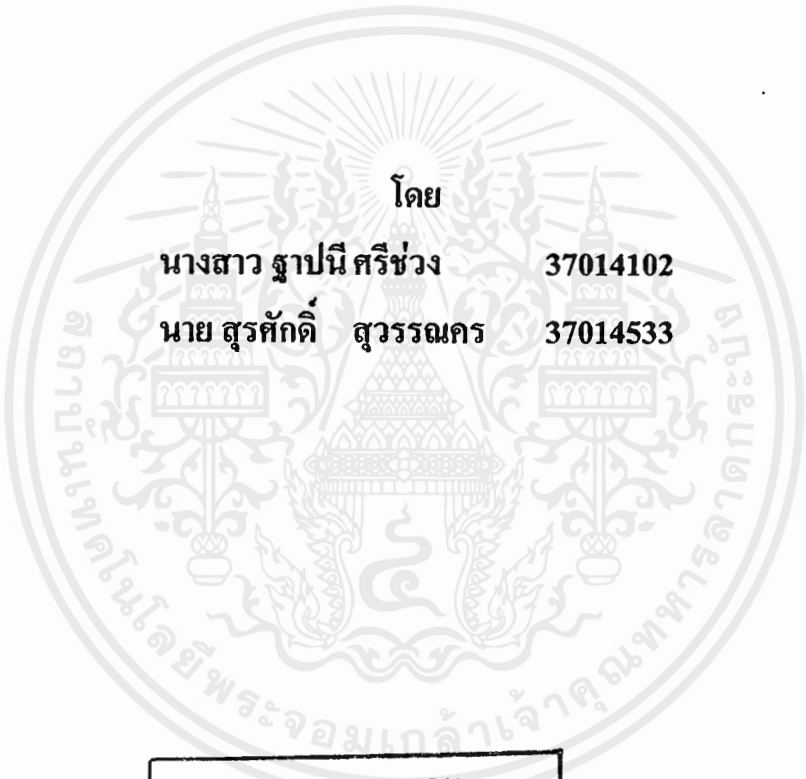
การสื่อสารแบบอนุกรมสำหรับไมโครคอนโทรลเลอร์

SERIAL COMMUNICATION CORE FOR MICROCONTROLLER

โดย

นางสาว ฐาปนี ศรีช่วง 37014102

นาย สุรศักดิ์ สุวรรณดร 37014533



ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

การสื่อสารแบบอนุกรมสำหรับไมโครคอนโทรลเลอร์

SERIAL COMMUNICATION CORE FOR MICROCONTROLLER

โดย

นางสาว ฐาปนี ศรีช่วง 37014102

นาย สุรศักดิ์ สุวรรณคร 37014533

อาจารย์ที่ปรึกษา

อาจารย์ อภิเนตร อูนากูล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่องการสื่อสารแบบอนุกรมสำหรับไมโครคอนโทรลเลอร์

ผู้จัดทำ

1. นางสาว ฐาปนี ศรีช่วง
2. นาย สุรศักดิ์ สุวรรณคร

..... อาจารย์ที่ปรึกษา

(อ. ดำเนตร ตุหาญ)

การสื่อสารแบบอนุกรมสำหรับไมโครคอนโทรลเลอร์

นางสาว ฐาปนี ศรีช่วง
นาย สุรศักดิ์ สุวรรณคร

อ. อภิเนตร อูนากูล
ปีการศึกษา 2540

บทคัดย่อ

ในปัจจุบันนี้ นับได้ว่าการสื่อสารแบบอนุกรมเป็นพื้นฐานของการสื่อสารของระบบการติดต่อสื่อสารโดยทั่ว ๆ ไป เนื่องจากข้อได้เปรียบหลายประการที่เหนือกว่า การสื่อสารแบบขนาน อาทิเช่น ความประหยัดในการวางสายสื่อสาร ประสิทธิภาพในการส่งข้อมูล เป็นต้น

นอกจากนี้การพัฒนานำส่วนของการสื่อสารไปใส่ไว้ในไมโครคอนโทรลเลอร์ถือว่าเป็นการเพิ่มความสามารถในการทำงานของตัวไมโครคอนโทรลเลอร์มากยิ่งขึ้น ซึ่งในโครงงานนี้ได้ทำการออกแบบ และสร้างวงจรหลัก สำหรับการสื่อสารแบบอนุกรมสำหรับไมโครคอนโทรลเลอร์ในระดับของส่วนประกอบที่สามารถนำกลับมาใช้ใหม่ได้ (Reusable Component) ซึ่งวงจรหลักดังกล่าวได้แก่ Universal Asynchronous Receiver/Transmitter (UART) , I^2C-bus Memory Interface และ I^2C-bus Controller และได้ทำการทดสอบในระดับของซอฟท์แวร์และฮาร์ดแวร์ โดยในระดับของซอฟท์แวร์ได้ทำการเขียนโปรแกรมทดสอบ. ส่วนในระดับของฮาร์ดแวร์ได้ใช้ FPGA ช่วยในการทดสอบ

คณะผู้จัดทำ
28 เม.ย. 41

Serial Communication Core for Microcontroller

Thapanee Srichoung

Surasak Suwannakorn

Apinatr Unakul

1997

Abstract

This project is Serial Communication Core for Microcontroller. In present serial communication is basic of general communication. Because of serial communication is more advantageous than parallel communication such as inexpensive and reliable than parallel communication over long distance and transmitting over longer distances requires thicker wires to reduce signal degradation .

In additional, If you want to increase efficiency of microcontroller. You can to add serial communication in it. In this project, we designed and implemented the three soft cores of serial communication for microcontroller. The soft cores are the reusable component . They are Universal Asynchronous Receiver/Transmitter (UART) , I^2C-bus Memory Interface and I^2C-bus Controller . And they were tested in software level and hardware level. Software level, they were tested by Test Bench. Hardware level, they were tested by FPGA and programming.

28 Apr. 98

กิตติกรรมประกาศ

ขอขอบพระคุณบุคคลที่มีรายนามดังต่อไปนี้ ที่ให้ความช่วยเหลือในการทำโครงงานนี้จนประสบความสำเร็จ

1. อาจารย์ อภิเนตร อูนากูล
   ให้โอกาสในการทำงานที่น่าสนใจ
   ให้คำปรึกษาและดูแลการทำงานอย่างต่อเนื่อง
2. คุณธนพร ทองเผือก
   สละเวลาในการทำงานมาให้คำปรึกษาและคำแนะนำ
   เอื้อเฟื้ออุปกรณ์ในการทดสอบ
3. น.ส. ปัญจภา เรืองสินทรัพย์
   เอื้อเฟื้อเครื่องคอมพิวเตอร์ในบางเวลาและให้คำแนะนำ
4. น.ส. วรรณรัช สันติอมรทัต
   เอื้อเฟื้อเครื่องคอมพิวเตอร์ในบางเวลาและให้คำแนะนำ
5. ห้องปฏิบัติการ EMMBEDDED SYSTEM LAB
6. ภาควิชาวิศวกรรมคอมพิวเตอร์

สุดท้ายขอขอบคุณเพื่อน ๆ พี่ ๆ และ น้อง ๆ ที่ไม่ได้เอ่ยนามในที่นี้ ที่คอยเป็นกำลังใจให้ทำโครงงานนี้จนสำเร็จ

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มา

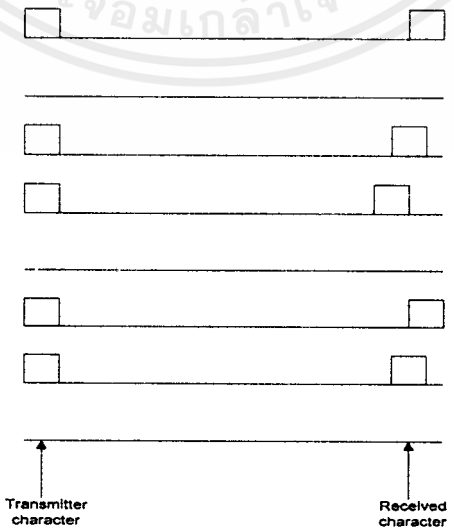
การสื่อสารโดยทั่ว ๆ ไปในปัจจุบันนี้จะมีอยู่ 2 ประเภทที่สำคัญ (จำแนกวิธีการส่งตามลักษณะการจัดข้อมูล) ได้แก่ การสื่อสารแบบอนุกรม และการสื่อสารแบบขนาน

ลักษณะของการสื่อสารแบบขนาน คือ การนำทุก ๆ บิท ของรหัสที่ประกอบเป็นอักษรหนึ่งตัว ส่งออกไปพร้อม ๆ กัน ในเวลาเดียวกัน ทำให้จะต้องใช้สายเท่ากับจำนวนบิทที่ประกอบเป็นตัวอักษรตัวนั้น

ส่วนการสื่อสารแบบอนุกรม มีลักษณะดังนี้คือ บิททั้งหมดของตัวอักษรหนึ่งตัว จะถูกนำมาส่งออกไปทีละบิท ติดต่อกันไปเรื่อย ๆ ตามสายส่งที่มีเพียงสายเดียว เมื่อทางด้านรับ รับข้อมูลมาแล้ว ก็ต้องนำมาจัดเป็นตัวอักษรขึ้นใหม่ ให้ตรงกับชุดตัวอักษรที่ทางด้านส่งได้ส่งมา ซึ่งวิธีการดังกล่าวนี้จะต้องประกอบด้วย ความสัมพันธ์ในการทำงานระหว่างด้านรับและด้านส่ง เพื่อที่รับข้อมูลได้ถูกต้อง

ซึ่งการสื่อสารทั้ง 2 ประเภทมีข้อดีข้อเสียของแต่ละแบบดังต่อไปนี้

1) การสื่อสารข้อมูลแบบขนาน จะส่งได้เร็วกว่าการส่งข้อมูลแบบอนุกรม เนื่องจากในการส่งแต่ละครั้งของการสื่อสารแบบขนาน จะส่งทีละ 1 ตัวอักษร แต่ถ้าเป็นการสื่อสารแบบอนุกรมจะส่งทีละบิท เพราะฉะนั้น ถ้าส่งข้อมูล 1 ตัวอักษร (character) ก็จะต้องส่งไปเป็นจำนวนครั้งเท่ากับจำนวนบิทที่ประกอบเป็น 1 ตัวอักษร
2) การสื่อสารแบบอนุกรมจะประหยัดกว่าเนื่องจากใช้สายเพียงเส้นเดียวในการสื่อสาร แต่การสื่อสารแบบขนานจะต้องใช้สายเป็นจำนวนเท่ากับ จำนวนบิทที่ประกอบเป็น ตัวอักษร
3) ในการสื่อสารที่ระยะทางไกล ๆ การสื่อสารแบบขนานจะเกิด skew คือ ข้อมูลแต่ละบิทที่ถูกส่งไปบนสายพร้อมกัน แต่ฝ่ายรับอาจจะได้รับไม่พร้อมกันเนื่อง ความต้านทานของแต่ละสายที่ใช้สื่อสารไม่เท่ากัน



รูปที่ 1-1 แสดงลักษณะการเกิด skew ในการสื่อสารข้อมูลแบบขนาน

จากการเปรียบเทียบข้อดี-ข้อเสียของการสื่อสารแต่ละแบบ ทำให้ทางผู้จัดทำโครงงานคิดว่าการสื่อสารแบบอนุกรม น่าจะพัฒนาเพื่อใช้เพิ่มประสิทธิภาพของการทำงานของ ไมโครคอนโทรลเลอร์ ได้ดีกว่าการสื่อสารแบบขนาน ซึ่งในปัจจุบันนี้ เทคโนโลยีของระบบการสื่อสารข้อมูลแบบอนุกรมก็มีหลายแบบ ซึ่งที่ใช้อยู่ในระบบคอมพิวเตอร์ทั่วไป ก็จะมีการสื่อสารข้อมูลแบบอนุกรม โดยใช้ชิปที่เรียกว่า UART (Universal Asynchronous Receiver/Transmitter) และในปัจจุบันนี้ยังมีรูปแบบการสื่อสารแบบอนุกรมอีกลักษณะหนึ่ง ที่เรียกว่า I^2C-bus ซึ่งเป็นการสื่อสารข้อมูลแบบอนุกรมอีกรูปแบบหนึ่งที่น่าสนใจ นอกจากนี้ยังมีการสื่อสารแบบอนุกรมในลักษณะอื่นๆ อีก เช่น Universal Serial Bus (USB) และ Smart-Card เป็นต้น

1.2 วัตถุประสงค์ของงานวิจัย

1) ศึกษาทฤษฎีเบื้องต้นของระบบการสื่อสารแบบอนุกรมในรูปแบบของ UART และ I^2C-bus
2) ทำการออกแบบวงจร โดยใช้ทฤษฎีเบื้องต้นของ UART และ I^2C-bus

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้จะทำการออกแบบวงจร 3 ตัว โดยตัวหนึ่งจะเป็นการออกแบบโดยใช้รูปแบบของ UART ส่วนอีก 2 ตัว จะออกแบบโดยใช้รูปแบบของ I^2C-bus เนื่องจากจะต้องมีการทดสอบว่าวงจรที่ออกแบบโดยใช้รูปแบบของ I^2C-bus สามารถที่จะสื่อสารกันได้จริงจึงต้องออกแบบ 2 ตัว โดยตัวหนึ่งจะใช้เชื่อมต่อกับหน่วยความจำ ส่วนอีกตัวหนึ่งใช้เชื่อมต่อ กับไมโครคอนโทรลเลอร์

เนื่องจากงานวิจัยตัวนี้ ยังไม่ได้เป็นการออกแบบให้รวมอยู่ในไมโครคอนโทรลเลอร์ตัวเดียว ดังนั้นถ้ามีการวิจัยต่อน่าจะเป็นการนำไปรวมไว้ในไมโครคอนโทรลเลอร์ตัวเดียว

1.4 วิธีการดำเนินงาน

1) ในขั้นแรกเป็นการเริ่มศึกษาหลักการ และทฤษฎีเบื้องต้น ต่างๆ ที่เกี่ยวกับโครงงานตัวนี้ ซึ่งจะมีเรื่องหลัก ๆ อยู่ 2 เรื่องด้วยกัน คือ ลักษณะและรูปแบบการสื่อสารข้อมูลแบบอนุกรมของ UART และ ลักษณะและรูปแบบการสื่อสารข้อมูลแบบอนุกรมของ I^2C-bus ซึ่งมีรายละเอียดอยู่ในบทที่ 2
2) เป็นการนำความรู้ทั้งหมดที่ได้จากการศึกษา มาออกแบบวงจรทั้ง 3 ตัว โดยเริ่มจากการออกแบบ Block Diagram และ แผนภูมิสถานะ (State Diagram) ซึ่งได้แสดงรายละเอียดต่าง ๆ ไว้ในบทที่ 3
3) เมื่อออกแบบเสร็จเรียบร้อยแล้ว ก็นำสิ่งที่ออกแบบมาสร้างจริง โดยเริ่มด้วยการเขียนโปรแกรมด้วยภาษา VHDL หลังจากนั้นก็ทำการทดสอบ เมื่อทดสอบผ่านเรียบร้อยแล้ว ก็สร้างออกมาเป็น FPGA ซึ่งเมื่อสร้างเป็น FPGA แล้วก็ต้องมีการทดสอบว่าใช้งานได้จริงหรือไม่
4) เป็นการทดสอบในระดับฮาร์ดแวร์ (Hardware) หรือ การทดสอบระบบ (System Test)

บทที่ 2

ทฤษฎีเบื้องต้น

สำหรับในบทนี้จะกล่าวถึงทฤษฎีต่าง ๆ ที่ใช้ในการออกแบบวงจรทั้ง 3 ตัว ซึ่งทฤษฎีที่ใช้ในการออกแบบหลัก ๆ ที่สำคัญได้แก่ ทฤษฎีของ UART (Universal Asynchronous Receiver/Transmitter) ทฤษฎีของ I^2C-bus ทฤษฎีของ VHDL และทฤษฎีของ Xilinx XC40000

2.1 Universal Asynchronous Receiver/Transmitter Concept

UART เป็นอุปกรณ์ที่ใช้ในการสื่อสารแบบอนุกรม ซึ่งมีการส่งเป็นแบบ อะซิงโครนัส (Asynchronous) สามารถที่จะทำการโปรแกรมได้ และรีจิสเตอร์ภายในสามารถอ้างอิงแอดเดรสได้ UART ถูกออกแบบให้มีการใช้งานกับไมโครโปรเซสเซอร์ขนาด 8 บิท แต่บางตัวก็สามารถใช้ได้กับ 16 บิทและ 32 บิท

2.1.1 การเชื่อมต่อภายนอกของ UART

UART จะมีส่วนที่ต่อกับไมโครโปรเซสเซอร์ และส่วนที่ต่อกับอุปกรณ์อื่นๆ

ขาที่ใช้ติดต่อกับไมโครโปรเซสเซอร์ :-

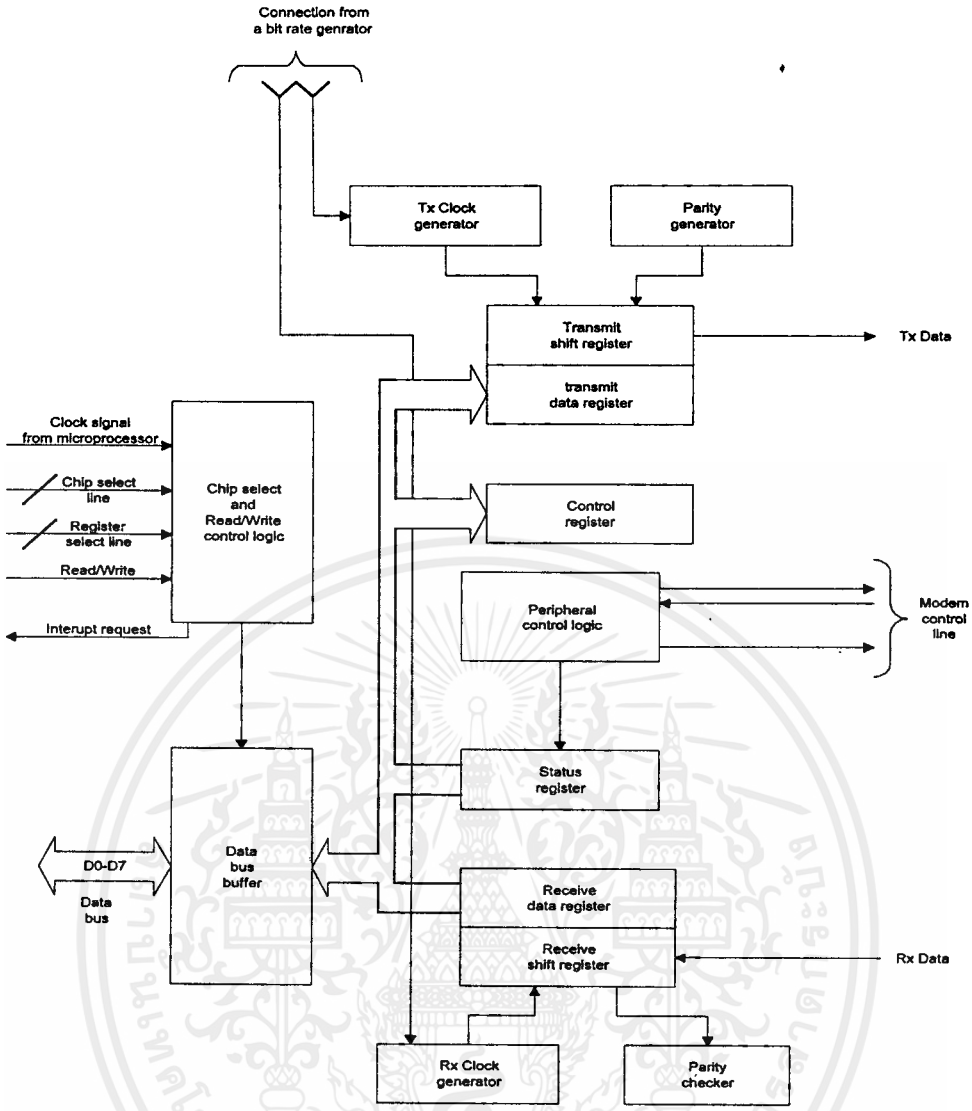
- **Chip select หรือ Chip enable :** เป็นขาที่ไมโครโปรเซสเซอร์ใช้เพื่อเลือกอุปกรณ์ที่จะติดต่อด้วย ขานี้จะต่อกับ แอดเดรส bus ของไมโครโปรเซสเซอร์ผ่านตัวถอดรหัส
- **Register select :** ไมโครโปรเซสเซอร์ใช้เพื่อเลือก รีจิสเตอร์ ภายใน UART ขานี้จะต่อกับ A0 หรือ A1 ของไมโครโปรเซสเซอร์
- **Data bus :** เป็น บัส ขนาด 8 บิท ใช้ขนถ่าย (transfer) ข้อมูลระหว่างไมโครโปรเซสเซอร์ และ UART การ ส่งข้อมูลจะต้องทำการซิงโครไนซ์ (synchronize) กับ สัญญาณนาฬิกา (clock) ของไมโครโปรเซสเซอร์
- **Read/Write :** ไมโครโปรเซสเซอร์ใช้เพื่ออ่านหรือเขียน UART
- **Interrupt Request (/ IRQ) :** เป็นสัญญาณที่ UART ส่งไปบอกไมโครโปรเซสเซอร์ว่า ขณะนี้ถึงเวลาที่จะส่งหรือรับข้อมูล หรือจะไม่ส่งหรือไม่รับข้อมูลเนื่องจากสถานะของโมเด็มยังไม่พร้อม ขาที่ใช้ติดต่อกับอุปกรณ์อื่น ๆ
- **Tx Data (Transmit Data) :** ใช้ส่งข้อมูลเป็นอนุกรมออกไป จากการรับเป็นแบบขนานมาจากไมโครโปรเซสเซอร์ ในการส่งข้อมูล UART จะทำการเพิ่มบิทสตาร์ท (start bit) , บิทพาริตี้ (parity bit) และ บิทสตอป (stop bit) โดยอัตโนมัติ
- **Rx Data (Receive Data) :** ใช้รับข้อมูลเป็นอนุกรมเข้ามา แล้วทำการส่งไปยังไมโครโปรเซสเซอร์เป็นแบบขนาน ในการรับข้อมูล UART จะทำการตัดบิทสตาร์ท , บิทพาริตี้ และบิทสตอป ออกก่อนที่ไมโครโปรเซสเซอร์จะอ่านข้อมูลไป
- **Peripheral Control :** ใช้ติดต่อกับโมเด็มหรืออุปกรณ์อื่น ๆ

อัตราการส่งและรับข้อมูลของ UART จะถูกกำหนดโดยสัญญาณนาฬิกาจากภายนอก

2.1.2 Block Diagram ภายในของ UART

- **Transmit shift register และ Transmit data register :** ไมโครโปรเซสเซอร์จะส่งข้อมูลที่ต้องการจะส่งออกไปมาไว้ใน transmit data register วงจรภายใน UART จะจัดการส่งข้อมูลแต่ละตัวอักษร จาก transmit data register ไปยัง transmit shift register โดยอัตโนมัติเมื่อ shift register นั้นว่าง
- **Receiver shift register และ Receive data register :** ไมโครโปรเซสเซอร์จะรับข้อมูลจาก receive data register เท่านั้น UART จะทำการส่งข้อมูลจาก receive shift register ไปยัง data register โดยอัตโนมัติ เมื่อ data register นั้นว่าง ข้อมูลจะยังคงอยู่ใน data register จนกว่าไมโครโปรเซสเซอร์จะอ่านไปแล้ว

การทำงานร่วมกันของรีจิสเตอร์ 2 ตัวนี้ จะเรียกว่า double buffering ในการทำงานของ UART ก็เหมือนกับอุปกรณ์ ที่เป็นอินพุท/เอาท์พุทอื่นๆ ที่จะใช้เวลานาน ดังนั้นถ้า ไมโครโปรเซสเซอร์ติดต่อกับ UART อยู่ มันจะไม่สามารถไปทำงานอย่างอื่นได้



รูปที่ 2-1 Typical External Pin Assignment for Internal Block Diagram of a UART

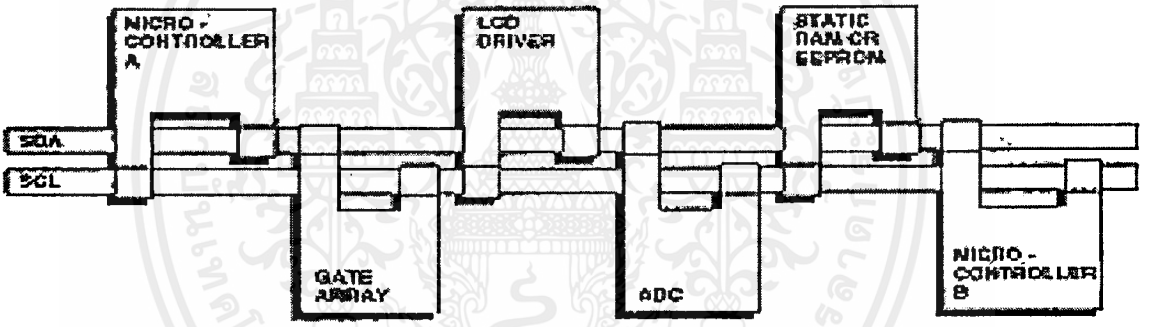
- **Control register :** จะเก็บรูปแบบการทำงานของ UART ซึ่งจะส่งมาจากไมโครโปรเซสเซอร์ เพื่อกำหนดค่าเริ่มต้นในการทำงานของ UART เช่น มีการกำหนดอัตราการส่งข้อมูล , จำนวนบิทข้อมูล , จำนวนบิทของ สตาร์ท , สตอป และ พาริตี้ หรือไม่ก็เป็นการที่ UART ส่งสัญญาณอินเตอร์รัปท์ (interrupt) ไปบอกไมโครโปรเซสเซอร์ เมื่อ transmit data register ว่าง หรือ receive data register เต็ม
- **Status register** : เป็นตัวเก็บสถานะต่าง ๆ ของ UART เช่น บิทของ parity error จะใช้ตรวจสอบความถูกต้องของบิทพาริตี้ เป็นต้น

ไมโครโปรเซสเซอร์จะติดต่อโดยตรงกับ รีจิสเตอร์ เพียง 4 ตัวเท่านั้น คือ transmit data register , control register , status register , receive data register

2.2 ทฤษฎีบทของ I²C-bus

จะมีสายที่ใช้ในการรับส่งข้อมูลระหว่างอุปกรณ์ที่เชื่อมต่อบนบัส อยู่ 2 สาย คือ SDA (Serial Data Line) และ SCL (Serial Clock Line) อุปกรณ์ต่าง ๆ ที่เชื่อมต่อบนบัส จะมีแอดเดรสของตัวเอง และสามารถทำการรับส่งได้โดยขึ้นอยู่กับหน้าที่ของอุปกรณ์แต่ละตัว นอกจากนั้นอุปกรณ์แต่ละตัวสามารถเป็นมาสเตอร์ (master) หรือ สเลฟ (slave) ก็ได้ โดย มาสเตอร์จะเป็นตัวหลักในการส่งข้อมูลและจะเป็นตัวสร้างสัญญาณนาฬิกาด้วย

I²C-bus มีลักษณะเป็นบัสที่มีมาสเตอร์หลายตัว (multi-mater) ซึ่งสามารถต่อกับอุปกรณ์ได้ทีละหลาย ๆ ตัวดังนั้นจึงเป็นไปได้ที่บน I²C-bus สามารถมีไมโครคอนโทรลเลอร์ได้ทีละหลาย ๆ ตัว ดังนั้นจึงมีโอกาสที่ไมโครคอนโทรลเลอร์เหล่านั้น ต้องการที่จะส่งหรือรับข้อมูลในเวลาเดียวกัน ซึ่งบน I²C-bus ได้จัดการเหตุการณ์แบบนี้โดยใช้การแข่งขันกัน (Arbitration)



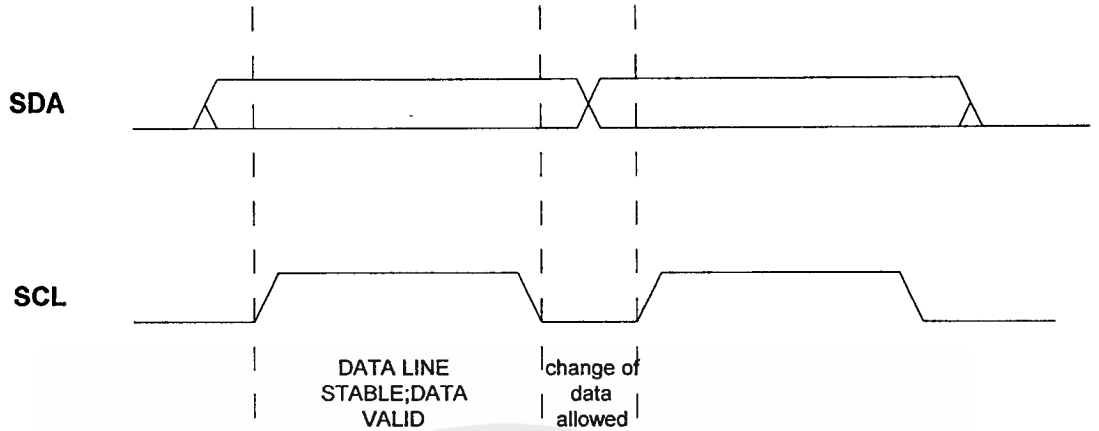
รูปที่ 2-2 ตัวอย่างการเชื่อมต่อ I²C Bus ซึ่งมีการใช้ไมโครคอนโทรลเลอร์ 2 ตัว

2.2.1 ลักษณะทั่ว ๆ ไปของ I²C-bus

ทั้งสาย SDA และ SCL เป็นสายที่ส่งได้ทั้งสองทาง โดยถูกต่อกับความต่างศักย์ที่เป็นบวก (positive supply voltage) ผ่านตัวต้านทานที่ pull-up ไว้ ในขณะที่บัสว่างทั้งสองสายจะมีสถานะสูง (High) ความเร็วในการส่งจะเป็น 100 Kbps เมื่ออยู่ในโหมดมาตรฐาน (standard mode) และจะเป็น 400 Kbps ในโหมดเร็ว (fast mode)

- **การขนส่งบิทข้อมูล (Bit Transfer)**

เนื่องจากมีอุปกรณ์หลายประเภทที่สามารถต่อกับ I²C-bus ได้ ดังนั้นค่าความต่างศักย์ของการเป็น 0 หรือ 1 จะขึ้นกับอุปกรณ์แต่ละตัว



รูปที่ 2-3 การส่งบิท บน I²C-bus

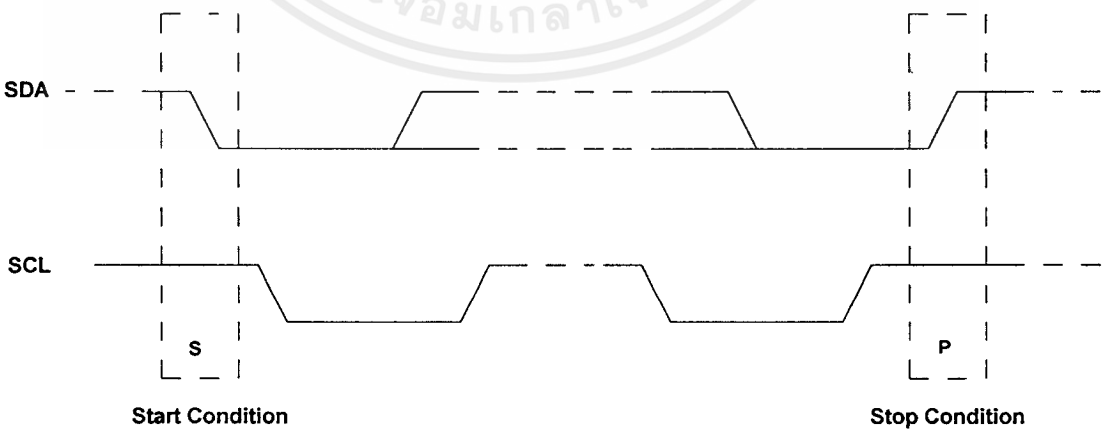
- **ความถูกต้องของข้อมูล (Data Validity)**

ข้อมูลที่อยู่บนสาย SDA จะต้องคงที่ ไม่มีการเปลี่ยนแปลงขณะที่ SCL มีสถานะสูง และจะเปลี่ยนสถานะจากสถานะต่ำ (Low) เป็นสถานะสูง ได้เฉพาะตอนที่ SCL มีสถานะต่ำเท่านั้น

- **เงื่อนไขการเริ่มต้นและการหยุด (START and STOP conditions)**

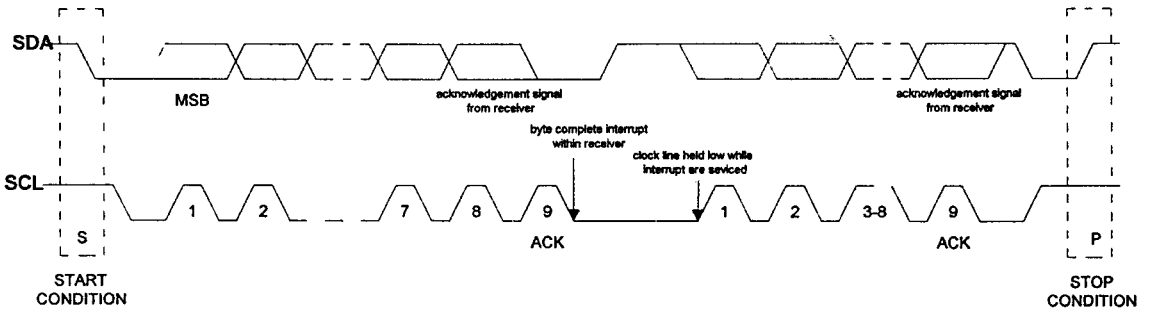
START conditions คือ การที่สาย SDA เปลี่ยนจาก สถานะสูงเป็นสถานะต่ำ ในขณะที่ SCL มีสถานะสูง

STOP conditions คือการที่สาย SDA เปลี่ยนจากสถานะต่ำเป็นสถานะสูง ในขณะที่ SCL มีสถานะสูง START และ STOP conditions จะถูกสร้างโดย ตัวที่เป็นมาสเตอร์เสมอ



รูปที่ 2-4 START และ STOP Condition

2.2.2 การขนส่งข้อมูล (Data Transfer)



