกตได้จากค่าใน rf_dataw ซึ่งเป็นค่า 00000010

ผลจากการจำลองการทำงานด้วยชุดคำสั่งในลักษณะของชุดทดสอบ ผลปรากฏว่าสามารถตรวจสอบข้อผิดพลาดที่เกิดจากคาบเวลาที่ใช้ผิดพลาด และคำสั่งของการ load/store ผิดพลาดจึงได้ทำการแก้ไขแล้วใช้ชุดทดสอบเดิมตรวจสอบการทำงานอีกครั้งจนกระทั่งไม่พบข้อผิดพลาดใดเกิดขึ้นในการจำลองการทำงาน

4.4 ผลการวิเคราะห์การจำลองการทำงาน

เนื่องจากการสร้างชุดทดสอบเพื่อการทดสอบ จะทำการคัดเลือกคำสั่งโดยการวิเคราะห์พฤติกรรมการส่งผ่านข้อมูลระหว่างโมดูลในไมโครโปรเซสเซอร์ ARM 7 ในระดับของคำสั่งการทำงาน ซึ่งมีทั้งหมด 2,660 คำสั่งจากคุณลักษณะที่กำหนดไว้ก่อนการออกแบบ โดยเป็นชุดทดสอบที่จัดแบ่งตามคำสั่งของ ARM 7 ทั้งหมด 26 คำสั่งตามคุณลักษณะของไมโครโพรเซสเซอร์ที่กำหนด

ไว้ (Specification) เนื่องจากคำสั่งของ ARM สามารถกำหนดให้ไม่มีการคำนวณค่าตัวบ่งชี้ (flag) ดังนั้นชุดทดสอบนี้จึงไม่คำนึงผลของตัวบ่งชี้ที่เกิดขึ้นจากการทำงาน ซึ่งทำให้ชุดทดสอบแบ่งเป็น 2 กลุ่มหลักคือ กลุ่มของการคำนวณที่ไม่มีผลกับค่าตัวบ่งชี้ และการคำนวณที่ส่งผลกับตัวบ่งชี้ โดยใน 16 คำสั่งแรกเป็นคำสั่งในการคำนวณทางคณิตศาสตร์และลอจิกซึ่งส่วนใหญ่จะอยู่ใน หน่วยคำนวณและตรรกะ โดยแต่ละคำสั่งจะแบ่งออกได้ 4 กรณีที่ตามการใช้งานของการอ้างอิง Operand ที่มาจากรีจิสเตอร์ทั้งคู่, มาจากรีจิสเตอร์และค่า Immediate, Operand มาจากรีจิสเตอร์และอีกค่าทำการเลื่อนค่าซึ่งได้ค่ามาจากรีจิสเตอร์, Operand มาจากรีจิสเตอร์แต่อีกค่าที่สามารถทำการเลื่อนค่าได้มาจากค่า Immediate ตัวอย่างเช่นการทดสอบการทำงานของคำสั่ง AND จะสามารถใช้ชุดทดสอบ 4 ชุดที่แทนการทำงานที่เกิดขึ้นทั้งหมดของ AND ซึ่งในแต่ละชุดสามารถแบ่งเป็น 2 กลุ่มของการมีค่าตัวบ่งชี้ ดังนั้นจึงได้ชุดคำสั่งของการทำงานในส่วนของ 16 คำสั่งแรกเป็น

จำนวนคำสั่ง x ชนิดของการอ้างอิง operand x การคำนึงถึงตัวบ่งชี้

= 16 x 4 x 2

= 128 คำสั่ง

เนื่องจากการทดสอบการทำงานของการประมวลผลทางคณิตศาสตร์และลอจิกนั้นเป็นการทดสอบในระดับของ Abstract เพราะขนาดของข้อมูลสามารถมีค่าได้ถึง 2^{32} ค่า และเกิดจาก Operand 2 ค่า ดังนั้นหากจะต้องทดสอบทุกค่าจะต้องใช้ชุดคำสั่งถึง 2^{64} ค่าในแต่ละคำสั่งซึ่งต้องใช้เวลานานมากขึ้นหลายเท่าตัว ดังนั้นการทดสอบการคำนวณทางคณิตศาสตร์หรือลอจิก จะใช้เฉพาะบางค่าเท่านั้นเช่น

ADD r1, r2, r3 ; R2 = 02h R3 = FFh

จากตัวอย่างข้างต้นสามารถทดสอบการทำงานของการบวกแบบไม่คิดค่าตัวบ่งชี้ ซึ่งจะมีการทำงานไม่แตกต่างหากเปลี่ยนค่า R2 และ R3 เป็นค่าอื่นๆ จึงสามารถทดสอบการทำงานได้ด้วยค่าเพียงชุดเดียว

คำสั่งการกระโดดข้ามการทำงาน (Branch) มีชุดทดสอบทั้งในกรณีที่มีการคืนค่ากลับตำแหน่งเดิมได้ (BL) และไม่สามารถกลับไปตำแหน่งเดิมได้ (B) โดดทั้ง 2 แบบจะมีชุดทดสอบการทำงานของไปป์ไลน์ซึ่งจะเกิดกรณีของการ Stalling ไปป์ไลน์ และค่าของ PC ให้มีการทำงานที่ถูกต้อง ดังนั้นเมื่อใช้ชุดทดสอบการทำงานที่มีคำสั่ง B จะทำให้สามารถครอบคลุมการทดสอบทั้งในส่วนของการอ้างอิงหน่วยความจำ, การทำงานไปป์ไลน์ และการทำงานของชุดคำสั่งทั้งหมด การทดสอบคำสั่ง Branch มีข้อแตกต่างเพิ่มตรงที่การป้อนชุดคำสั่งทดสอบอย่างต่อเนื่องเพื่อให้สามารถตรวจสอบการทำงานของ Pipeline ว่ามีการคำนวณค่า PC ได้ถูกต้องหรือไม่ เพื่อให้ได้ตำแหน่งของหน่วยความจำต่อไปที่ต้องการทำงาน

ชุดทดสอบมีคำสั่งของการทำงานในแบบที่เป็นการคูณตามปกติ (MUL) และการคูณที่สามารถมีการคำนวณการบวกได้ (MLA) คำสั่งของการย้ายข้อมูลระหว่างหน่วยความกับรีจิสเตอร์มีชุดทดสอบที่คำนึงข้อมูลที่มีการแลกเปลี่ยน รวมทั้งค่า PC ที่เปลี่ยนไป คำสั่งการกระโดดข้ามการทำงาน การคูณและการย้ายข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์มีชุดทดสอบในการอ้างอิงหน่วยความจำทั้ง 3 แบบ ณIndirect, Index และ Base Index ดังนั้นจึงได้ชุดทดสอบดังนี้ 2 x 3 x 2 = 12 ชุดคำสั่ง

ชุดทดสอบคำสั่งการ Load / Store / Swp ทั้ง 3 ชุดคำสั่งนี้ แบ่งออกเป็น 4 กลุ่มย่อย ตามชนิดของการอ้างอิงหน่วยความจำ ดังนี้ indirect addressing mode, index addressing mode, base index addressing mode และ byte and word โดยในกลุ่มนี้ให้ทำการทดสอบเป็นชุดคำสั่งต่อเนื่องเพื่อตรวจสอบการทำงานของ Pipeline ด้วยเนื่องจากในแต่ละคำสั่งจะมีการใช้งาน Cycle ในการทำงานที่แตกต่างกันออกไป