รูปที่ 2-5 Data transfer on I^2C - bus

- **รูปแบบของไบต์ข้อมูล (byte format)**

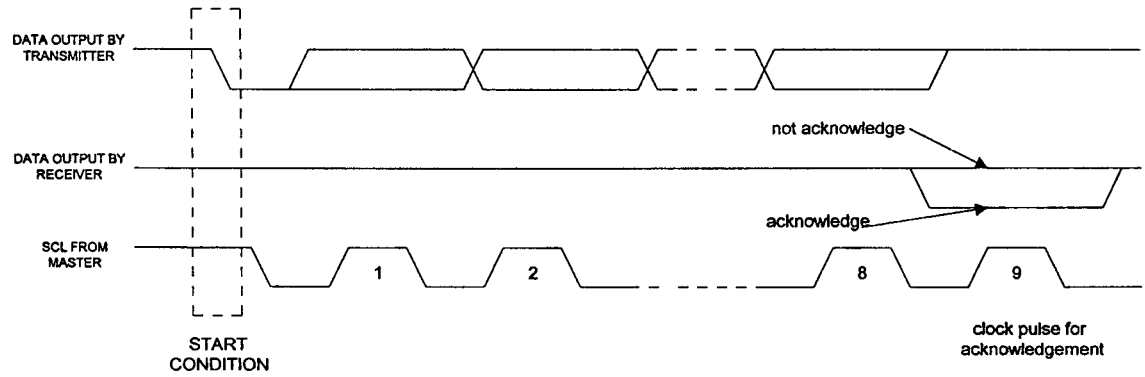
ทุก ๆ ไบต์ บน SDA จะมีความยาว 8 บิท หลังจาก START condition แล้ว 7 บิท ต่อมาจะเป็น แอดเดรส ของอุปกรณ์ที่ตัวมาสเตอร์ต้องการที่จะคุยด้วย บิท ที่ 8 จะบอกทิศทางการไหลของข้อมูลว่าจะเป็นการอ่านหรือการเขียน การส่งข้อมูลแต่ละบิท ต้องใช้สัญญาณนาฬิกา 1 ลูก พอถึงลูกที่ 9 จะต้องมีการตอบทุก ๆ ไบต์ (Acknowledge) การ ACK ของตัวส่งก็คือการปล่อยสาย SDA ให้มีสถานะสูง ในช่วงของสัญญาณนาฬิกาลูกที่จะให้มีการ Acknowledge ถ้าไม่มีการ ACK จะต้องหยุดติดต่อกันทันทีไม่ว่าจะเป็นกรณีที่ ตัวรับที่เป็นสเลฟ (slave receiver) ไม่ตอบรับการคุยด้วย หรือระหว่างที่มีการส่งข้อมูลแล้ว เกิดไม่ได้รับข้อมูลในบางไบต์ (ยกเว้นไบต์สุดท้ายที่จะไม่มีการตอบรับอยู่แล้ว เพื่อเป็นการบอกว่าจะสิ้นสุดการส่งข้อมูล) แต่ถ้าเป็นตัวส่งที่เป็นสเลฟ (slave-transmitter) มันจะต้องปล่อยสาย SDA เพื่อที่จะให้ มาสเตอร์ สร้าง STOP condition หรือ repeated START condition

- **การตอบรับ (Acknowledge)**

ตัวที่เป็นตัวส่ง (Transmitter) จะปล่อยสาย SDA ให้มีสถานะสูง ระหว่างที่มีการ Acknowledge ตัวรับ (Receiver) จะดึงสาย SDA ให้มีสถานะต่ำ ระหว่างที่ SCL มีสถานะสูง ซึ่งจะทำให้เสียเวลาในช่วงนี้ไป

เมื่อตัวรับที่เป็นสเลฟไม่ตอบ Acknowledge ไม่ว่ากรณีใด ๆ เช่น ตัวรับที่เป็นสเลฟไม่พร้อมที่จะคุยกับมาสเตอร์ หรือไม่ได้รับข้อมูลที่มาสเตอร์ส่งไปให้ เป็นต้น สาย SDA จะถูกปล่อยให้มีสถานะสูง เพื่อให้มาสเตอร์ สร้าง STOP Condition เพื่อยกเลิกการติดต่อ

ในกรณีที่เป็นตัวส่งที่เป็นมาสเตอร์ (Master-Receiver) ในไบต์สุดท้ายจะไม่ตอบ Acknowledge ไปยังตัวรับที่เป็นสเลฟ แต่จะสร้าง STOP หรือ repeated START Condition แทน

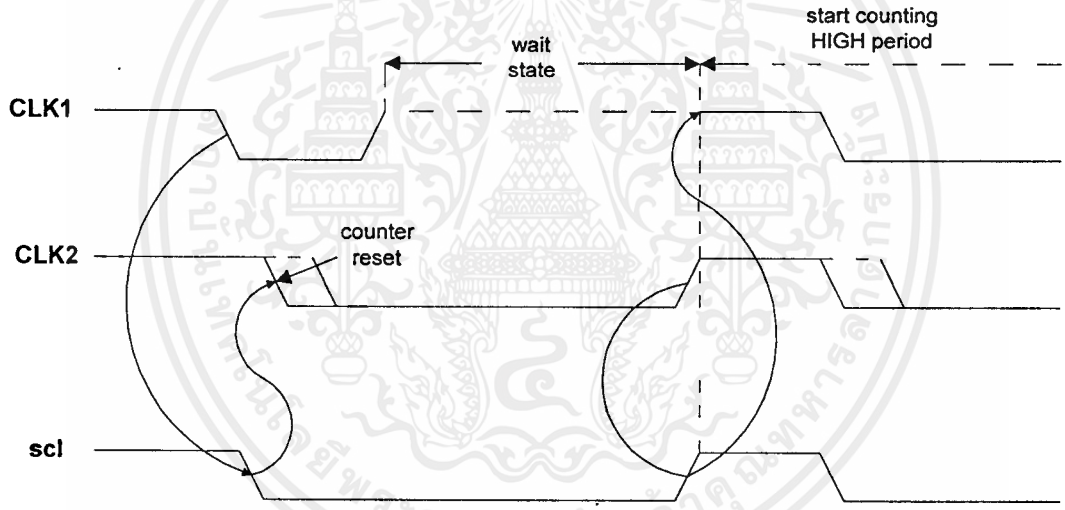


รูปที่ 2-6 Acknowledge on the I²C bus

2.2.3 การแข่งขันและการสร้างสัญญาณนาฬิกา (Arbitration and Clock Generation)

- **การซิงโครไนซ์ (Synchronization)**

มาสเตอร์ ทุกตัวเป็นตัวสร้างสัญญาณนาฬิกา ซึ่งการซิงโครไนซ์ของสัญญาณนาฬิกาของ มาสเตอร์ แต่ละตัวจะอธิบายได้ดังรูป



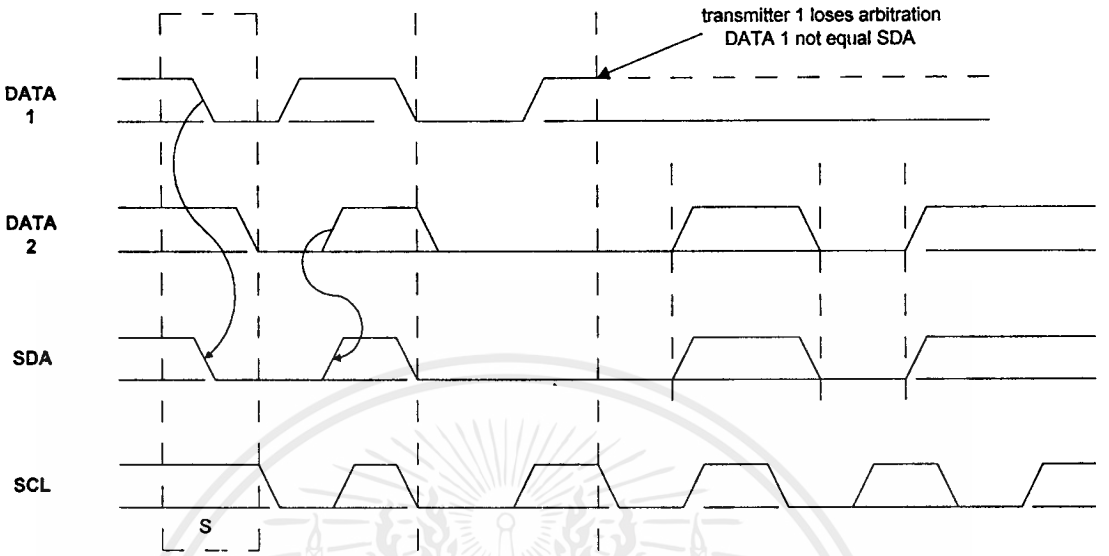
รูปที่ 2-7 Clock Synchronization the arbitration procedure

เมื่อ CLK1 มีสถานะต่ำ ก็จะดึงสาย SCL ที่มีสถานะสูง ให้มีสถานะต่ำด้วยแล้วเริ่มนับเวลาของคาบที่มีสถานะต่ำ ต่อมาเมื่อ CLK2 มีสถานะต่ำ ในขณะที่ SCL มีสถานะต่ำอยู่แล้ว มันก็ยังคงมีสถานะต่ำต่อไป เมื่อ CLK1 มีสถานะสูง แล้วแต่ CLK2 ยังมีสถานะต่ำอยู่ SCL ก็ยังคงมีสถานะต่ำต่อไป จนกว่า CLK2 จะมีสถานะสูง จึงจะเริ่มนับเวลาของคาบที่มีสถานะสูง ช่วงที่ CLK1 รอ CLK2 มีสถานะสูง จะเรียกว่า สถานะรอ (Wait state)

เพราะฉะนั้น SCL จะมีสถานะต่ำนานตามสัญญาณนาฬิกาของอุปกรณ์ที่มีคาบ 0 นานที่สุด และจะมีสถานะสูงตามสัญญาณนาฬิกาของอุปกรณ์ที่มีคาบเป็น 1 สั้นที่สุด

- **การแข่งขัน (Arbitration)**

การแข่งขันจะเกิดขึ้นในสาย SDA ขณะที่สาย SCL มีสถานะสูง ตามรูปที่ 2-8



รูปที่ 2-8 Arbitration procedure of two masters

ให้ DATA1 และ DATA2 เป็นของ มาสเตอร์1 และ มาสเตอร์2 ตามลำดับ โดยเมื่อ มาสเตอร์1 ส่ง START condition มาก็จะทำให้ SDA มีสถานะต่ำ และในขณะที่ SCL ยังมีสถานะสูงอยู่นั้น มาสเตอร์ ก็ส่ง START condition ของตัวเองมา ต่อมาทั้งสองตัวก็จะส่ง แอดเดรส (7 บิท) ของอุปกรณ์ที่จะคุยด้วย ถ้าอุปกรณ์ที่จะคุยด้วยเป็นคนละตัวและ เกิดมีบิท 1 โดยที่อีกตัวเป็นบิท 0 มาสเตอร์ที่ส่ง บิท 1 ก่อนก็จะแพ้ไป มาสเตอร์ที่แพ้จะสามารถส่งข้อมูลในไบต์ที่แพ้จนหมดและจะสร้างสัญญาณนาฬิกาได้จนถึงบิทสุดท้ายของไบต์ที่แพ้แล้วจึงหยุดส่งข้อมูลและสร้างสัญญาณนาฬิกาทันที แต่ถ้าปรากฏว่าแอดเดรสของอุปกรณ์ที่มาสเตอร์ทั้งคู่จะคุยด้วยเป็นตัวเดียวกัน การแข่งขันก็จะต้องมีต่อไปเรื่อย ๆ โดยจะแข่งกันแบบ บิทต่อบิท (bit-by-bit) ในช่วงที่ SCL มีสถานะเป็นสูงด้วย

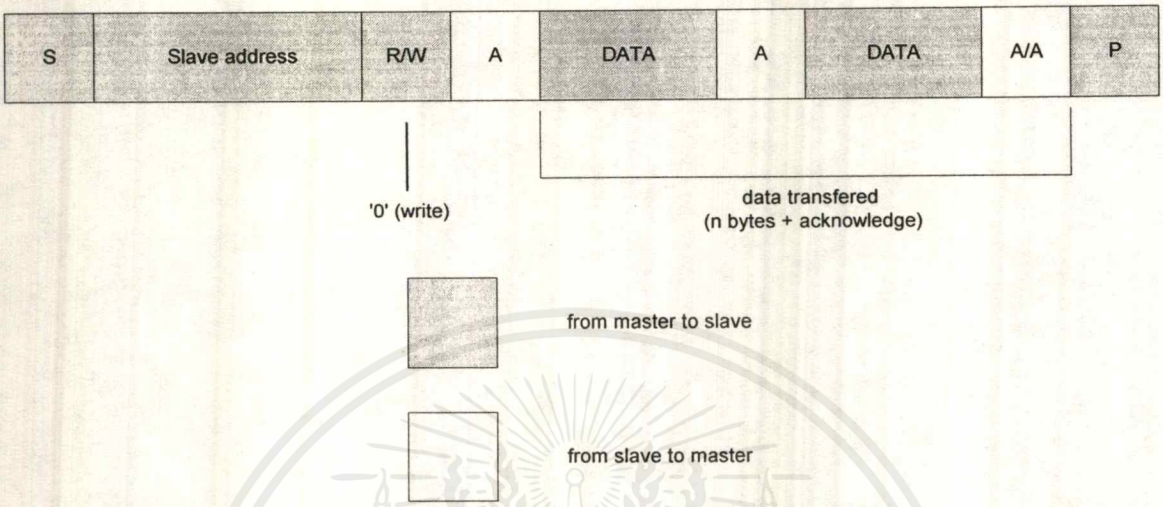
ถ้ามาสเตอร์ที่ชนะยังต้องการที่จะส่งข้อมูลอีก มันก็จะส่งสัญญาณ Repeated START conditions (Sr) ก่อน แล้วจึงติดต่อกับอุปกรณ์ที่ต้องการจนเสร็จแล้วจึงส่ง STOP condition

- **Formats with 7-bits addresses**

หลังจาก START Condition แล้วจะเป็น แอดเดรส ของตัวสเลฟ ซึ่งมีขนาด 7 บิท แล้วตามด้วยบิทที่ 8 ซึ่งจะบอกทิศทางการไหลของข้อมูล (R / $\overline{W}$) - " 0 " เป็นการเขียนข้อมูล ส่วน " 1 " เป็นการอ่านข้อมูล ในการส่งข้อมูล มาสเตอร์จะต้องสร้าง STOP Condition เสมอเมื่อต้องการเลิกการติดต่อ แต่ถ้ามาสเตอร์ยังต้องการที่จะใช้บัสอยู่ ก็จะส่ง repeated START แล้วตามด้วยแอดเดรสของสเลฟ อีกตัวหนึ่ง ก่อนที่จะสร้าง STOP Condition

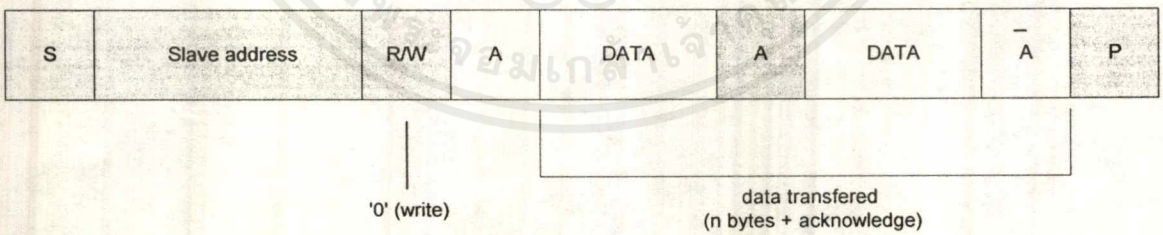
2.2.4 รูปแบบของการส่งข้อมูลที่เป็นไปได้ได้แก่

- **ตัวส่งที่เป็นมาสเตอร์ส่งข้อมูลไปยังตัวรับที่เป็นสเลฟ (Master-transmitter transmits to slave-receiver)** ทิศทางการส่งข้อมูลไม่มีการเปลี่ยนแปลง



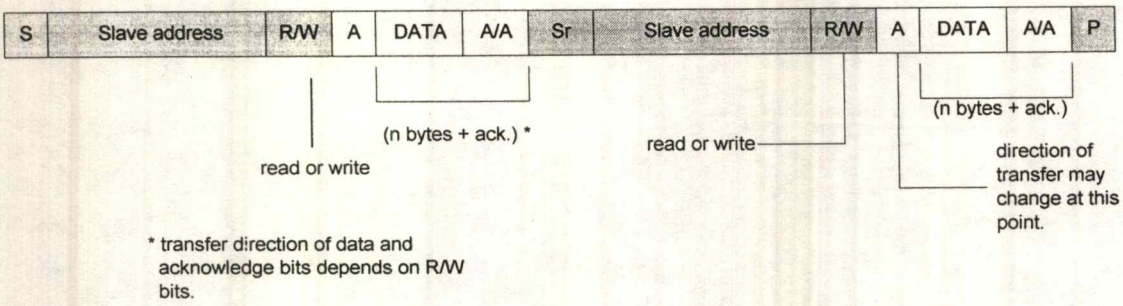
รูปที่ 2-9 A master-transmitter addresses a slave receiver with a 7-bit address

- **ตัวมาสเตอร์อ่านตัวสเลฟทันทีหลังจากที่เกิดไบต์แรก (Master reads slave immediately after first byte)** เมื่อมีการ Acknowledge ครั้งแรกตัวส่งที่เป็นมาสเตอร์ จะกลายเป็นตัวรับที่เป็นมาสเตอร์ และตัวรับที่เป็นสเลฟ จะกลายเป็นตัวส่งที่เป็นสเลฟ การ Acknowledge ยังคงถูกสร้างโดยสเลฟ ส่วน มาสเตอร์ จะสร้าง STOP Condition



รูปที่ 2-10 A master reads a slave immediately after the first byte

- **รูปแบบผสม (Combined format)** ในระหว่างที่มีการเปลี่ยนทิศทางการส่งข้อมูล ทั้ง START Condition และ แอดเดรส ของสเลฟจะถูกส่งไปอีกครั้ง แต่ R / $\overline{W}$ จะกลับกัน



รูปที่ 2-11 Combined format

2.3 ทฤษฎีเบื้องต้นของ VHDL

2.3.1 VHDL คืออะไร

VHDL ย่อมาจาก Very high speed integrated circuits Hardware Description Language เป็นภาษาที่ใช้อธิบายฮาร์ดแวร์ ซึ่งสามารถถูกใช้เพื่อจำลองระบบดิจิตอล (digital) ที่มีช่วงของระดับ algorithmic จนถึงระดับเกท (gate) อยู่หลายระดับ ความซับซ้อนของระบบดิจิตอลที่ถูกออกแบบ สามารถที่จะแปรเปลี่ยนจากเกท ง่าย ๆ ไม่ซับซ้อน จนถึงระบบอิเลกทรอนิกส์ดิจิตอล (digital electronics) ที่สมบูรณ์ ระบบดิจิตอลสามารถอธิบายได้เป็นลำดับขั้นตอน ไทม์มิง (timing) ก็สามารถที่จะจำลองได้อย่างชัดเจน

ภาษา VHDL สามารถถูกมองว่าเป็นการรวมกันของภาษาเหล่านี้คือ :=
sequential language + concurrent language + net-list language + timing specifications + waveform generation language => VHDL

นอกจากนี้ภาษา VHDL ยังสามารถแสดงการทำงานพร้อมกัน (concurrent) หรือ เป็นลำดับ (sequential behaviour) ของระบบดิจิตอลโดยจะมีหรือไม่มีไทม์มิง ได้อย่างชัดเจน และสามารถจำลองระบบที่มีการเชื่อมต่อของส่วนประกอบต่างๆ (components) ได้

ตัวภาษา VHDL ไม่ได้กำหนดเพียงแค่ไวยากรณ์ (syntax) เท่านั้น แต่ยังกำหนดการจำลอง (simulate) ได้ด้วย เพราะฉะนั้นการเขียนแบบจำลองด้วย VHDL จึงสามารถที่จะทดสอบได้โดยใช้ VHDL simulator

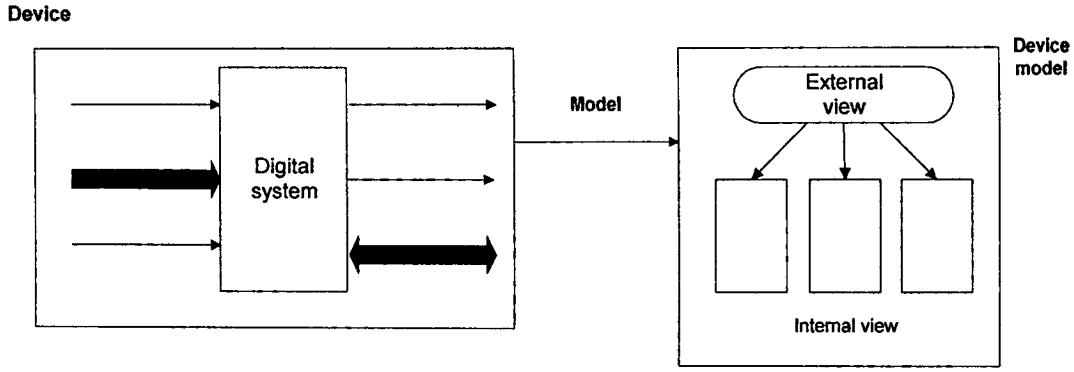
2.3.2 ความสามารถของภาษา VHDL

1) ใช้เป็น Exchange medium ระหว่างตัวแทนจำหน่ายชิป (chip venders) และผู้ใช้เครื่องมือ (tools users) โดยตัวแทนจำหน่าย สามารถใช้ภาษา VHDL อธิบายการทำงานของส่วนประกอบแต่ละส่วน ให้กับผู้ออกแบบระบบได้ และผู้ใช้ CAD ก็สามารถใช้ VHDL capture behaviour ของการออกแบบในระดับสูง (high level) ของการจำลองการทำงานต่าง ๆ ได้
2) ใช้เป็นตัวกลางในการสื่อสาร (communication medium) ระหว่าง CAD และ CAE ที่ต่างกันได้
3) ภาษา VHDL สนับสนุนการออกแบบที่เป็นลำดับขั้นตอน (hierarchy) นั่นคือระบบดิจิตอลสามารถจำลองเป็นชุดขององค์ประกอบ ที่เชื่อมต่อกันได้ และแต่ละองค์ประกอบ ก็สามารถถูกจำลองเป็นชุดขององค์ประกอบย่อย (sub-component) ที่เชื่อมต่อกันภายใน

4) ภาษา VHDL เป็นวิธีการออกแบบที่มีความยืดหยุ่น (flexible design methodologies) คือ จะออกแบบเป็น top-down, bottom-up หรือ ผสมกันก็ได้
5) ใช้ได้กับเทคโนโลยีทางด้านฮาร์แวร์หลายประเภท
6) ไทม์มิงเป็นทั้งซิงโครนัส (Synchronous) และ อะซิงโครนัส (Asynchronous)
7) เทคนิคต่าง ๆ ในทางดิจิตอล ใช้ได้กับภาษา VHDL เช่น finite - state machine model, algorithmic descriptions และ Boolean equation เป็นต้น
8) มีความเป็น proprietary คือมนุษย์ก็สามารถอ่านได้เข้าใจ เครื่องก็สามารถอ่านได้
9) อยู่ในมาตรฐานของ IEEE และ ANSI
10) ภาษา VHDL จะใช้รูปแบบในการอธิบายได้ทั้งเป็นโครงสร้าง (structural) , การไหลของข้อมูล (dataflow) และ เป็น บิแฮฟวิเออร์ (behavioural) หรือจะใช้ทั้งสามรูปแบบร่วมกันก็ได้
11) ใช้ออกแบบได้ตั้งแต่ระดับบิแฮฟวิเออร์จนถึงระดับเกท
12) สามารถจำลองการออกแบบระบบขนาดใหญ่ได้
13) ในภาษา VHDL มีส่วนประกอบ (element) ต่าง ๆ ที่ช่วยในการออกแบบระบบขนาดใหญ่ให้ง่ายขึ้น เช่น component, function, procedure และ package เป็นต้น
14) ในการจำลอง (simulation) ไม่จำเป็นต้องเรียนรู้ภาษาตัวใหม่ เนื่องจากสามารถเขียนเป็น Testbench ในรูปของภาษา VHDL ได้เลย
15) nominal เวลาประวิงการแพร่กระจาย (propagation delays), min-max delays, เวลาจัดเตรียม (set-up time) และ เวลาคงไว้ (hold timing), เงื่อนไขบังคับเรื่องเวลา (timing constraint) และ spike detection สามารถอธิบายได้โดยใช้ภาษา VHDL
16) การใช้แบบจำลอง (model) ที่อยู่ในรูปของ generic และลักษณะเฉพาะ (attribute) ทำให้ง่ายต่อการ back-annotation ของข้อมูลแบบสถิต (static information) เช่น ไทม์มิง หรือ placement information
17) generic และ ลักษณะเฉพาะ ยังใช้ประโยชน์ในเรื่องของการอธิบายการออกแบบแบบพารามิเตอร์ (parameterized designs)
18) แบบจำลองของภาษา VHDL ไม่ใช่เป็นการอธิบายถึงหน้าที่ของการออกแบบเท่านั้น แต่ยังอธิบายถึงข้อมูลที่เกี่ยวกับ ตัวที่ได้ออกแบบ ในรูปแบบของลักษณะเฉพาะ ที่กำหนดโดยผู้ใช้ (user) เช่น แอเรียทั้งหมด (Total area) และ ความเร็ว เป็นต้น

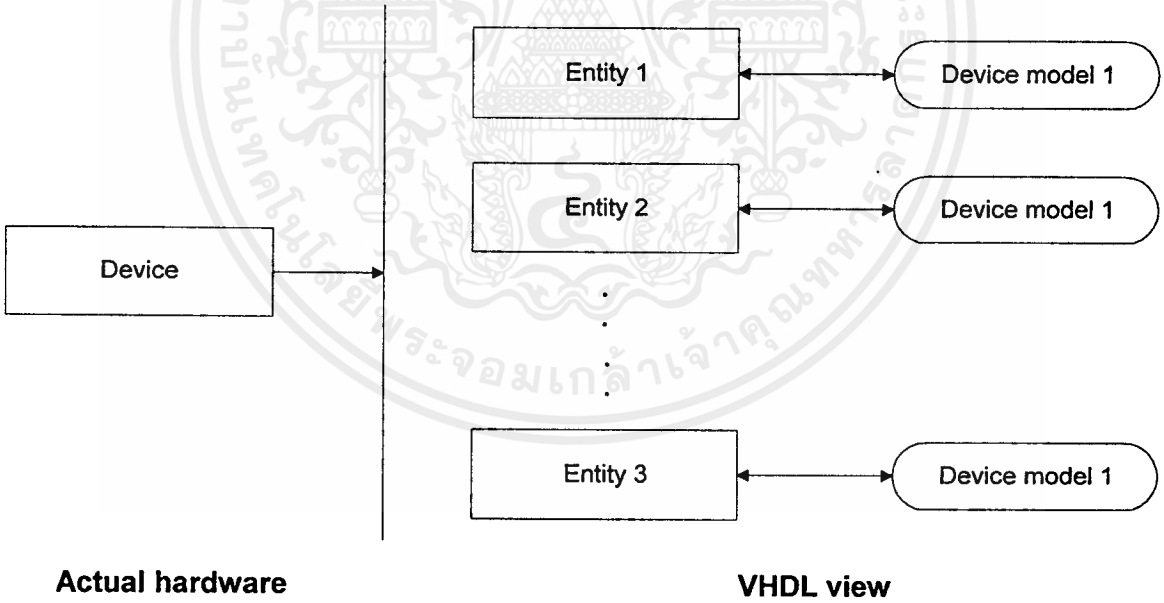
2.3.3 HARDWARE ABSTRACTION

VHDL ถูกใช้อธิบายแบบจำลองของอุปกรณ์ฮาร์แวร์ทางดิจิตอล (digital hardware device) ซึ่งแบบจำลองนี้จะกำหนดมุมมองภายนอก (external view) และมุมมองภายใน (internal view) โดยที่มุมมองภายใน จะใช้กำหนดหน้าที่และโครงสร้างของอุปกรณ์ ส่วนมุมมองภายนอกจะกำหนดการเชื่อมต่อของอุปกรณ์ กับ แบบจำลองตัวอื่น ๆ รูปที่ 2-12 แสดงอุปกรณ์ฮาร์แวร์ และแบบจำลองของซอฟท์แวร์ (software) ที่เกี่ยวข้องกับมัน



รูปที่ 2-12 Device versus device model

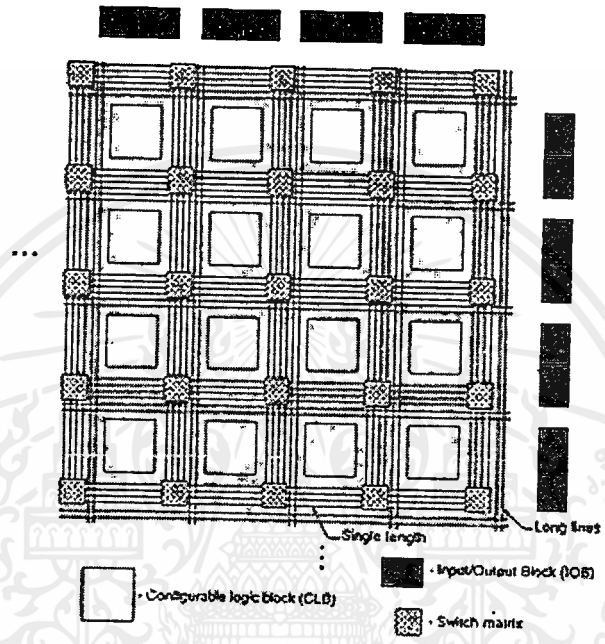
ความสัมพันธ์ของแบบจำลองแต่ละตัวที่ทำการแทนข้อมูล (map) ไปอีกแบบจำลองหนึ่งอยู่ในรูปของ one to many นั่นคือ อุปกรณ์ 1 ตัว สามารถมีแบบจำลองของอุปกรณ์ได้หลายแบบจำลอง ใน VHDL แต่ละแบบจำลองของอุปกรณ์ จะแสดงอยู่ในรูปแบบที่ไม่เหมือนกัน คือ มีความเป็นอุปกรณ์เดี่ยว (unique device) ซึ่งเรียกว่า เอนติตี (entity) รูปที่ 2-13 แสดงมุมมองของ VHDL ของ อุปกรณ์ฮาร์ดแวร์ ซึ่งมีแบบจำลองของอุปกรณ์อยู่หลายตัว ซึ่งแบบจำลองของอุปกรณ์แต่ละตัวก็จะมี 1 เอนติตี



รูปที่ 2-13 a VHDL view of a device

เนื่องจากในโครงงานนี้ได้ใช้ FPGA ช่วยในการทดสอบระดับฮาร์ดแวร์ โดย FPGA ที่ใช้เป็นของ Xilinx ตระกูล XC4000 ดังนั้นจึงขอกล่าวถึงทฤษฎีของ Xilinx XC4000 โดยคร่าว ๆ ดังหัวข้อ 2.4

2.4 โครงสร้างของ Xilinx XC4000



รูปที่ 2-14 Xilinx XC4000 FPGA Structure

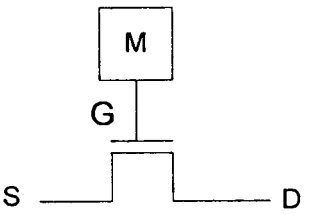
โครงสร้างภายในของ Xilinx XC4000 FPGA เป็นดังรูปที่ 2- 14 ลอจิก (Logic) ภายใน FPGA จะถูกสร้างในกลุ่มของแถวลำดับ (array blocks) ซึ่งเรียกว่า Configurable logic blocks (CLBs) ส่วน อินพุท และ เอาท์พุท จะถูกจัดการโดย Input/Output blocks (IOBs) ซึ่งจะอยู่บริเวณขอบของแถวลำดับ

การเชื่อมต่อ CLBs และ IOBs สามารถถูกโปรแกรมได้ในลักษณะของเส้นทางจากบล็อกหนึ่งไปอีกบล็อกหนึ่ง โดยการใช้อาร์เรย์ของเมตริกซ์ของสวิตช์ (switch matrices)

Xilinx ใช้ SRAM ในการเก็บข้อมูล เมื่อมีการป้อนกระแสให้กับตัว Xilinx ตัวโปรแกรมที่ได้ป้อนเข้าไปจะถูกโหลด (load) เข้าไปเก็บไว้ใน SRAM เมื่อโปรแกรมถูกโหลดเข้าไปแล้ว FPGA จะเปลี่ยนจากโหมดการโปรแกรม (programming mode) มาเป็นโหมดการทำงาน (operation mode) แล้วค่าต่างๆ จะยังคงอยู่จนกว่าจะมีการโปรแกรม FPGA ใหม่หรือไม่จ่ายกระแสไฟให้กับตัว FPGA

การเก็บค่าทางลอจิกของ SRAM ใน FPGA ทำได้ 3 วิธี ได้แก่

- การควบคุมโดยผ่านตัวทรานซิสเตอร์ (pass transistor control)
- การควบคุมโดยใช้ตัวมัลติเพล็กซ์ (multiplexer control)
- การทำตารางเปิดดู (look up table implementation)

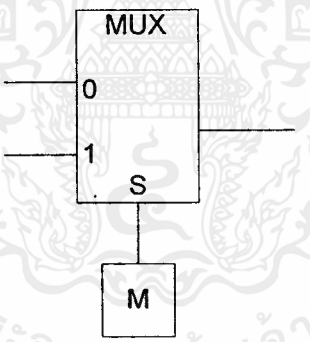


รูปที่ 2-15 การควบคุมโดยผ่านตัวทรานซิสเตอร์

- **การควบคุมโดยผ่านตัวทรานซิสเตอร์**

เมื่อ SRAM เก็บลอจิกที่เป็น ' 0 ' ตัวทรานซิสเตอร์จะปิด (OFF) เส้นทางระหว่าง S (Source) กับ D (Drain) ก็จะเปิด (OPEN) ทำให้ไม่มีสัญญาณผ่านจาก S ไปยัง D เมื่อ SRAM เก็บลอจิกที่เป็น ' 1 ' ตัวทรานซิสเตอร์จะเปิด (ON) เส้นทางระหว่าง S และ D ก็จะถูกปิด (CLOSED) ทำให้สัญญาณ ผ่านระหว่าง 2 ส่วน (segment) นี้ได้

- **การควบคุมโดยใช้ตัวมัลติเพล็กซ์**

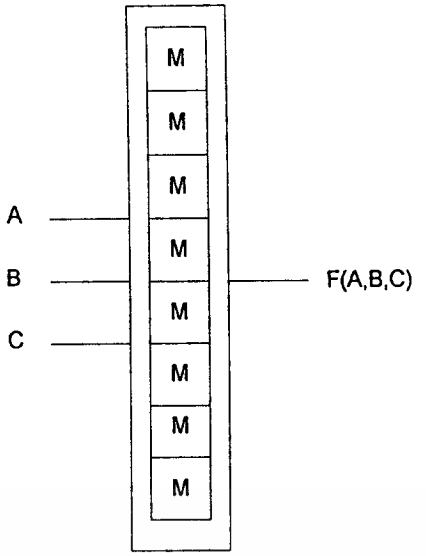


รูปที่ 2-16 การควบคุมโดยใช้ตัวมัลติเพล็กซ์

SRAM จะต่อกับขา S (Select input) ของตัวมัลติเพล็กซ์ชนิด 2-to-1 (2-to-1 multiplexer) ถ้า SRAM เป็น ' 0 ' ค่าที่ขา อินพุท 0 ของ ตัวมัลติเพล็กซ์จะถูกส่งไปยังขา เอาท์พุท ของตัวมัลติเพล็กซ์ ถ้า SRAM เก็บค่า ' 1 ' ค่าที่จะออกไปที่ขา เอาท์พุท จะเป็นค่าที่อยู่บนขา อินพุท 1