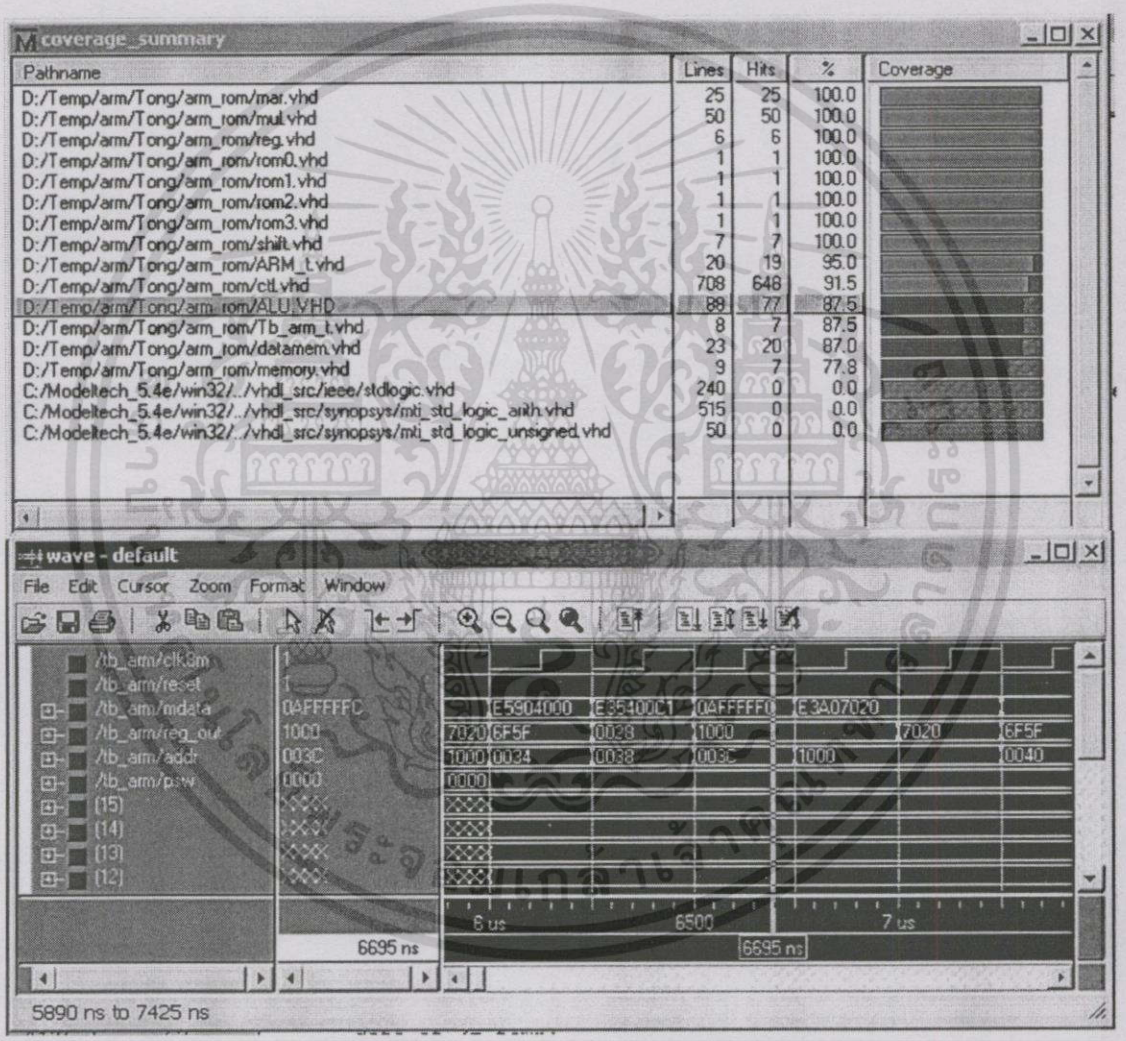
จากชุดทดสอบทั้งหมดที่ได้สร้างขึ้นมาและทำการป้อนให้เป็นอินพุทของระบบแล้วทดสอบการทำงานโดยวิธีการจำลองการทำงานเทียบกับ ARM7 Debugger สามารถวิเคราะห์ความสามารถของชุดทดสอบได้ว่ามีความสามารถในการทดสอบการทำงานของระบบได้มากน้อยเพียงไร โดยใช้เทคนิคของ Functional Code Coverage

Functional Code Coverage เป็นเทคนิคการวิเคราะห์คุณภาพของชุดทดสอบว่าสามารถแทนอินพุทของระบบทั้งหมดได้มากน้อยเพียงไร โดยการนำชุดทดสอบป้อนเข้ากับระบบแล้วติดตามดูผลกระทบของชุดทดสอบนั้นต่อระบบว่ามีมีผลกระทบมากน้อยเพียงไร เช่นการตรวจสอบจำนวนบรรทัดของโค๊ดโปรแกรมที่มีการทำงานอันเนื่องมากจากชุดทดสอบที่ป้อนเข้าไป ซึ่งชุดทดสอบทั้ง 2,660 ชุดทดสอบที่ป้อนเข้าไปในการทดสอบในวิทยานิพนธ์นี้ สามารถวิเคราะห์โดยใช้ฟังก์ชั่น Code Coverage ในซอฟแวร์ ModelSim เวอร์ชั่น 5.4 ได้ผลปรากฏว่าสามารถ ทดสอบการทำงานของระบบได้กว่า 91.9 % โดยนับจากจำนวนบรรทัดที่ของโค๊ดที่มีการทำงานดังรูปที่ 4.30 และสามารถตรวจสอบจำนวนครั้งของโค๊ดโปรแกรมที่ถูกกระทบโดยชุดทดสอบได้ ซึ่งโปรแกรมจะแสดงเลขจำนวนครั้งในคอลัมน์แรกของหน้าต่างซ๊อสโค๊ด ดังรูปที่ 4.31

จากรูปที่ 4.30 องค์ประกอบย่อย MAR มีโค๊ดภาษาวีเฮชดีแอลที่สามารถทำงานได้อยู่ 25 บรรทัด และจำนวนบรรทัดที่มีการทำงานเนื่องจากชุดทดสอบนี้มีทั้งหมด 25 บรรทัดเช่นกัน ทำให้ค่าของ Functional Code Coverage ในองคประกอบย่อย MAR เป็น 100% แต่อย่างไรก็ตามในองค์ประกอบย่อย Ctl, ALU และ DataMem มีค่าของ Functional Code Coverage ไม่ถึง 100% เนื่องจากการทำงานบางส่วนที่เป็นฟังก์ชั่นการทำงานย่อยได้ถูกตรวจสอบด้วยเทคนิคจำลองการทำงานไปแล้วในขั้นตอนการตรวจสอบความถูกต้องของแต่ละองค์ประกอบย่อย ทำให้ไม่จำเป็นต้องมีการตรวจสอบซ้ำในระดับของการรวมทุกองค์ประกอบย่อยเข้าด้วยกัน เช่นการตรวจสอบเงื่อนไขการทำงานของคำสั่งทุกๆ คำสั่งก่อนที่จะมีการทำงานจริง เนื่องจาก ARM7 จะมีการตรวจสอบเงื่อนไข

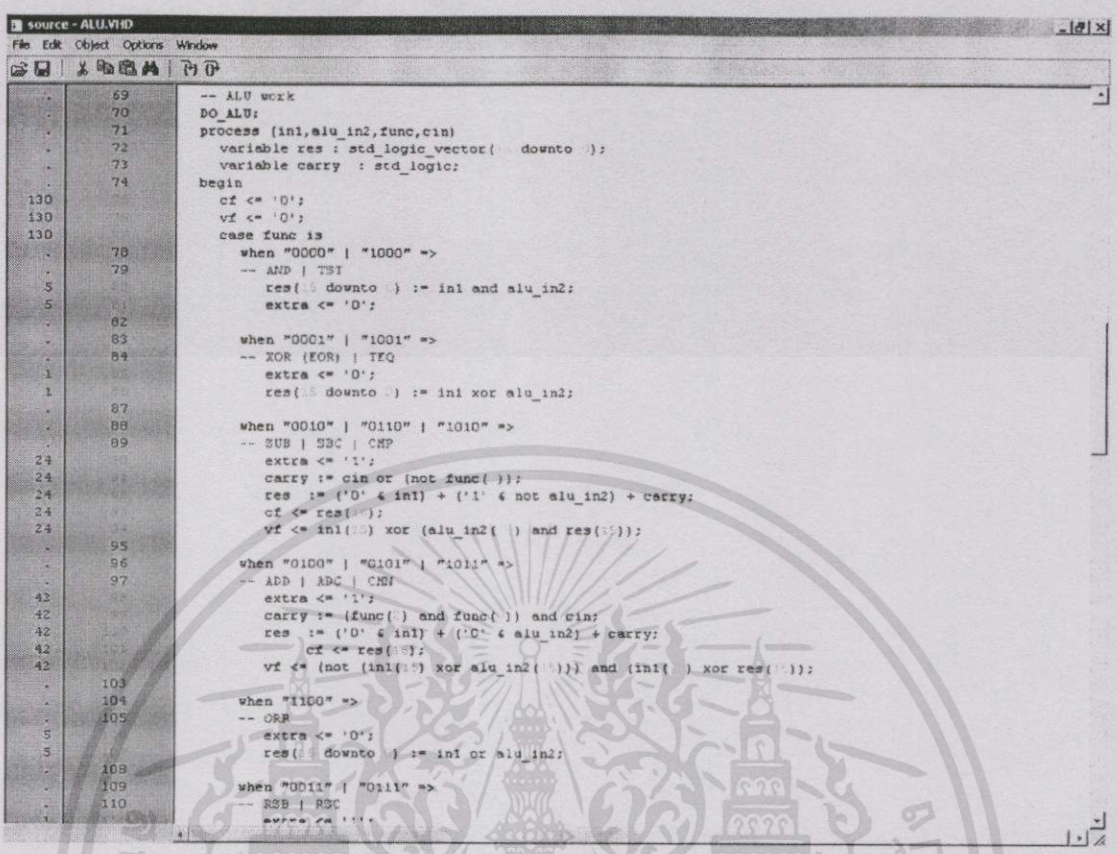
ที่ระบุก่อนการทำงานทุกคำสั่ง หากเงื่อนไขไปไม่เป็นจริงก็จะไม่ทำงานตามคำสั่งนั้นๆ ในส่วนตรวจสอบนี้จึงละเว้นส่วนนี้ไว้ เพื่อให้ทุกคำสั่งได้มีการทำงานทั้งหมด ทำให้โค๊ดบางส่วนของภาษาวีเฮชดีแอลในบางองค์ประกอบย่อยไม่มีการทำงานเนื่องจากชุดทดสอบนี้