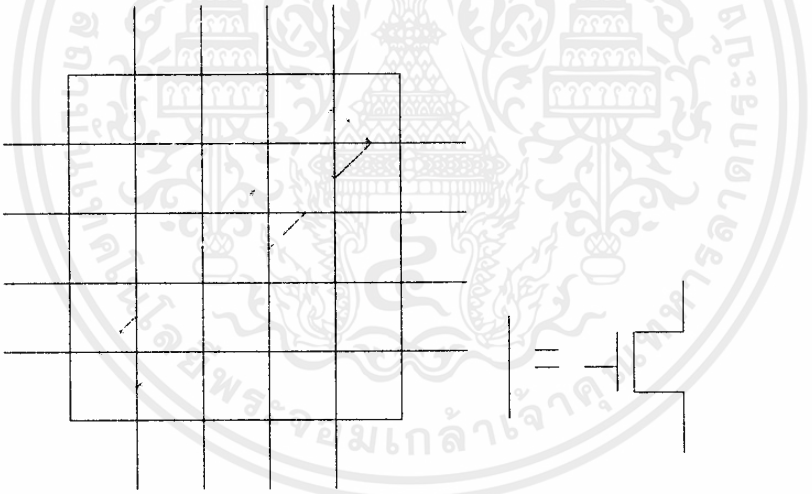
- **การทำตารางเปิดดู**

วิธีนี้จะนำ SRAM มาทำเป็น look up table ซึ่งใน SRAM จะเก็บตารางค่าความจริง (truth table) ของฟังก์ชั่น ดังนั้นในแต่ละ cell จะเก็บค่าของ ฟังก์ชั่น F ในรูปของ min-term ตัวตารางเปิดดูจะทำงานคล้ายกับตัวมัลติเพล็กซ์ ดังแสดงในรูปที่ 2-17



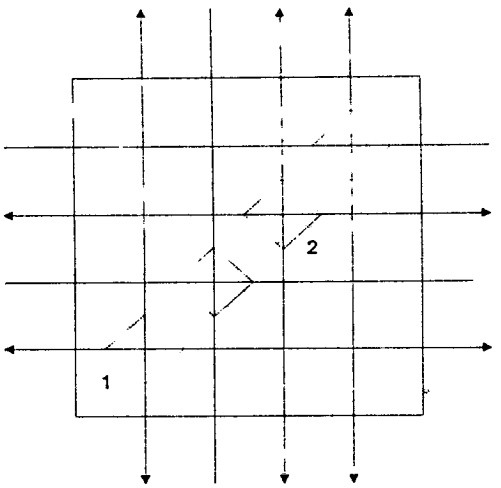
รูปที่ 2-17 การทำตารางเปิดดู

2.4.1 การเชื่อมต่อกันของ Xilinx



รูปที่ 2-18 Switch Matrix Transistor

การเชื่อมต่อระหว่าง CLBs กับ CLBs และ CLBs กับ IOBs ทำได้โดยการนำเซกเมนต์ (segment) ทั้งแนวตั้งและแนวนอนที่อยู่ระหว่างบล็อกมาเชื่อมต่อกัน เซกเมนต์ที่มีความห่างกันมากจะเรียกว่า long lines ซึ่งแสดงดังในรูปที่ 2-13 ส่วนเซกเมนต์ที่เหลือจะมีความยาวที่กระจายไปแค่ CLBs เดียว ดังนั้นเซกเมนต์ที่มีความห่างกันไม่มากนี้จะสามารถเชื่อมต่อกันได้โดยใช้เมตริกซ์ของสวิตช์



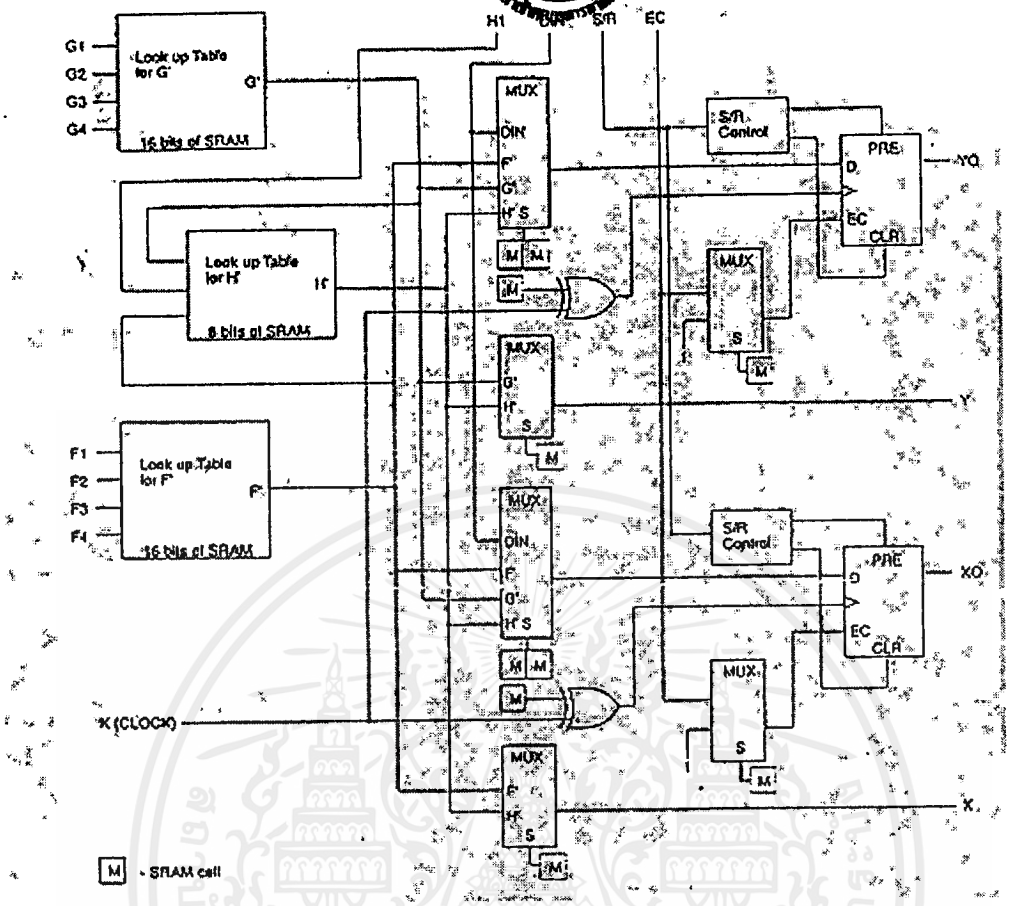
รูปที่ 2-19 ตัวอย่างการเชื่อมต่อ

2.4.2 ลอจิกของ Xilinx

วงจรทาง ลอจิก ของ Xilinx FPGAจะทับกันอยู่ใน CLBs และ IOBs ซึ่งทั้งสองส่วนนี้สามารถทำการโปรแกรมได้ และมีความซับซ้อน

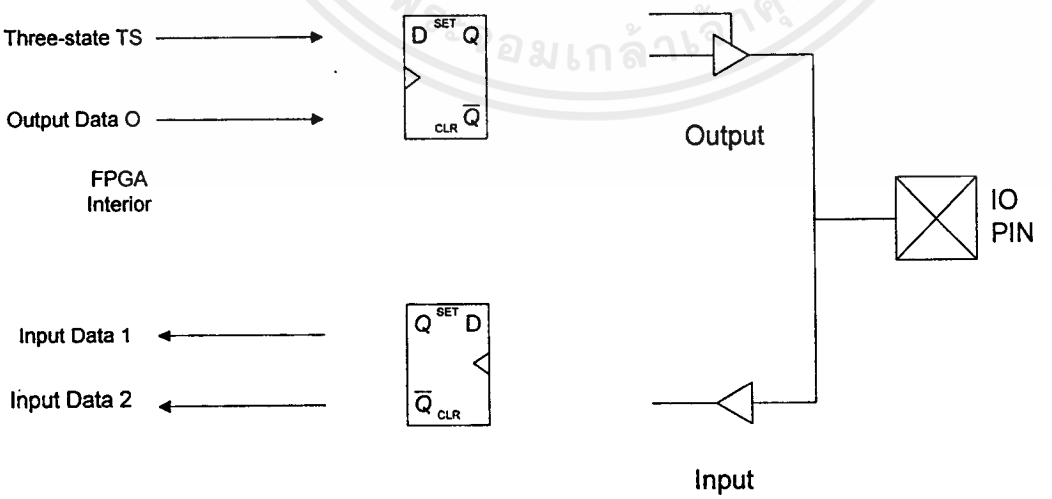
จากรูป 2-20 แสดงแผนภาพ (diagram) อย่างง่าย ๆ ของ CLBs ใน Xilinx โดย CLB จะมี อินพุท ทั้งหมด 13 อินพุท (รวมขา clock) มีตารางเปิดดูอยู่ 3 ตาราง โดย 2 ตารางจะใช้สร้างฟังก์ชั่นระหว่างอินพุท ทั้ง 4 ตัวที่เข้ามา ซึ่งจะได้ ฟังก์ชั่น มา 2 ฟังก์ชั่น คือ F' และ G' แล้ว ฟังก์ชั่น ที่ได้จากทั้ง 2 บล็อก นี้จะถูกป้อนกลับไปเป็นอินพุทให้กับตารางเปิดดูอีกตารางหนึ่ง ซึ่งตารางเปิดดูตารางสุดท้ายจะสร้างฟังก์ชั่นระหว่างตัวแปร 3 แปรได้แก่ F', G' และ H1 แล้วจะได้ เอาท์พุท เป็น ฟังก์ชั่น H' ซึ่งถ้าทำการกำหนดค่าของตัวมัลติเพล็กซ์ ให้เหมาะสม เอาท์พุท X และ Y ที่ได้จะได้มาจาก ฟังก์ชั่น F' , G' หรือ H' คู่ใดคู่หนึ่ง

ดี-ฟลิปฟลอป (D flip-flops) 2 ตัวใช้ในการขับค่า (drive) XQ และ YQ ได้โดยตรง ซึ่ง อินพุท ของ ดี-ฟลิปฟลอป จะได้มาจากตัวมัลติเพล็กซ์ 2 ตัว ซึ่งตัวมัลติเพล็กซ์ก็จะเลือกมาจาก F', G' , H' หรือ DIN ตัว มัลติเพล็กซ์จะเลือกมาเพียงฟังก์ชั่นเดียวเท่านั้นที่จะมาขับให้ฟลิปฟลอปทำงาน ในกรณีที่ไม่ได้เลือกฟังก์ชั่นใดให้เป็นอินพุทของดี-ฟลิปฟลอป ขา DIN จะถูกเลือกให้เป็นอินพุทของดี-ฟลิปฟลอปได้โดยตรง ดังนั้นจึงสามารถเก็บค่าที่มาจากต้นกำเนิดตัวอื่น ๆ ได้ XOR 2 ตัวใช้เป็นตัวเลือกว่าจะให้ฟลิปฟลอปทำงาน ที่ ขอบขาขึ้นหรือขอบขาลง (positive or negative edge triggered) ตัว S/R control จะควบคุมให้ ฟลิปฟลอป จะทำการ SET หรือ RESET อะซิงโครนัส ซึ่งจะขึ้นอยู่กับผลของ Global SET/Reset ในสถานะของฟลิปฟลอปด้วย ตัวมัลติเพล็กซ์ 2 ตัวที่เหลือ ใช้เป็นตัวเลือก (option) ของการทำงาน (enable) ของสัญญาณที่ให้กับฟลิปฟลอปแต่ละตัว



รูปที่ 2-20 Simplified Diagram of a Xilinx Configurable Logic Block

ต่อไปจะกล่าวถึง IOBs ส่วน IOB ของ Xilinx นี้ก็สามารถโปรแกรมได้เช่นเดียวกันกับส่วนของ CLBs



รูปที่ 2-21 โครงสร้าง IOB ของ Xilinx

ส่วน เอาท์พุท ของ IOB จะให้ output data O ซึ่งมาจากภายในของ FPGA และอีกทางเลือกหนึ่งคือการให้ output data O ที่มาจากฟลิปฟลอป ตัว Three-state driver ทางด้าน เอาท์พุท จะใช้เลือกว่าจะให้ IO PIN ทำหน้าที่เป็น อินพุท , เอาท์พุท หรือจะเป็นทั้งอินพุท/เอาท์พุท

ส่วนของ อินพุท ของ IOB สัญญาณที่ IO PIN จะเข้าไปในบัฟเฟอร์ข้อมูลออก (input buffer) สัญญาณนี้สามารถป้อนให้ Input Data 1 และ Input Data 2 ได้โดยตรง ซึ่ง อินพุท ทั้งสองนี้จะอยู่ภายใน FPGA หรือ Input Data จะได้มาจาก ค่าที่เก็บไว้ใน IO PIN หรือคอมพลีเมนต์ (complement) ของมัน

บทที่ 3

หลักการออกแบบ

3.1 ขั้นตอนในการออกแบบ

- **Preliminary Design**
  - Development Plan วางแผนในการออกแบบ
  - Design Partition แบ่งการออกแบบเป็นส่วน ๆ
  - Design Specification ระบุการทำงานที่คิดจะออกแบบของแต่ละส่วน
- **Test Plan**
  - Test Strategy วางแผนการทดสอบ
  - Test Design อธิบายวิธีการตรวจสอบ
  - Test system design ทำการออกแบบระบบที่ตรวจสอบ
- **RTL Block Coding**
  - เขียน code ภาษา VHDL สำหรับแต่ละ บล็อก ที่ได้ทำการแบ่งไว้แล้ว ของ วงจร แต่ละตัว
- **Test program design**
  - เขียน code ภาษา VHDL เพื่อตรวจสอบทั้งระบบ
- **System Simulation**
  - ทำการจำลองและตรวจสอบทุกบล็อกว่าทำงานได้ถูกต้อง
- **VHDL Synthesis**
  - ทำการสังเคราะห์ VHDL code ของแต่ละบล็อกให้เป็น logic gate schematic
- **Gate level simulation**
  - ทำการจำลองและ ตรวจสอบ schematic ที่สังเคราะห์แล้ว
- **System Test**
  - วิเคราะห์ Gate delay และ static timing และทำ FPGA mapping
  - นำ FPGA มาต่อกับระบบที่จะทดสอบเพื่อทำการตรวจสอบการทำงาน

3.2 ขั้นตอนในการออกแบบวงจร

ได้แบ่งการออกแบบเป็น 3 ส่วน ใหญ่ ๆ ได้แก่

- ออกแบบ Universal Asynchronous Receiver / Transmitter (UART)
- ออกแบบ Memory Interface with I2C-bus
- ออกแบบ Microcontroller Interface with I2C-bus

ในการออกแบบ UART และ Microcontroller Interface with I2C-bus นั้น ได้อาศัยทฤษฎีเบื้องต้นมาช่วยทำการออกแบบ นอกจากนั้นในการออกแบบบางส่วนของวงจร ได้ทำการเลียนแบบจากชิปที่มีอยู่แล้ว โดย UART ได้ใช้ MC6850 ACIA และ UART ของ MCS51 Microcontroller เป็นต้นแบบ ส่วน Microcontroller Interface ได้ใช้ PCF8584 ส่วนการออกแบบ Memory Interface with I2C-bus นั้นไม่ได้เลียนแบบจากวงจรที่มีอยู่แล้ว ได้ทำการออกแบบโดยหลักการและทฤษฎีเบื้องต้นของ I2C-bus เพียงอย่างเดียว ต่อไปเราจะมาดูรายละเอียดของการออกแบบแต่ละตัวว่าเป็นเช่นไร

3.2.1 ขั้นตอนการออกแบบ Universal Asynchronous Receiver / Transmitter (UART)

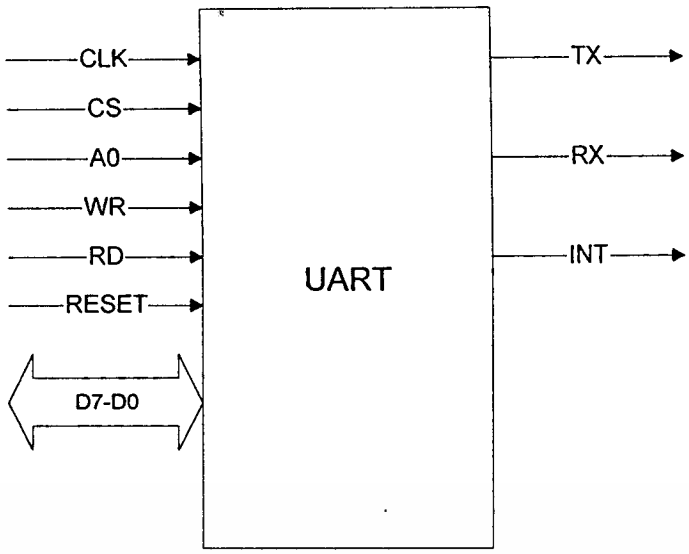
สำหรับการออกแบบ UART นั้น จากที่ได้กล่าวมาข้างต้นแล้วว่าได้เลียนแบบจาก MC6850 ACIA และ UART ใน MCS51 Microcontroller โดยจะมีสิ่งที่แตกต่างและสิ่งที่เหมือนกันดังรายละเอียดต่อไปนี้

- Block Diagram จะมีเหมือนกับ UART ทั่ว ๆ ไปดังรายละเอียดที่จะได้กล่าวต่อไป
- ไม่มีความสามารถในการติดต่อกับ peripheral device อื่น ๆ ซึ่งจะเหมือนกับของ MCS51
- ใน Control Register จะมีการทำงานในแต่ละบิทคล้ายกับ MC6850 ACIA แต่เนื่องจากไม่มีการติดต่อกับ peripheral device อื่น ๆ จึงไม่มี Control bit ที่ต้องควบคุมส่วนนี้ และใน Word select bits จะกำหนดการส่ง - รับข้อมูลขนาด 8 บิทเท่านั้น และที่แตกต่างกันอีกคือใน Counter divide/Master reset bits จะกำหนดค่าที่จะใช้หารเป็นสัญญาณนาฬิกาคือ 16 , 32 และ 64
- ใน Status Register จะไม่มีบิทที่ใช้รายงานสถานะของโมเด็ม คือ บิทที่ 2 และ 3 นอกนั้นจะเหมือนกับใน MC6850 ACIA

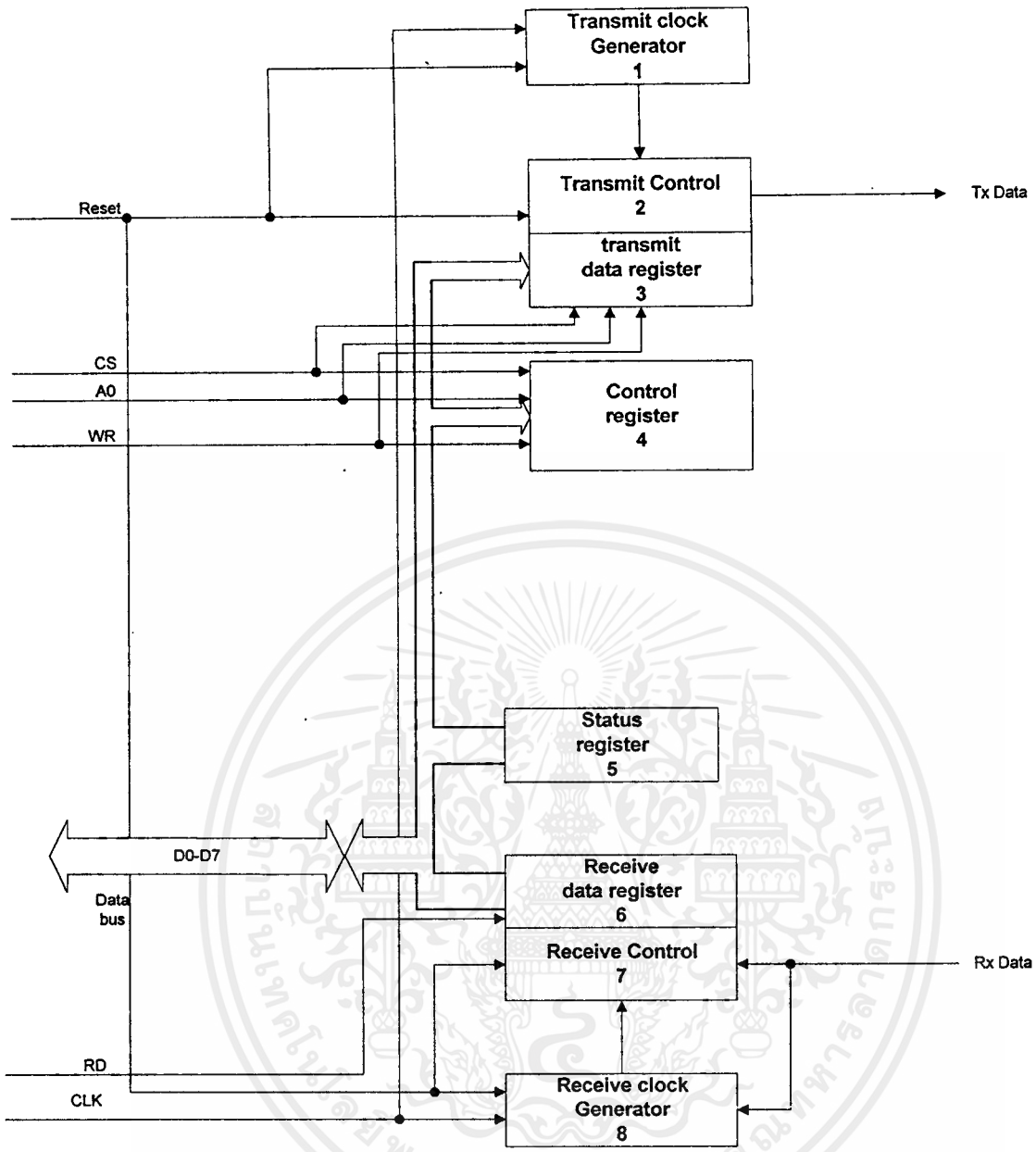
3.2.1.1 Design Specification

Features

- Complete asynchronous communication protocol includes :-
  - 8 bit data transfer
  - Even/odd or no parity bit generation and detection
  - Start and stop bit generation and detection
  - Receive overrun , framing errors and parity errors detection
- Buffered transmit and receive registers
- Exception handling using interrupt/polled modes
- Complete status reporting capabilities



รูปที่ 3-1 UART interface block

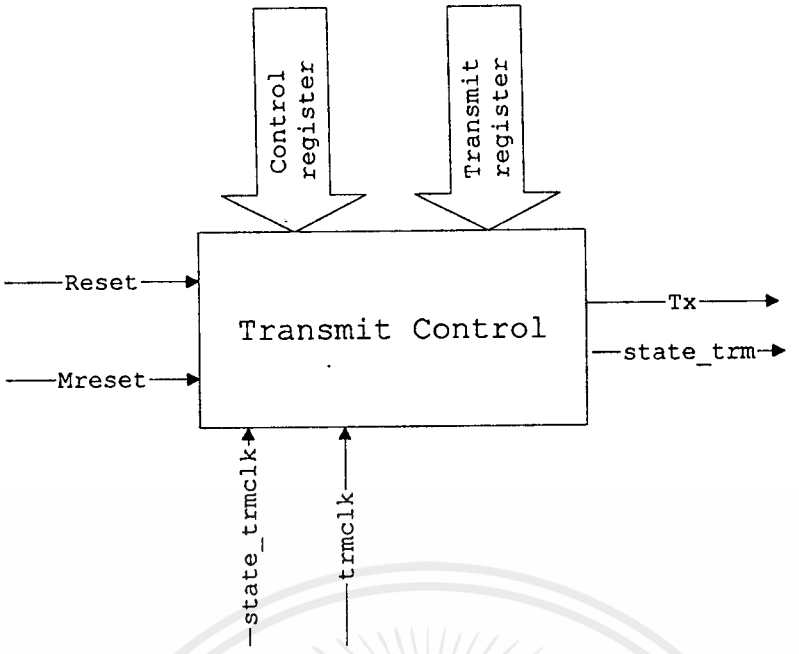


รูปที่ 3-2 แสดง Block Diagram สำหรับ UART

3.2.1.2 Design Partition ได้ทำการออกแบบแผนภูมิสถานะและ Timing Diagram โดยแบ่งการออกแบบเป็นส่วน ๆ ดังต่อไปนี้

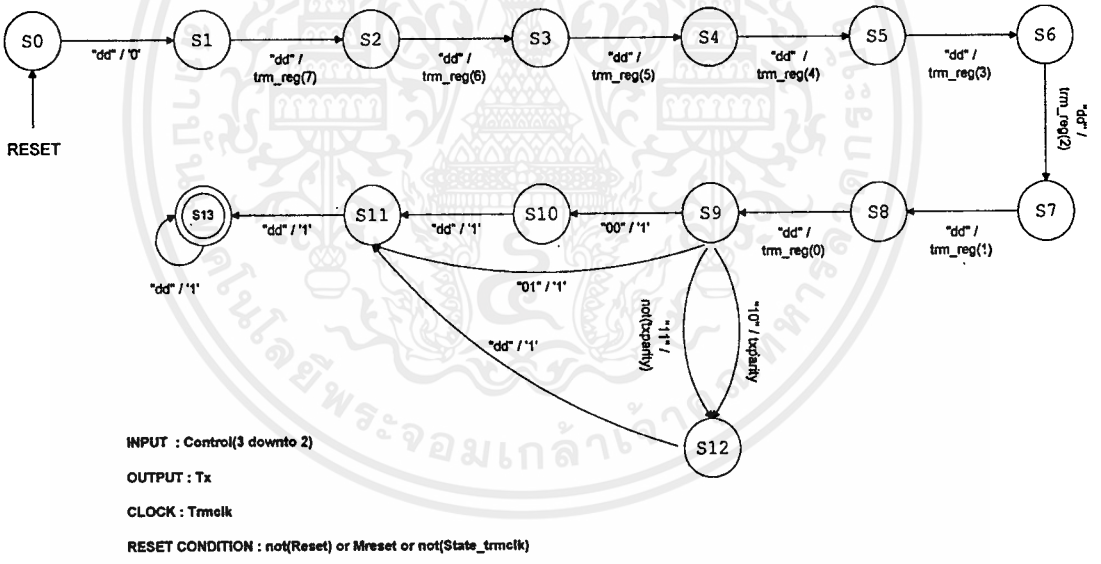
- **Transmit Control Block**

เป็นส่วนที่ใช้ในการควบคุมการส่งข้อมูลออกไปบนสายสื่อสาร ซึ่งก่อนส่งจะต้องทำการสร้างบิทพาริตี้ ใส่เข้าไปในข้อมูลที่จะส่งด้วย รูปที่ 3-3 แสดงลักษณะของ Block Diagram ของ Transmit Control



รูปที่ 3-3 Block Diagram ของ Transmit Control

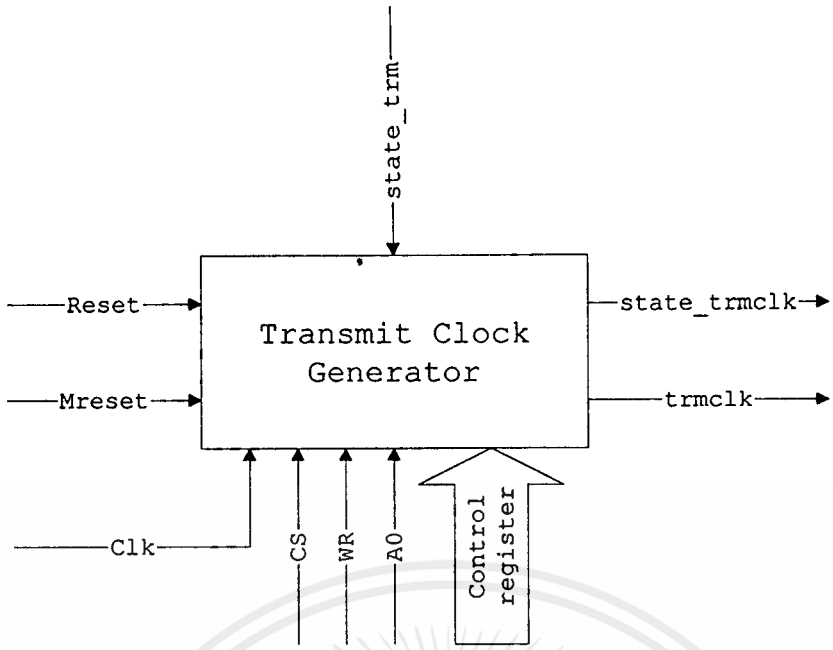
รูปที่ 3-4 แสดงแผนภูมิสถานะของ Transmit Control



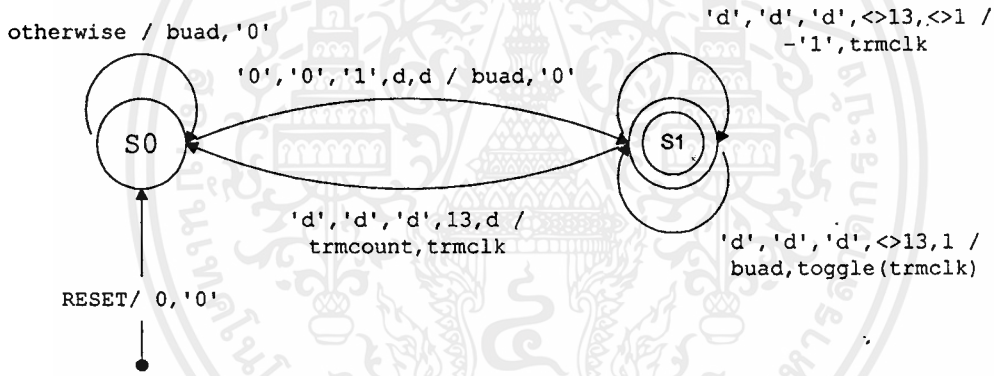
รูปที่ 3-4 แผนภูมิสถานะของ Transmit Control

- **Transmit Clock Generator Block**

ทำหน้าที่สร้างสัญญาณนาฬิกาที่ใช้ในการส่งข้อมูล ซึ่งสามารถกำหนดได้โดยโปรแกรม ซึ่งกำหนดได้ 3 ค่า คือ หารด้วย 16 , 32 และ 64 รูปที่ 3-5 แสดงลักษณะ Block Diagram ของ Transmit Clock Generator



รูปที่ 3-5 แสดงลักษณะ Block Diagram ของ Transmit Clock Generator



***Buad = 8 if control(1 downto 0) = "00"**

Buad = 16 if control(1 downto 0) = "01"

Buad = 32 if control(1 downto 0) = "10"

INPUT : CS,WR,A0,State_trm,Trmcount

OUTPUT : Trmcount,Trmclock

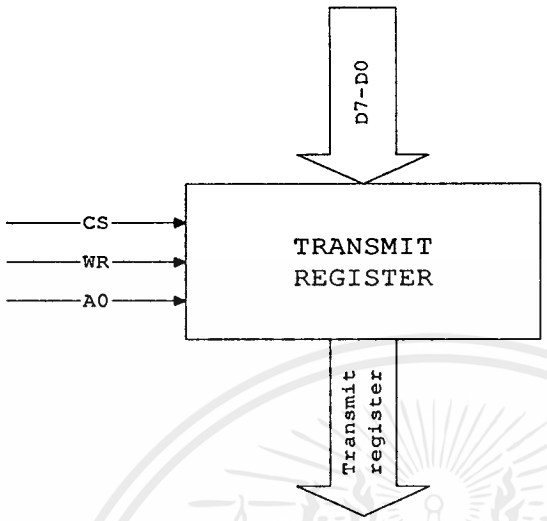
CLOCK : Clk

RESET CONDITION : not(Reset) or Mreset

รูปที่ 3-6 แสดงแผนภูมิสถานะของ Transmit Clock Generator

- **Transmit Data Register Block**

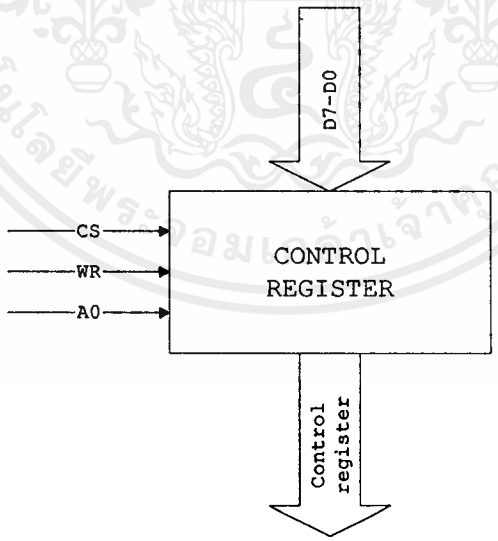
ส่วนนี้จะทำงานร่วมกับ Transmit Control Block เนื่องจากส่วนนี้จะเป็นส่วนที่ใช้เก็บข้อมูลที่จะส่งออกไป ส่วน Transmit Control Block จะทำหน้าที่ในการ shift ข้อมูลออกไปบนสาย



รูปที่ 3-7 แสดงลักษณะของ Block Diagram ของ Transmit Data Register

- **Control Register Block**

บล็อกนี้ทำหน้าที่ในการกำหนดว่าจะให้มีการรับ-ส่งข้อมูลแบบใด ซึ่งแต่ละบิทจะมีหน้าที่ดังแสดงในรูปที่ 3 - 8