หมายเหตุ โมดูล Reg, Rom0-3, Tb_Arm_t และ Memory เป็นโมดูลที่เขียนขึ้นมาเพื่อช่วยในการตรวจสอบการทำงานของ ARM7 ดังนั้นจึงไม่รวมในการวิเคราะห์และการตรวจสอบค่า Functional Code Coverage นี้



รูปที่ 4.30 ผลการวิเคราะห์จากฟังก์ชั่น Code Coverage

ค่าผลการตรวจสอบการทำงาน 91.9 % ที่ได้เกิดมาจากค่าเฉลี่ยของ Functional Code Coverage ในองค์ประกอบย่อยต่างๆ ดังนี้ MAR, MUL, Shift, ARM_t, Ctl, ALU และ DataMem คือ 25, 50, 19, 648, 77, 20 บรรทัดตามลำดับ รวมแล้ว 846 บรรทัดจากทั้งหมด 921 บรรทัด



รูปที่ 4.31 ผลกระทบจากชุดทดสอบต่อโค๊ดโปรแกรม

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

งานวิจัยในวิทยานิพนธ์นี้สนใจในเรื่องการตรวจสอบความถูกต้องของฟังก์ชั่นการทำงานของไมโครโพรเซสเซอร์ต่างๆ ที่ออกแบบด้วยภาษาวีเฮชดีแอล โดยได้ยกตัวอย่างเอาไมโครโพรเซสเซอร์ ARM7 เป็นกรณีศึกษา โดยได้ประยุกต์เอาการตรวจสอบการทำงานในสองเทคนิคมาใช้คือ การตรวจสอบสถานะนามธรรม (Abstract State Machine) และเทคนิคจำลองการทำงาน (Simulation) เพื่อตรวจสอบความถูกต้องในระดับต่างๆ กัน

5.1.1 Abstract State Machine

ในเทคนิคของ Abstract State Machine เป็นเทคนิคที่นำเสนอโดย James K. Huggins และ David Van Campenhout เพื่อใช้ในการตรวจสอบการทำงานของไมโครโพรเซสเซอร์ ARM2 ในส่วนของไปป์ไลน์ ในงานวิจัยนี้จึงได้นำเอาวิธีการดังกล่าวมาประยุกต์ใช้ร่วมกับการทำให้ระบบเป็นนามธรรมโดยใช้ Sementic Extraction เพื่อลดความซับซ้อนและจำนวนสถานะที่เป็นไปได้ของ State Machine โดยจะทำการเปลี่ยนจากข้อกำหนดคุณลักษณะและโค๊ดภาษาวีเฮชดีแอลให้เป็น Abstract State Machine เพื่อทำการตรวจสอบความถูกต้องซึ่งกันและกัน โดยจะใช้ในการตรวจสอบในระดับขององค์ประกอบย่อยต่างๆ

จากผลการตรวจสอบของการทำงานในส่วนขององค์ประกอบย่อยต่างๆ โดยใช้ Abstract State Machine นั้นสรุปผลได้ว่าการทำงานในส่วนต่างๆ สามารถทำงานได้อย่างถูกต้อง ยกเว้นในส่วนของการประมวลผลทางตรรกะที่ไม่ได้มีการตรวจสอบเงื่อนไขก่อนการทำงานจริงทำให้วงจรที่สร้างออกมาจะมีการทำงานตลอดเวลา ทำให้เปลืองพลังงานโดยใช่เหตุ แต่ก็ไม่ได้ทำให้การทำงานผิดพลาดแต่ประการใด

5.1.2 Simulation-Based Technique

เป็นวิธีในการทดสอบความถูกต้องของระบบโดยการสร้างชุดทดสอบเข้าไปทำงานในระบบที่ต้องการทดสอบ แล้วเปรียบเทียบผลที่ได้กับผลจากการทดสอบโดยใช้ชุดทดสอบเดียวกันป้อนไปยังชุดอ้างอิงเพื่อตรวจสอบความถูกต้อง โดยใช้งานวิจัยนี้ได้เลือกชุดอ้างอิงโดยใช้โปรแกรมตรวจสอบหาข้อบกพร่อง (Debugger) ของ ARM 7 ซึ่งการเลือกชุดทดสอบนั้นจะใช้วิธีการเลือกที่ทำให้ชุดทดสอบนั้นสามารถครอบคลุมการทำงานของระบบให้ได้มากที่สุด ผลจากการทดสอบโดยวิธีการทำงานดังกล่าวปรากฏว่าระบบสามารถทำงานได้อย่างถูกต้องในทุกๆ ชุด

ทดสอบและความสามารถในการครอบคลุมการทำงานของระบบของชุดทดสอบที่สร้างขึ้นมาเป็น 91.9 % จากการคำนวณค่าเฉลี่ยโดยอาศัยค่า Functional Code Coverage ที่ได้ในแต่ละองค์ประกอบย่อย ซึ่งสามารถทำให้ค่าเฉลี่ยที่ได้มีค่าสูงขึ้นโดยการสร้างชุดสอบเพิ่มเติมเพื่อให้ครอบคลุมการทำงานทั้งหมดได้

5.1.3 การตรวจสอบและการทดสอบระบบดิจิตอล

จากผลการวิจัยพบว่าการที่จะรับประกันความถูกต้องของวงจรดิจิตอลที่ได้ออกแบบด้วยภาษาระดับสูงอย่าง VHDL นั้น จำเป็นต้องทำการตรวจสอบโดยการแบ่งเป็นองค์ประกอบย่อยเพื่อทำการตรวจสอบโดยใช้เทคนิคของ Abstract State Machine ซึ่งจะสามารถรับประกันความถูกต้องของการทำงานขององค์ประกอบย่อยนั้นๆ ได้ทั้งหมด หลังจากนั้นจึงนำองค์ประกอบย่อยต่างๆ ที่ผ่านการตรวจสอบโดยเทคนิคดังกล่าวมารวมกันแล้วตรวจสอบอีกครั้งด้วยการจำลองการทำงานโดยเลือกชุดคำสั่งในการตรวจสอบเพื่อให้ครอบคลุมทั้งส่วนของการทำงานทั้งหมด ซึ่งจะช่วยให้ผู้ออกแบบมั่นใจในความถูกต้องของระบบมากขึ้นและลดเวลาในการตรวจสอบความถูกต้องของระบบโดยรวม ทั้งยังลดการสูญเสียจากกระบวนการผลิตโดยการค้นหาข้อบกพร่องของระบบก่อนที่จะทำงานผลิตจริง รวมทั้งยังสามารถประยุกต์ใช้ได้กับการออกแบบในปัจจุบันที่ผู้ออกแบบนิยมใช้ภาษาระดับสูงในการบรรยายพฤติกรรมการทำงานเช่น VHDL ซึ่งแต่ละเทคนิคที่ใช้ในการตรวจสอบจะมีข้อดีข้อเสียแตกต่างกันดังตารางที่ 5.1

ดังนั้นจากข้อได้เปรียบของการจำลองการทำงาน จึงจำเป็นต้องใช้การจำลองการทำงานในส่วนของวงจรที่มีขนาดใหญ่ และซับซ้อน หน่วยคำนวณทางคณิตศาสตร์และตรรกะ, หน่วยการเลื่อนค่า, หน่วยการคูณ และหน่วยอื่นๆ ในไมโครโพรเซสเซอร์ จึงจำเป็นต้องมีการแบ่งแยกการตรวจสอบเป็นแต่ละโมดูลเช่น โมดูลของการรับค่า โมดูลของการประมวลผลในแต่ละหน่วยนั้นๆ โมดูลของการแสดงผลลัพธ์ เป็นต้น การตรวจสอบจึงมีหน้าที่ทดสอบการทำงานของวงจรที่ออกแบบให้ถูกต้องตามคุณลักษณะที่ต้อง แต่การจำลองการทำงานจะช่วยในการตรวจสอบหน้าที่โดยละเอียดที่หน่วยนั้นๆสามารถทำได้ เช่น หน่วย alu ได้ผ่านการตรวจสอบแล้วว่าสามารถรับข้อมูลเข้าสู่หน่วย, ถอดรหัสเพื่อทำงานและตั้งค่า flag ด้วยวิธีการที่ตรงตามการทำงานของไมโครโพรเซสเซอร์ ARM อย่างถูกต้องแล้ว ดังนั้นการจำลองการทำงานจึงมีการทดสอบโดยการสมมติค่าป้อนเข้าสู่หน่วย alu แล้วตรวจสอบเฉพาะผลลัพธ์ที่ได้โดยไม่จำเป็นต้องตรวจสอบการทำงานภายในเนื่องจากผ่านการตรวจสอบแล้ว ทำให้การจำลองการทำงานทำได้รวดเร็วยิ่งขึ้น ไม่จำเป็นต้องตรวจสอบการทำงานทุกขั้นตอนและทุกค่าที่เป็นได้ อันเป็นสาเหตุของการช่วยให้การจำลองการทำงานร่วมกับการตรวจสอบโมดูลเชิงนามธรรมสามารถกระทำได้อย่างรวดเร็วกว่าขั้นตอนการจำลองการทำงานเพียงอย่างเดียว