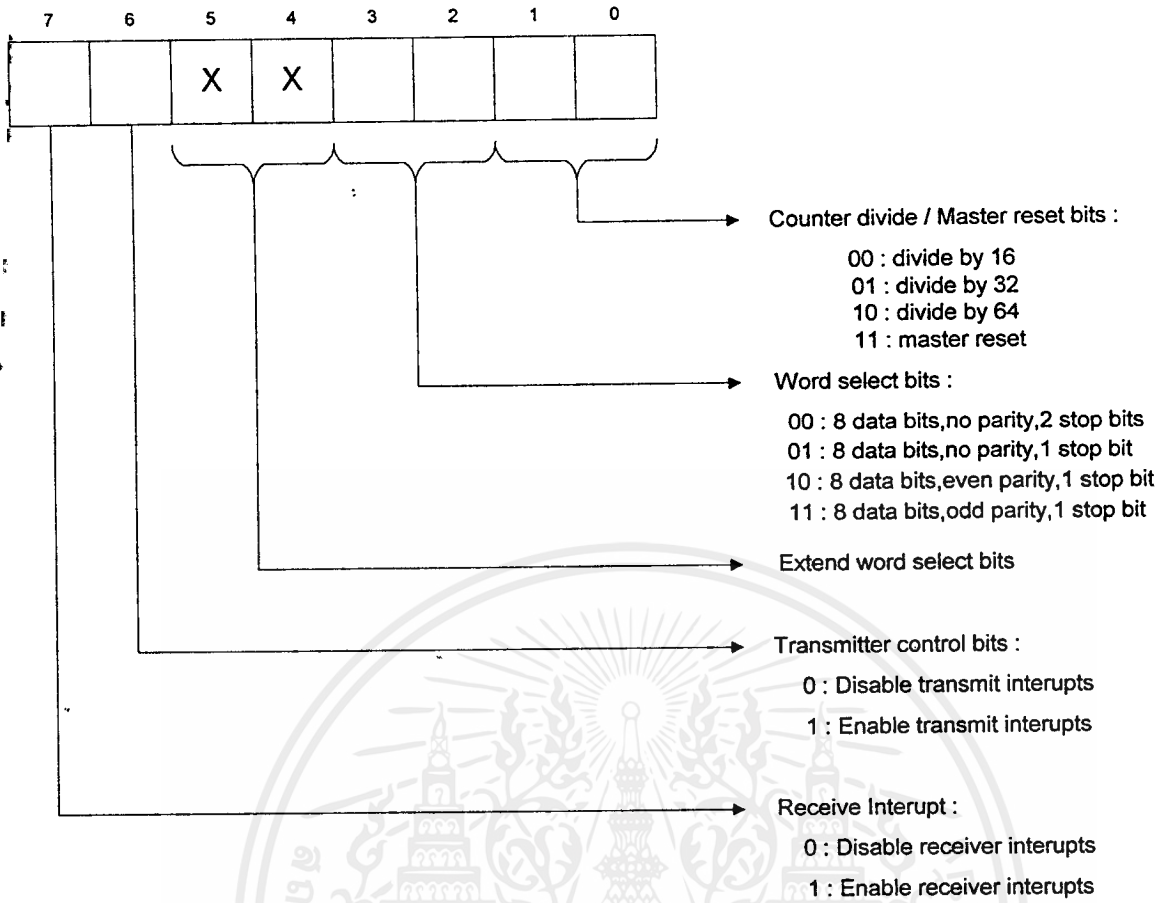
รูปที่ 3-8 แสดงลักษณะ Block Diagram ของ Control Register

- **Counter divide and Master reset (บิท1 and บิท0) :** เป็นบิทที่ใช้กำหนดสัญญาณนาฬิกาที่ใช้ในการรับ - ส่งข้อมูล โดยจะมีการกำหนดแต่ละบิทเป็นดังตารางที่ 3 - 1

ตาราง 3-1 แสดงการกำหนดสัญญาณนาฬิกาสำหรับ UART

บิท 1	บิท 0	Function
0	0	divide by 16
0	1	divide by 32
1	0	divide by 64
1	1	Master reset

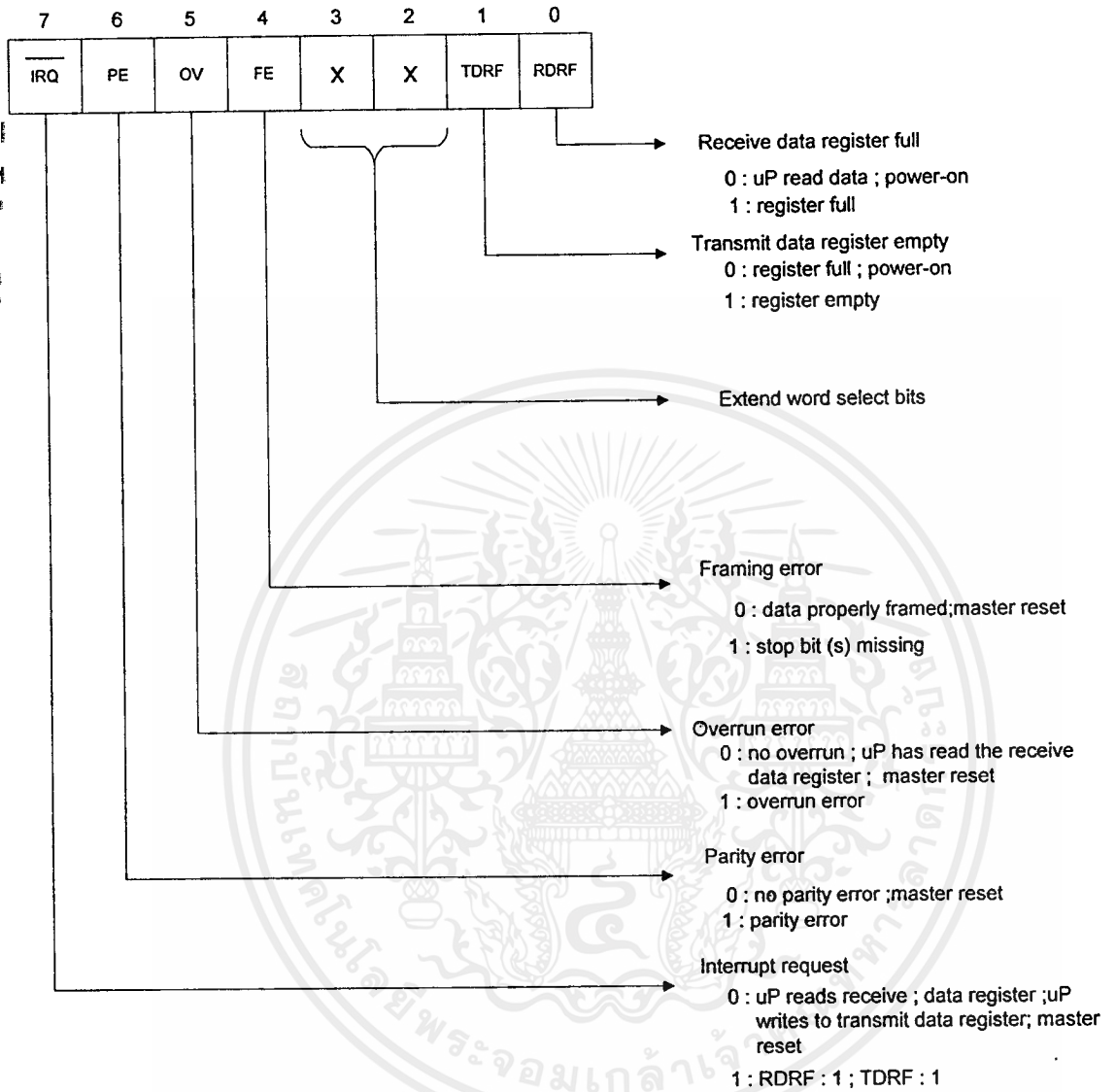
- **Word Select bit (บิท3 and บิท2) :** ใช้กำหนดขนาดของข้อมูลที่จะรับ - ส่งว่าจะมีความยาวเท่าใด และใช้กำหนดว่าจะมี Stop Bit เท่าใด นอกจากนั้นยังใช้กำหนดว่าจะมี parity bit หรือไม่ ถ้ามีจะเป็น odd parity หรือ even parity
- **Transmit Control bit (บิท6) :** เป็นบิทที่ใช้กำหนดการอินเตอร์รัปท์ไมโครคอนโทรลเลอร์ว่าขณะนี้ได้ส่งข้อมูลออกไปหมดแล้ว โดยจะมีค่าเป็น 0 เมื่อไม่อนุญาตให้มีการอินเตอร์รัปท์ และมีค่าเป็น 1 เมื่ออนุญาตให้มีการอินเตอร์รัปท์
- **Receive Interrupt bit (บิท7) :** เป็นบิทที่ใช้กำหนดการอินเตอร์รัปท์ไปบอกไมโครคอนโทรลเลอร์ว่าขณะนี้ได้รับข้อมูลมาจากฝ่ายส่งเรียบร้อยแล้ว โดยจะมีค่าเป็น 0 เมื่อไม่อนุญาตให้มีการ อินเตอร์รัปท์ และจะมีค่าเป็น 1 เมื่ออนุญาตให้มีการอินเตอร์รัปท์



รูปที่ 3-9 แสดงการทำงานในแต่ละบิทของ Control Register

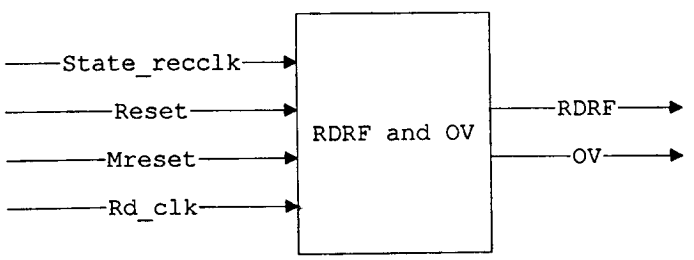
- **Status Register Block**

ส่วนนี้ใช้รายงานสถานะการทำงานของการสื่อสารระหว่างฝ่ายรับและฝ่ายส่ง โดยแต่ละบิทจะมีหน้าที่การทำงานดังแสดงในรูปที่ 3 - 10 ซึ่งมีรายละเอียดการเปลี่ยนแปลงในแต่ละบิทเป็นดังนี้

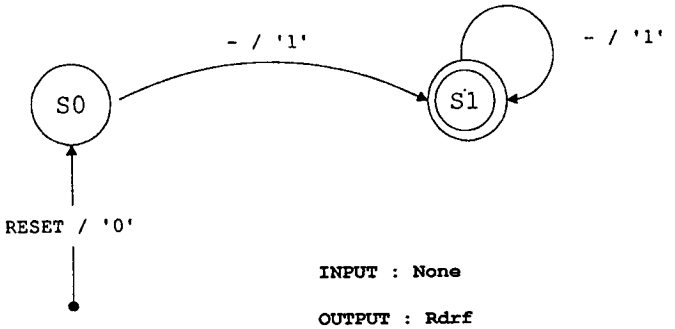


รูปที่ 3-10 แสดงการทำงานในแต่ละบิทของ Status Register

- RDRF จะเป็น ' 0 ' ในขณะที่ chip reset หรือ power-on หรือ CPU อ่านค่าจาก Receive register และจะเป็น ' 1 ' เมื่อรับบิทข้อมูลครบ 1 เฟรม (frame)

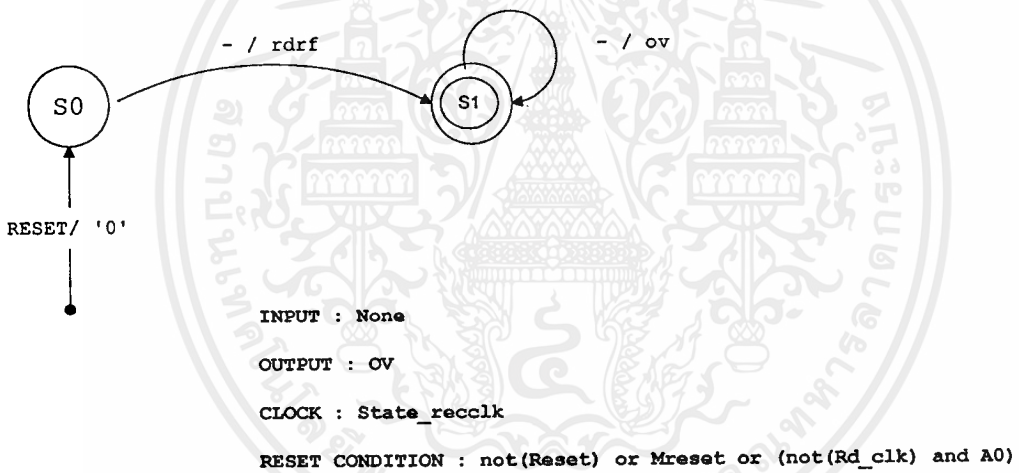


รูปที่ 3-11 แสดงลักษณะ Block Diagram ของบิท RDRF และ บิท OV



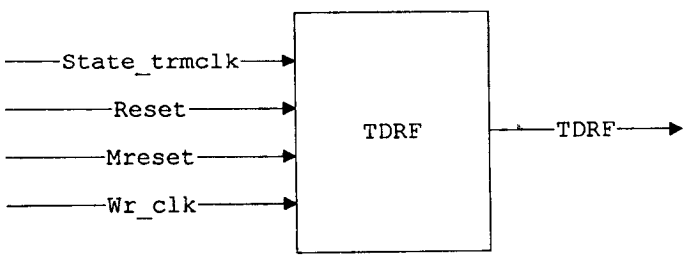
รูปที่ 3-12 แสดงการเปลี่ยนแปลงค่าของบิท RDRF ใน Status Register

- OV (overrun) จะเป็น ' 0 ' ในขณะที่ chip reset หรือ power-on และจะเป็น ' 1 ' ถ้าข้อมูลใหม่เข้ามาในขณะที่ยังไม่ได้อ่านข้อมูลเก่า

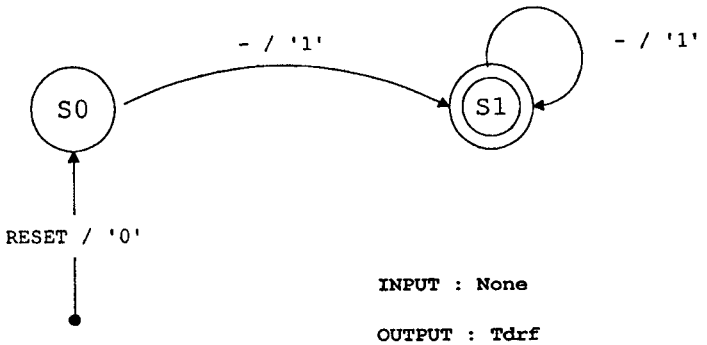


รูปที่ 3-13 แผนภูมิสถานะแสดงการเปลี่ยนแปลงบิท OV ใน Status Register

- TDRF จะเป็น ' 0 ' ในขณะที่ chip reset หรือ power-on หรือ CPU เขียนค่าใส่ Transmit register และจะเป็น ' 1 ' เมื่อส่ง บิทข้อมูลครบ 1 เฟรม



รูปที่ 3-14 แสดงลักษณะ Block Diagram ของบิท TDRF



INPUT : None

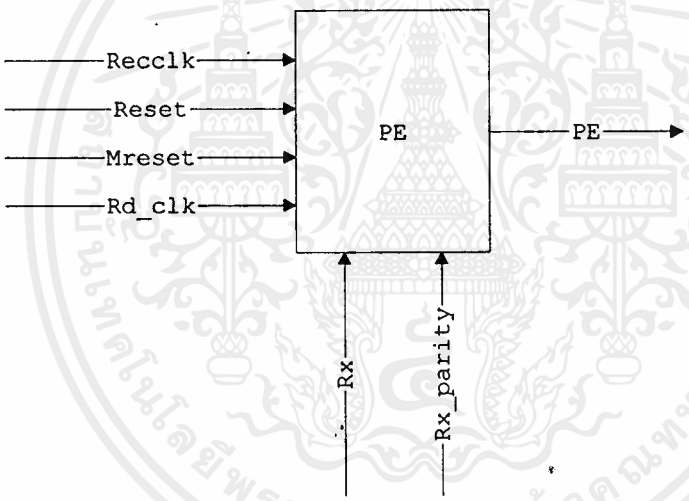
OUTPUT : Tdrf

CLOCK : State_trmclk

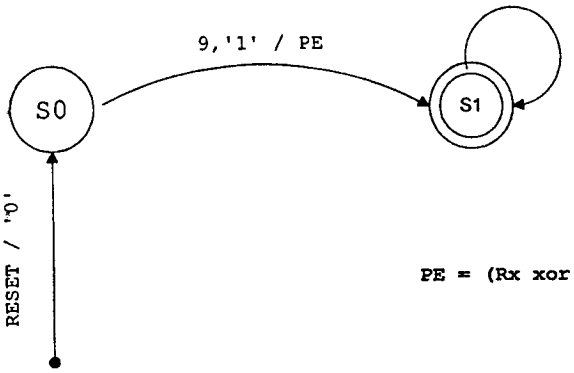
RESET CONDITION : not(Reset) or Mreset or not(Wr_clk)

รูปที่ 3-15 แผนภูมิสถานะแสดงการเปลี่ยนแปลงบิท TDRF ใน Status Register

- PE (parity error) จะเป็น ' 0 ' ในขณะที่ chip reset หรือ power-on และจะเป็น ' 1 ' เมื่อ parity ที่รับเข้ามาผิดพลาด



รูปที่ 3-16 แสดงลักษณะ Block Diagram ของบิท PE



PE = (Rx xor Rxparity) xor Control(2)

INPUT : State_rec,Control(3)

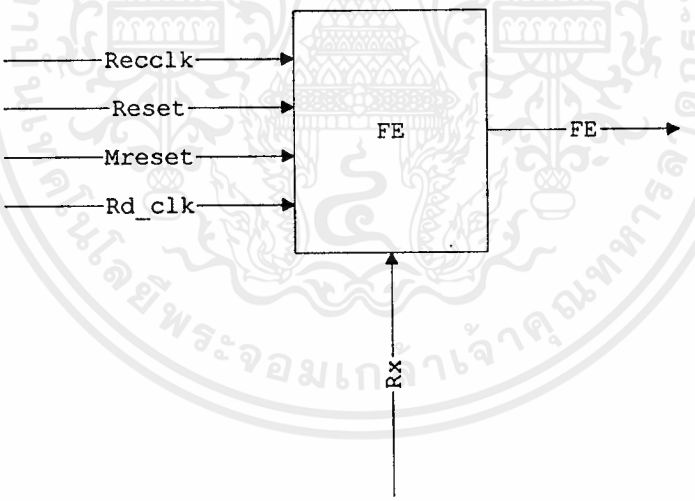
OUTPUT : PE

CLOCK : Recclk

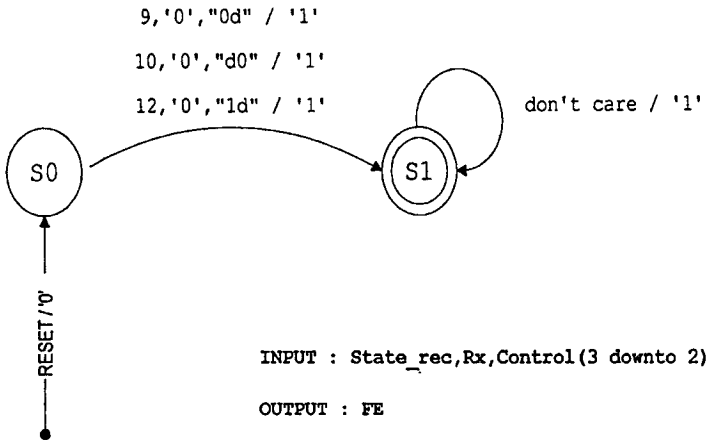
RESET CONDITION : not(Reset) or Mreset or (not(Rd_clk) and A0)

รูปที่ 3-17 แผนภูมิสถานะแสดงการเปลี่ยนแปลงบิท PE ใน Status Register

- FE (frame error) จะเป็น ' 0 ' ในขณะที่ chip reset หรือ power-on และจะเป็น ' 1 ' เมื่อเฟรมที่รับเข้ามาผิดพลาด ซึ่งเกิดจากการมี stop bit ไม่ครบตามที่กำหนด



รูปที่ 3-18 แสดงลักษณะ Block Diagram ของบิท FE



INPUT : State_rec,Rx,Control(3 downto 2)

OUTPUT : FE

CLOCK : Recclk

RESET CONDITION : not(Reset) or Mreset or (not(Rd_clk) and A0)

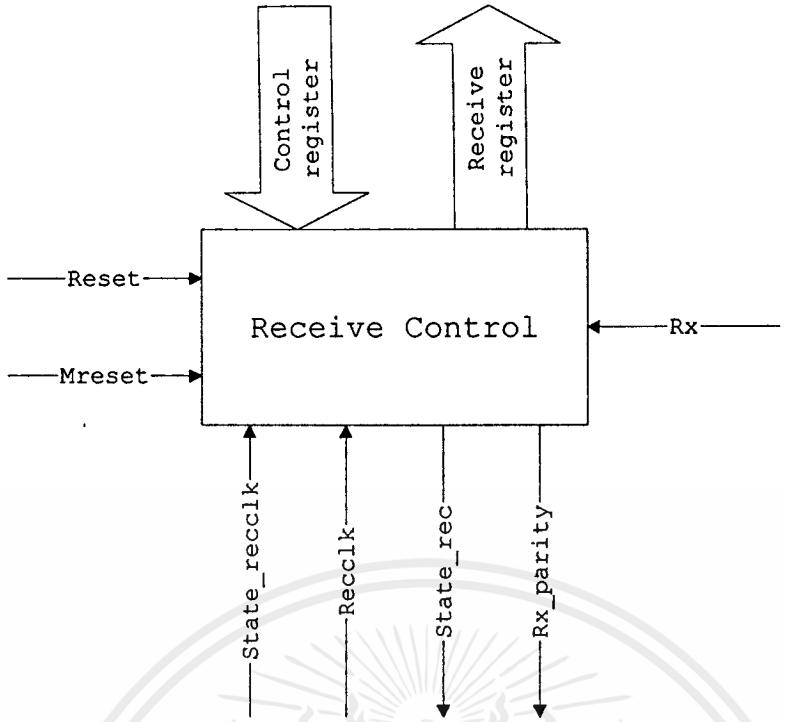
รูปที่ 3-19 แผนภูมิสถานะแสดงการเปลี่ยนแปลงบิท FE ใน Status Register

- IRQ (interrupt request) จะเป็น ' 0 ' ในขณะที่ chip reset หรือ power-on และเป็น ' 1 ' เมื่อ UART ต้องการอินเตอร์รัปท์

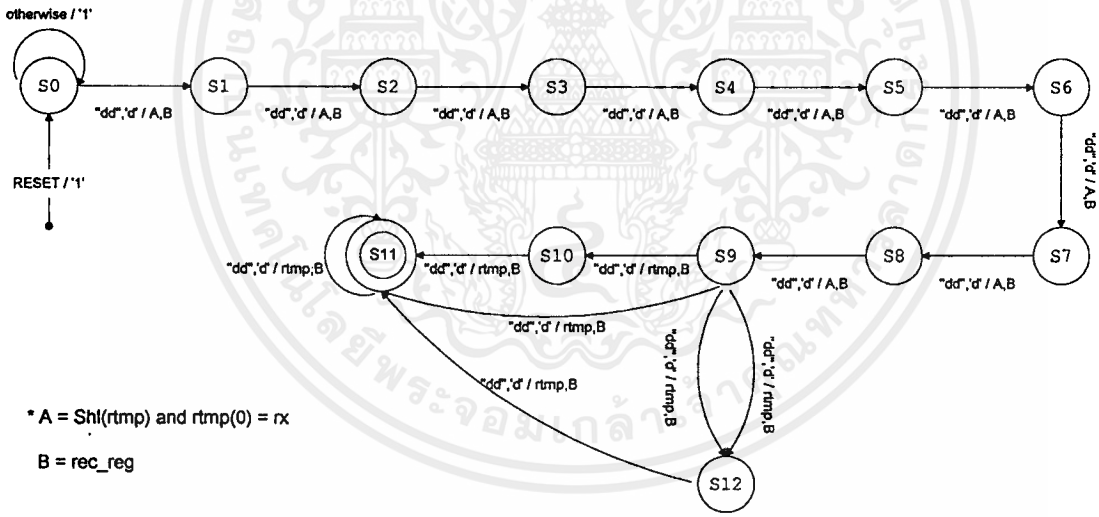
รูปที่ 3-20 แสดงการเปลี่ยนแปลงบิท IRQ ใน Status Register

- **Receive Control Block**

ส่วนนี้จะใช้ควบคุมการรับข้อมูลทั้งหมด ซึ่งเมื่อรับแล้วจะมีการตรวจสอบความถูกต้องของข้อมูลที่ได้รับมา แล้วจึงไปทำการเซ็ทบิทต่าง ๆ ใน Status Register



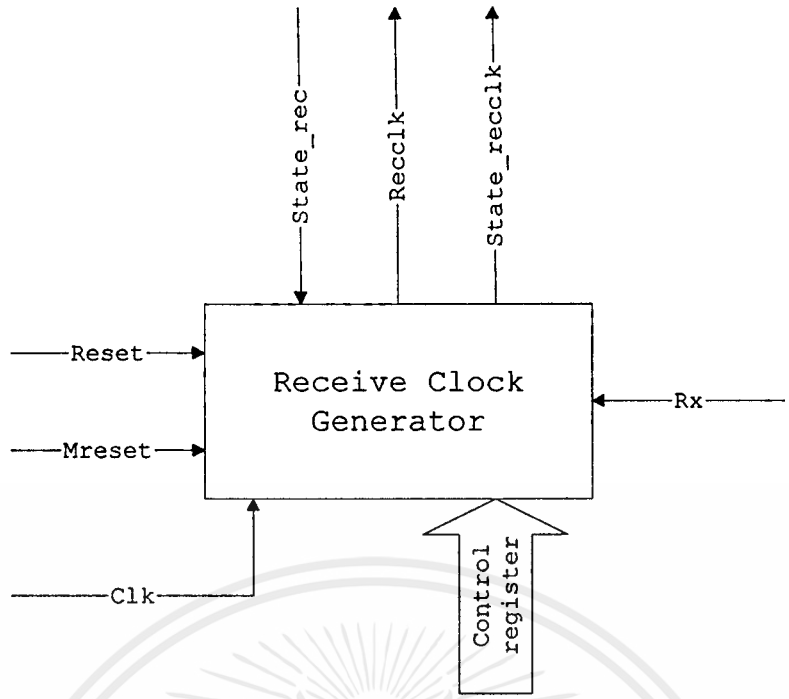
รูปที่ 3-21 แสดงลักษณะ Block Diagram ของ Receive Control



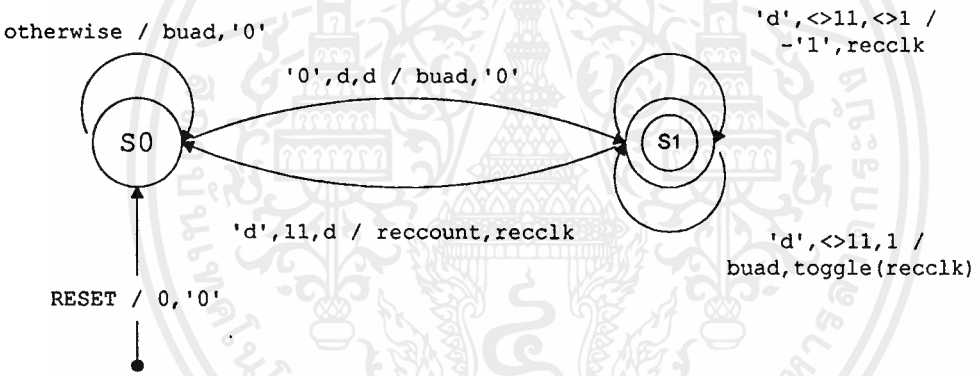
รูปที่ 3-22 แผนภูมิสถานะของ Receive Control

- **Receive Clock Block**

ใช้สร้างสัญญาณนาฬิกาสำหรับซิงโครไนซ์กับทางฝ่ายที่ทำการส่ง ซึ่งมีแผนภูมิสถานะเป็นดังรูปที่ 3-23



รูปที่ 3-23 แสดงลักษณะ Block Diagram ของ Receiver Clock Generator



***Buad = 8 if control(1 downto 0) = "00"**

Buad = 16 if control(1 downto 0) = "01"

Buad = 32 if control(1 downto 0) = "10"

INPUT : RX,State_rec,Reccount

OUTPUT : Reccount,Recclock

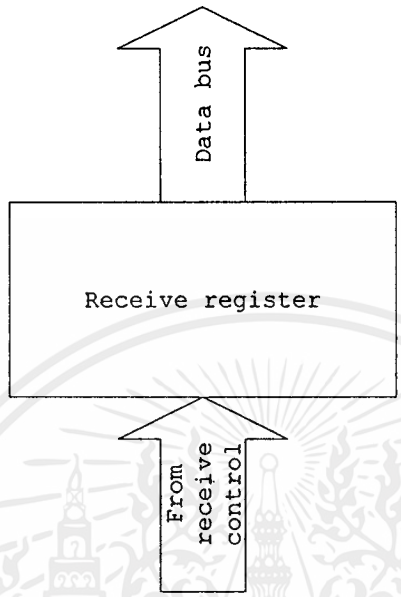
CLOCK : Clk

RESET CONDITION : not(Reset) or Mreset

รูปที่ 3-24 แผนภูมิสถานะของ Receive Clock

- **Receive Data Register Block**

เป็นส่วนที่เก็บข้อมูลซึ่งรับมา เมื่อส่วนนี้เต็มจะมีการส่งสัญญาณอินเตอร์รัปท์ไปบอกไมโครคอนโทรลเลอร์ ซึ่งถ้าไมโครคอนโทรลเลอร์ อนุญาตให้มีการอินเตอร์รัปท์ก็จะอ่านข้อมูลส่วนนี้ไป



รูปที่ 3-25 แสดงลักษณะ Block Diagram ของ Receive data register

ตาราง 3-2 Core Signal Pinout of UART

Signal	Signal Direction	Description
System Interface Signals		
A0	Input	Address signal to select and internal registers for read/write operations.
CS	Input	Chip Select. Indicates read/write selection of UART , active low.
WR	Input	Write Control, active low. Pulled high if unused.
RD	Input	Read Control, active low. Pulled high if unused.
D0 - D7	In/Out	Bidirectional data bus carries data to be written to internal registers; also reports status of

		registers during read cycle.
CLK	Input	Master clock input.
RESET	Input	Chip reset. This input forces UART into predefined state; all flags are reset.
Transmit/Receive Signals		
TX	Output	Serial data output from transmit data register block.
RX	Input	Serial data input for receive data register block.