ตารางที่ 5.1 ข้อดีและข้อเสียจากการตรวจสอบอย่างมีแบบแผนและการจำลองการทำงาน

การตรวจสอบ	ข้อดี	ข้อเสีย
การตรวจสอบในรูปแบบมาตรฐาน	- ตรวจสอบการทำงานของวงจรได้โดยไม่ต้องสร้างชุดทดสอบ - ทำให้ช่วงเวลาในการตรวจสอบวงจรลดน้อยลง - เหมาะสมที่จะใช้กับองค์ประกอบย่อยทางคณิตศาสตร์ - สามารถบอกประสิทธิภาพของวงจรที่ออกแบบได้เช่นการกินไฟในการทำงาน เป็นต้น	- มีความซับซ้อนและยาก - การตรวจสอบวงจรขนาดใหญ่และซับซ้อนทำได้ยาก - ไม่สามารถตรวจสอบวงจรที่เกี่ยวข้องกับเวลาได้ - เครื่องมือที่ใช้ในการตรวจสอบกับการออกแบบระดับสูงมีน้อยและไม่สะดวกในการใช้งาน
การจำลองการทำงาน	- ง่ายและสามารถใช้ในทุกองค์ประกอบย่อยของวงจร รวมถึงการทำงานที่เกี่ยวข้องกับเวลา - มีเครื่องมือให้ใช้กับการออกแบบระดับสูงมากและสะดวกในการใช้งาน	- เสียเวลาในการจำลองการทำงานตามชุดทดสอบค่อนข้างสูง และแปรผันกับประสิทธิภาพของระบบที่ใช้จำลองการทำงาน - ต้องสร้างชุดทดสอบจำนวนมากเพื่อให้ครอบคลุมการทำงานของวงจรให้มากที่สุด - ไม่สามารถบอกประสิทธิภาพของ วงจรเกี่ยวกับพลังงานที่ใช้

5.2 ข้อเสนอแนะ

การตรวจสอบการทำงานของวงจรดิจิตอลที่ได้มีการประยุกต์ใช้ในวิทยานิพนธ์ฉบับนี้สามารถช่วยย่นระยะเวลาในการตรวจสอบการทำงานให้สั้นลง เมื่อเทียบกับการตรวจสอบการทำงานโดยใช้การจำลองการทำงานแต่เพียงอย่างเดียว อีกทั้งยังสามารถวิเคราะห์ประสิทธิภาพของการทำงานในแต่ละส่วนประกอบย่อยได้อีกด้วย รวมทั้งวิธีการในการเลือกชุดทดสอบที่ใช้จำลองการทำงานในวิทยานิพนธ์นี้สามารถช่วยลดจำนวนชุดทดสอบที่จำเป็นต้องใช้ในการตรวจสอบความถูกต้องของระบบทั้งในระดับองค์ประกอบย่อยและระบบโดยรวมได้อีกด้วย

ซึ่งนักออกแบบสามารถประยุกต์ใช้วิธีการดังกล่าวในการตรวจสอบการทำงานของระบบที่ออกแบบขึ้นเองได้ โดยเฉพาะไมโครโพรเซสเซอร์ เนื่องจากโดยพื้นฐานขององค์ประกอบย่อยต่างๆ ของไมโครโพรเซสเซอร์โดยทั่วไป จะมีลักษณะใกล้เคียงกัน ทำให้ข้อกำหนดคุณลักษณะและวิธีการตรวจสอบสถานะนามธรรมสามารถนำไปประยุกต์ใช้ได้ในหลายๆ ส่วน

เทคนิคการเลือกชุดทดสอบโดยการวิเคราะห์โครงสร้างและฟังก์ชั่นการทำงานหลักที่ได้จาก Abstract State Machine จะช่วยให้อินพุทที่ต้องใช้ในการจำลองการทำงานของไมโครโพรเซสเซอร์ลดลง เนื่องจากการทำงานในแต่ละอย่างมีข้อมูลอินพุทที่เป็นไปได้หลายรูปแบบ ดังนั้นจึงสามารถแทนรูปแบบข้อมูลอินพุทที่เป็นไปได้ทั้งหมดด้วยบางอินพุทเท่านั้น ทำให้ระยะเวลาที่ใช้ในการตรวจสอบการทำงานโดยใช้ การจำลองการทำงานลดลง