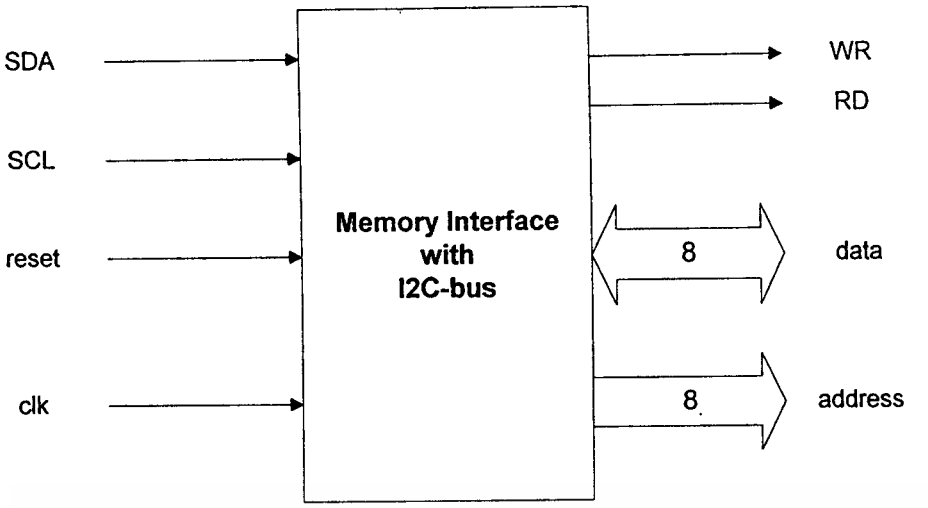
3.2.2 ขั้นตอนการออกแบบ I2C-bus Memory Interface

สำหรับการออกแบบ I2C-bus memory Interface นั้น จากที่ได้กล่าวมาตั้งแต่ต้นแล้วว่าไม่ได้มีการเลียนแบบชิปตัวใดเลย ซึ่งตัวนี้จะเป็นเพียงตัวเชื่อมต่อกับหน่วยความจำภายนอกโดยตรง จะไม่มีหน่วยความจำอยู่ภายใน ซึ่งจะแตกต่างกับชิปที่มีอยู่แล้วที่เป็นของ Philips ซึ่งจะมีหน่วยความจำอยู่ภายในตัวเลย เพราะฉะนั้นในการทำงานสำหรับ I2C-bus memory Interface ในโครงงานนี้จะทำงานได้จะต้องต่อกับหน่วยความจำอีกตัวหนึ่ง

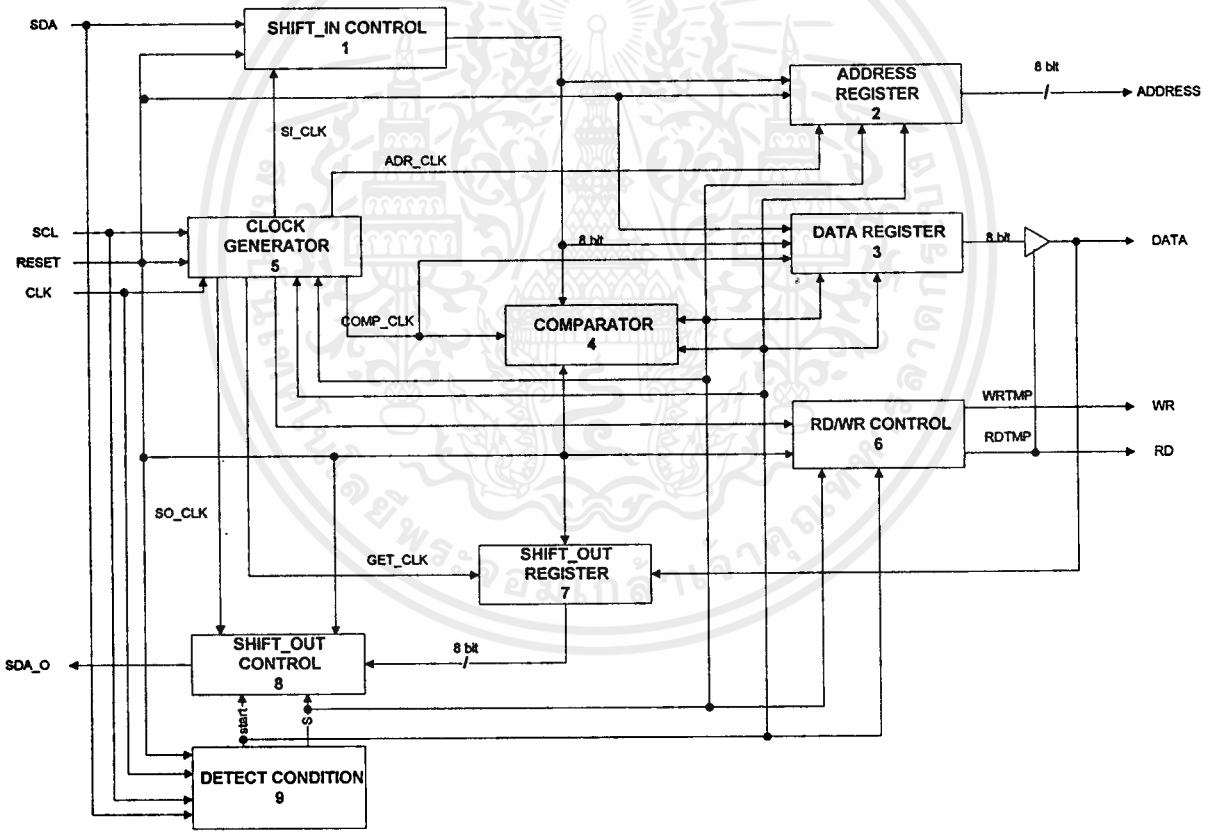
3.2.2.1 Design Specification

Features

- Serial input/output bus
- Automatic word address incrementing
- Power on reset
- 2 modes operation : slave transmitter and slave receiver



รูปที่ 3-26 Memory Interface Block Diagram

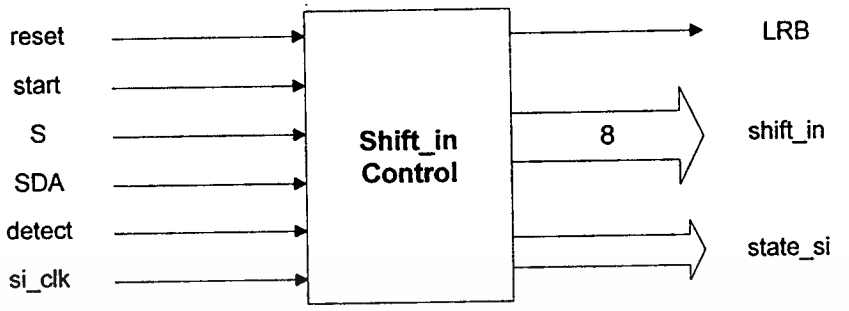


รูปที่ 3-27 แสดง Block Diagram ภายในของ I^2C-bus Memory Interface

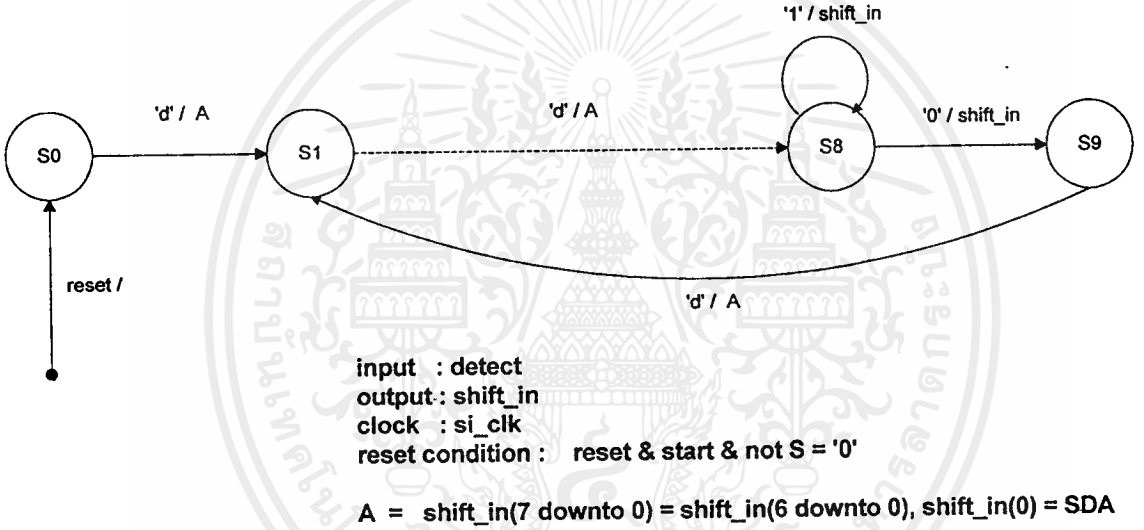
3.2.2.2 Design Partition ได้ทำการออกแบบแผนภูมิสถานะและ Timing Diagram โดยทำการแบ่งการออกแบบเป็นแต่ละส่วนดังนี้

- **Shift_in Control Block**

เป็นส่วน ที่ใช้ในการ shift ข้อมูลที่อยู่บนสาย SDA เข้ามาทีละบิท แล้วจึงรวมเป็นตัวอักษร ซึ่งการทำงานของ Shift_in register นี้จะถูกควบคุมโดย si_clk ซึ่ง จะมีBlock Diagramและ แผนภูมิสถานะ เป็นดังรูปที่ 3- 28 และ 3-29 ตามลำดับ



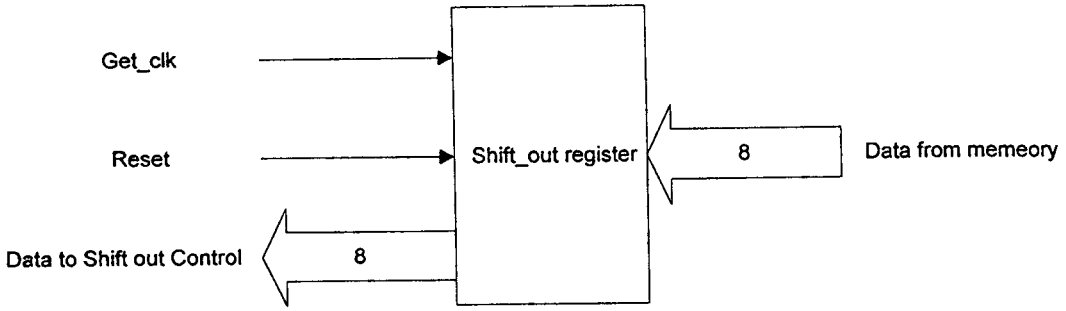
รูปที่ 3-28 แสดง Block Diagram ของ Shift_in register



รูปที่ 3-29 แผนภูมิสถานะสำหรับ shift_in register

- **Shift_out Register Block**

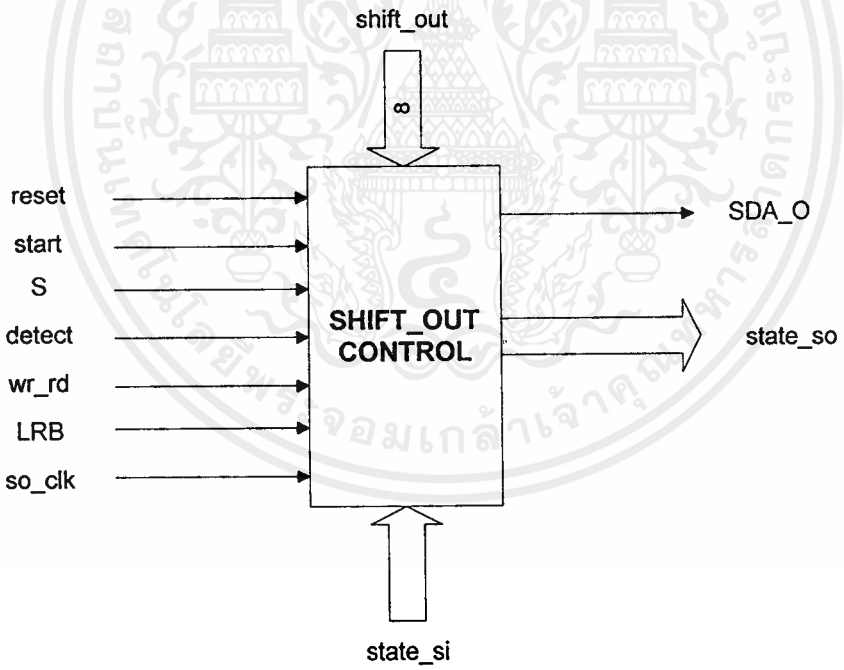
ส่วนของ Shift_out register จะทำหน้าที่รับข้อมูลจากหน่วยความจำ ซึ่งจะทำเมื่อมาสเตอร์ในระบบต้องการอ่านข้อมูลจากหน่วยความจำ โดย Shift_out register จะทำงานเมื่อได้รับสัญญาณ Get_clk จาก Clock Generator การทำงานทั้งหมดของ Shift_out register จะขึ้นอยู่กับ Shift_out Control ดังนั้นแผนภูมิสถานะจึงรวมอยู่กับ Shift_out Control ซึ่ง Shift_out register จะมี Block Diagram เป็นดังรูปที่ 3- 30



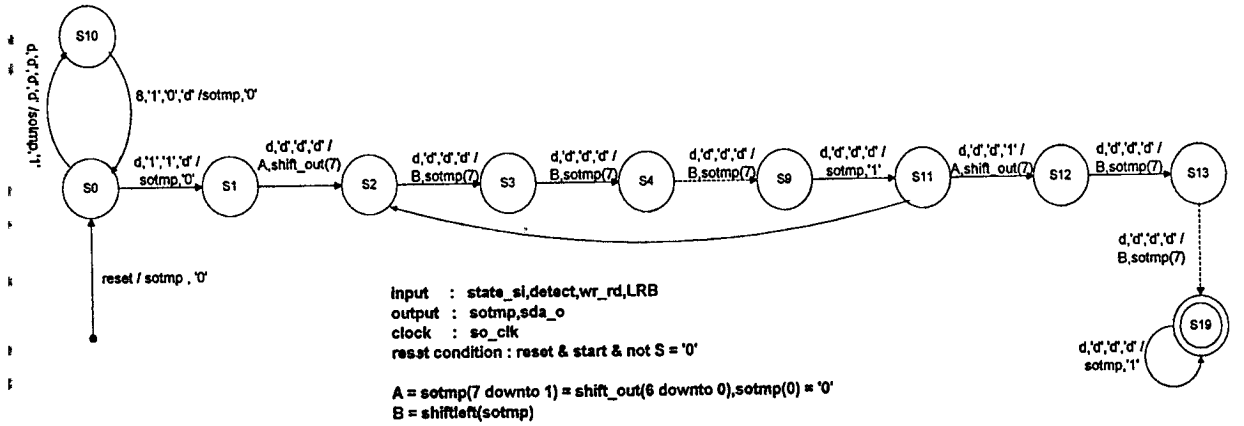
รูปที่ 3-30 Block Diagram สำหรับ Shift_out Register

- **Shift_out Control Block**

เป็นบล็อกที่ทำหน้าที่ในการ Shift ข้อมูลที่ได้มาจาก Shift_out register ออกไปบนสาย SDA ทีละบิท ถ้าวงจรอยู่ในโหมดของ ตัวส่งที่เป็นสเลฟ แต่ถ้าวงจรอยู่ในโหมดของตัวรับที่เป็นสเลฟ ตัว Shift_out Control จะทำหน้าที่ในการ Acknowledge เมื่อได้รับข้อมูลแล้ว ซึ่ง Shift_out Control มี Block Diagram และแผนภูมิสถานะ เป็นดังรูปที่ 3-31 และ รูปที่ 3-32



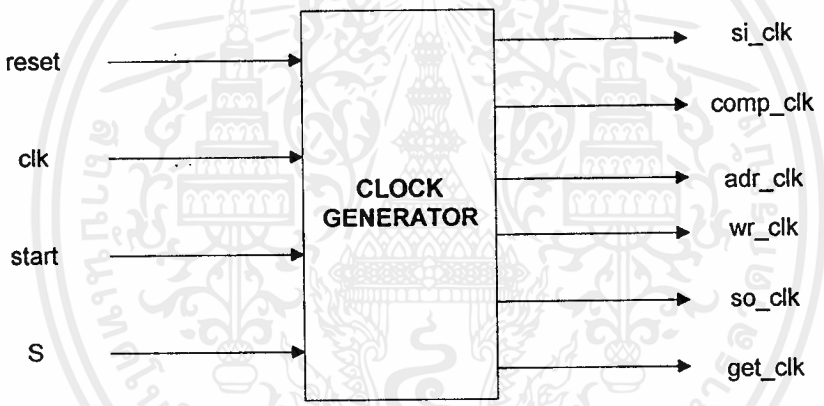
รูปที่ 3-31 แสดง Block Diagram ของ Shift_out Control



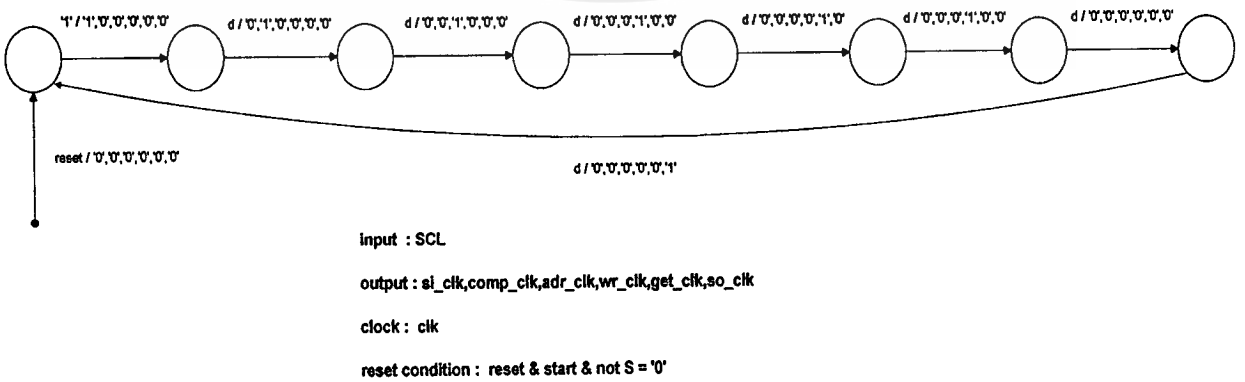
รูปที่ 3-32 แผนภูมิสถานะสำหรับ Shift_out Control

- **Clock Generator Block**

ทำหน้าที่สร้างสัญญาณนาฬิกาให้กับบล็อกต่าง ๆ ในตัววงจรตามสัญญาณ SCL ที่ได้รับมาซึ่งมี Block Diagram และแผนภูมิสถานะดังรูปที่ 3-33 และ 3-34 ตามลำดับ



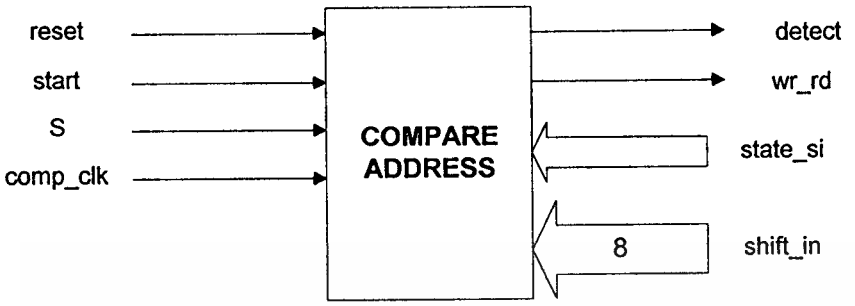
รูปที่ 3-33 Block Diagram of Clock Generator



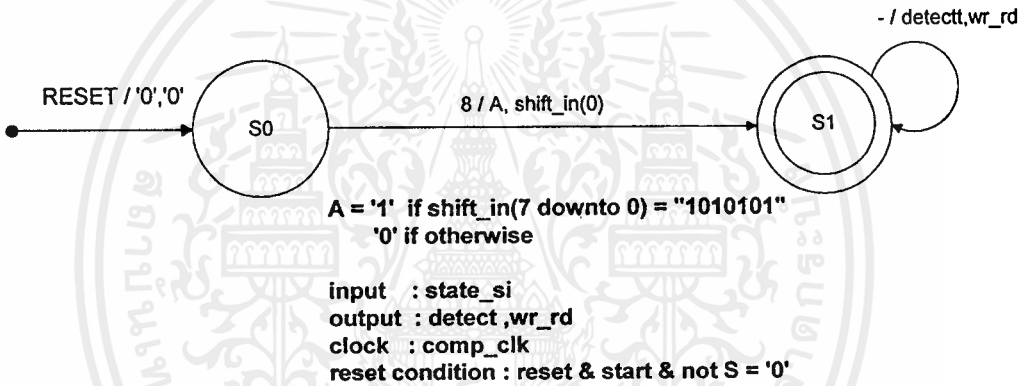
รูปที่ 3-34 แสดงแผนภูมิสถานะของ Clock Generator

- **Compare Address Block**

บล็อกนี้มีหน้าที่ในการเปรียบเทียบว่า 7 บิทแรกหลังจากที่เกิด Start Condition แล้วเป็นแอดเดรสของวงจรหรือไม่ โดยแอดเดรสของ Chip Memory Interface with I2C-bus คือ " 1010101 "



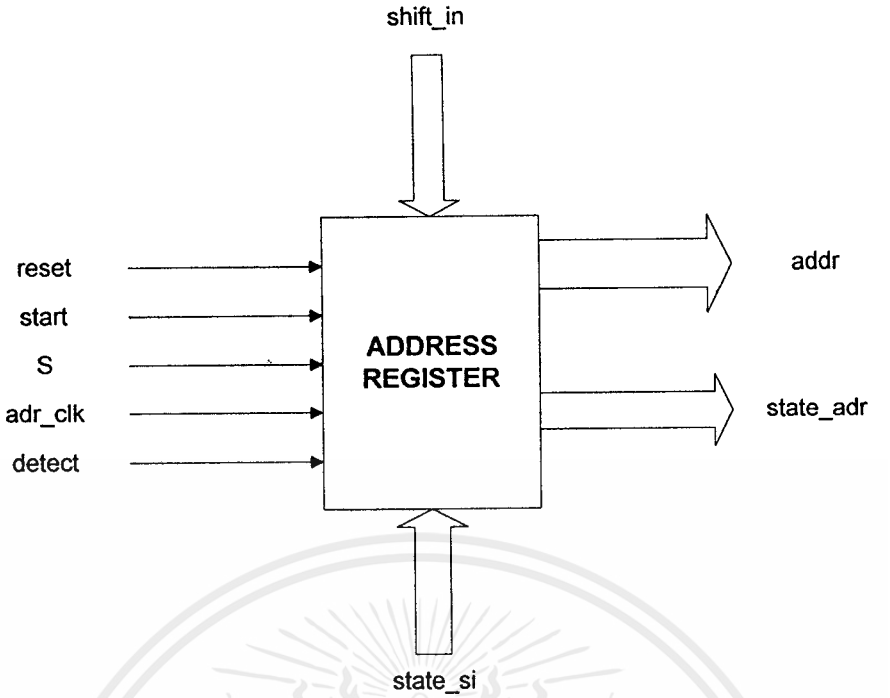
รูปที่ 3-35 แสดง Block Diagram ของ Compare Address



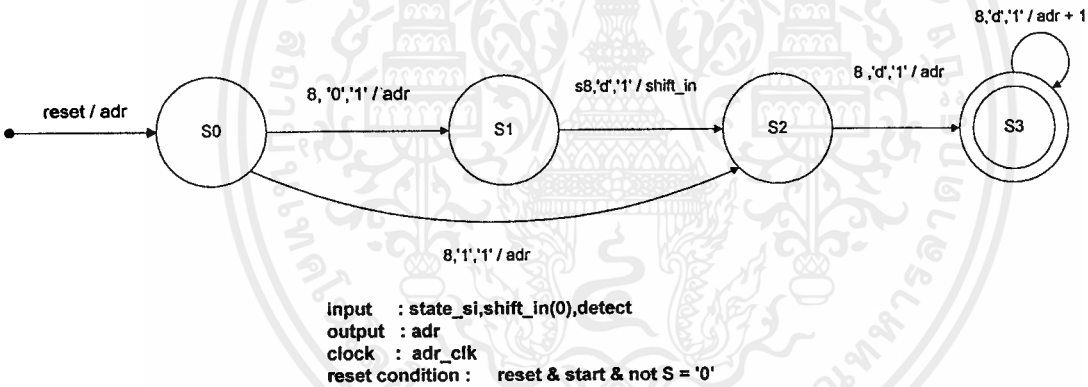
รูปที่ 3-36 แสดงแผนภูมิสถานะของ Compare Address

- **Address Register Block**

ทำหน้าที่ในการเก็บค่าของแอดเดรสที่จะทำการเข้าถึง (access) หน่วยความจำ โดยจะทำการบวกแอดเดรสของหน่วยความจำขึ้นทีละหนึ่งโดยอัตโนมัติ จะเริ่มทำงานเมื่อได้รับสัญญาณ Adr_clk จาก Clock Generator



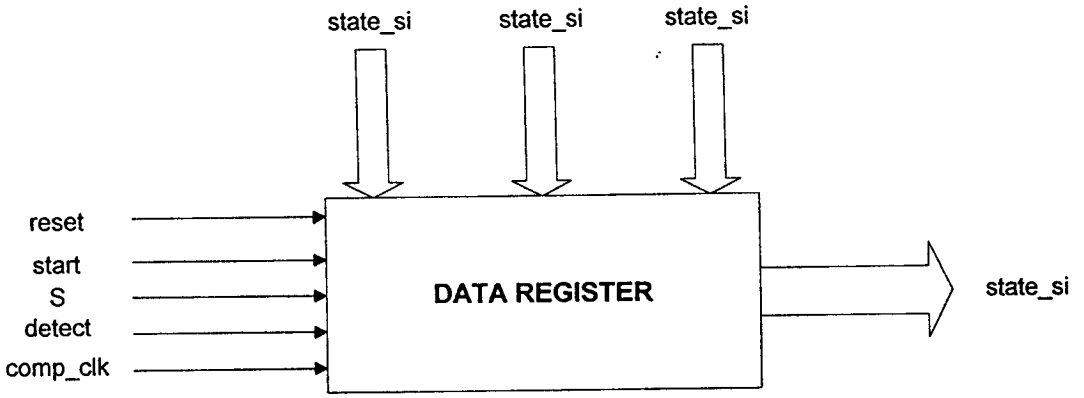
รูปที่ 3-37 แสดง Block Diagram ของ Address Register



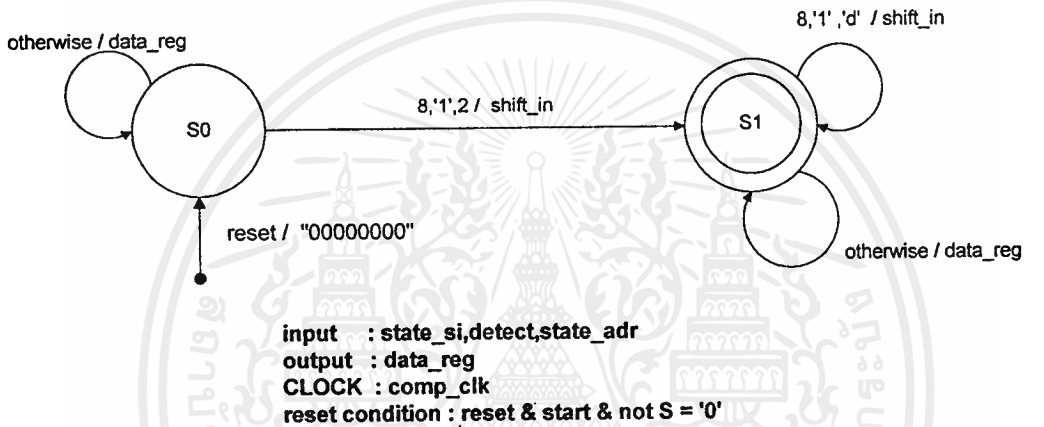
รูปที่ 3-38 แผนภูมิสถานะของ Address Register

- **Data Register Block**

ทำหน้าที่ในการรับข้อมูลจาก Shift_in Register เพื่อที่จะนำไปเก็บในหน่วยความจำต่อไป Data Register เป็นเสมือนที่พักข้อมูลชั่วคราว



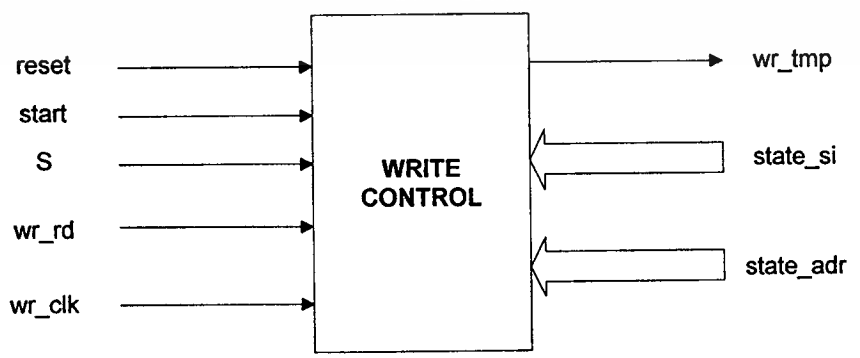
รูปที่ 3-39 แสดง Block Diagram ของ Data Register



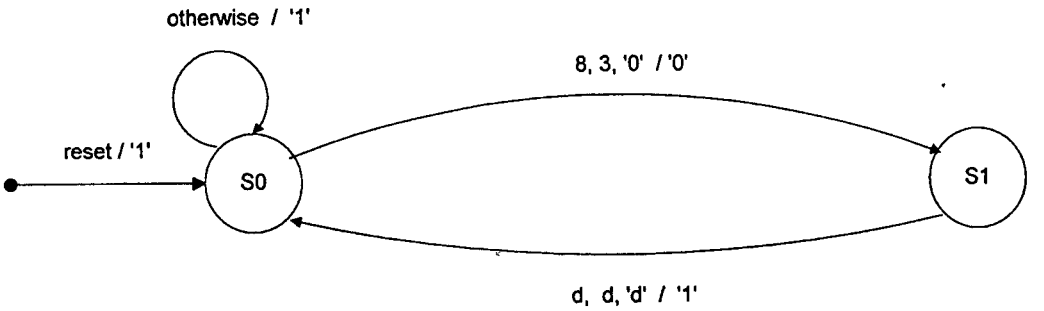
รูปที่ 3-40 แผนภูมิสถานะของ Data Register

- **WR/RD Control Block**

บล็อกนี้จะทำหน้าที่ในการควบคุมการอ่านหรือเขียนหน่วยความจำ ซึ่งจะขึ้นอยู่กับว่าขณะนั้นวงจรทำงานอยู่ในโหมดของตัวรับที่เป็นสเลฟ หรือตัวส่งที่เป็นสเลฟ



รูปที่ 3-41 แสดง Block Diagram ในส่วนของ Write Control ใน WR/RD Control



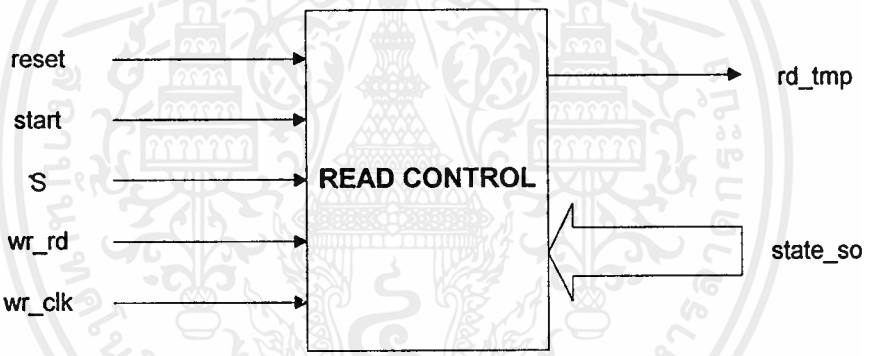
input : state_si,state_adr,wr_rd,

output : wrtmp

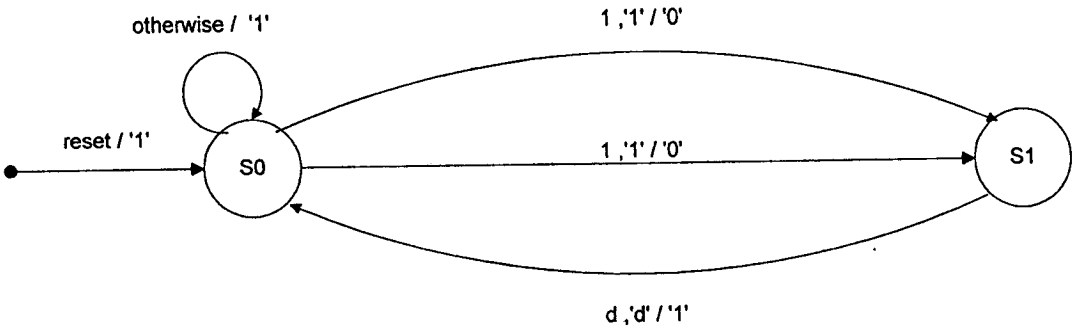
clock : wr_clk

reset condition : reset & start & not S = '0'

รูปที่ 3-42 แผนภูมิสถานะสำหรับการ Write ของ WR/RD Control



รูปที่ 3-43 แสดง Block Diagram ในส่วนของ Read Control ใน WR/RD Control



input : state_so,wr_rd

output : rdtmp

clock : wr_clk

reset condition : reset & start & not S = '0'

รูปที่ 3-44 แผนภูมิสถานะสำหรับการ Read ของ WR/RD Control

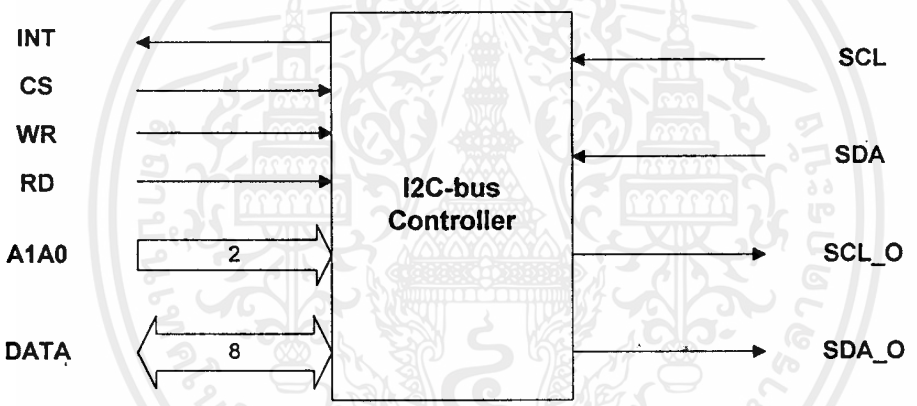
ตาราง 3-3 Core Signal Pinout of Memory Interface with I2C-bus Core

Signal	Signal Direction	Description
System Interface Signal		
SDA_O	Output	Serial Data line Out.
SDA	Input	Serial Data Line In
SCL	Input	Serial Clock Line
RESET	Input	This input forces the I2C-bus memeory into a predefined state , all flags are reset.
Memory Interface Signal		
WR	Output	WR control signal, active low. It is used in slave receiver mode.
RD	Output	RD control signal, active low. It is used in slave transmitter mode.
ADDRESS	Output	Address bus to memory.
DATA	In/Out	Bidirection data bus carries data to be written to memory ; also it carries data to be read from memory

3.2.3 ขั้นตอนการออกแบบวงจร Controller Interface with I2C-bus

สำหรับขั้นตอนในการออกแบบ I2C-bus Controller Interface นั้นจากที่กล่าวมาในข้างต้นแล้วว่าได้เลียนแบบการทำงานของ PCF8584 แต่จะมีบางอย่างที่ไม่เหมือนกัน ดังรายละเอียดต่อไปนี้คือ

- Clock Register สำหรับ PCF8584 จะมี 2 บิท ในการกำหนดความเร็วของสาย SCL และอีก 3 บิท ในการกำหนดสัญญาณนาฬิกาที่ใช้ภายในตัววงจร แต่สำหรับ core ที่ทำในโครงงานนี้จะใช้ทั้ง 8 บิท ในการกำหนดความถี่ของสัญญาณนาฬิกาบนสาย SCL และภายในตัววงจรเลยทำให้มีความยืดหยุ่นของสัญญาณนาฬิกามากกว่า
- การอ้างอิงรีจิสเตอร์ภายในตัว PCF8584 จะใช้ Input AO ร่วมกับ ES0 - ES2 ใน Control Register โดยถ้า A0 เป็น 1 จะเป็นการ RD/WR ลง Control แต่ถ้า A0 เป็น 0 จะต้องดูใน ES0 - ES2 ว่าจะทำการ RD/WR ลงรีจิสเตอร์ใด แต่สำหรับ I2C-bus controller ซึ่งทำในโครงงานนี้จะใช้ Input A0A1 เป็นตัวเข้าถึงรีจิสเตอร์ภายในตัว core ทำให้ใน Control register จึงไม่ต้องมี ES0 - ES2 BIT
- ใน Status register จะไม่มี Bus Error Bit

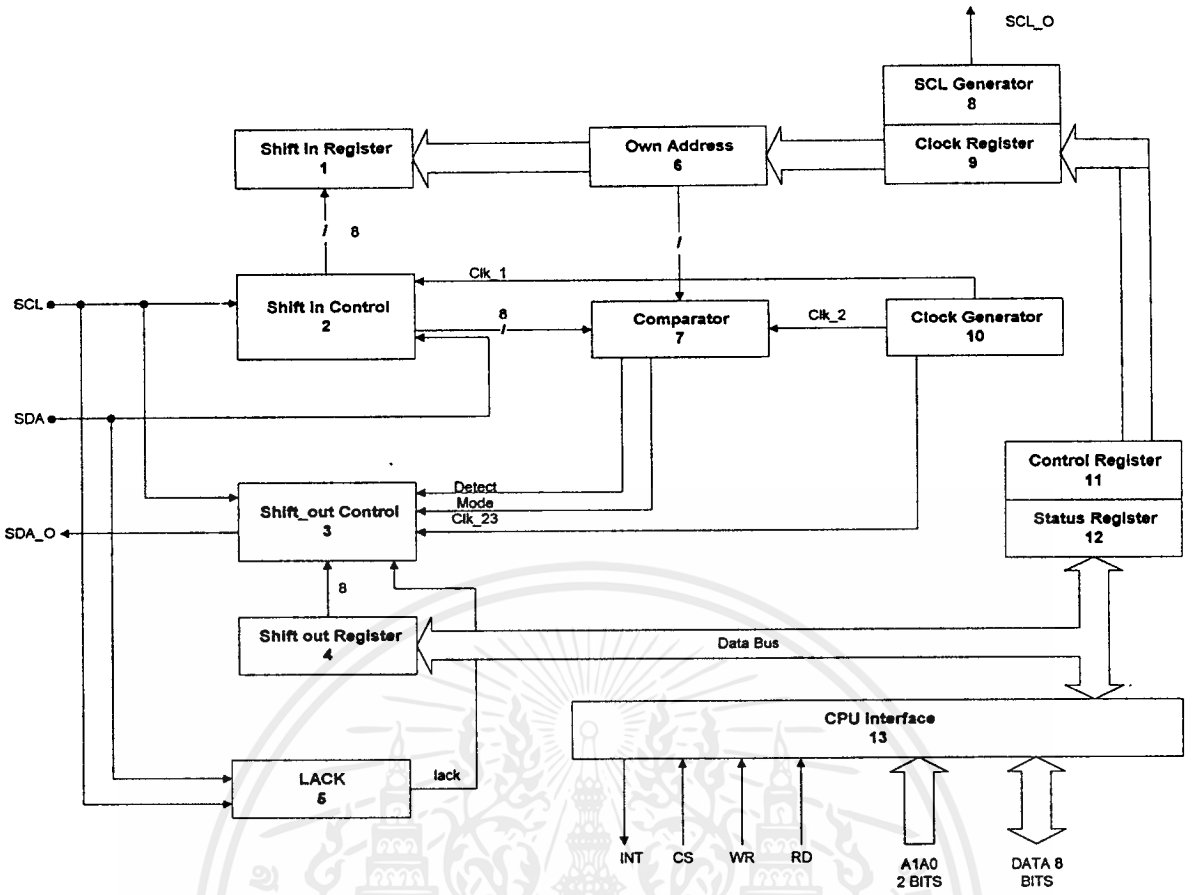


รูปที่ 3-45 I2C-bus Controller Interface Block

3.2.3.1 Design Specification

Features

- Both master and slave function
- Multi - master capability
- Parallel bus to I2C-bus protocol converter and interface
- Communication rate 100 Kbps.
- 4 modes operation : master transmitter , master receiver , slave transmitter and slave receiver



รูปที่ 3-46 Block Diagram ของ I2C-bus Controller

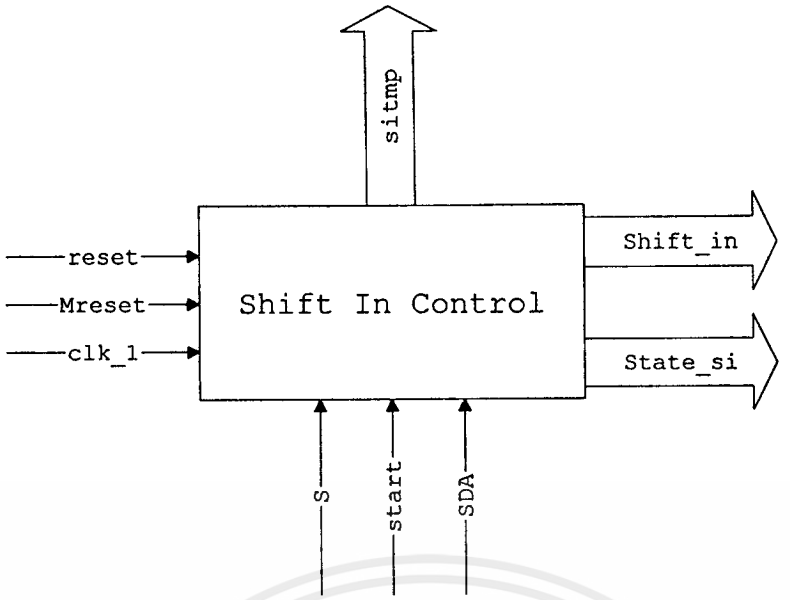
3.2.3.2 Design Partition ได้ทำการออกแบบแผนภูมิสถานะ และ Timing Diagram ของแต่ละส่วน โดยแบ่งการออกแบบเป็นแต่ละส่วนดังต่อไปนี้

- **Shift in Register Block**

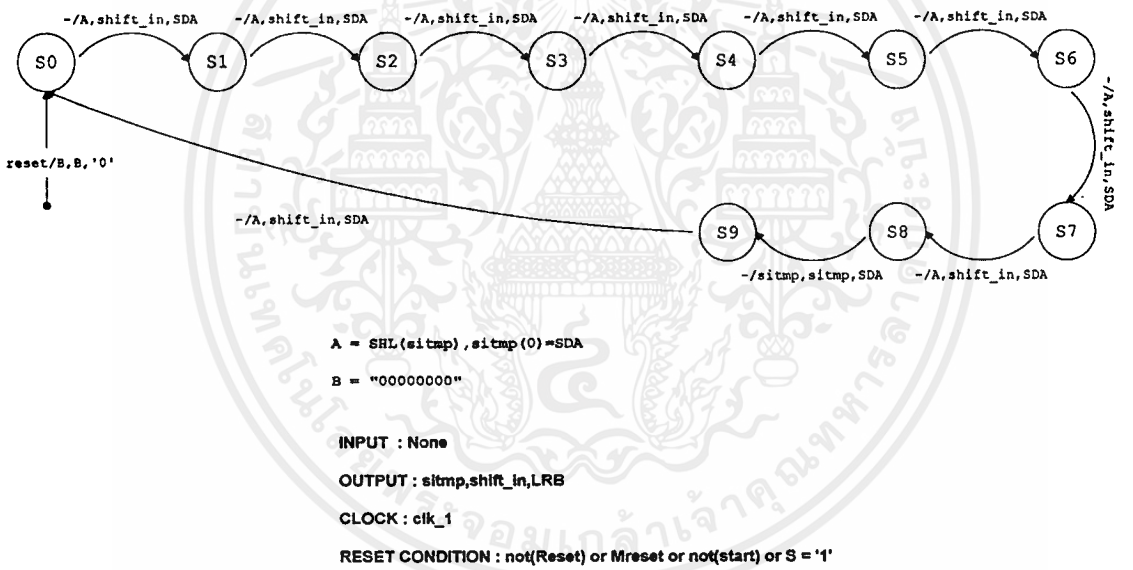
บล็อกนี้ทำหน้าที่นำข้อมูลจาก Shift In Control มา เป็นที่พักข้อมูลเพื่อรอให้ไมโครคอนโทรลเลอร์มาอ่านไป

- **Shift in Control Block**

ทำหน้าที่ควบคุมการ shift ข้อมูลจาก I2C-bus ในกรณีที่ I2C-bus controller อยู่ในโหมดของ การรับข้อมูล รูปที่ 3 - 47 และ 3 - 48 แสดง Block diagram และแผนภูมิสถานะสำหรับ Shift In Control



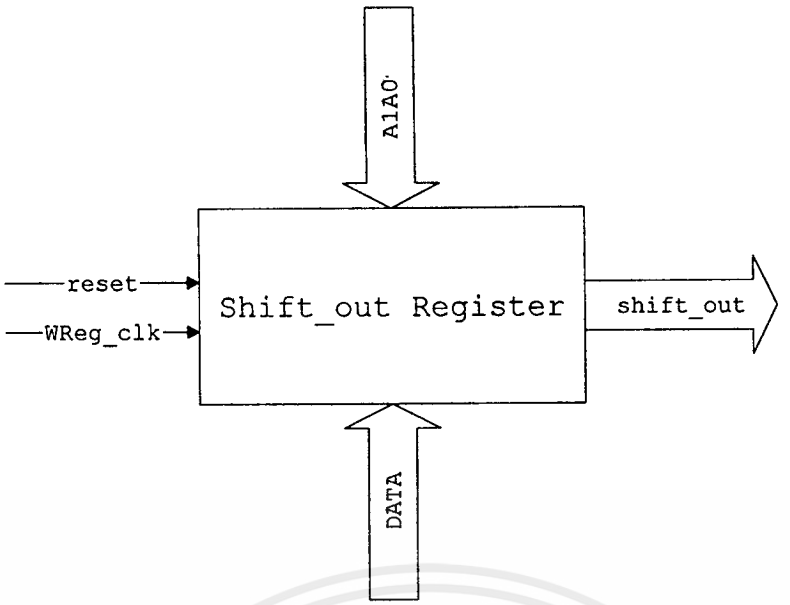
รูปที่ 3-47 แสดง Block diagram สำหรับ Shift In Control



รูปที่ 3-48 แผนภูมิสถานะของ Shift In Control

- **Shift out Register Block**

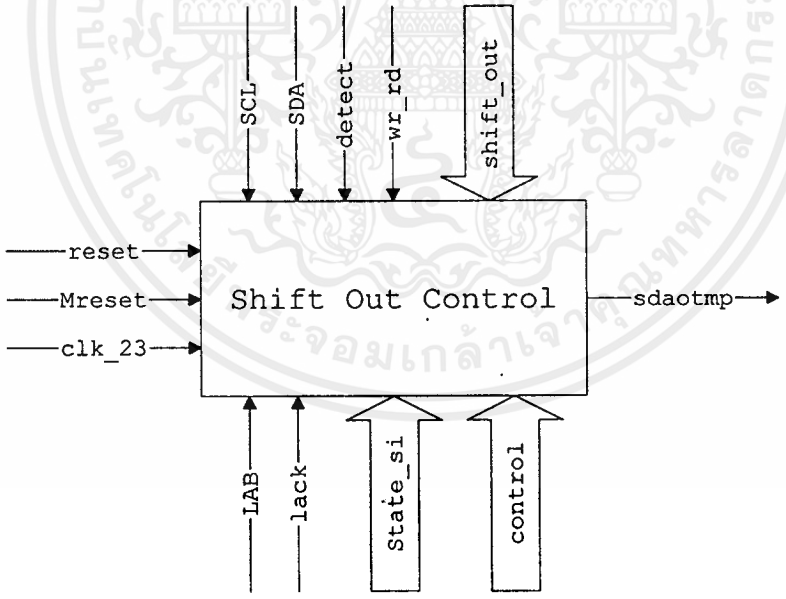
ทำหน้าที่เก็บข้อมูลที่ไมโครคอนโทรลเลอร์ส่งมา เพื่อที่จะส่งต่อไปให้ Shift Out Control นำส่งออกไปยังสาย SDA ต่อไป



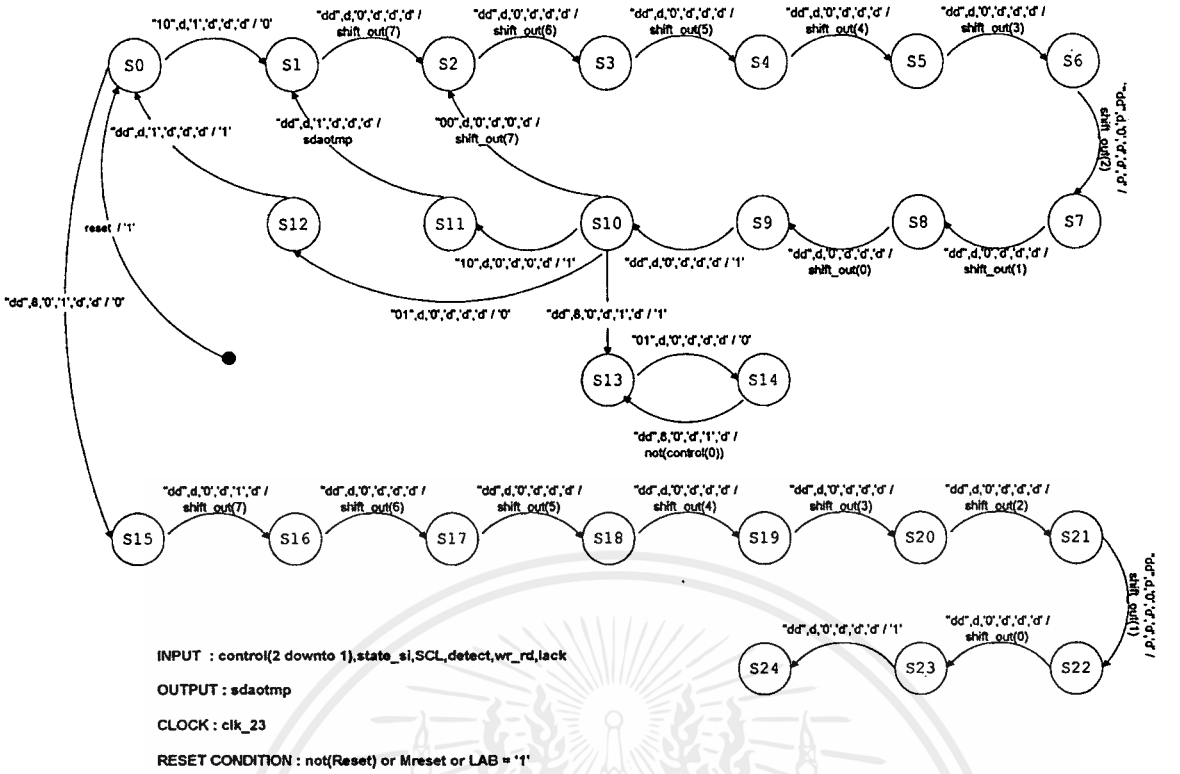
รูปที่ 3-49 แสดง Block diagram สำหรับ Shift_out Register

- **Shift Out Control Block**

ทำหน้าที่ควบคุมการส่งข้อมูลไปบนสาย SDA ซึ่งข้อมูลดังกล่าวจะมี ข้อมูลที่ไมโครคอนโทรลเลอร์ต้องการจะส่งออกไป , การตอบ Acknowledges ตัวส่ง , Start Condition , Stop Condition และ Repeated Start ซึ่งจะมีรายละเอียดของ Block Diagram และแผนภูมิสถานะเป็นดังรูปที่ 3-50 และ 3-51 ตามลำดับ



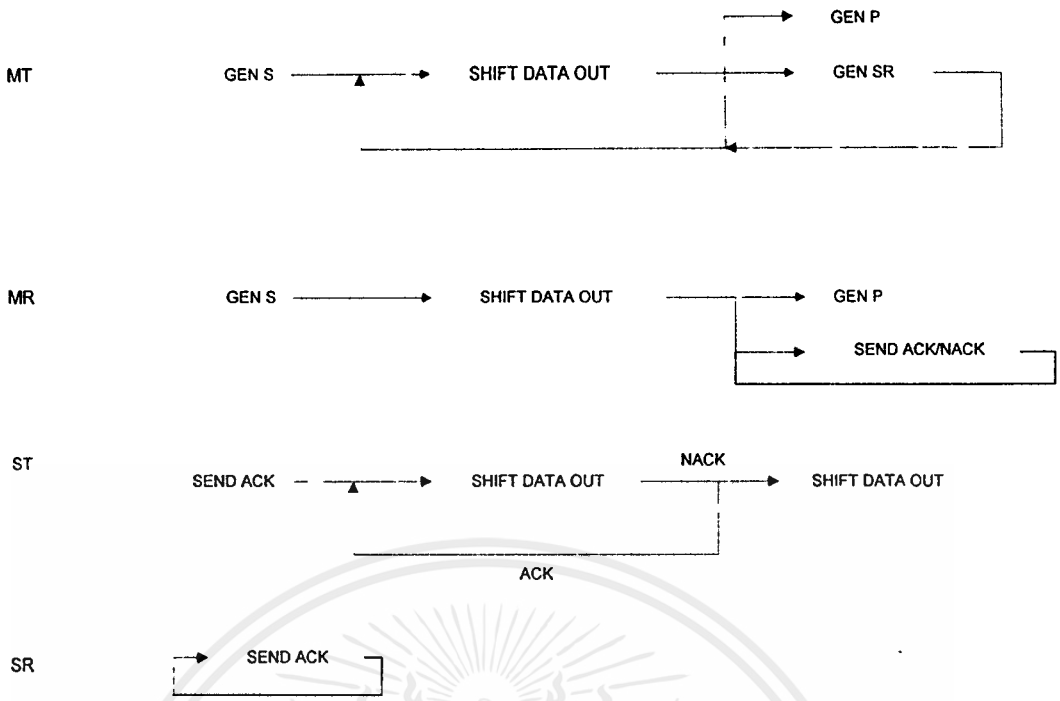
รูปที่ 3-50 แสดง Block Diagram สำหรับ Shift Out Control



รูปที่ 3-51 แสดงแผนภูมิสถานะสำหรับ Shift_out Control

หน้าที่ของ Shift Out Control จะขึ้นอยู่กับว่า I2C-bus control นี้ทำงานอยู่ในโหมดใด :-

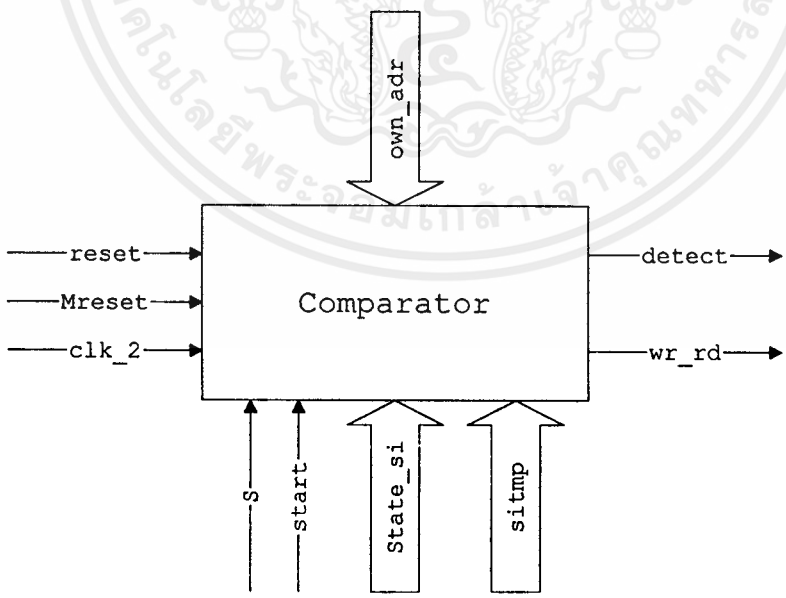
- **Transmitter mode :** จะทำหน้าที่นำข้อมูลที่ไมโครคอนโทรลเลอร์ต้องการส่งไปให้ตัวรับ นำลงไปในสาย SDA ทีละ บิท
- **Receiver mode :** จะทำหน้าที่ตอบ ACK ไปให้ตัวส่ง หรือตอบ NACK ไปให้ตัวส่ง เมื่อมีนต้องการข้อมูลอีกเพียงไบต์เดียว
- **Master mode :** จะทำหน้าที่ส่ง START Condition , REPEAT Condition หรือ STOP Condition ไปบนสาย SDA



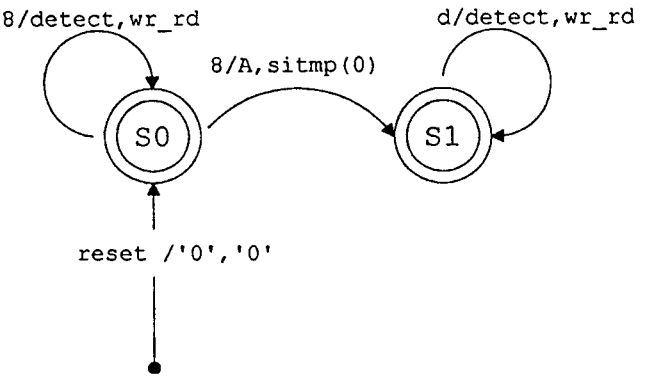
รูปที่ 3-52 แสดงโหมดการทำงานของ Shift Out Control Register

- **Comparator Block**

ทำหน้าที่เปรียบเทียบว่าแอดเดรสที่รับมานี้เป็นแอดเดรสของตัว core นี้หรือไม่ ใช้กรณีที่อยู่ในโหมดของสเลฟ ซึ่งจะเปรียบเทียบเพียง 7 บิท ส่วนบิทที่ 8 จะใช้บอกว่าจะเป็นการรับหรือการส่ง



รูปที่ 3-53 แสดง Block Diagram ของ Comparator



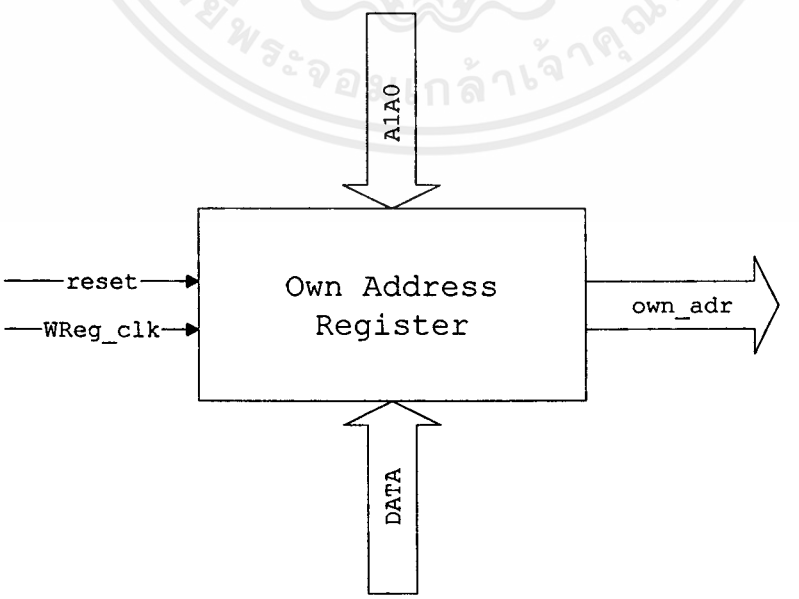

```
A = {'1' if sitmp(7 downto 1) = own_adr(6 downto 0)
      '0' if otherwise}
```

INPUT : state_si
OUTPUT : detect,wr_rd
CLOCK : clk_2
RESET CONDITION : not(Reset) or Mreset or not(start) or S = '1'

รูปที่ 3-54 แผนภูมิสถานะของ Comparator

- **Own Address Block**

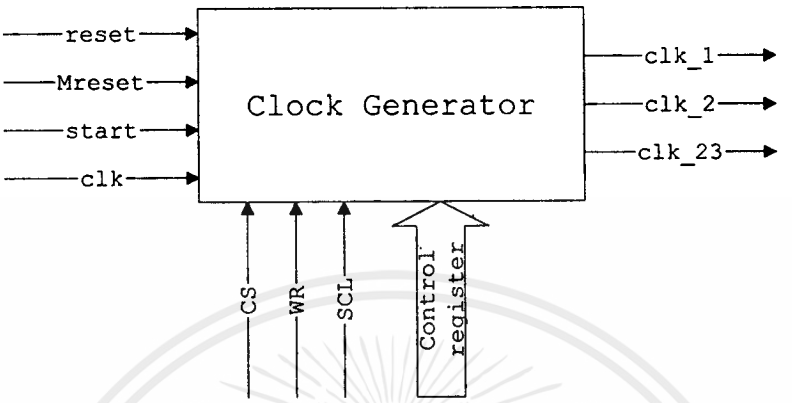
ทำหน้าที่เก็บแอดเดรสของ Microcontroller Interface Core ในกรณีที่เป็นสเลฟ ซึ่ง address as slave นี้จะมีขนาด 7 บิท บล็อกนี้สามารถใช้โปรแกรมกำหนดได้ เพราะฉะนั้น Slave address ของ Core นี้จะสามารถเปลี่ยนแปลงได้ขึ้นอยู่กับการโปรแกรม



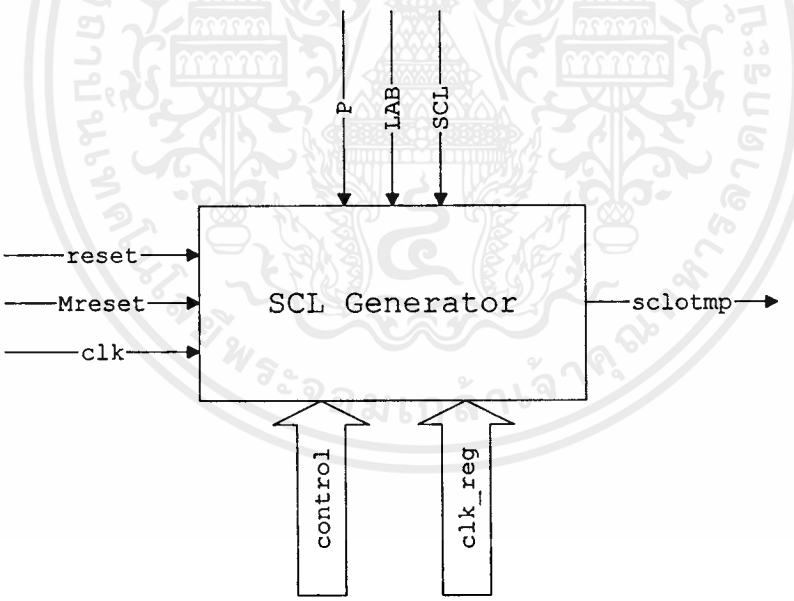
รูปที่ 3-55 แสดง Block Diagram สำหรับ Own Address Register

- **Clock Generator and SCL Generator Block**

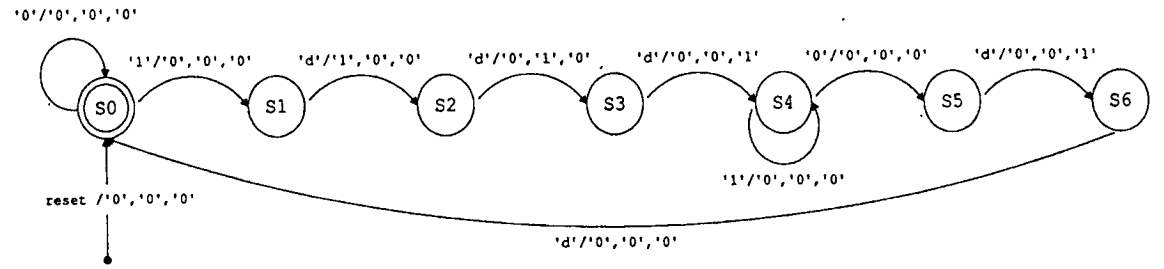
สำหรับ Clock Generator จะทำหน้าที่สร้างสัญญาณนาฬิกาให้กับบล็อก ต่าง ๆ ภายใน Core ส่วน SCL Generator จะทำหน้าที่สร้างสัญญาณนาฬิกาให้กับระบบ ในกรณีที่อยู่ในโหมดของมาสเตอร์โดยสัญญาณนาฬิกานี้สามารถกำหนดได้โดยโปรแกรมผ่าน Clock Register Block



รูปที่ 3-56 Block Diagram ของ Clock Generator



รูปที่ 3-57 Block Diagram ของ SCL Generator



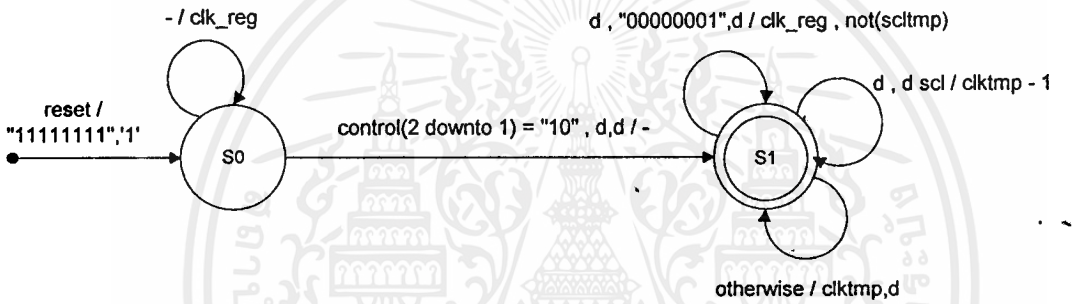
input : SCL

output : clk_1,clk_2,clk_23

clock : clk

reset condition : not(Reset) or Mreset or not(start or cs or wr or not(control(1) or control(0)) = '1'

รูปที่ 3-58 แสดงแผนภูมิสถานะของ Clock Generator

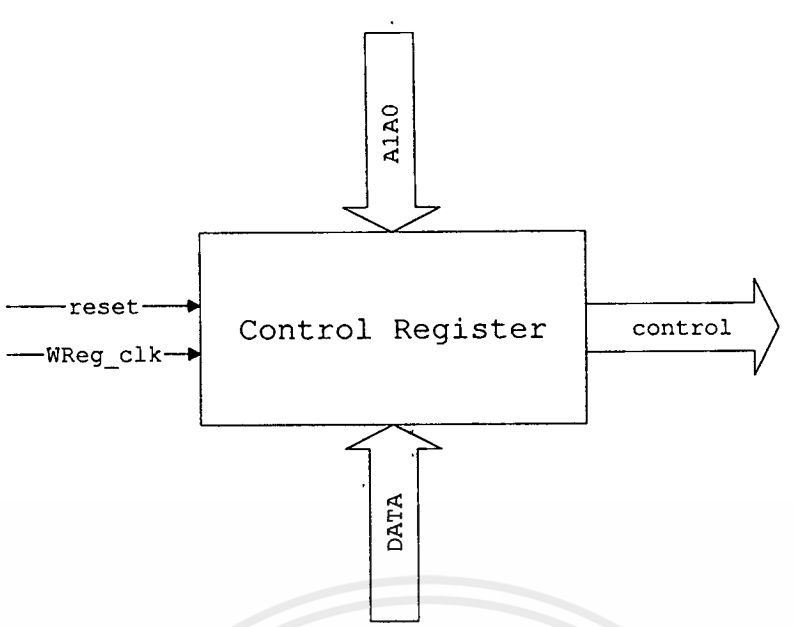


input : control(2 downto 1) , clktmp,scltmp
output : clktmp,scltmp
clock : clk
reset condition : not reset or Mreset or P or LAB = '1'

รูปที่ 3-59 แสดงแผนภูมิสถานะของ SCL Generator

- **Control Register Block**

เป็นส่วนที่ใช้บอกโหมดการทำงานของ I^2C-bus Controller ซึ่งในส่วนนี้สามารถโปรแกรมได้



รูปที่ 3-60 แสดง Block Diagram สำหรับ Control Register

X	X	X	MRES ET	ENI	STA	STO	ACK

รูปที่ 3-61 แสดง บิท ต่าง ๆ ใน Control Register

การทำงานแต่ละบิทสำหรับ Control Register เป็นดังต่อไปนี้ :-

- **ACK (Acknowledge : บิท 0) :** โดยปกติจะมีค่าเป็น ' 1 ' เป็นบิทที่กำหนดให้ I2C-bus controller ทำการตอบ Acknowledge โดยอัตโนมัติ เมื่อรับข้อมูลแต่ละไบต์มาเรียบร้อยแล้ว และจะมีค่าเป็น ' 0 ' เมื่ออยู่ในโหมดของตัวรับที่เป็นมาสเตอร์ และต้องการข้อมูลอีกเพียงไบต์เดียวจาก ตัวส่งที่เป็นสเลฟ ซึ่งการทำเช่นนี้จะทำให้บน I2C-bus จะเป็นการ negative acknowledge ซึ่งจะทำให้ทางฝ่ายที่เป็นสเลฟจะหยุดการส่งข้อมูล
- **STA and STO (Start and Stop Condition : บิท 2 and บิท 1) :** 2 บิทนี้จะทำหน้าที่ควบคุมการสร้าง Start Condition บน I2C-bus และการส่ง slave address และ R/W bit ควบคุมการสร้าง Repeat Start Condition และ การสร้าง Stop Condition

ตาราง 3-4 แสดงโหมดการทำงานของบิท STA และ STO

STA	STO	PRESENT MODE	FUNCTION	OPERATION
1	0	SLV/REC	START	transmit START + address, remain MST/TRM if R/W = 0; go to MST/REC if R/W = 1
1	0	MST/TRM	REPEAT START	same as for SLV/REC
0	1	MST/REC; MST/TRM	STOP READ; STOP WRITE	transmit STOP go to SLV/REC mode;
1	1	MST	DATA CHAINING	send STOP;START and address after last master frame without STOP sent
0	0	ANY	NOP	no operation

- **ENI (Enable Interrupt : บิท 3) :** บิทนี้จะถูกกำหนดเป็น ' 1 ' เมื่อต้องการใช้หรือติดต่อกับไมโครคอนโทรลเลอร์ โดยการส่งสัญญาณอินเตอร์รัปท์ แต่ถ้าบิทนี้ถูกกำหนดเป็น ' 0 ' ไมโครคอนโทรลเลอร์จะทำการติดต่อด้วยโดยการ polling
- **MRESET (Master Reset : บิท4) :** บิท นี้ใช้ในการรีเซ็ท (reset) ตัววงจรด้วยโปรแกรม

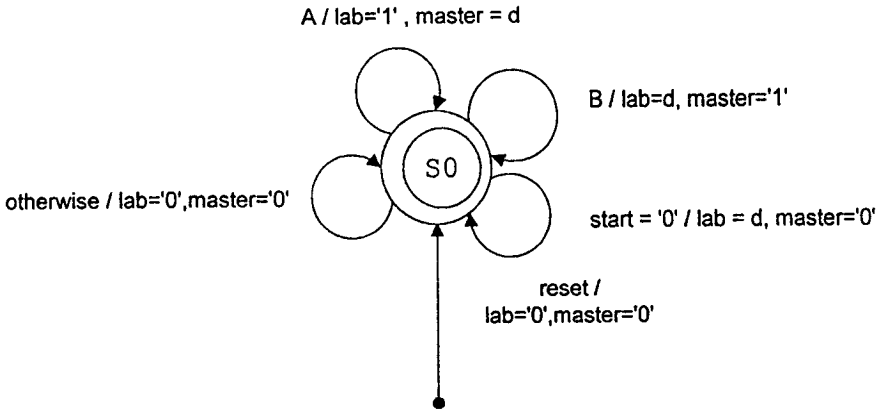
- **Status Register Block**

PIN	X	STS	X	AD0/ LRB	AAS	LAB	BB

รูปที่ 3-62 แสดง บิท ต่าง ๆ ใน Status Register

การทำงานในแต่ละบิทสำหรับ Status Register เป็นดังนี้ :-

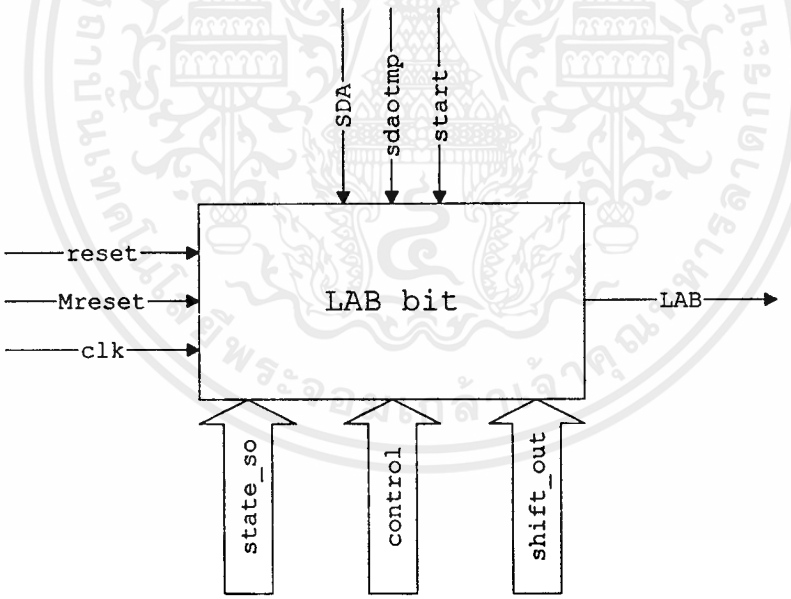
- **BB (Bus Busy : บิท 0) :** เป็นบิทที่ใช้บอกว่า I2C-bus ถูกใช้งานอยู่ โดยบิทนี้จะถูกกำหนดเป็น '0' เมื่อ I2C-bus ถูกใช้งานซึ่งจะไม่สามารถใช้ I2C-bus ได้ บิทนี้จะถูก Set/Reset โดย STOP/START Condition
- **LAB (Lost Arbitration : บิท1) :** บิทนี้จะมีค่าเป็น ' 1 ' เมื่อในระบบมีอุปกรณ์ที่เป็นมาสเตอ อยู่หลายตัว จึงต้องมีการแข่งกันแย่งใช้สาย แล้วเกิดแพ้ให้กับมาสเตอร์ตัวอื่น



input : A , B , start
output : lab , master
clock : clk
reset condition : not reset or Mreset or not start = '1'

A =lab='0' & sda='0' & sdotmp='1' & not(state_so=10 or state_so=24) & master='1'
B =master='0' & start='1' & control(2 downto 1)="10" & shift_out(0)='0'

รูปที่ 3-63 แผนภูมิสถานะของ LAB bit in Status Register

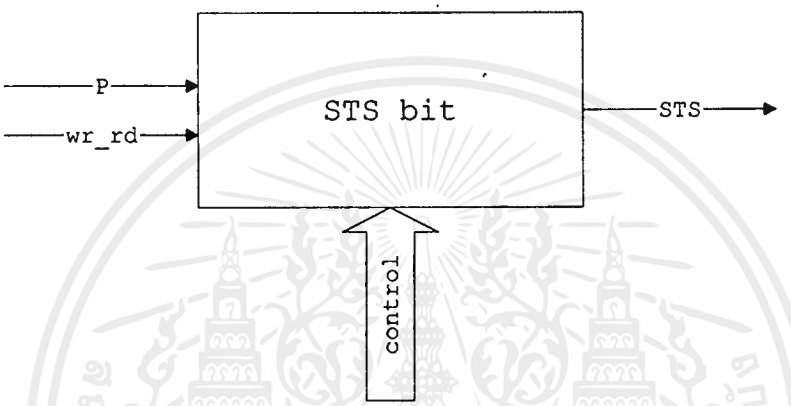


รูปที่ 3-64 Block Diagram of LAB bit ใน Status Register

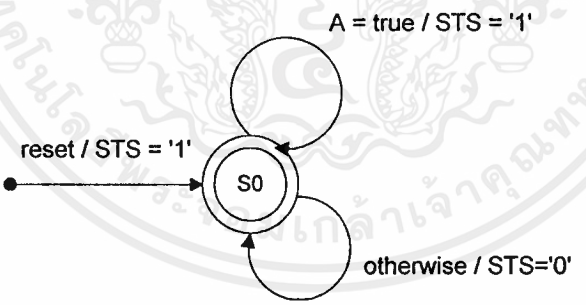
- **AAS (Address As Slave : บิท2) :** ใช้ได้เมื่อ PIN เป็น ' 0 ' เท่านั้น บิทนี้จะถูกกำหนดให้เป็น ' 1 ' เมื่อแอดเดรสที่รับเข้ามาเป็นแอดเดรสของตัวเอง จะใช้ในโหมดที่เป็นสเลฟ เท่านั้น
- **LRB/AD0 (Last Received Bit or Address 0 bit : บิท 3) :** ใช้เมื่อ PIN เป็น ' 0 ' เท่านั้น โดย

1. LRB จะเก็บบิทสุดท้ายที่รับมาบน I2C-bus ในขณะที่ AAS = ' 0 ' โดยปกติแล้วจะเป็นค่า Acknowledge ของอุปกรณ์ที่เป็นสเลฟ ดังนั้นถ้าจะทำการตรวจสอบว่ามีการ ACK จากสเลฟหรือไม่ สามารถตรวจสอบผ่านบิทนี้ได้
2. เมื่อ AAS เป็น 1 I2C-bus Controller จะถูกกำหนดให้เป็นสเลฟ บิทนี้จะถูกกำหนดให้เป็น AD0 และจะถูกเซ็ทให้เป็น ' 1 ' เมื่อได้รับแอดเดรส 00H (general call) และจะเป็น '0 ' ได้รับแอดเดรสที่เป็นของตัวเอง

- **STS (บิท 5) :** ในโหมดตัวรับที่เป็นสเลฟ จะทำการเซ็ทบิทนี้ให้เป็น '1 ' เมื่อ พบ STOP Condition



รูปที่ 3-65 แสดง Block Diagramของ STS bit ใน Status Register

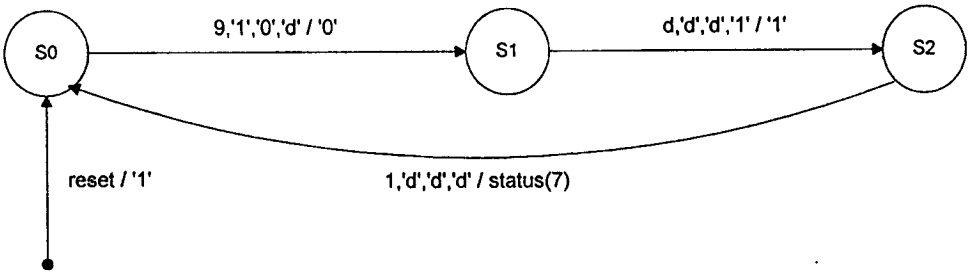


A = P='1' & not (control(2 downto 1)="01") & wr_rd = '0'

รูปที่ 3-66 แผนภูมิสถานะของ LAB bit in Status Register

- **PIN (Pending Interrupt Not : บิท 7) :** บิทนี้จะมีการทำงานดังต่อไปนี้ :-
  1. เมื่อ Power On บิทนี้จะมีค่าเป็น ' 1 '
  2. เมื่อมีการ reset หรือ master reset จะมีค่าเป็น ' 1 '
  3. เมื่อส่งข้อมูลแต่ละไบต์เสร็จแล้ว บิทนี้จะมีค่าเป็น ' 1 ' และจะมีค่าเป็น ' 0 ' เมื่อไมโครโปรเซสเซอร์เขียนค่าใหม่ (WRITE ค่าลง register ใดก็ได้)

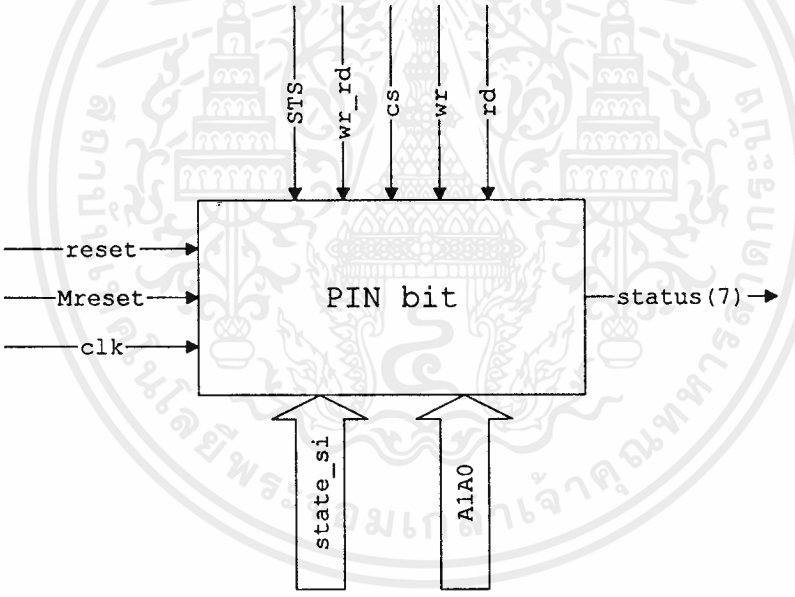
4. เมื่อรับข้อมูลแต่ละไบต์ เสร็จแล้วบิทนี้จะมีค่าเป็น ' 1 'และจะมีค่าเป็น ' 0 ' เมื่อไมโครโปรเซสเซอร์ทำการอ่านค่าใน Shift In Register




```
input  : state_si,STS,wr_rd,A
output : PIN
clock  : clk
reser condition : not reset or Mreset = '1'

A = not(CS) & (not(WR) or (not(RD) & not(A1A0(1) & A1A0(0))))
```

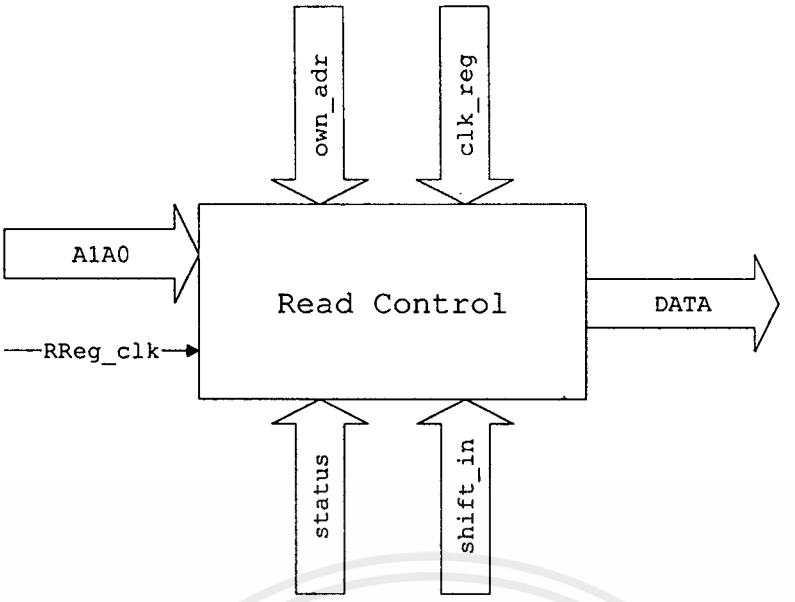
รูปที่ 3-67 แผนภูมิสถานะของ PIN bit ใน Status Register



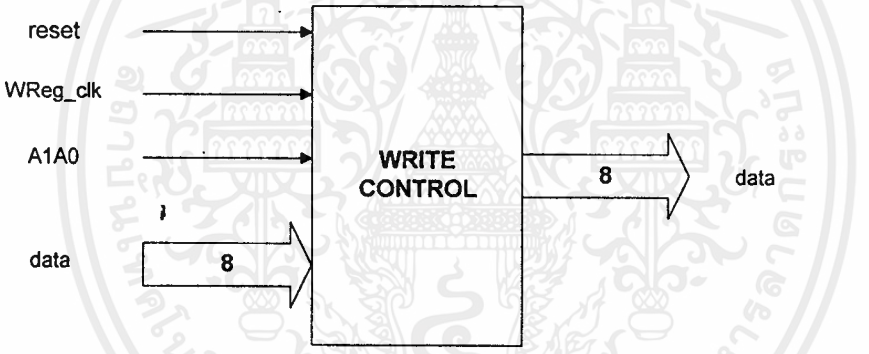
รูปที่ 3-68 แสดง Block diagram สำหรับบิท PIN ใน Status Register

- **CPU Interface Block**

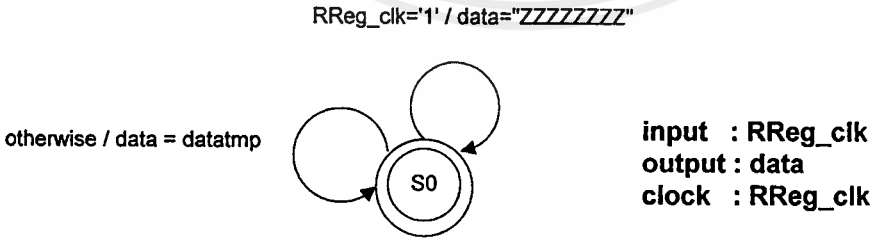
เป็นส่วนที่ใช้ในการติดต่อกับไมโครคอนโทรลเลอร์โดยตรง ไม่ว่า I2C-bus controller ทำอะไรกับไมโครคอนโทรลเลอร์ จะต้องทำผ่านบล็อกนี้ ในบล็อกนี้จะมีบล็อกย่อยที่สำคัญอยู่ 2 บล็อกได้แก่ Read Control และ Write Control ซึ่งจะมี Block Diagram เป็นดังรูปที่ 3- 69 และ 3- 70 ตามลำดับ และจะมีแผนภูมิสถานะแสดงดังรูปที่ 3-71 และ 3-72 สำหรับ Read Control และ Write Control ตามลำดับ



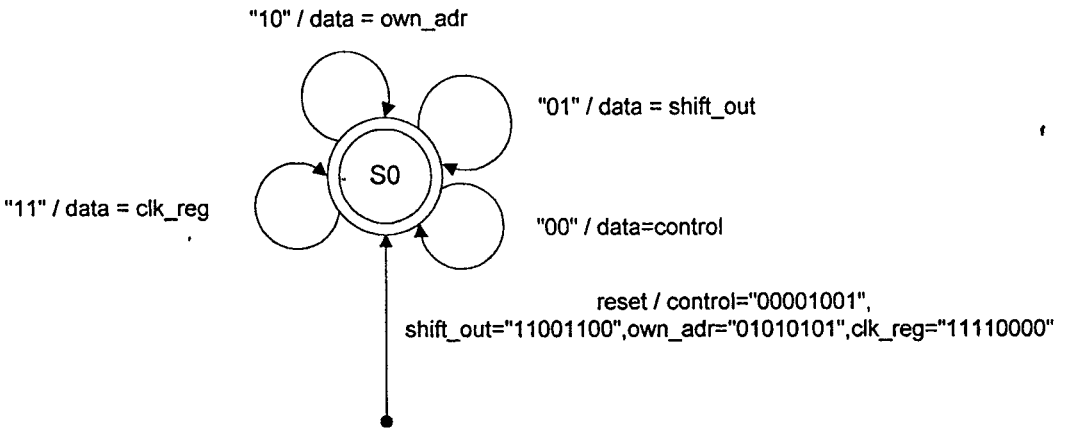
รูปที่ 3-69 แสดง Block Diagram ของ Read Control ใน CPU Interface



รูปที่ 3-70 แสดง Block Diagram ของ Write Control ใน CPU Interface



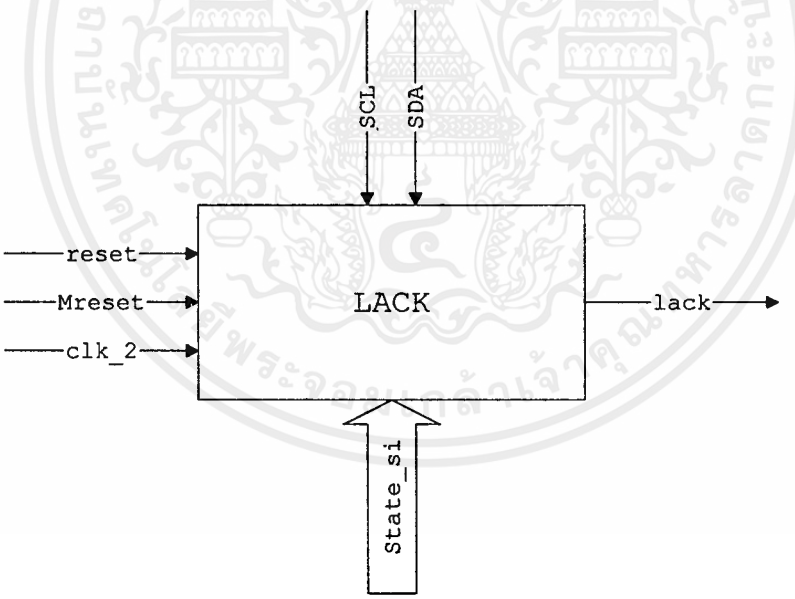
รูปที่ 3-71 แสดงแผนภูมิสถานะสำหรับ Read Control ใน CPU Interface



input : A1A0
output : data
clock : WReg_clk
reset condition : reset = '0'

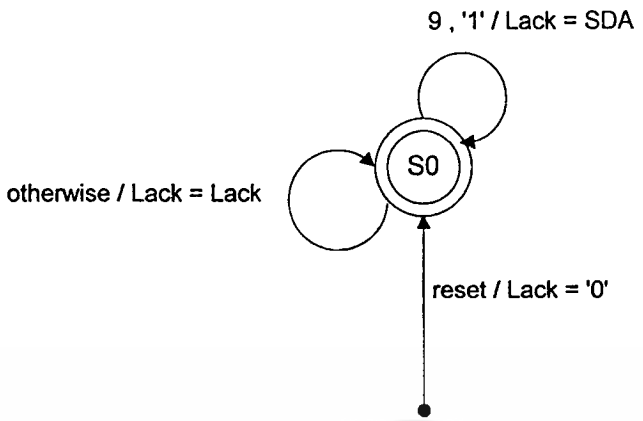
รูปที่ 3-72 แสดงแผนภูมิสถานะสำหรับ Write Control ใน CPU Interface

- **LACK**



รูปที่ 3-73 แสดงลักษณะของ Block Diagram สำหรับ LACK

- **Designed State Machine**



input : state_si , SDA
output : LACK
clock : clk_s
reset condition : not reset or Mreset = '1'

รูปที่ 3-74 แสดงแผนภูมิสถานะสำหรับ LACK

ตาราง 3-5 Core Signal Pinout of Microcontroller Interface with I2C-bus

Signal	Signal Direction	Description
System Interface Signal		
SDA_O	Output	Serial Data line Out.
SDA	Input	Serial Data Line In
SCL_O	Output	Serial Clock Line Out. It is used in master mode.
SCL	Input	Serial Clock Line .
Microcontroller Interface Signal		
WR	Input	WR control signal, active low.
RD	Input	RD control signal, active low.
CS	Input	Chip Select.
INT	Output	Interrupt Signal. It is used when it want to service from microcontroller.
A1A0	Input	It is used to select register in this core
DATA	In/Out	Bidirection data bus carries data to microcontroller or from microcontroller.

3.3 การทดสอบการทำงาน

ได้แบ่งการทดสอบออกเป็น 2 ส่วน ได้แก่

- การทดสอบในระดับ Software
- การทดสอบในระดับ Hardware

3.3.1 การทดสอบในระดับ Software

งานในขั้นนี้จะแบ่งเป็นการทดสอบก่อนการสังเคราะห์และการทดสอบหลังการสังเคราะห์ ซึ่งการทดสอบทั้งสองครั้งจะมีสิ่งที่ต้องทดสอบเหมือนกันคือ จะทำการตรวจสอบการทำงานของวงจร 3 ตัวดังรายละเอียดต่อไปนี้

3.3.1.1 การทดอบการทำงานของ UART

ได้แบ่งการทดสอบเป็นของแต่ละ บล็อก ดังต่อไปนี้

1) **Status Register** ทดสอบว่าแต่ละบิทใน status register มีการเปลี่ยนแปลงค่าได้ถูกต้องตามข้อผิดพลาดที่เกิดขึ้นหรือไม่ โดยมีการทดสอบดังนี้

- RDRF จะต้องเป็น '0' ในขณะที่ chip reset หรือ power-on หรือ CPU อ่านค่าจาก Receive register และจะเป็น '1' เมื่อรับบิทข้อมูลครบ 1 เฟรม
- TDRF จะต้องเป็น '0' ในขณะที่ chip reset หรือ power-on หรือ CPU เขียนค่าใส่ Transmit register และจะเป็น '1' เมื่อส่งบิทข้อมูลครบ 1 เฟรม
- PE (parity error) จะต้องเป็น '0' ในขณะที่ chip reset หรือ power-on และจะเป็น '1' เมื่อบิทพาริตี้ที่รับเข้ามาผิดพลาด
- FE (frame error) จะต้องเป็น '0' ในขณะที่ chip reset หรือ power-on และจะเป็น '1' เมื่อ frame ที่รับเข้ามาผิดพลาด ซึ่งเกิดจากการมี stop bit ไม่ครบตามที่กำหนด
- OV (overrun) จะต้องเป็น '0' ในขณะที่ chip reset หรือ power-on และจะเป็น '1' ถ้าข้อมูลใหม่เข้ามาในขณะที่ยังไม่ได้อ่านข้อมูลเก่า
- IRQ (interrupt request) จะต้องเป็น '0' ในขณะที่ chip reset หรือ power-on และเป็น '1' เมื่อ UART ต้องการ อินเตอร์รัปท์

2) **Tx and Rx Clock Generator** จะทดสอบว่ามีการสร้างสัญญาณนาฬิกาตามที่ได้กำหนดไว้ใน Control register หรือไม่

3) **Transmit control** ภายในจะประกอบด้วย Transmit Shift register และสร้างพาริตี้ ดังนั้นจะสามารถส่งเฟรมของข้อมูลได้อย่างถูกต้องตามโหมดการทำงานที่กำหนดไว้ใน Control Register และจะต้องสร้างบิทพาริตี้ที่ถูกต้องด้วย

4) **Receive control** ภายในจะประกอบด้วย Receive Shift register และ parity checker ดังนั้นจะต้องตรวจสอบความถูกต้องของข้อมูลที่รับเข้ามา

3.3.1.2 การทดสอบการทำงานของ I²C-bus memory Interface

จะมีรายละเอียดการทดสอบตามตารางที่ 3 - 6

ตาราง 3-6 แสดงการทดสอบการทำงานของ I²C-bus memory Interface

Component	การทำงานที่ถูกต้องของแต่ละตัว
Start	detect S , detect Sr
Stop	detect P
Comparator register	ส่งสัญญาณ detect เป็น 1 เมื่อ เป็น แอดเดรส ของตัวเอง
	ส่งสัญญาณ detect เป็น 0 เมื่อ ไม่เป็น แอดเดรส ของตัวเอง
Address Register	เก็บ first address ของการ write หรือ read ได้
	บวกค่าของ แอดเดรส ได้ทุกครั้งเมื่อมีการ write หรือ read เสร็จ
Shift-in Register	รับ บิท ต่าง ๆ จากสาย SDA ในขณะที่ SCL เป็น HIGH ได้ถูกต้อง
Shift-out Register	ส่งสัญญาณ ACK ได้เมื่อรับข้อมูลครบ 8 บิท
	ทำการส่งข้อมูลเมื่ออยู่ในโหมดของตัวส่งที่เป็นสเลฟ
Data Register	ทำการรับข้อมูลเมื่ออยู่ในโหมดของตัวรับที่เป็นสเลฟ ได้
Clock Generator	สามารถสร้างสัญญาณนาฬิกาให้กับบล็อกต่าง ๆ ทำงานได้ ตามเวลาที่เหมาะสม
Rd/Wr Control	การส่งสัญญาณ read หรือ write ไปบอกหน่วยความจำเมื่อได้รับสัญญาณจาก Clock Generator

3.3.1.3 การทดสอบ I²C-bus controller Interface

ตาราง 3-7 แสดงการทดสอบการทำงานของ I²C-bus controller Interface

Component	การทำงานที่ถูกต้องของแต่ละตัว
Shift in Control	จะต้องทำการ shift ข้อมูลจากสาย SDA เข้ามาเมื่ออยู่ใน receiver mode
Shift out Control	จะต้องทำการส่งข้อมูลลงไปบนสาย SDA ตามช่วงเวลาที่ SCL เป็น LOW ทีละบิท ใน Transmitter mode และจะต้องทำการ Acknowledge เมื่ออยู่ใน Receiver mode ได้
Comparator	ทำการเปรียบเทียบแอดเดรส เมื่อทำงานอยู่ใน Slave receiver mode โดยเปรียบเทียบกับค่า แอดเดรส ที่อยู่ใน own address register
Clock Generator	ทำการสร้างสัญญาณนาฬิกาให้กับ บล็อก ต่าง ๆ ได้
SCL Generator	สร้างสัญญาณความถี่ของสาย SCL ได้ตามที่กำหนดไว้ใน Clock Register เมื่อทำงานอยู่ใน โหมดของมาสเตอร์
Status Register	เมื่อเกิดเหตุการณ์ต่าง ๆ ในขณะทำการสื่อสารข้อมูล ก็สามารถรายงานสถานะของสาย ได้ถูกต้องตามเหตุการณ์ที่เกิดขึ้น

3.3.2 การทดสอบในระดับฮาร์ดแวร์

การทดสอบทางฮาร์ดแวร์ของวงจรทั้ง 3 จะต้องนำวงจรทั้ง 3 วงจรที่ออกแบบไว้มาเชื่อมต่อกับไมโครคอนโทรลเลอร์ซึ่งเป็นไปตามวัตถุประสงค์ของโปรเจคนี้ การทดสอบวงจรแต่ละวงจรก็มีรายละเอียดต่าง ๆ กันดังนี้

3.3.2.1 การทดสอบ UART

ขั้นตอนการทดสอบ UART จะนำ FPGA ที่ถูกโปรแกรมเป็น UART มาต่อกับวงจรของไมโครคอนโทรลเลอร์ 89C51 ในส่วนของ System Interface และในส่วนของการสื่อสารจะต่อกับพอร์ตสื่อสารแบบอนุกรมของ คอมพิวเตอร์ การตรวจสอบความถูกต้องของการสื่อสารจะใช้โปรแกรม Xtalk เพื่อดูค่าของข้อมูลที่สื่อสารกับไมโครคอนโทรลเลอร์

3.3.2.2 การทดสอบ I^2C-bus memory Interface

ขั้นตอนการทดสอบ I2C Bus Memory Interface จะนำ FPGA ที่ถูกโปรแกรมเป็น I2C Bus Memory Interface มาต่อกับ Memory และส่วนของการสื่อสารจะต่อกับ I2C Bus เพื่อทำการสื่อสารกับ I2C Bus Controller การทดสอบจะต้องทดสอบการทำงาน 2 ลักษณะคือ

1. Slave Receiver
2. Slave Transmitter

3.3.2.3 การทดสอบ I^2C-bus Controller

ขั้นตอนการทดสอบ I2C Bus Controller จะนำ FPGA ที่ถูกโปรแกรมเป็น I2C Bus Controller มาต่อกับไมโครคอนโทรลเลอร์ 89C51 ในส่วนของ System Interface และในส่วนของการสื่อสารจะต่อกับ I2C Bus ซึ่งจะต้องสื่อสารกับ Serial EPROM ที่เป็น I2C ได้ การทดสอบจะต้องทำงานได้ใน 2 ลักษณะคือ

1. Master Transmitter
2. Master Receiver

บทที่ 4

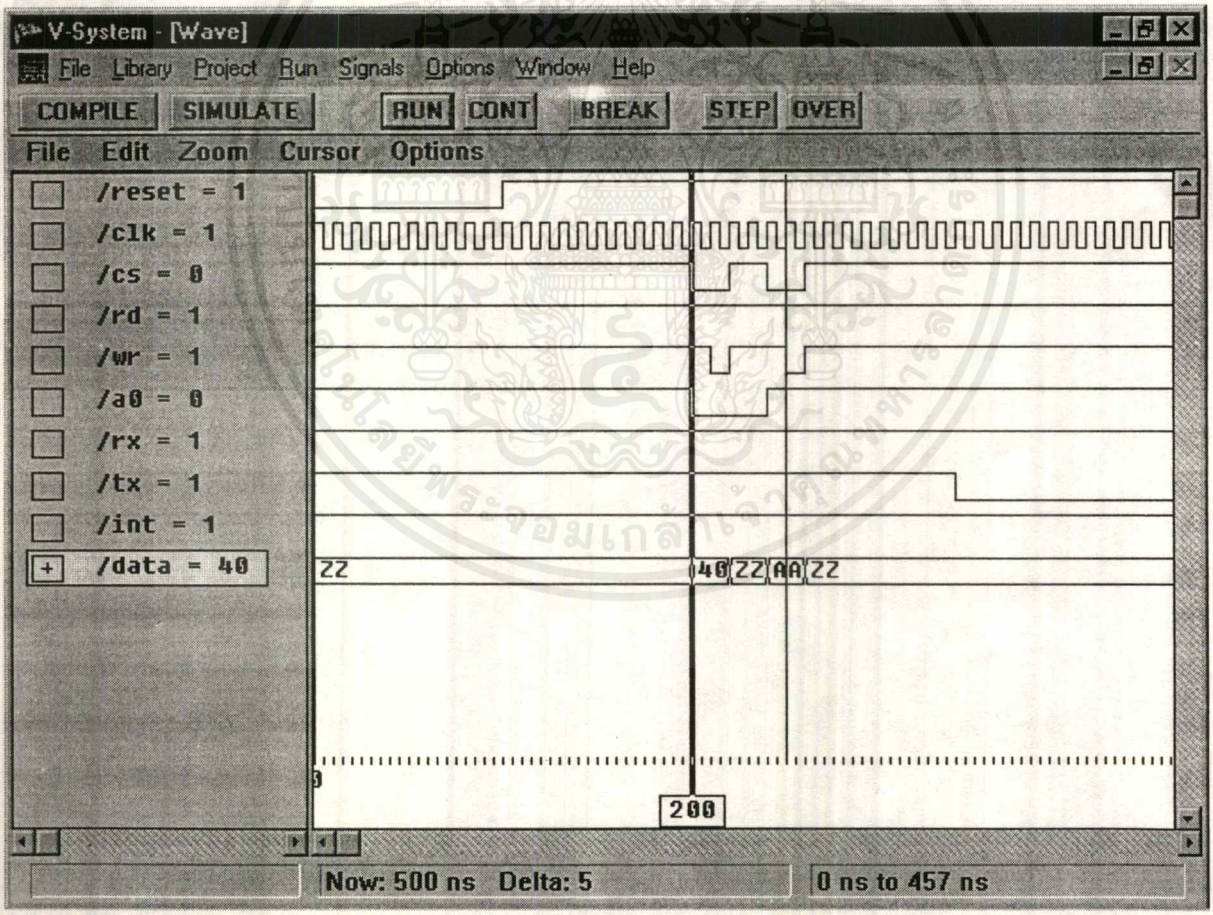
ผลการทดสอบ

ในบทที่ผ่านมาได้กล่าวถึงขั้นตอนการทดสอบวงจรทั้ง 3 ตัว ทั้งในระดับของซอฟท์แวร์และฮาร์ดแวร์ สำหรับบทนี้จะขอกล่าวถึงผลการทดสอบที่ได้

4.1 Functional Simulation

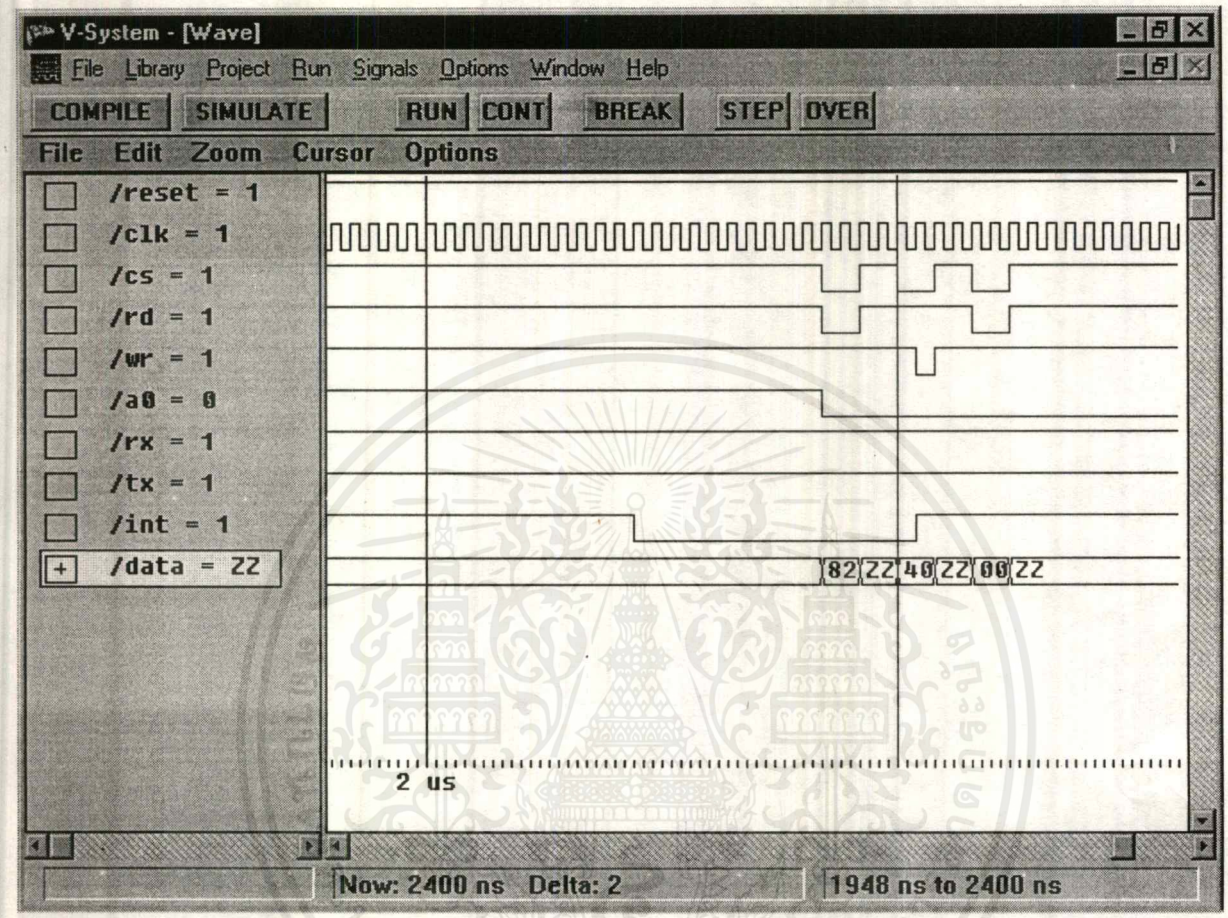
หลังจากที่ได้ทำการเขียน code เป็นภาษา VHDL และทำการ Simulation แล้วผลการทดลองของ Chip Core แต่ละตัวเป็นดังนี้

1) ผลการทดสอบสำหรับ UART ในการทดสอบการเป็นฝ่ายส่ง ได้กำหนดให้ส่งข้อมูลขนาด 1 ไบต์ ไม่มี parity bit และ มี 2 stop bits และอนุญาตให้มีการอินเตอร์รัปต์ ได้ รูปที่ 4 - 1 จะแสดงผลก่อนการส่งข้อมูล ซึ่งก่อนการส่งจะมีการเขียนไปที่ Control Register ว่าจะส่งแบบใด หลังจากนั้นจึงเขียนค่าที่ส่งลงไปที่ Transmit Register ซึ่งในการทดสอบจะให้ส่งค่า AAH (ดูที่ data)



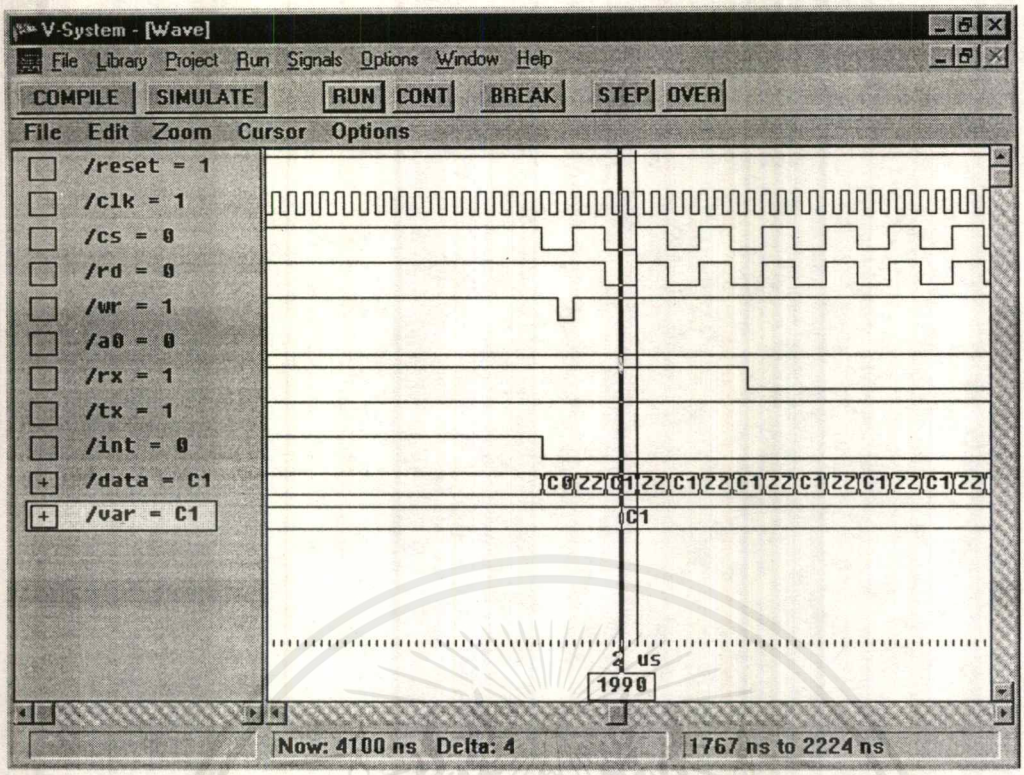
รูปที่ 4-1 แสดงผลการทดสอบก่อนการส่งข้อมูล

รูปที่ 4-2 แสดงผลของการส่งของ UART สังเกตที่ data จะเห็นค่า 82H ซึ่ง read มาจาก Status Register หลังจากนั้นก็เขียนค่า 40H กลับไปที่ Control Register เพื่อที่เริ่มทำการส่งใหม่ จากนี้จะสังเกตเห็นว่าค่าใน Status Register จะถูกเคลียร์ด้วย

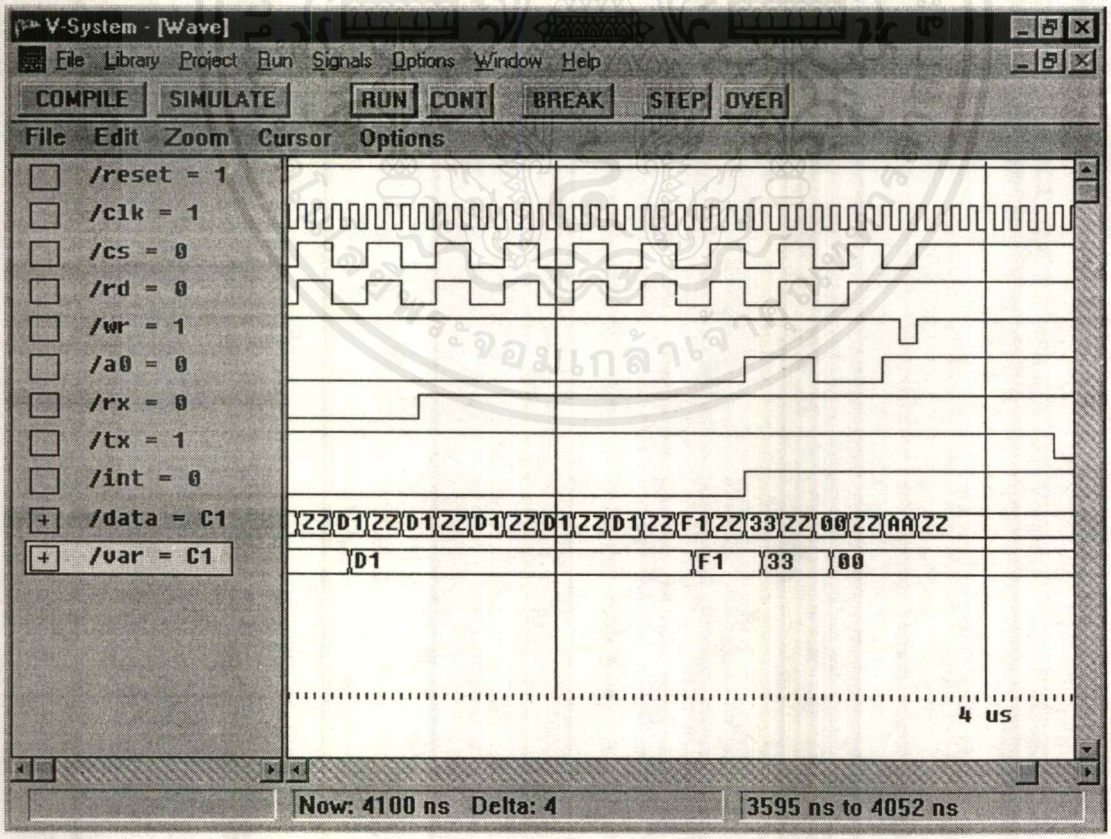


รูปที่ 4-2 แสดงผลการทดสอบหลังการส่งข้อมูล

รูปที่ 4-3 และ 4-4 จะเป็นการรับข้อมูลของ UART ซึ่งจะดูการเปลี่ยนแปลงของ Status Register ในบางกรณีเท่านั้น ซึ่งในการทดสอบจะทำการอ่าน Status Register แล้ว จึ่งอ่าน Data จากนั้นก็กลับไปอ่าน Status อีกครั้ง ซึ่งในรูปที่ 4-3 จะแสดงการเกิด Parity Error (ที่ data จะเห็นค่า C1H) ส่วนรูปที่ 4-4 จะแสดงการเกิด Frame Error (ที่ data จะเห็นค่า F1H) หลังจากอ่านข้อมูลเข้ามาแล้ว ค่าใน Status Register จะถูกเคลียร์



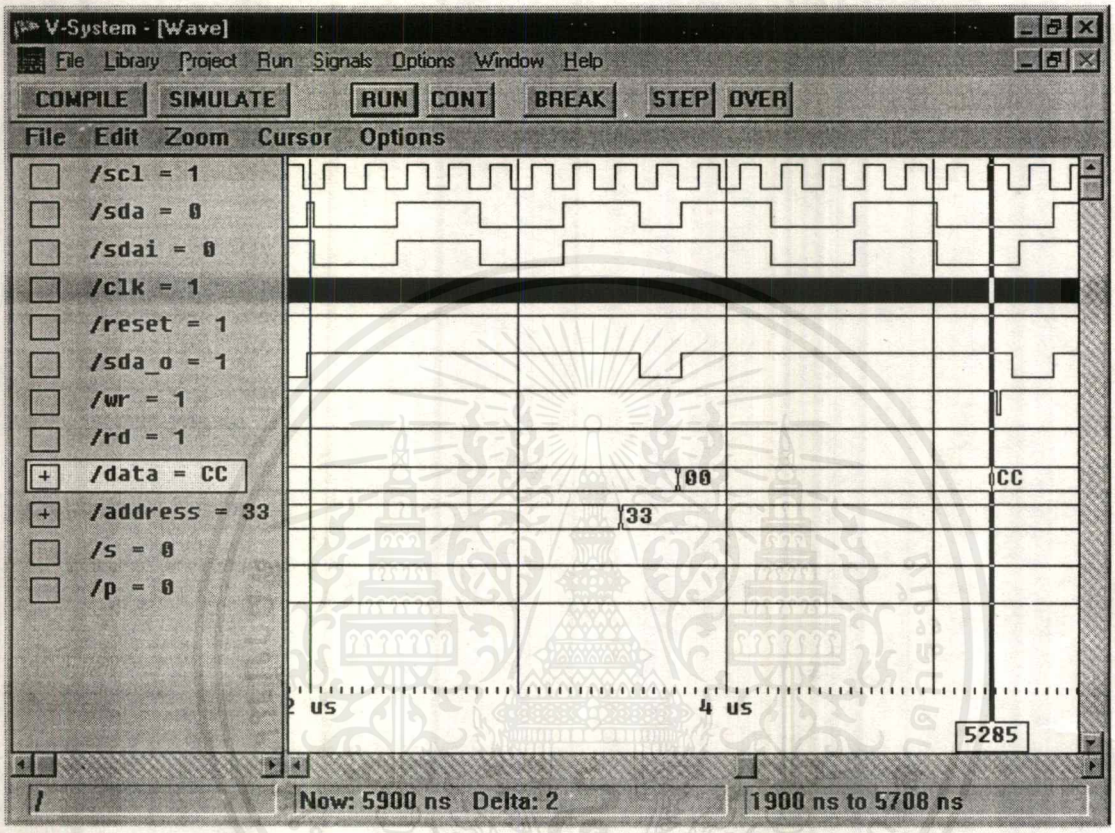
รูปที่ 4-3 แสดงผลการทดสอบการรับข้อมูลเมื่อเกิด Parity Error



รูปที่ 4-4 แสดงผลการทดสอบการรับข้อมูลเมื่อเกิด Frame Error

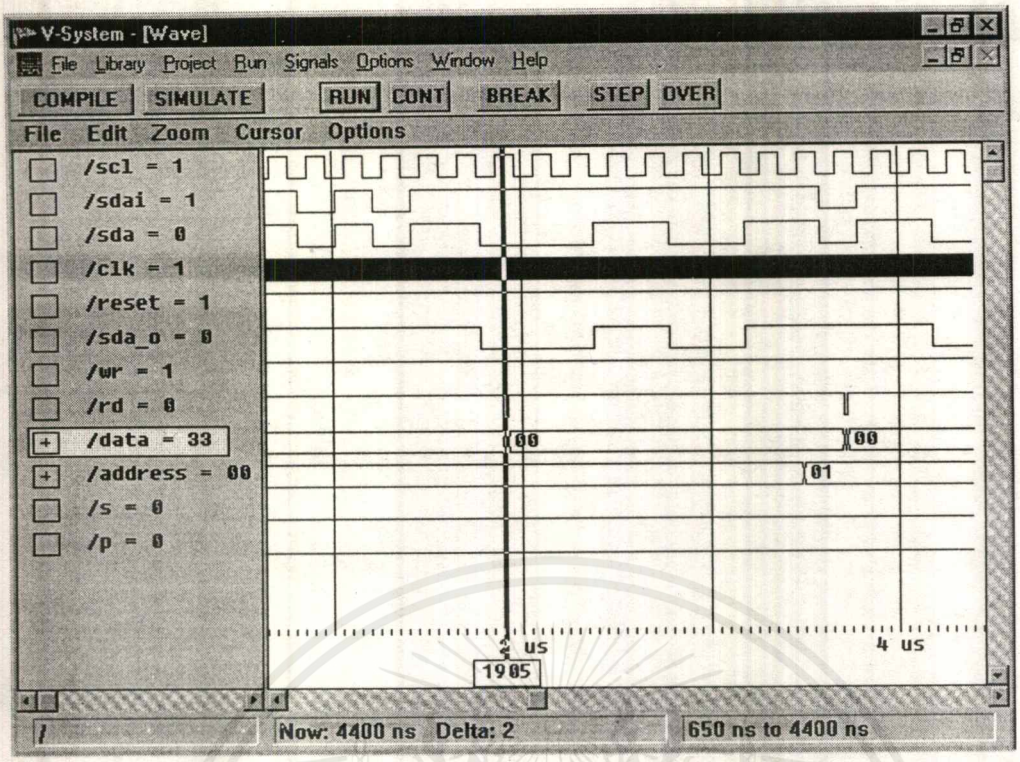
2) ผลการทดสอบสำหรับ I^2C-bus memory Interface จะมีการทดสอบ 2 modes การทำงาน คือ

● Slave receiver ครั้งแรกที่รับข้อมูล จะต้องดูว่าเป็นแอดเดรสของตัวเองหรือไม่ หลังจากนั้นก็รับข้อมูลที่เป็นแอดเดรสเริ่มต้นที่จะทำการเขียนข้อมูลลงไปในหน่วยความจำ ซึ่งในการทดสอบให้เริ่มที่ แอดเดรส 33H แล้วจากนั้นจึงเป็นข้อมูลที่จะเขียน ผลการทดสอบเป็นดังรูปที่ 4-5

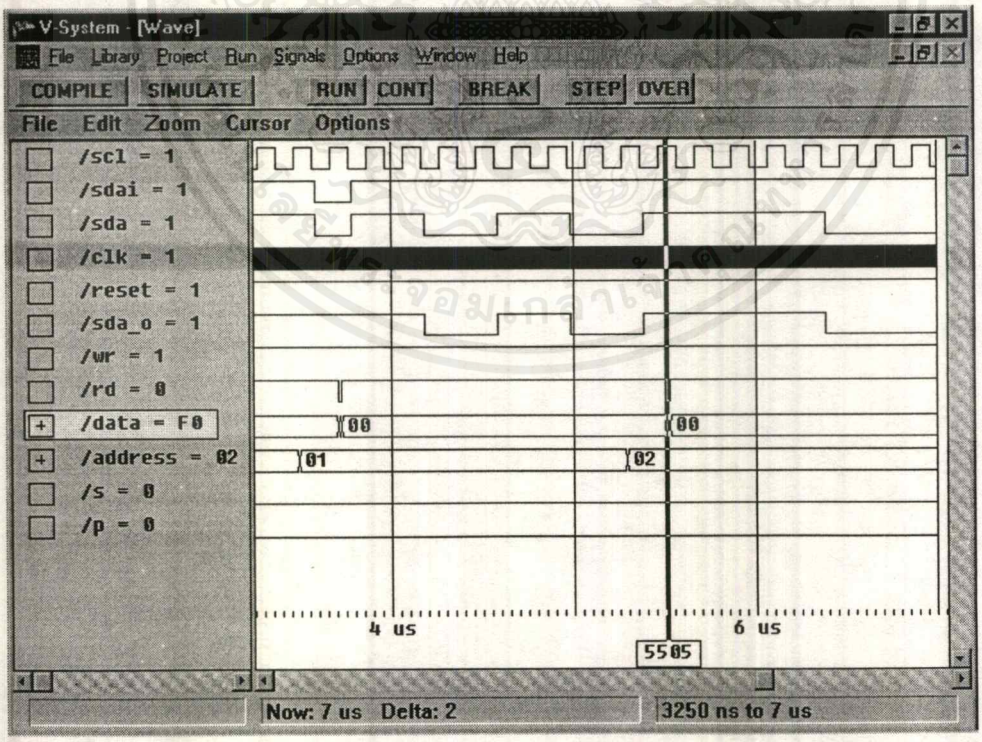


รูปที่ 4-5 แสดงผลการทดสอบการทำงานใน Slave Receiver mode ของ I^2c-bus Memory Interface

● Slave Transmitter จะทดสอบโดยการอ่านข้อมูลมาจากหน่วยความจำแล้วส่งออกไป ซึ่งในการทดสอบให้ส่งข้อมูลที่อยู่ใน แอดเดรส ที่ 00H - 03H ออกไป ผลการทดสอบได้ดังรูปที่ 4-6 และ 4.7

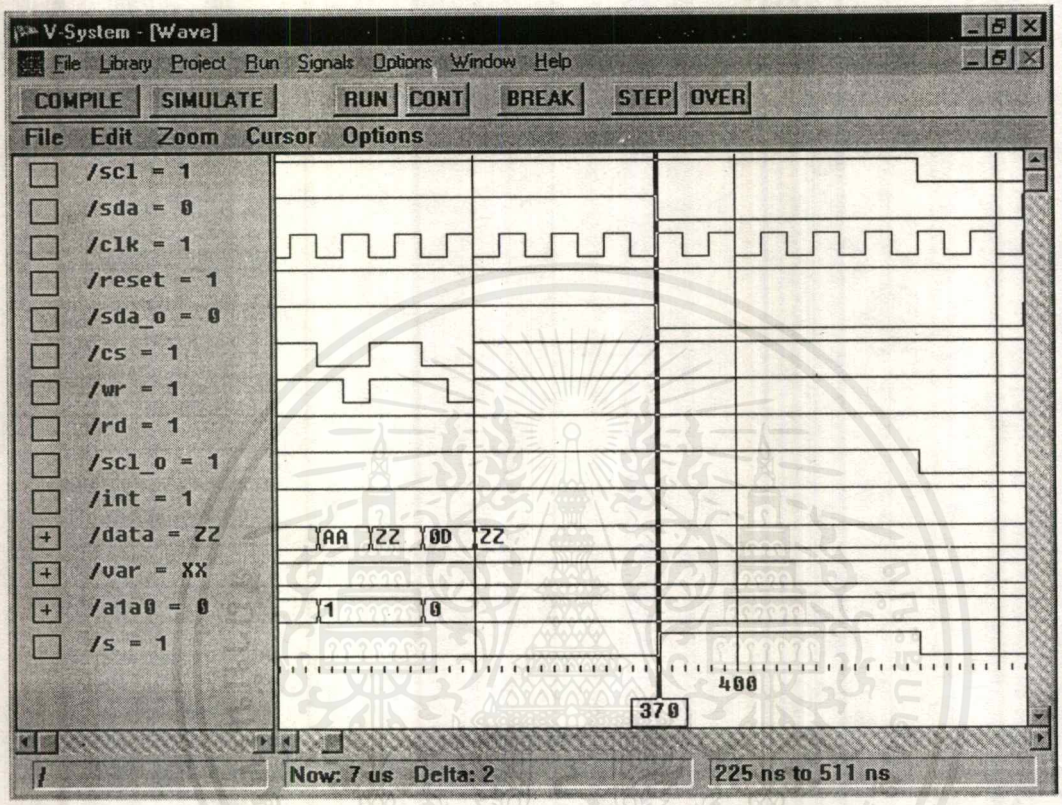


รูปที่ 4-6 แสดงผลการทดสอบการทำงานใน Slave Transmitter mode ของ I²c-bus Memory Interface

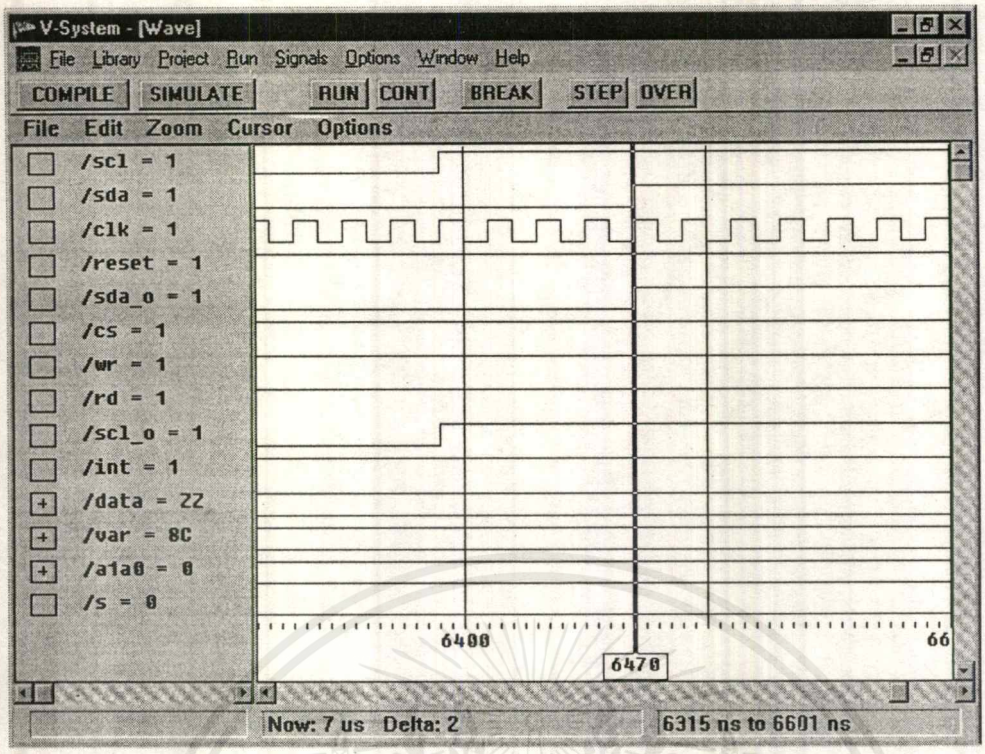


รูปที่ 4-7 แสดงผลการทดสอบการทำงานใน Slave Transmitter mode ของ I²c-bus Memory Interface (ต่อ)

3) ผลการทดสอบ I^2C-bus controller Interface จะต้องมีโหมดในการทดสอบอยู่ 4 โหมด แต่จะยกตัวอย่างให้ดูเพียงแค่การสร้าง START Condition และ Stop Condition ของการทำงานที่เป็น Master ดังแสดงในรูปที่ 4-8และ 4-9 ตามลำดับ

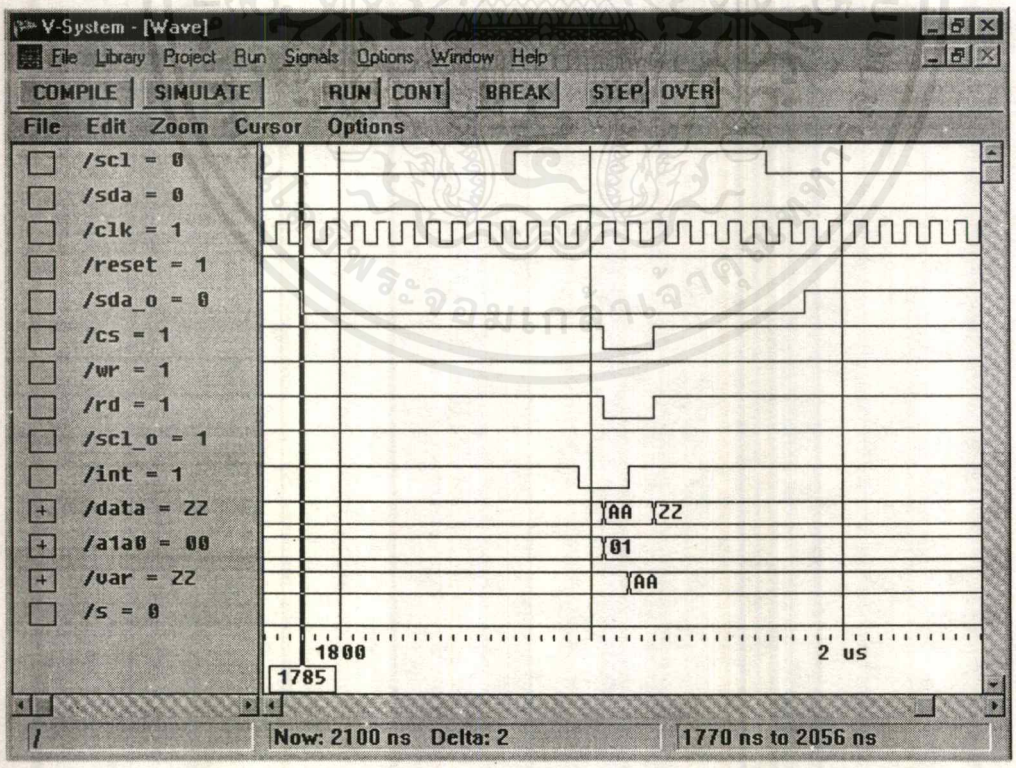


รูปที่ 4-8 การสร้าง Start Condition ของ Master



รูปที่ 4-9 การสร้าง Stop Condition ของ Master

ส่วนในรูปที่ 4-10 แสดงการตอบ Acknowledge ของการทำงานในโหมด Slave Receiver



รูปที่ 4-10 การ Acknowledge ของ Slave Receiver

4.2 Post Synthesis Simulation

หลังจากที่แปลงเป็น XNF netlist แล้วทำการแปลงกลับ แล้วทำการ Simulation อีกครั้งหนึ่ง ซึ่งจะได้ผลการทดลองของ Chip Core จะได้ผลเหมือนกับก่อนทำการสังเคราะห์

4.3 FPGA Prototype

เนื่องจากปัญหาของการสั่งซื้อชิปสำหรับทดสอบ I^2C-bus ทำให้ยังไม่สามารถทดสอบในส่วนนี้ได้ แน่สำหรับ UART ได้ทำการทดสอบ ผลที่ได้เป็นไปตามที่ต้องการ

บทที่ 5

บทสรุป และวิจารณ์

5.1 สิ่งที่ได้จากการทำโครงงาน

จากการศึกษาและออกแบบการสื่อสารแบบอนุกรมทั้ง UART และ I^2C ทำให้ได้ทราบถึงรูปแบบการสื่อสารอนุกรมในลักษณะต่าง ๆ กันออกไป และทำให้ทราบความแตกต่างของการสื่อสารแบบ Point-to-point และ Multipoint โดยการสื่อสารแบบ multipoint นอกจากจะต้องมีโปรโตคอล (protocol) ที่ใช้ในการรับส่งแล้ว ยังต้องมีโปรโตคอล ที่ใช้ในการแข่งขันที่จะใช้สายกันอีกด้วย ซึ่งเป็นส่วนที่ point-to-point ไม่มี ดังนั้นการศึกษา I^2C จึงทำให้ทราบถึงเทคนิคต่าง ๆ ที่ใช้ในการสื่อสารที่ไม่เคยทราบมาก่อน

สิ่งที่สรุปได้จากการศึกษา UART

ข้อดีของ UART

- เป็นการสื่อสารที่มีรูปแบบที่ง่าย จึงทำให้เป็นที่ใช้งานกันอย่างแพร่หลาย
- มีการส่ง parity bit ทำให้ตรวจสอบความถูกต้องได้

ข้อเสียของ UART

- ไม่เหมาะต่อการส่งข้อมูลที่มีขนาดใหญ่ เนื่องจากจะช้าเพราะมี over head สูง
- ฝ่ายส่งกับฝ่ายรับ ต้องใช้ baud rate เท่ากัน ถ้าไม่เท่าจะไม่สามารถ รับ และ ส่งข้อมูลได้

สิ่งที่สรุปได้จากการศึกษา I^2C

ข้อดีของ I^2C

- เป็นการเชื่อมต่อแบบ multipoint ทำให้สามารถนำอุปกรณ์ต่างๆมาต่อ หรือ ถอดออกจาก bus ได้โดยไม่มีผลต่ออุปกรณ์อื่นๆ
- มีการจัดการ การแข่งขัน การใช้สายโดยใช้หลักการที่ง่ายๆ
- สามารถส่งได้หลาย baud rate โดยขึ้นกับ มาสเตอร์ และ slave ของแต่ละการสื่อสาร
- เป็นรูปแบบการสื่อสารที่สามารถทำ hand shaking ได้

ข้อเสียของ I^2C

- ไม่มีการทำ error checking เพราะไม่มี parity bit ดังเช่นที่ UART มี แต่สามารถทำ error detection ได้โดยการทำ frame check sequence
- ไม่เหมาะสำหรับระบบที่มี traffic สูงๆ

5.2 วิธีการทำงาน

ในการทำงานในช่วงแรกเป็นการศึกษารูปแบบการสื่อสารแบบอนุกรมสำหรับ UART และ I2C-bus ซึ่งงานในส่วนของการศึกษาได้ทำในภาคการเรียนที่ผ่านมา

สำหรับงานในขั้นของการออกแบบ การสร้าง และการทดสอบได้ทำในภาคการเรียนนี้ทั้งหมด ซึ่งจะเริ่มด้วยการออกแบบ State Diagram สำหรับแต่ละ Component ของ Chip แต่ละตัว หลังจากนั้นก็ทำการโค้ดด้วยภาษา VHDL แล้วทำการทดสอบก่อนที่จะไปทำการ Synthesize ซึ่งงานเหล่านี้ในช่วงแรกจะมีการเปลี่ยนแปลง State Diagram หลายครั้ง เนื่องจากเมื่อมาทำการทดสอบ จะเกิดข้อผิดพลาดของสัญญาณบางตัว

งานในขั้นต่อมาคือ ทำการ สังเคราะห์ แล้วท
ปัญหาค่อนข้างมาก เริ่มด้วยตัว Code VHDL ที่เขียนไม่สามารถทำการ สังเคราะห์ ได้ทำให้ต้องไปแก้โค้ดซึ่งในบางครั้งเมื่อ แก้โค้ดแล้วทำการ สังเคราะห์ ผ่าน แต่เมื่อมา ตรวจสอบการทำงาน ปรากฏว่าได้ผลการทำงานไม่ตรงกับที่ต้องการทำจนทำให้ต้องไปรื้อโค้ดใหม่อีกครั้ง ซึ่งงานในช่วงนี้จะทำกลับไปกลับมาระหว่าง การเขียนโค้ดใหม่ทั้งหมด กับทดสอบหลังการสังเคราะห์อยู่ถึง 4 ครั้งด้วยกัน ทำให้เสียเวลาในช่วงนี้นานมาก

งานในขั้นสุดท้ายคือ การโปรแกรมลง FPGA และทำการทดสอบในระดับของ Hardware เนื่องจากไปเสียเวลาในงานระดับ Software มาก ทำให้การทดสอบระดับ Hardware ทำได้บางส่วนเท่านั้น

5.3 การออกแบบ ทดสอบ และอุปสรรคในการทำงาน

5.3.1 UART

ถึงแม้ว่า UART จะมีรูปแบบการสื่อสารที่ง่าย และมี state machine ที่ไม่ยุ่งยาก แต่ก็มีปัญหาในเรื่องการเขียนโค้ดเช่นเดียวกัน และมักเกิดกรณีที่คาดไม่ถึงเมื่อนำโค้ดที่สังเคราะห์ไปทำการทดสอบอีกครั้ง ในขั้นตอนการออกแบบเป็นไปโดยราบรื่น และมีปัญหาในการทดสอบบ้างเล็กน้อย แต่ปัญหาส่วนใหญ่มักอยู่ที่การทดสอบโค้ดที่ สังเคราะห์แล้ว

เมื่อทดสอบโค้ดที่สังเคราะห์ผ่านแล้ว ก็ทำการโปรแกรมลง FPGA และต่อวงจรทดสอบ ผลที่ได้พบว่าบางคอมพอเนนท์ ทำงานถูกต้อง และบางคอมพอเนนท์ ทำงานไม่ตรงกับที่ออกแบบไว้ ทั้งนี้สันนิษฐานว่าความผิดผลาดน่าจะเกิดในช่วงที่ Place and Route เพราะจากการทดสอบ โค้ด ที่สังเคราะห์แล้ว สามารถทำงานถูกต้องตรงตามที่ออกแบบไว้ตั้งแต่แรก การประเมินผลสำหรับ UART นี้น่าจะเสร็จสมบูรณ์ไปกว่า 85 % ของการทำงานทั้งหมด

5.3.2 I^2C-bus memory Interface

ขั้นตอนการออกแบบของวงจรตัวนี้ ค่อนข้างจะมีปัญหาพอสมควร เนื่องจากรูปแบบการสื่อสารที่ซับซ้อนมากขึ้น ทำให้ต้องพิจารณากรณีต่างๆมากขึ้นด้วย การออกแบบต้องปรับเปลี่ยนหลายครั้ง เนื่องจากมีความรู้ทางเทคนิคไม่เพียงพอ เมื่อทราบถึงเทคนิคใหม่ๆก็ต้องปรับเปลี่ยนใหม่เช่นกัน ปัญหาที่สำคัญอีกอย่างคือ ปัญหาด้านไทม์มิง เนื่องจากเมื่อทดสอบโค้ดที่สังเคราะห์มักพบว่า สัญญาณต่างๆไม่ตรงกับที่ออกแบบไว้ จึงต้องกลับมาพิจารณา source code หลายครั้ง

การทดสอบโค้ด ที่สังเคราะห์แล้วยังไม่สามารถทำได้ เนื่องจากข้อจำกัดทางด้านเวลา และ อุปกรณ์โดยเฉพาะการหา Serial EPROM ที่เป็น I2C bus การประเมินการทำงานของวงจรนี้สามารถทำไปได้ประมาณ 80 % ของการทำงาน

5.3.3 I^2C-bus controller Interface

ขั้นตอนการออกแบบวงจรตัวนี้มีความยุ่งยากมากที่สุด เพราะมีรูปแบบการสื่อสารถึง 4 รูปแบบด้วยกัน ส่วนประกอบแต่ละตัวต้องมีความซับซ้อนมากขึ้น การทดสอบโค้ดก็ยุ่งยากเพราะมีกรณีต่างๆมาก การทดสอบโค้ดที่สังเคราะห์แล้วก็มีปัญหาที่ยังแก้ไม่ได้อีกมากเช่นเดียวกัน

การทดสอบโค้ดที่สังเคราะห์แล้วยังไม่สามารถทำได้เช่นเดียวกัน เนื่องจากมีปัญหาเช่นเดียวกับ การทดสอบ I2C bus memory Interface การประเมินการทำงานของวงจรนี้สามารถทำไปได้ประมาณ 80 % ของการทำงาน

5.4 แนวทางในการพัฒนาต่อไป

เนื่องจากในโครงงาน นี้ได้ทำถึงขั้น Soft Core แต่ยังไม่ได้ทดสอบในทุกกรณีที่เป็นไปได้ และยังเป็นเพียงแค่ส่วนที่เชื่อมต่อกับไมโครคอนโทรลเลอร์ ดังนั้นถ้ามีการพัฒนาต่อ ควรจะนำไปใส่เป็นส่วนหนึ่งของไมโครคอนโทรลเลอร์ ซึ่งจะเป็นการเพิ่มประสิทธิภาพในการทำงานของตัวไมโครคอนโทรลเลอร์มาก

บรรณานุกรม

[1] Fredrick F. Driscoll , Data Communication , (International Ed.) , Saurders College Publishing , 1992

[2] Paul H. Young , Electronic Communication Techniques , (3rd Ed.) , Merrill Publishing company , 1994 , pp.546-548

[3] William Schweber , Electronic Communication Systems A Complete Course , (2nd Ed.) , Prentice Hall, Inc. , 1996

[4] Gary M. Miller , Modern Electronic Communication , (4th Ed.) , Prentice-Hall, Inc. , 1993

[5] Philips Semiconductors , Data Handbook IC12 " I^2C Peripheral ", 1996

[6] Barry B. Brey , Intel Microprocessors 8086/8088 , 80186 , 80286 , 80386 , and 80486

[7] William A. Shay , Understanding Data Communications and Networks, PWS Publishing Company, 1995 , pp.136-137

[8] D. C. Green , Data Communication , (2nd Ed.) , Longman Scientific & Technical , 1995 , pp.3-6

[9] David H. Stein , Introduction to Digital Data Communications , Delmar Publishers Inc. , 1985

[11] Jayaram Bhasker ,A VHDL Primer , Prentice-Hall ,Inc. , 1992

[12] Steve Carlson , Introduction to HDL-based Design Using VHDL , Synopsys Inc. , 1991

[13] จิรศักดิ์ เหลืองอุไร , คัมภีร์การใช้งานการสื่อสารอนุกรมบน PC , บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)