เอกสารอ้างอิง

1. George Milne, "Formal Specification Verification of Digital Systems.", McGraw-Hill Book company, 1994.
2. Masahiro Fujita, "Model Checking : Its Basics and Reality.", IEEE Trans. Design and tes*t*, pp. 217-222, 1998.
3. K.L. MaMillan, "Symbolic model checking : an approach to the state explosion problem.", Kluwer, 1993.
4. E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications.", ACM Trans. Program Lang. Syst., Vol. 8, No. 2, pp. 244-263, 1986.
5. J. Queille and J. Sifakis, "Specification and verification of concurrent systems in CAESAR," Proceedings of Fifth ISP, 1982.
6. R. Gerth, D. Peled, M. Vardi, "Simple on the fly automatic verification of linear temporal logic.", Proceedings of IFIP/WG6.1 Symposium on Protocol Specification, Testing and Verification, June, 1995.
7. D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang, "Protocol verification as a hardware design aid," Int. conference on Computer Design, pp. 522-525, 1992.
8. M. Kaltenbach, "Model checking for UNITY.", Tehn. Rep. TR94-31. University of Texas at Austin, Dec., 1994.
9. R. Cleaveland, J. Parrow, and B. Steffen, "The concurrency Workbench : a semantics-based tool for the verification of concurrent systems," *ACM Trans. Program Lang. Syst.,* Vol. 15, No. 1, pp. 36-72, 1993.
10. T. Filkorn, H. Scheneider, A. sholz, A. Strasser, and P. Warkentin, "SVE user' guide," Tech. Rep. ZFE BT SE1-SVE-1. Siemens AG, Corporate Research and Development, Munich, Germany.
11. D. Deharbe, B. Josko, and R. Schlor, "Specification and validation methods for programming language and systems," Chap. Specification and verification of VHDL-based system-level hardware designs, Oxford University Press, New York, pp. 331-410, 1995.
12. C. Daws and S. Yovine, "Two examples of verification of multirate timed automata with KRONOS," Proceeding of 1995 IEEE Realtime systems symposium, RTSS'95, 1995.

13. Z. Har'el and R.P. Kurshan, "Software for analytical development of communications protocols," *AT&T Bell Lab. Tech. J.* Vol.69, No. 1, pp. 45-59, 1990.A. Roscoe, "Model checking CSP," In A classical mind: essays in honor of C.A.R. Hoare, A. Rosoe, Ed., Prentice-Hall, 1994.
14. A. Roscoe, "Model checking CSP," In a classical mind : essays in honor of C.A.R. Hoare, A. Rosoe Ed., Prentice-Hall, 1994.
15. Robert V. Binder, "Testing Object-Oriented Systems Models, Patterns, And Tools" Addison Wesley, 1999
16. Lan Somerville, "Software Engineering", fifth edition, Addison-Wesley, 1996
17. Tsu-Hua Wang, Chong Guan Tan, "Practical code coverage for Verilog"., Verilog HDL Conference Proceedings,IEEE International,1995, Pages 99 –104
18. Williams, T.W, Mercer, "Code coverage, what does it mean in terms of quality?", Reliability and Maintainability Symposium 2001, Proceedings Annual , 2001, Pages 420 –424
19. Ahmad, I, Dhodhi, M.K., "State assignment of finite-state machines" , Computers and Digital Techniques, IEE Proceedings, Volume: 147 Issue: 1 , Jan. 2000, Pages 15 –22
20. Sasaki H., "A formal semantics for Verilog-VHDL simulation interoperability by abstract state machine", Design Automation and Test in Europe Conference and Exhibition 1999 Proceedings , 1999, Pages 353 –357
21. L. Hennessy & Davis A Patterson , "Computer Architecture A Quantitative Approach", pp250-343 Morgan Kaufmann Publishers ,Inc.
22. John V. Oldfield, Richard C. Dorf, "Field Programmable Gate Arrays reconfigurable logic for rapid prototyping and implementation of digital systems.", A Wiley-Interscience Publication, 1995.
23. Sajjan G. Shiva , "Computer Design & Architecture.", University of Alabama in Huntsville, Second Edition 1991. pp. 430-432.
24. เจริญ วงษ์ชุ่มเย็น, สมศักดิ์ มิตะถา, วรรณรัช สันติอมรทัต, อาทิตย์ ทองทักษ์, "การทวนสอบไมโครโพรเซสเซอร์ ARM7 ที่ออกแบบด้วยภาษา VHDL.", วารสารลาดกระบัง, มิถุนายน, 2542.

ประวัติผู้เขียน

นายเจริญ วงษ์ชุ่มเย็น เกิดเมื่อวันที่ 27 กันยายน 2517 ที่จังหวัดลำพูน สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมคอมพิวเตอร์(เกียรตินิยม อันดับ 1) คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง ปีการศึกษา 2540 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตร์มหาบัณฑิต เมื่อ พ.ศ. 2541 ปัจจุบันเป็นอาจารย์พิเศษ ประจำภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง