เครื่องสั่งอาหารแบบไร้สาย

WIRELESS MENU ORDERING EQUIPMENT

โดย

นายมรกต กรพิพัฒน์

นายศรัณย์วุฒิ ชื่นถนอมบุญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

เครื่องสั่งอาหารแบบไร้สาย

WIRELESS MENU ORDERING EQUIPMENT

โดย

นายมรกต กรพิพัฒน์ 46010602

นายศรัณย์วุฒิ ชื่นฉนอมบุญ 46010765

อาจารย์ที่ปรึกษา

ผศ.ดร. พิพัฒน์ พรหมมี

อ.ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **เครื่องสั่งอาหารแบบไร้สาย**

WIRELESS MENU ORDERING EQUIPMENT

ผู้จัดทำ

1. นายมรกต กรพิพัฒน์ 46010602

2. นายศรัณย์วุฒิ ชื่นถนอมบุญ 46010765

..... อาจารย์ที่ปรึกษา

(ผศ.ดร.พิพัฒน์ พรหมมี)

..... อาจารย์ที่ปรึกษา

(อ.ศรวัฒน์ ชีวปรีชา)

เครื่องสั่งอาหารแบบไร้สาย

WIRELESS MENU ORDERING EQUIPMENT

โดย นายบรกต กรพิพัฒน์ 46010602

นายศรัณย์วุฒิ ชื่นถนอมบุญ 46010765

อาจารย์ที่ปรึกษา ผศ.ดร.พิพัฒน์ พรหมมี

อ.ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

โครงงานนี้เสนอการศึกษาและการสร้างเครื่องสั่งอาหารแบบไร้สาย ซึ่งประกอบด้วยชุดสั่งงานส่วนของลูกค้า และส่วนประมวลผลกลาง โดยใช้การป้อนรหัสของรายการอาหารเข้าสู่เครื่องสั่งอาหารประกอบด้วยไมโครคอนโทรเลอร์และแสดงผลทางจอLCD ชุดสั่งงานจะทำการส่งรายการอาหาร เข้าสู่เครื่องรับของหน่วยประมวลผลกลางแบบไร้สาย และแสดงผลบนจอคอมพิวเตอร์ พร้อมทั้งคิดเงินได้

ABSTRACT

This project describes a study and implementation of wireless menu ordering equipment. It consists of user command and server units. The user command unit is including a LCD display, keypad, microcontroller and RF unit. The order command is sent from user command unit to server unit based on radio frequency. The server is able to make a billing reported.

กิตติกรรมประกาศ

ปริญญานิพนธ์ชิ้นนี้ สำเร็จลุล่วงได้ด้วยดี ด้วยความกรุณาเป็นอย่างดียิ่งจาก ผศ.ดร. พิพัฒน์ พรหมมี และ อ. ศรวัฒน์ ชีวปรีชา ที่กรุณาให้ทั้งในด้านความรู้ ความช่วยเหลือ ให้คำปรึกษาแนะนำที่ดีเสมอมา รวมถึงการอนุเคราะห์พื้นที่ห้อง เครื่องมือ และอุปกรณ์ต่างๆ สำหรับใช้ในการทำงาน

สุดท้ายนี้ ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ และขอขอบคุณเพื่อนๆทุกคน ที่คอยช่วยเหลือให้การสนับสนุน และให้กำลังใจที่ดีเสมอมา จนการจัดทำปริญญานิพนธ์ครั้งนี้สำเร็จลงได้

สารบัญ

บทที่ 1 บทนำ 1

บทที่ 2 ทฤษฎีและหลักการ 2

2.1 ทฤษฎีของไมโครคอนโทรลเลอร์ตระกูล MCS-51 2

2.1.1 คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ตระกูล MCS-51 2

2.1.2 ลักษณะการจัดขาของMCS-51 3

2.1.3 การจัดหน่วยความจำ 6

2.1.3.1 หน่วยความจำโปรแกรม 7

2.1.3.2 หน่วยความจำข้อมูล 7

2.1.3.3 รีจิสเตอร์ใช้งานทั่วไป 7

2.1.3.4 รีจิสเตอร์หน้าที่พิเศษ (SFR) 7

2.1.4 การใช้งานรีจิสเตอร์ 13

2.1.5 พอร์ทอินพุทและพอร์ทเอาท์พุท 14

2.1.5.1 การใช้งานพอร์ทเป็นอินพุท 14

2.1.5.2 การใช้งานพอร์ทเป็นเอาท์พุท 15

2.1.6 การเชื่อมต่อไมโครคอนโทรลเลอร์ตระกูล 8051 กับคอมพิวเตอร์ผ่านพอร์ตอนุกรม 15

2.1.6.1 การสื่อสารแบบอนุกรม 15

2.1.6.2 การสื่อสารข้อมูลแบบอะซิงโครนัส 16

2.2 การสื่อสารข้อมูลอนุกรม 18

2.2.1 กระบวนการรับและส่งข้อมูลอนุกรมของ 8051 20

2.2.2 การสื่อสารพอร์ตอนุกรม RS-232 21

2.2.3 ขั้นตอนการติดต่อระหว่างอุปกรณ์ DTE และ DCE 25

2.3 Single chip 2.4 GHz Transceiver nRF2401 27

2.3.1 Overview 27

2.3.2 Active modes 27

2.3.3 Shock Burst 27

2.3.3.1 หลักการของ Shock Burst 28

2.3.3.2 การส่ง Shock Burst 30

2.3.3.3 การรับ Shock Burst 32

2.3.4 DUOCiever Simultaneous two channel Receive mode 32

2.3.5 Device configuration 33

2.3.5.1 Configuration for ShockBurst operation 33
(Configuration สำหรับ การดำเนินการ ShockBurst)
2.3.5.2 Configuration for Direct Mode operation 34
(Configuration สำหรับการดำเนินการใน Direct Mode)
2.3.6 Configuration Word Detailed Description (อธิบายรายละเอียดของ configuration word) 35
2.3.6.1 Shock Burst configuration 36
2.3.6.1.1 PLL_CTRL 36
2.3.6.1.2 DATAx_W 36
2.3.6.1.3 ADDRx 37
2.3.6.1.4 ADDR_W & CRC 37
2.3.6.2 General RF configuration 38
2.3.6.2.1 Modes 38
2.3.6.2.2 RF channel & direction 39
2.3.7 Data package Description. 40
2.3.8 Configuration mode timing 41
2.3.9 ShockBurst Mode Timing 42
2.4 การเชื่อมต่อคีย์แพดเข้ากับไมโครคอนโทรลเลอร์ MCS-51 43
2.5 การเชื่อมต่อกับโมดูลแอลซีดี(LCD Module) 44
2.5.1 โครงสร้างภายในของตัวควบคุมโมดูล LCD 44
2.5.2 คำสั่งควบคุมโมดูล LCD 45
2.5.3 คำสั่งควบคุมการแสดงผล 46
2.5.4 คำสั่งควบคุมการเลื่อนเคอร์เซอร์และข้อมูลตัวอักษร 46
2.5.5 คำสั่งกำหนดฟังก์ชั่นการทำงาน 47
2.5.6 คำสั่งเลือกแอดเดรสของ DDRAM 47
2.5.7 คำสั่งอ่านบิตซีแฟลกและแอดเดรส 48
2.6 Delphi 7 48
2.6.1 ความสามารถของ Delphi 7 49
2.6.1.1 สร้างแอพพลิเคชันสำหรับ Windows 49
2.6.1.2 สร้างระบบงานด้านฐานข้อมูล 49
2.6.1.3 สร้างแอพพลิเคชันรองรับ .NET Web Service 49
2.6.1.4 ขั้นตอนการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชันกับ Delphi7 49
2.6.1.5 ภาษาออบเจ็กต์ปาสคาล 51

2.6.1.6 โครงสร้างของยูนิต 51
2.6.1.7 ไฟล์โปรเจ็กต์ 52
2.6.1.8 การเขียนโปรแกรมแบบ OOP 52
2.6.1.9 การเข้าถึงข้อมูลที่เก็บภายในคลาส 53
2.6.1.10 Constructor และ Destructor 53

บทที่ 3 การคำนวณและการสร้าง 54
3.1 ชุดสั่งงานของลูกค้า 55
3.2 วงจร TRW-2.4GHz ฝั่งลูกค้า 57
3.3 วงจร TRW-2.4GHz ฝั่งเครื่องserver 58
3.4 การใช้งาน RF Module (TRW-2.4GHz) 60
3.4.1 การทำงานของ Mode ShockBurst 60
3.4.2 การ Config TRW-2.4GHz 61
3.5 ส่วนของเครื่อง Server 63

บทที่4 การทดลองและผลการทดลอง 69
การทดลอง
4.1 การทดลองเรื่อง การวัดสัญญาณการรับ - ส่งระหว่าง TRW-2.4GHz ด้านส่งและด้านรับ 69
4.2 การทดลองเรื่อง การวัดความถี่ที่ใช้ส่งและกำลังของสัญญาณของTRW-2.4GHz 75
4.3 การทดลองเรื่อง การแสดงผลของเครื่อง Server และการแสดงผลตอบรับที่จอLCD กรณีที่ข้อมูลที่ส่งมาจากชุดสั่งงานของลูกค้าถูกต้อง 76
4.4 การทดลองเรื่อง การแสดงผลของเครื่อง Server และการแสดงผลตอบรับที่จอLCD กรณีที่ข้อมูลที่ส่งมาจากชุดสั่งงานของลูกค้าไม่ถูกต้อง 82

บทที่5 บทวิจารณ์และสรุป 85
ภาคผนวก
หนังสืออ้างอิง

สารบัญรูปภาพ

รูปที่ 2.1	แสดงรายละเอียดโครงสร้างหลักของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช ของ ATMEL	3
รูปที่ 2.2	การจัดขามาตรฐานของไมโครคอนโทรลเลอร์ MCS-51 ในอนุกรม AT89C5x	4
รูปที่ 2.3	การจัดโครงสร้างของหน่วยความจำทั้งในส่วนของหน่วยความจำโปรแกรม และหน่วยความจำข้อมูล	6
รูปที่ 2.4	การจัดหน่วยความจำและตำแหน่งรีจิสเตอร์หน้าที่พิเศษต่างๆ	9
รูปที่ 2.5	แสดงค่าของรีจิสเตอร์เฉพาะหลังการรีเซต	10
รูปที่ 2.6	แสดงแฟลกต่างๆของรีจิสเตอร์ PSW	12
รูปที่ 2.7	การใช้ไมโครคอนโทรลเลอร์เป็นอินพุทและเอาท์พุทพอร์ท	14
รูปที่ 2.8	รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม	16
รูปที่ 2.9	รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส	16
รูปที่ 2.10	คอนเน็กเตอร์ 9 ขาหรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์)	21
รูปที่ 2.11	คอนเน็กเตอร์อนุกรม 25 ขาหรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์)	22
รูปที่ 2.12	การต่ออุปกรณ์ภายนอกกับพอร์ตอนุกรมของคอมพิวเตอร์ในลักษณะต่างๆ	26
รูปที่ 2.13	จับเวลาข้อมูลโดย CPU ส่ง โดยเทคโนโลยี Shock B	28
รูปที่ 2.14	การใช้กระแส RF โดยใช้และไม่ใช้เทคโนโลยี Shock Burst	28
รูปที่ 2.15	แสดงแผนผังงาน(Flow Chart) ShockBurst การส่งของระบบย่อย nRF2401	29
รูปที่ 2.16	แสดงแผนผังงาน(Flow Chart) ShockBurst การรับระบบย่อย nRF2401	31
รูปที่ 2.17	Simultaneous 2 channel receiver on nRF24E1	32
รูปที่ 2.18	DUOCiever โดยมี 2 ช่องสัญญาณรับข้อมูลที่เป็นอิสระต่อกัน	33
รูปที่ 2.19	DATA package set-up	34
รูปที่ 2.20	Data package Diagram	40
รูปที่ 2.21	Timing Diagram สำหรับ configuration ระบบย่อย nRF2401	41
รูปที่ 2.22	Timing ของ ShockBurst ใน TX	42
รูปที่ 2.23	Timing ของ ShockBurst ใน Rx	43
รูปที่ 3.1	แสดงบล๊อกไดอะแกรมของเครื่องสั่งอาหารแบบไร้สาย	54
รูปที่ 3.2	แสดงวงจรชุดสั่งงานของลูกค้า	55
รูปที่ 3.3	แสดงแผนผังงาน(Flow Chart) การทำงานในไมโครคอนโทรเลอร์ของชุดสั่งงานของลูกค้า	56
รูปที่ 3.4	แสดงวงจร TRW-2.4GHz ด้านส่ง	57
รูปที่ 3.5	แสดงวงจร TRW-2.4GHz ด้านรับ	58
รูปที่ 3.6	แสดงแผนผังงาน(Flow chart) การรับ-ส่งของTRW-2.4GHz	59
รูปที่ 3.7	แสดง RF Module (TRW-2.4GHz)	60

รูปที่ 3.8 แสดงแผนผังงาน(Flow Chart) การทำงานในส่วนของเครื่อง Server 64
รูปที่ 3.9 แสดงแผนผังงาน(Flow Chart) การทำงานในส่วนของเครื่อง Server ในการพิจารณาตัวเลขชุดที่หนึ่ง(รหัสอาหาร) 65
รูปที่ 3.10 แสดงแผนผังงาน(Flow Chart) การทำงานในส่วนของเครื่อง Server ในการพิจารณาตัวเลขชุดที่สาม(โต๊ะที่) 66
รูปที่ 3.11 แสดงข้อมูลที่รับได้มีค่าเท่ากับ 002001003 67
รูปที่ 3.12 แสดงข้อมูลจากลูกค้าเพื่อเรียกเก็บเงิน 67
รูปที่ 3.13 แสดงข้อมูลจากลูกค้าเพื่อลบรายการอาหาร 68
รูปที่ 3.14 แสดงข้อมูลจากลูกค้าเพื่อสั่งเพิ่มรายการอาหาร 68
รูปที่ 4.1 การวัดสัญญาณที่ขาData (บน) เทียบกับสัญญาณนาฬิกาที่ขาCLK1 (ล่าง) ฝั่งชุดของลูกค้า 70
รูปที่ 4.2 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCS (ล่าง) ฝั่งชุดของลูกค้า 70
รูปที่ 4.3 ภาพขยายการวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCS (ล่าง) ฝั่งชุดของลูกค้า 71
รูปที่ 4.4 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCE (ล่าง) ฝั่งชุดของลูกค้า 71
รูปที่ 4.5 ภาพขยายการวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCE (ล่าง) ฝั่งชุดของลูกค้า 72
รูปที่ 4.6 การวัดสัญญาณที่ขาData (บน) กับสัญญาณนาฬิกาที่ขาCLK1 (ล่าง) ฝั่งเครื่อง server 72
รูปที่ 4.7 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาDR1 (ล่าง) ฝั่งเครื่อง server 73
รูปที่ 4.8 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCS (ล่าง) ฝั่งเครื่อง server 73
รูปที่ 4.9 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCE (ล่าง) ฝั่งเครื่อง server 74
รูปที่ 4.10 แสดงผลการวัดความถี่และกำลังของสัญญาณที่ใช้ส่งของ TRW-2.4GHz 75
รูปที่ 4.11 แสดงหน้าจอ LCD เมื่อชุดสั่งงานของลูกค้าอยู่ในสถานะพร้อมทำงาน 76
รูปที่ 4.12 แสดงหน้าจอแสดงผลของเครื่อง Server ในสถานะพร้อมทำงาน 76
รูปที่ 4.13 แสดงหน้าจอ LCD พร้อมที่จะส่งข้อมูลที่มีค่าเท่ากับ 002001003 77
รูปที่ 4.14 แสดงหน้าจอ LCD หลังจากการกดปุ่ม # 77
รูปที่ 4.15 แสดงหน้าจอ LCD เมื่อมีกดตัวเลขไม่ครบ 9 ตัว 77
รูปที่ 4.16 แสดงหน้าจอแสดงผลของเครื่อง Server หลังจากรับข้อมูลที่มีค่าเท่ากับ 002001003 78
รูปที่ 4.17 แสดงหน้าจอ LCD เมื่อได้รับการยืนยันการสั่งเพิ่มรายการอาหารที่ 002 78
รูปที่ 4.18 แสดงหน้าจอแสดงผลของเครื่อง Server หลังจากรับข้อมูลที่มีค่าเท่ากับ 021002003, 045001003 ตามลำดับ 79
รูปที่ 4.19 แสดงหน้าจอแสดงผลของเครื่อง Server หลังจากรับข้อมูลที่มีค่าเท่ากับ 021000003 80
รูปที่ 4.20 แสดงหน้าจอ LCD เมื่อได้รับการยืนยันการสั่งลบรายการอาหารที่ 021 จากเครื่อง Server 81
รูปที่ 4.21 แสดงหน้าจอแสดงผลของเครื่อง Server หลังจากรับข้อมูลที่มีค่าเท่ากับ 000000003 81
รูปที่ 4.22 แสดงใบเสร็จที่ใช้สำหรับการเรียกเก็บเงินกับลูกค้าโต๊ะ 3 82
รูปที่ 4.23 แสดงหน้าจอ LCD เมื่อได้รับการยืนยันการเรียกเก็บเงินของลูกค้าโต๊ะที่ 3 จากเครื่อง Server 82

รูปที่ 4.24 แสดงหน้าจอ LCD เมื่อไม่ได้รับการยืนยันการรับข้อมูลจากเครื่อง Server เนื่องจาก รหัสอาหารไม่มีในฐานข้อมูล 83
รูปที่ 4.25 แสดงหน้าจอ LCD เมื่อไม่ได้รับการยืนยันการรับข้อมูลจากเครื่อง Server เนื่องจาก หมายเลขโต๊ะที่ไม่มีในโปรแกรม 83
รูปที่ 4.26 แสดงชุดสั่งงานของลูกค้า 84
รูปที่ 4.27 แสดงภาครับทางฝั่ง Server 84

สารบัญตาราง

ตารางที่ 2.1	แสดงการเลือก Register	13
ตารางที่ 2.2	แสดงบิตพาริตี้ของข้อมูล	17
ตารางที่ 2.3	การเลือกโหมดการทำงาน	20
ตารางที่ 2.4	ข้อกำหนดของขาสัญญาณต่างๆ	22
ตารางที่ 2.5	แสดงรายละเอียดของสัญญาณที่ต่อจาก DTE ไปยัง DCE โดยใช้คอนเน็กเตอร์แบบ DB-25	23
ตารางที่ 2.6	รายละเอียดการต่อคอนเน็กเตอร์แบบ DB9 มาตราฐาน RS-232	23
ตารางที่ 2.7	nRF2401 subsystem main modes	27
ตารางที่ 2.8	ตารางของ Configuration word	34
ตารางที่ 2.9	Configuration data word	35
ตารางที่ 2.10	PLL setting	36
ตารางที่ 2.11	จำนวนของบิตใน payload	36
ตารางที่ 2.12	แอดเดรสของตัวรับ 1 และ 2	37
ตารางที่ 2.13	จำนวนบิตที่ต้องคงไว้สำหรับ RX address + CRC setting	37
ตารางที่ 2.14	RF operational setting	38
ตารางที่ 2.15	crystal frequency setting	39
ตารางที่ 2.16	RF output power setting	39
ตารางที่ 2.17	ช่องความถี่ และ การ set Rx / Tx	39
ตารางที่ 2.18	รายละเอียดของ Data package	40
ตารางที่ 3.1	วิธีการ Config TRW-2.4GHz	62

บทที่ 1

บทนำ

สิ่งที่มีผลต่อการดำรงชีวิตและตอบสนองความต้องการของมนุษย์ที่สำคัญสิ่งหนึ่ง นั่นก็คือ ความต้องการความสะดวกรวดเร็วในด้านต่างๆ เครื่องสั่งอาหารแบบไร้สายเป็นตัวอย่างหนึ่งซึ่งตอบสนองความต้องการเหล่านั้น

จากที่เห็นว่าปัจจุบันนี้ ร้านอาหารได้เปิดกิจการขึ้นเป็นจำนวนมากมาย ทำให้เกิดการแข่งขันกันสูงระหว่างร้านอาหารต่างๆ ยิ่งถ้าร้านไหนมีการบริการที่ดี อาหารอร่อยถูกใจลูกค้า ก็จะเป็นสิ่งที่ทำให้มีลูกค้าเข้ามารับประทานอาหารมากขึ้น และเมื่อมีลูกค้ามากขึ้น ก็จะทำให้เกิดปัญหาในด้านการบริการลูกค้า เช่น บริกรรับรายการอาหารจากลูกค้าได้ช้า ลูกค้าต้องรอนาน จนลูกค้าเกิดความไม่พอใจ อาจทำให้ร้านอาหารร้านนั้นเสียลูกค้าได้ จึงมีความจำเป็นอย่างยิ่งที่จะต้องมีเครื่องสั่งอาหารแบบ ไร้สายในการที่จะให้ลูกค้าสั่งอาหารได้อย่างสะดวกและรวดเร็วขึ้น

การทำงานของเครื่องสั่งอาหารแบบไร้สายนั้น ประกอบด้วย 2 ส่วน ได้แก่ ส่วนที่หนึ่ง เป็นส่วนของชุดสั่งงานของลูกค้า ซึ่งจะใช้ไมโครคอนโทรเลอร์ตระกูล MCS-51 มาควบคุมคีย์แพด ให้ทำหน้าที่รับค่ารหัสที่ลูกค้ากด แล้วนำค่านั้นเก็บไว้ในหน่วยความจำพร้อมทั้งแสดงผลที่หน้าจอ LCD ส่วนค่าที่เก็บไว้นำหน่วยความจำนั้น จะถูกส่งออกไปผ่าน วงจร RF โมดูลของตัวส่ง ในส่วนที่สองคือ ส่วนของหน่วยประมวลผลกลาง ในส่วนนี้จะมี RF โมดูลตัวรับทำหน้าที่รับข้อมูลจากส่วนของชุดสั่งงานของลูกค้า แล้วส่งผ่านพอร์ตอนุกรมเข้าไปยังเครื่อง Server ซึ่งเครื่อง Server จะทำการประมวลผลของรหัสต่างๆ นอกจากนี้ ยังสามารถตอบกลับไปยังชุดสั่งงานของลูกค้าเพื่อยืนยันการสั่งอาหาร ว่าได้รับแล้วหรือต้องการยกเลิกรายการอาหารที่ได้สั่งไป และยังสามารถเรียกเก็บเงินพร้อมสั่งให้พิมพ์เพื่อใช้สำหรับการคิดเงินกับลูกค้าได้

บทที่ 2

ทฤษฎีและหลักการ

2.1 ทฤษฎีของไมโครคอนโทรลเลอร์ตระกูล MCS-51

ในเครื่องสั่งอาหารแบบไร้สายนั้นจะมีการนำไมโครคอนโทรเลอร์มาใช้งานด้วย ดังนั้นจึงจำเป็นต้องทราบถึงทฤษฎีต่างๆของไมโครคอนโทรเลอร์ซึ่งมีดังนี้

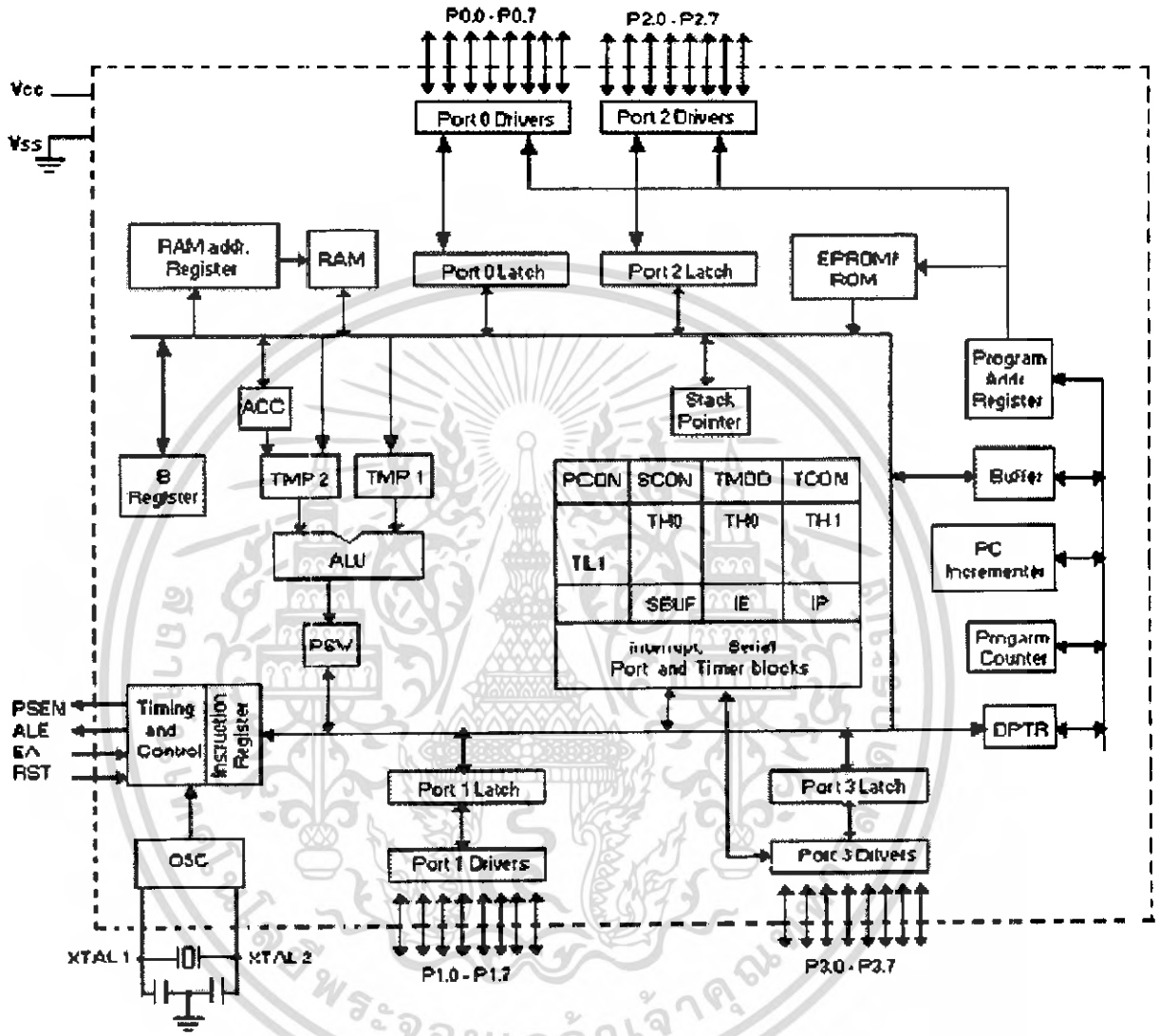
2.1.1 คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ตระกูล MCS-51

คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีดังต่อไปนี้

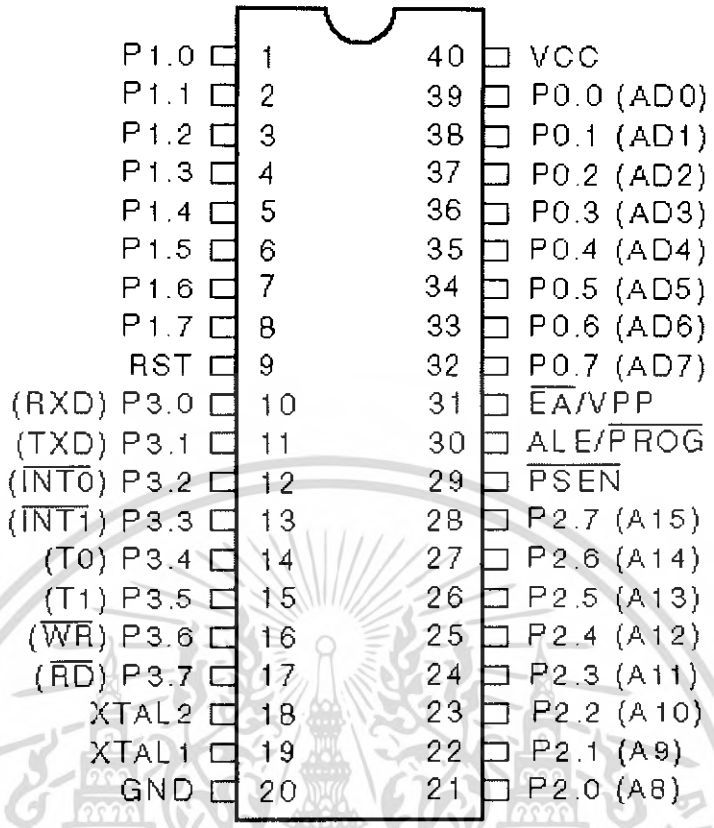
- เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต
- มีวงจรออสซิลเลเตอร์และวงจรผลิตสัญญาณนาฬิกาภายในไอซี
- มีขาสัญญาณอินพุตและเอาต์พุตจำนวน 32 บิต
- สามารถเชื่อมหน่วยความจำข้อมูลภายนอก (External Data Memory) โดยการอ้างตำแหน่งแอดเดรสได้ถึง 64 K
- สามารถเชื่อมต่อหน่วยความจำโปรแกรมภายนอก (External Program Memory) โดยการอ้างตำแหน่งแอดเดรสได้ถึง 64 K
- มีหน่วยความจำข้อมูลภายในตัว (On-Chip Data Memory) ขนาด 128 ไบต์ โดยเฉพาะหมายเลข 8032 และ M8052 จะมีหน่วยความจำในส่วนนี้ถึง 256 ไบต์
- มีหน่วยความจำโปรแกรมภายในตัว (On-Chip Program Memory) ขนาด 4 K โดยเฉพาะหมายเลข 8052 จะมีหน่วยความจำในส่วนนี้ถึง 8 K สำหรับหมายเลข 8031 และ M8032 จะไม่มีหน่วยความจำในส่วนนี้
- หน่วยความจำข้อมูลภายในบางส่วนสามารถเข้าถึงข้อมูลระดับบิตได้ด้วย ทำให้สามารถควบคุมหรือตรวจสอบสถานะบิตได้ง่าย ส่งผลให้สามารถเขียนโปรแกรมได้ง่ายขึ้น
- มีไทเมอร์/เคาน์เตอร์ (Timer/Counter) ขนาด 16 บิตจำนวน 2 ตัว โดยเฉพาะหมายเลข 8032 หรือ 8052 จะมีไทเมอร์/เคาน์เตอร์จำนวน 3 ตัว
- สามารถทำการอินเตอร์รัพท์ได้จากแหล่งกำเนิด 5 แหล่ง โดยเฉพาะหมายเลข 8032 และ 8052 จะทำการอินเตอร์รัพท์ได้จากแหล่งกำเนิด 6 แหล่ง พร้อมทั้งยังสามารถจัดลำดับความสำคัญของการอินเตอร์รัพท์ได้ 2 ระดับ
- มีพอร์ตสื่อสารอนุกรมภายในตัวเอง ซึ่งการทำงานจะเป็นแบบฟูลดูเพล็กซ์ (Full Duplex)
- มีคำสั่งในการคำนวณทางคณิตศาสตร์และทางตรรกศาสตร์
- คำสั่งส่วนใหญ่จะใช้เวลาในการประมวลผลเพียง 1 ไมโครวินาที เมื่อใช้คริสตอลออสซิลเลเตอร์ความถี่ 12 เมกะเฮิรตซ์
- ต้องการแหล่งจ่ายไฟกระแสตรงแรงดัน 5 โวลต์

2.1.2 ลักษณะการจัดขาของ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 ทุกเบอร์จะมีสถาปัตยกรรมและขาใช้งานพื้นฐานเหมือนกัน ดังแสดงในรูปที่ 2.1 และ 2.2 โดยมีรายละเอียดขั้นต้นดังนี้



รูปที่ 2.1 แสดงรายละเอียดโครงสร้างหลักของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช ของ ATMEL



รูปที่ 2.2 การจัดขามาตรฐานของไมโครคอนโทรลเลอร์ MCS-51 ในอนุกรม AT89C5x

ขา Vcc ใช้สำหรับต่อไฟเลี้ยง +5V

ขา GND เป็นขากราวด์ สำหรับต่อกับกราวด์ของระบบ

ขาพอร์ท 0 (P0.0-P0.7) มีขา 8 ขา สามารถกำหนดให้เป็นได้ทั้งอินพุทและเอาท์พุท สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ท 0 ขาใดขาหนึ่งเป็นอินพุท สามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ทที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ทนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุทอิมพิแดนซ์สูง สามารถใช้งานเป็นขาพอร์ทอินพุทได้ นอกจากนั้นขาพอร์ทนี้ยังถูกใช้งานในการคิดต่อกับขาแอดเดรสไบต์ต่ำของหน่วยความจำภายนอก (A0-A7) และขาข้อมูล (D0-D7) โดยใช้กระบวนการมัลติเพล็กซ์เข้าช่วย เพื่อสลับการทำงานเป็นได้ทั้งขาติดต่อแอดเดรส และขาข้อมูล

ขาพอร์ท 1 (P1.0-P1.7) มีขา 8 ขา แต่ละขาสามารถกำหนดให้เป็นทั้งอินพุทและเอาท์พุทสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ทใดเป็นอินพุท สามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ทที่ต้องการติดต่อด้วย

ขาพอร์ท 2 (P2.0-P2.7) มีขา 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุทและเอาท์พุทสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ทใดเป็นอินพุทสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ทที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ทนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุท

อิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ทอินพุทได้ นอกจากนั้นขาพอร์ทนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์สูงของหน่วยความจำภายนอก (A8-A15)

ขาพอร์ท 3 (P3.0-P3.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุทและเอาท์พุทสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ทใดเป็นอินพุทสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ทที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ทนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุทอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ทอินพุทได้ นอกจากนั้นขาพอร์ท 3 ยังเป็นขาที่มีหน้าที่การใช้งานพิเศษ ดังมีรายละเอียดขั้นตอนต่อไปนี้

P3.0 ใช้เป็นขาอินพุทสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RxD

P3.1 ใช้เป็นขาอินพุทสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา TxD

P3.2 ใช้เป็นขาอินพุทรับสัญญาณอินเตอร์รัพต์จากภายนอกช่อง 0 หรือขา $\overline{\text{INT0}}$

P3.3 ใช้เป็นขาอินพุทรับสัญญาณอินเตอร์รัพต์จากภายนอกช่อง 1 หรือขา $\overline{\text{INT1}}$

P3.4 ใช้เป็นขาอินพุทสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 0 หรือขา T0

P3.5 ใช้เป็นขาอินพุทสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 1 หรือขา T1

P3.6 ใช้เป็นขาสัญญาณ $\overline{\text{WR}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

P3.7 ใช้เป็นขาสัญญาณ $\overline{\text{RD}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

ขารีเซต (Reset) ใช้ในการรีเซ็ทการทำงานของไมโครคอนโทรลเลอร์ โดยใช้การป้อนสัญญาณเพื่อรีเซ็ทสถานะที่ขานี้ต้องอยู่ในระดับรีเซ็ทอย่างน้อย 2 แมชชีนไซเคิล โดยที่วงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างเป็นปกติ

ขา **$\overline{\text{ALE}}$ /PROG (Address Latch Enable/Program pulse input)** เป็นขาที่ใช้ในการควบคุมการแลตช์ของขาพอร์ท 0 เมื่อมีการใช้งานหน่วยความจำภายนอก นอกจากนั้นขานี้ยังใช้เป็นขาสำหรับรับพัลส์ของการโปรแกรมสำหรับโปรแกรมข้อมูลลงในไมโครคอนโทรลเลอร์ MCS-51 ในรุ่นที่มีหน่วยความจำโปรแกรมเป็นแบบอีพรอม

ขา **$\overline{\text{PSEN}}$ (Program Store Enable)** ขานี้ใช้ในการส่งสัญญาณเพื่อร้องขอติดต่อกับหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายนอก ตัวไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายนอกตัวไมโครคอนโทรลเลอร์จะส่งสัญญาณออกมาที่ขานี้ 2 ครั้ง ในแต่ละแมชชีนไซเคิล แต่ถ้าหากติดต่อกับหน่วยความจำข้อมูลภายนอกขานี้จะไม่มีการส่งสัญญาณใดๆออกมา

ขา **$\overline{\text{EA}}$ /Vpp (External Access enable/Programming voltage input)** ใช้สำหรับเลือกการติดต่อหน่วยความจำโปรแกรมจากภายนอกหรือภายในตัวไมโครคอนโทรลเลอร์ ถ้าหาขานี้เป็น "0" เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำโปรแกรมภายนอก แต่ถ้าหากขานี้เป็น "1" เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำภายในตัวไมโครคอนโทรลเลอร์ นอกจากนี้ที่ขานี้ยังใช้เป็นขาอินพุทสำหรับรับแรงดันไฟสูงสำหรับการโปรแกรมหน่วยความจำภายในไมโครคอนโทรลเลอร์ สำหรับไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชต้องการแรงดันสำหรับการโปรแกรม +5V

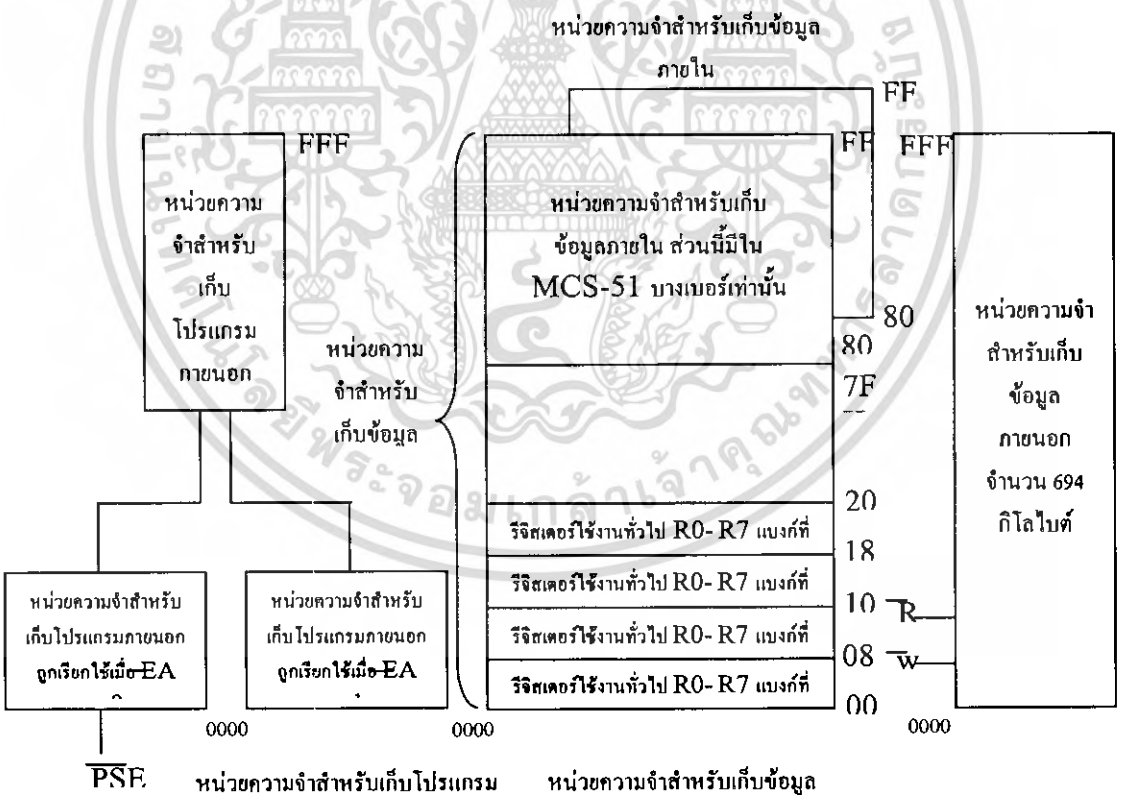
ขา XTAL1 และ XTAL2 เป็นขาสำหรับต่อคริสตัลเพื่อสร้างสัญญาณนาฬิกาในการกำหนดจังหวะการทำงานของไมโครคอนโทรลเลอร์

2.1.3 การจัดหน่วยความจำ

ในไมโครคอนโทรลเลอร์ตระกูล MCS-51 แบ่งชนิดหรือหน้าที่ของหน่วยความจำออกเป็น 2 ส่วน คือ หน่วยความจำโปรแกรม (Program Memory) และ หน่วยความจำข้อมูล (Data Memory)

หน่วยความจำโปรแกรมใช้สำหรับเก็บโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์ซึ่งบางหมายเลขจะมีหน่วยความจำนี้อยู่ในตัว โดยอาจจะมีขนาดไม่เท่ากันหรือเป็นหน่วยความจำต่างชนิดกัน เช่น บางหมายเลขเป็น ROM แต่บางหมายเลขอาจเป็น EPROM และบางหมายเลขอาจไม่มีหน่วยความจำในส่วนนี้เลย โปรแกรมการทำงานจะถูกเก็บไว้ยังหน่วยความจำโปรแกรมภายนอกทั้งหมด

สำหรับหน่วยความจำข้อมูลจะใช้สำหรับเก็บข้อมูลหรือค่าตัวแปรต่างๆ จากการทำงานของโปรแกรมซึ่งใน MCS-51 ทุกหมายเลขจะมีหน่วยความจำส่วนนี้อยู่จำนวนหนึ่ง แต่อาจมีขนาดมากน้อยต่างกันไปในแต่ละหมายเลข สำหรับการจัดโครงสร้างของหน่วยความจำทั้งในส่วนของหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลแสดงไว้ในรูปที่ 2.3



รูปที่ 2.3 การจัดโครงสร้างของหน่วยความจำทั้งในส่วนของ
หน่วยความจำโปรแกรมและหน่วยความจำข้อมูล

2.1.3.1 หน่วยความจำโปรแกรม

หน่วยความจำโปรแกรมสามารถแบ่งออกได้เป็น 2 ส่วนคือ หน่วยความจำโปรแกรมภายในและหน่วยความจำโปรแกรมภายนอก หน่วยความจำภายในจะถูกเลือกใช้งานถ้าขาสัญญาณ EA มีค่าเป็น 1 โดยจะถูกใช้งานในช่วงแอดเดรส 0-0FFFH (หรือช่วงแอดเดรส 0-1FFFH ในหมายเลข 8052) ในกรณีตรงข้ามถ้าขาสัญญาณ EA มีค่าเป็น 0 ในช่วงแอดเดรส 0-0FFFH (หรือช่วงแอดเดรส 0-1FFFH ในหมายเลข 8052) จะถูกใช้จากหน่วยความจำภายนอก หรือกล่าวได้ว่าถ้าขาสัญญาณ EA มีค่าเป็น 0 จะเป็นการเลือกใช้หน่วยความจำโปรแกรมภายนอกทั้งหมดตลอดช่วงแอดเดรส

2.1.3.2 หน่วยความจำข้อมูล

หน่วยความจำข้อมูลสามารถแบ่งออกได้เป็น 2 ส่วนคือ หน่วยความจำข้อมูลภายในและหน่วยความจำข้อมูลภายนอก สำหรับหน่วยความจำข้อมูลภายในยังแบ่งออกได้เป็น 2 ส่วนย่อยคือ ส่วนที่ใช้เก็บข้อมูลทั่วไปและส่วนที่ใช้เป็นรีจิสเตอร์หน้าที่พิเศษหรือ SFR (Special Function Register) โดยส่วนที่ทำหน้าที่เก็บข้อมูลทั่วไปจะถูกใช้สำหรับเก็บข้อมูลหรือค่าตัวแปรต่างๆ จากการทำงานของโปรแกรม ส่วนรีจิสเตอร์หน้าที่พิเศษจะถูกใช้งานเป็นรีจิสเตอร์ควบคุมการทำงาน และแสดงสถานะการทำงานของไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกหมายเลขจะมีหน่วยความจำข้อมูลภายในขนาด 128 ไบต์เป็นอย่างน้อย และบางหมายเลขอาจมีถึง 256 ไบต์

2.1.3.3 รีจิสเตอร์ใช้งานทั่วไป

รีจิสเตอร์ใช้งานทั่วไปมีไว้สำหรับให้ผู้เขียนโปรแกรมสามารถนำข้อมูลไปพักไว้ชั่วคราวหรือใช้งานทั่วไปได้ตามต้องการ ซึ่งรีจิสเตอร์ใช้งานทั่วไปนี้มีอยู่ด้วยกัน 8 ตัวคือ รีจิสเตอร์ R0-R7 โดยรีจิสเตอร์ทั้ง 8 ตัวถูกจัดให้อยู่รวมกันและมีให้เลือกใช้ถึง 4 แบงก์ (bank) นั่นคือ มีรีจิสเตอร์ใช้งานทั่วไปถึง 32 ตัวให้ใช้งาน เพียงแต่การเลือกใช้รีจิสเตอร์ R0-R7 ในแบงก์ใดแบงก์หนึ่งจะถูกกำหนดจากบิต RS0, RS1 ในรีจิสเตอร์หน้าที่พิเศษ PSW ดังนั้นการเลือกใช้จึงเลือกได้เพียงแบงก์เดียวในขณะใดขณะหนึ่ง อย่างไรก็ตาม ค่าของข้อมูลที่เก็บไว้ในรีจิสเตอร์มีชื่อเดียวกันแต่อยู่คนละแบงก์จะไม่มีผลซึ่งกันและกันเลย ทำให้ผู้เขียนโปรแกรมสามารถใช้งานรีจิสเตอร์ทั่วไปได้ทั้ง 32 ตัวอย่างเต็มที่และไม่ยุ่งยากในการเขียนโปรแกรม

2.1.3.4 รีจิสเตอร์หน้าที่พิเศษ (SFR)

รีจิสเตอร์หน้าที่พิเศษมีบทบาทอย่างมาก ในการควบคุมการทำงานของไมโครคอนโทรลเลอร์และทำให้การเขียนโปรแกรมสามารถทำได้สะดวกขึ้น รีจิสเตอร์หน้าที่พิเศษนี้ทำหน้าที่สำคัญคือควบคุมการทำงานในส่วนต่างๆ ภายในไมโครคอนโทรลเลอร์และยังทำหน้าที่แสดงสถานะการทำงาน ซึ่งในรีจิสเตอร์หน้าที่พิเศษบางตัวยังสามารถเข้าถึงได้ในระดับบิต (Bit addressable) ด้วย ดังแสดงการจัดหน่วยความจำและตำแหน่งรีจิสเตอร์หน้าที่พิเศษต่างๆ ในรูปที่ 2.4

DIRECT Byte Address	Bit Address								Special Function Register Symbol
	(MSB)							(LSB)	
	WDT	T32	SERR	IZC	P3HZ	P2HZ	P1HZ	ALF	
0F8H	FF	FE	FD	FC	FB	FA	F9	F8	IOCON
0F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	CY	AC	F0	RS1	RS0	0V	F1	P	
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0CDH	Not Bit Addressable								TH2
0CCH	Not Bit Addressable								TL2
0CBH	Not Bit Addressable								RCAP2H
0CAH	Not Bit Addressable								RCAP2L
	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	$C/\overline{T}$	$CP/\overline{RL}$	
0C8H	CF	CE	CD	CC	CB	CA	C9	C8	T2CON
	PCT		PT2	PS	PT1	PX1	PT0	PX0	
0B8H	BF	–	BD	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA		ET2	ES	ET1	EX1	ET0	EX0	
0A8H	AF	–	AD	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
99H	Not Bit Addressable								SBUF
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	

98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
8DH	Not Bit Addressable								TH1
8CH	Not Bit Addressable								TH0
8BH	Not Bit Addressable								TL1
8AH	Not Bit Addressable								TL0
89H	Not Bit Addressable								TMOD
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
87H	Not Bit Addressable								PCON
83H	Not Bit Addressable								DPH
82H	Not Bit Addressable								DPL
81H	Not Bit Addressable								SP
80H	87	86	85	84	83	82	81	80	P0

รูปที่ 2.4 การจัดหน่วยความจำและตำแหน่งรีจิสเตอร์หน้าที่พิเศษต่างๆ

ค่า Special Function Register หลังการรีเซตจะมีค่าดังแสดงในรูปที่ 2.5

Register	ค่าไบนารี่	Register	ค่าไบนารี่
*ACC	00000000	TMOD	00000000
*B	00000000	*TCON	00000000
*PSW	00000000	*+T2CON	00000000
SP	00000111	THO	00000000
DPTR		TLO	00000000
DPL	00000000	TH1	00000000
DPH	00000000	TL1	00000000
*PO	11111111	+TH2	00000000
*P1	11111111	+TL2	00000000
*P2	11111111	+RCAP2H	00000000
*P3	11111111	+RCAP2L	00000000
*IP	8051 XXX00000	*SCON	00000000
	8052 XX000000	SBUF	Indeterminate
*IE	8051 0XX00000	PCON	HMOS 0XXXXXXX
	8052 0X000000		CHMOS 0XXXXXXX

X = Undefined * = Bit addressable + = 8052 only

* = สามารถอ้างตำแหน่งข้อมูลแบบบิตได้

+ = มีเฉพาะใน 8052 และ 8032 เท่านั้น

รูปที่ 2.5 แสดงค่าของรีจิสเตอร์เฉพาะหลังการรีเซต

การทำงานของรจิสเตอร์ต่างๆ เป็นดังนี้

ACC : แอคคิวมูเลเตอร์มีขนาด 8 บิตทำหน้าที่เช่นเดียวกับแอคคิวมูเลเตอร์ของซีพียูอื่น ๆ คือ ใช้เป็นตัวกระทำร่วมทางคณิตศาสตร์ของเลข 2 จำนวน เช่น การ บวก ลบ คูณ หาร เป็นต้น และทำหน้าที่เป็นตัวเก็บผลลัพธ์ที่ได้จากการคำนวณทางคณิตศาสตร์

Register B : เป็นรีจิสเตอร์ที่ใช้งานเฉพาะในคำสั่งการคูณ (MUL) และการหาร (DIV) เท่านั้น โดยจะทำงานร่วมกับ แอคคิวมูเลเตอร์ ในการเก็บผลลัพธ์ของการคูณและเศษที่ได้จากการหาร

SP : (Stack Pointer) มีขนาด 8 บิต ทำหน้าที่ชี้ตำแหน่งสแตก ซึ่งไมโครคอนโทรลเลอร์ MCS- 51 ใช้พื้นที่ของหน่วยความจำภายในเป็นสแตก โดยตำแหน่งเริ่มต้นหลังจากการรีเซตจะเป็น 07 การทำงานของ SP จะเพิ่มขึ้นหรือลดลงอย่างอัตโนมัติเมื่อมีการเก็บข้อมูลลงในสแตกหรือนำข้อมูลออก

จาก สแตก การเก็บข้อมูลลงในสแตกจะเก็บได้ครั้งละ 8 บิต โดยใช้คำสั่ง PUSH โดยค่าของ SP จะเพิ่มขึ้น 1 แล้วเก็บข้อมูลลงในสแตกที่ตำแหน่งที่ SP ชี้อยู่ ซึ่งจะทำให้ตำแหน่งแรกของการเก็บข้อมูลของสแตกคือ 08 การนำข้อมูลออกจากสแตก ทำได้จากการทำคำสั่ง POP โดยจะดึงข้อมูลออกจากสแตกในตำแหน่งที่ SP ชี้อยู่ออกไปกำหนดให้กับหน่วยความจำที่รับข้อมูล แล้วค่าของสแตกจะลดลง 1 หลังจากมีการนำข้อมูลออกจากสแตกไปแล้ว สามารถกำหนดพื้นที่ของแสตกไปอยู่ในตำแหน่งใดๆ ในหน่วยความจำข้อมูลภายในตัวไมโครคอนโทรลเลอร์ได้โดยการกำหนดค่าให้กับ SP ตามที่ต้องการแต่ไม่ควรใช้ในตำแหน่งเริ่มต้น 00 เนื่องจากเป็นตำแหน่งเดียวกับรีจิสเตอร์ R0-R7 ในแบงค์ 0 และไม่ควรกำหนดตำแหน่งสูงเกินไป เพราะจะทำให้มีพื้นที่ของสแตกน้อยซึ่งอาจไม่เพียงพอกับใช้งาน

DPTR : (Data Pointer) เป็นรีจิสเตอร์ขนาด 16 บิตใช้สำหรับเป็นตัวชี้ตำแหน่งของหน่วยความจำภายนอกหรือตำแหน่งของอุปกรณ์อินพุตเอาต์พุตที่ไมโครคอนโทรลเลอร์ต้องการติดต่อด้วย และใช้เป็นตัวกำหนดตำแหน่งเริ่มต้น (Base) ของตารางในการทำงานเกี่ยวกับ Look up table รีจิสเตอร์ DPTR ประกอบด้วยรีจิสเตอร์ขนาด 8 บิต 2 ตัวคือรีจิสเตอร์ DPH และ DPL ซึ่งเราสามารถเลือกการใช้งานในลักษณะ 8 บิต 2 ตัวหรือ 16 บิต 1 ตัวก็ได้

P0-P3 : พอร์ต P0 - พอร์ต P3 เป็นรีจิสเตอร์ขนาด 8 ซึ่งค่าที่อยู่ในรีจิสเตอร์เหล่านี้จะเป็นค่าเดียวกับค่าของสัญญาณที่ขาต่างๆ ของพอร์ตดังนั้นการส่งข้อมูลออกไปที่พอร์ตจะทำได้โดยการกำหนดค่าให้กับรีจิสเตอร์ P0 - P3 ในกรณีที่ใช้พอร์ตเป็นอินพุตเราสามารถอ่านสัญญาณที่ต่อกับขาของพอร์ตได้จากการอ่านค่าของรีจิสเตอร์ P0 - P3 เช่นกัน

SBUF : เป็นรีจิสเตอร์ขนาด 8 บิต ทำหน้าที่เป็นบัฟเฟอร์ของการรับส่งข้อมูลแบบอนุกรม โดยมีทั้งบัฟเฟอร์ด้านรับข้อมูลและบัฟเฟอร์ด้านส่งข้อมูล บัฟเฟอร์ด้านรับข้อมูลจะทำหน้าที่เก็บข้อมูลที่ส่งเข้ามาให้กับไมโครคอนโทรลเลอร์ทางพอร์ตอนุกรม เมื่อข้อมูลเข้ามาครบจำนวนบิตแล้วค่าที่อยู่ในบัฟเฟอร์ตัวนี้จะถูกไมโครคอนโทรลเลอร์อ่านออกไปอีกทีหนึ่ง สำหรับบัฟเฟอร์ด้านส่งข้อมูลจะรับข้อมูลขนาด 8 บิตที่ไมโครคอนโทรลเลอร์ต้องการส่งออกไปทางพอร์ตอนุกรม โดยข้อมูลในบัฟเฟอร์จะถูกส่งออกไปยังพอร์ตอนุกรมทีละบิตต่อไป

RCAP2H, RCAP2L : เป็นรีจิสเตอร์ขนาด 8 บิต ที่มีอยู่เฉพาะในเบอร์ 8032 และ 8052 ซึ่งใช้ร่วมกับ Timer/Counter 2 ในโหมด Capture รีจิสเตอร์ RCAP2H, RCAP2L จะทำหน้าที่เก็บค่าปัจจุบันของรีจิสเตอร์ TH2 และ TL2 ตามลำดับเมื่อมีการเปลี่ยนแปลงของสัญญาณ T2EX สำหรับการทำงานในโหมด Auto reload รีจิสเตอร์ RCAP2H, RCAP2L จะทำหน้าที่เก็บค่าเริ่มต้นของตัวตั้งเวลา และค่านี้จะถูกส่งไปยัง TH2 และ TL2 เมื่อมีการเปลี่ยนแปลงของสัญญาณ T2EX

IP : (Interrupt Priority Control) เป็นรีจิสเตอร์ขนาด 8 บิตทำหน้าที่ควบคุมการจัดลำดับความสำคัญของ การอินเตอร์รัพต์

IE : (Interrupt Enable) เป็นรีจิสเตอร์ขนาด 8 บิตทำหน้าที่ควบคุมการส่งสัญญาณและการตอบรับอินเตอร์รัพต์ของไมโครคอนโทรลเลอร์

TMOD : (Timer Mode Control) เป็นรีจิสเตอร์ขนาด 8 บิตใช้สำหรับควบคุมการเลือกโหมดการทำงานของ Timer/Counter0 และ Timer/Counter1

TCON : (Timer Control) เป็นรีจิสเตอร์ขนาด 8 บิตใช้สำหรับควบคุมการทำงานของ Timer/Counter0 และ Timer/Counter1

T2CON : เป็นรีจิสเตอร์ขนาด 8 บิตใช้สำหรับควบคุมการทำงานของ Timer/Counter2

SCON : (Serial Control) เป็นรีจิสเตอร์ขนาด 8 บิตใช้สำหรับควบคุมการทำงานของ พอร์ตอนุกรม

PCON : เป็นรีจิสเตอร์ทำหน้าที่ควบคุมการใช้กำลังไฟของตัวไมโครคอนโทรลเลอร์

PSW : (Program Status Word) เป็นรีจิสเตอร์ขนาด 8 บิตทำหน้าที่แสดงสถานะการทำงานของโปรแกรม หรือเรียกว่าแฟลก ซึ่งจะมีการเปลี่ยนแปลงหลังจากมีการทำงานในคำสั่งต่างๆ ที่มีผลต่อแฟลกและยังใช้เป็นตัวเลือกตำแหน่งแบงค์ของรีจิสเตอร์ (Register Bank) R0-R7 อีกด้วย ผลของบิตต่างๆ สามารถนำไปเป็นเงื่อนไขในการกระโดด (Jump) ได้ ค่าของบิตต่างๆ ใน PSW สามารถเซต/เคลียร์ ด้วยคำสั่งทางซอฟต์แวร์ได้ แฟลกต่าง ๆ ของรีจิสเตอร์ PSW จะอยู่ในตำแหน่งของบิตต่างๆ ดังรูปที่ 2.6

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	0V	-	P

รูปที่ 2.6 แสดงแฟลกต่าง ๆ ของรีจิสเตอร์ PSW

CY : (Carry Flag) เป็นแฟลกตัวทดทำหน้าที่หลายอย่าง เช่น เป็นตัวทดในกรณีของการบวก ทำหน้าที่เป็นตัวยืมในกรณีของการลบ ใช้เป็นตัวร่วมกับแอคคิวมูเลเตอร์ในการหมุนบิต และเราสามารถใช้ค่าของ CY เป็นเงื่อนไขของการกระโดด (Jump) ได้

AC : (Auxiliary Carry Flag) เป็นแฟลกตัวทดช่วยระหว่าง บิตที่ 3 และบิตที่ 4 ซึ่งแฟลกนี้จะถูกเซตเป็น 1 หากทำการบวกเลขแล้วมีการทดจากบิตที่ 3ไปยังบิตที่ 4 หากไม่มีการทดแฟลกนี้จะเป็น 0 แฟลก AC จะใช้สำหรับการทำงานในลักษณะของเลข BCD ซึ่งจะถูกใช้โดยซีพียู

F0 : (Flag 0) เป็นแฟลกที่ใช้งานทั่วไปซึ่งเราสามารถใช้เป็นแฟลกสภาวะ (Status flag) ของโปรแกรมได้โดยการเซตหรือรีเซตด้วยคำสั่งทางซอฟต์แวร์

RS1-RS0 : (Register Bank Select) เป็นตัวกำหนดการเลือกพื้นที่ใช้งานของกลุ่มรีจิสเตอร์ R0 - R7 ซึ่งเป็นรีจิสเตอร์ที่ใช้งานทั่วไป พื้นที่ที่เลือกใช้ได้แสดงในตารางต่อไปนี้

ตารางที่ 2.1 แสดงการเลือก Register

RS1	RS0	แบงค์	ตำแหน่ง
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

OV : (Overflow Flag) เป็นแฟลกที่แสดงค่าโอเวอร์โฟลว์ ซึ่งจะถูกเซตหรือเคลียร์จากการทำงานของคำสั่งทางคณิตศาสตร์ การเปลี่ยนแปลงของแฟลกจะพิจารณาในลักษณะของ เลข 8 บิต เครื่องหมาย (8 bits sign number) โดยบิตซ้ายมือสุดเป็นเครื่องหมาย (0 = บวก 1 = ลบ) เลขลบจัดเก็บในลักษณะของเลข 2 ' s complement ค่าของจำนวนเลขที่เก็บด้วย 8 บิต เครื่องหมายจะมีค่าบวกสูงสุด 127 (0 1 1 1 1 1 1 1) และมีค่าค่าลบต่ำสุด คือ -128 (1 0 0 0 0 0 0 0) ในการนำเลข 2 จำนวนมารวมกันแล้วได้ผลลัพธ์มากกว่า +127 หรือต่ำกว่า -128 (เกินความสามารถของ 8 บิต) จะทำให้แฟลก OV ถูกเซตเป็น 1

แฟลกโอเวอร์โฟลว์ จะเกิดเมื่อนำเลขลบ 2 จำนวนมารวมกันแล้วได้ค่าลบต่ำกว่า -128 หรือนำเลขบวก 2 จำนวนมารวมกันแล้วได้ค่าเป็นบวกมากกว่า 127 การนำเลข 2 จำนวนที่จำนวนหนึ่งเป็นค่าลบ และอีกจำนวนหนึ่งเป็นค่าบวก มารวมกันจะไม่ทำให้เกิดโอเวอร์โฟลว์

P : (Parity Flag) เป็นแฟลกที่แสดงจำนวนบิตที่เป็น 1 ในแอคคิวมูเลเตอร์ โดยแฟลก P จะถูกเซตเป็น 1 เมื่อแอคคิวมูเลเตอร์มีจำนวนบิตที่เป็น 1 เป็นคี่ (odd) และแฟลก P จะถูกเคลียร์เป็น 0 เมื่อแอคคิวมูเลเตอร์มีจำนวนบิตที่เป็น 1 เป็นคู่ (even) ดังนั้นจำนวนบิตในแอคคิวมูเลเตอร์รวมกับแฟลก P จะมีจำนวนบิตที่เป็น 1 ทั้งหมดเป็นจำนวนคู่

2.1.4 การใช้งานรีจิสเตอร์

โดยปกติไมโครคอนโทรลเลอร์ตระกูล MCS-51 จะทำการประมวลผลข้อมูลครั้งละ 1 ไบต์ ซึ่งกระทำกับรีจิสเตอร์ภายในโดยที่รีจิสเตอร์แต่ละตัวเก็บข้อมูลได้ขนาด 1 ไบต์เช่นกัน เช่น รีจิสเตอร์ A ซึ่งเป็นแอคคิวมูเลเตอร์ (accumulator) ทำหน้าที่เป็นรีจิสเตอร์กลางสำหรับการคำนวณทางคณิตศาสตร์หรือทางลอจิกของตัวกระทำ 2 ตัว ตัวอย่างเช่น ถ้าต้องการบวกค่า 10 กับข้อมูลตัวหนึ่ง ให้ทำการโหลดข้อมูลไปเก็บไว้ในรีจิสเตอร์ A ก่อน จากนั้นให้ใช้คำสั่งนำค่า 10 ไปบวกกับ A ผลที่ได้จกการบวกข้อมูลและค่า 10 จะถูกเก็บไว้ในรีจิสเตอร์ A นอกจากรีจิสเตอร์ A ทำการบวกด้วยการกำหนดค่าโดยตรงแล้ว มันยังทำการคำนวณร่วมกับรีจิสเตอร์ขนาด 8 บิตตัวอื่นๆ ได้อีกด้วย

ทั้งไมโครโปรเซสเซอร์ และไมโครคอนโทรลเลอร์จะมีรีจิสเตอร์สำหรับใช้งานในคำสั่งพิเศษ โดยผู้เขียนโปรแกรมอาจกำหนดขึ้นเองได้ โดยที่กำหนดให้อยู่ในตำแหน่งแอดเดรสพิเศษซึ่งในที่นี้มีค่า

มากกว่า 07FH ขึ้นไป ตัวอย่างเช่น แอคคิวมูเลเตอร์ถูกกำหนดให้ใช้หน่วยความจำภายในที่ 0E0H รีจิสเตอร์เหล่านี้เรียกว่า รีจิสเตอร์หน้าที่พิเศษ (Special Function Registers: SFRs) จำนวนของรีจิสเตอร์หน้าที่พิเศษอาจจะมีไม่เท่ากันในไมโครคอนโทรลเลอร์แต่ละหมายเลขในตระกูล MCS-51 ขึ้นอยู่กับคำสั่งที่ได้ทำการตั้งค่าไว้ เพราะรีจิสเตอร์หน้าที่พิเศษเหล่านี้ถูกรวม หรือใช้พื้นที่ในหน่วยความจำภายในของไมโครคอนโทรลเลอร์ ซึ่งหน่วยความจำภายในหรือแรมภายในจะมีขนาดไม่เท่ากันในแต่ละหมายเลข

นอกจากรีจิสเตอร์หน้าที่พิเศษหรือ SFR แล้วยังมีรีจิสเตอร์สำหรับใช้งานทั่วไปอีก 8 ตัวคือ รีจิสเตอร์ R0-R7 รีจิสเตอร์ทั้ง 8 ตัวนี้ถูกบรรจุอยู่ภายในแรมภายในไมโครคอนโทรลเลอร์ในรูปของแบงก์ และใช้สำหรับเก็บข้อมูลชั่วคราวระหว่างการประมวลผล

2.1.5พอร์ทอินพุทและพอร์ทเอาท์พุท

พอร์ท คือ แอดเดรสหนึ่งที่ได้รับการกำหนดไว้เพื่อการโอนย้ายข้อมูลระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์ภายนอก การกำหนดประเภทของการติดต่อขึ้นอยู่กับทิศทางการไหลของข้อมูล เมื่อพิจารณาจากไมโครคอนโทรลเลอร์เป็นหลัก จากรูปที่ 2.7(a) เป็นการใช้ไมโครคอนโทรลเลอร์เป็นเอาท์พุทพอร์ท และจากรูป 2.7(b) เป็นการใช้ไมโครคอนโทรลเลอร์เป็นอินพุทพอร์ท

รูปที่ 2.7 การใช้ไมโครคอนโทรลเลอร์เป็นอินพุทและเอาท์พุทพอร์ท

2.1.5.1 การใช้งานพอร์ทเป็นอินพุท

การใช้งานพอร์ทเป็นการอินพุทข้อมูลจะต้องเริ่มต้นด้วยการส่งข้อมูลที่มีค่าเป็น 1 ออกมาทางบิตของพอร์ทสั้นก่อนเป็นอันดับแรก เพื่อหยุดการทำงานของทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทของบิตนั้น ทำให้ขาสัญญาณของบิตถูกต่อเข้ากับตัวต้านทานซึ่งทำหน้าที่ Pull-up ภายในซึ่งมีผลทำให้บิตนั้นของพอร์ท 1 , 2 และ 3 เป็นสภาวะลอจิกสูง ตัวต้านทานนี้มีค่าประมาณ 50 k ohm ซึ่งเป็นค่าที่สูงมาก และทำให้อุปกรณ์ภายนอกสามารถขับสัญญาณของพอร์ทเหล่านี้เป็นลอจิกต่ำได้ง่าย สำหรับบิตของพอร์ท 0 นั้นแม้ว่าจะมีหลักการทำงานที่คล้ายคลึงกันกับบิตของพอร์ทอื่นๆ แต่เนื่องจากไม่มีตัวต้านทานซึ่งทำหน้าที่ Pull-up ภายในไว้ ทำให้เมื่อทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นหยุดการทำงาน ก็จะเป็นผลให้สัญญาณนี้อยู่ในสภาวะอิมพีแดนซ์สูงแทน

2.1.5.2 การใช้งานพอร์ทเป็นเอาท์พุท

เมื่อมีการส่งข้อมูลที่มีค่าเป็น 0 ให้กับแต่ละบิตของพอร์ททุกพอร์ท ข้อมูลนี้จะถูกส่งให้กับฟลิปฟลอปซึ่งจะค้างค่านี้ไว้ และมีผลทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นทำงาน ดังนั้นขาสัญญาณก็จะมีสภาวะลอจิกเป็นลอจิกต่ำด้วย

ส่วนการส่งข้อมูลที่มีค่าเป็น 1 ออกมานั้น ในกรณีที่เป็นการทำงานในแต่ละบิตของพอร์ท 1 , 2 หรือ 3 จะทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นหยุดทำงาน มีผลทำให้ขาของสัญญาณเป็นลอจิกสูงด้วยตัวต้านทานที่ Pull-up อยู่ภายในนั้น แต่สำหรับการใช้งานในแต่ละบิตทางพอร์ท 0 นั้นจะมีผลแตกต่างออกไป โดยขาสัญญาณจะมีสภาวะอิมพีแดนซ์สูงแทน เนื่องจากไม่มีตัวต้านทานภายในเชื่อมต่ออยู่นั้นเอง ดังนั้นการใช้งานพอร์ท 0 เป็นการนำข้อมูลออกทางเอาท์พุท จึงจำเป็นต้องใช้ตัวต้านทานภายนอก Pull-up สัญญาณไว้กับลอจิกสูงแทน

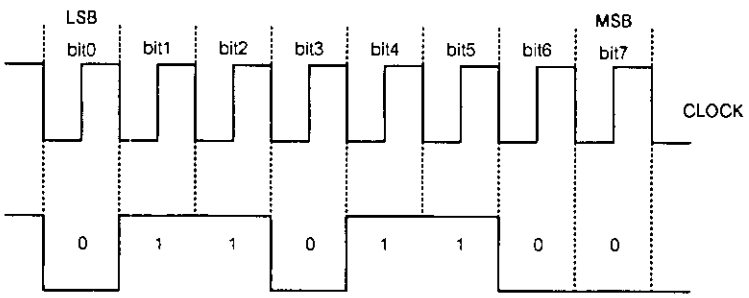
2.1.6 การเชื่อมต่อไมโครคอนโทรลเลอร์ตระกูล 8051 กับคอมพิวเตอร์ผ่านพอร์ตอนุกรม

การเคลื่อนย้ายข้อมูลจากคอมพิวเตอร์ไปยังอุปกรณ์ต่อพ่วงอื่นๆ หรือคอมพิวเตอร์ด้วยกันนั้น มี 2 วิธี คือ การรับส่งข้อมูลแบบขนานและการรับส่งข้อมูลแบบอนุกรม การรับส่งข้อมูลแบบขนานจะเป็นการรับส่งข้อมูลคราวละ 4 หรือ 8 บิต ในเวลาเดียวกันซึ่งจะทำให้การรับและส่งข้อมูลทำได้ที่ความเร็วสูง ซึ่งหมายความว่า จำนวนสายที่ใช้ในการส่งจะต้องมีมากเท่ากับจำนวนบิตของข้อมูลที่จะส่งด้วย นอกจากนี้ยังจะต้องรวมถึงสายที่ใช้สำหรับควบคุมและการตรวจสอบการรับส่งข้อมูลด้วย ซึ่งอาจจะต้องใช้สายมากเป็น 2 เท่าของจำนวนบิตข้อมูลที่จะส่งก็ได้ ซึ่งก็เป็นปัญหาในเรื่องของราคาของสายที่ใช้ในการเชื่อมต่อแบบขนานมักจะมีราคาแพง

ในขณะที่การรับส่งข้อมูลแบบอนุกรมเป็นการรับส่งข้อมูลครั้งละ 1 บิต แต่ก็สามารถรับส่งข้อมูลได้คราวละหลายๆ บิตได้ หากแต่จะต้องมีการตกลงกันระหว่างตัวส่งและตัวรับว่า จะรับส่งข้อมูลคราวละกี่บิต ตัวรับจะต้องรอข้อมูลมาให้ครบทุกบิตเสียก่อนจึงจะทำการประมวลผล ส่งผลให้การสื่อสารข้อมูลแบบอนุกรมอาจมีความเร็วต่ำกว่าแบบขนาน อย่างไรก็ตาม การรับส่งข้อมูลแบบอนุกรมนั้นสามารถใช้สายสัญญาณที่มีความยาวมากกว่าแบบขนาน ทำให้ระยะทางในการสื่อสารข้อมูลแบบอนุกรมสามารถทำได้มากกว่า

2.1.6.1 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมนั้นจะแบ่งออกได้เป็น 2 แบบ คือ การสื่อสารอนุกรมแบบซิงโครนัส และ การสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วย ตัวอย่างการส่งข้อมูลแบบซิงโครนัสคือคีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของสัญญาณนาฬิกา ส่วนสายอีกเส้นหนึ่งจะเป็นสายของข้อมูล ดังนั้นการติดต่อแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้น คือ สัญญาณนาฬิกา, ข้อมูล และกราวด์



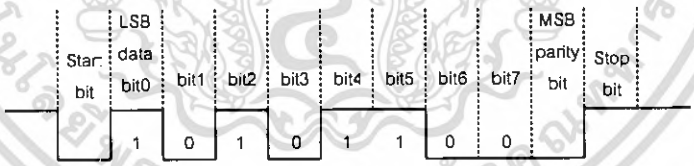
รูปที่ 2.8 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม

2.1.6.2 การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัส คือการรับส่งข้อมูลไปในสายสัญญาณ โดยไม่จำเป็นต้องส่งสัญญาณนาฬิการ่วมด้วยเหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับและภาคส่งนี้ว่า อัตราการถ่ายเทข้อมูล หรือ อัตราบอด (baudrate) ที่มีหน่วยเป็น บิตต่อวินาที (bit per second : bps)

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนด้วยกัน คือ

1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลอนุกรมจะมีขนาด 5, 6, 7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Bit) จะมีขนาด 1 บิตหรือไม่มี
4. บิตปิดท้าย (Stop Bit) จะมีขนาด 1, 1.5 หรือ 2 บิต



รูปที่ 2.9 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส

รูปที่ 2.9 แสดงรูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส ซึ่งเมื่อไม่มีข้อมูลที่จะส่ง ขาดาต้าจะมีสถานะลอจิก "1" ซึ่งจะเรียกสถานะนี้ว่าสถานะว่าง (idle stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขาดาต้ามีลอจิก "0" ด้วยช่วงระยะเวลา 1 บิต ซึ่งจะเรียกบิตนี้ว่าบิตเริ่มต้น จากนั้นบิตข้อมูลจะถูกส่งออกไปโดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจจะมีจำนวนบิต 5, 6, 7 หรือ 8 บิตก็ได้ จากนั้นจะตามด้วยบิตพาริตี้ ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่จะส่งคือบิตปิดท้าย ซึ่งจะให้ขาดาต้ามีสถานะลอจิก 1 อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต, 1.5 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับและการส่งข้อมูลแบบอะซิงโครนัส เรียกว่า Universal Asynchronous Receiver / Transmitter หรือ UART อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัส คือ ค่าอัตราบอด ซึ่งก็คือค่าอัตราการเข้ารหัสที่ใช้ในการรับและส่งข้อมูล อัตราบอดมาตรฐานที่ใช้สำหรับพอร์ตอนุกรม RS-232 ได้แก่ 110, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที และมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดค่าอัตราบอดได้สูงถึง 115200 บิตต่อวินาที เนื่องจากอัตราบอดคือจำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิตและบิตปิดท้าย 1 บิตความยาวของข้อมูลที่รับส่งนี้เท่ากับ 10 บิต ถ้าใช้อัตราบอดในการส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก็จะสามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที และถ้ามีการใช้พาริตี้ความเร็วในการรับส่งข้อมูลจะเหลือเป็น 872 ไบต์ต่อวินาที

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นแบบคี่ (odd), แบบคู่ (even) หรืออาจไม่มีการตรวจสอบพาริตี้ก็ได้ การตรวจสอบพาริตี้เป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิก "1" ภายในข้อมูลที่ส่งไป 1 ไบต์ว่ามีจำนวนรวมเป็นเลขคู่หรือเลขคี่โดยต้องรวมบิตพาริตี้เข้าไปด้วย ยกตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิตและมีค่าเท่ากับ 99 ฐานสิบหก หรือ 10011001 ฐานสอง จะเห็นว่าข้อมูลในไบต์นี้มีลอจิก "1" จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้น ถ้ากำหนดค่าพาริตี้เป็นคู่ ค่าในบิตพาริตี้จะต้องมีลอจิก "0" แต่ถ้าพาริตี้เป็นคี่ ค่าที่บิตพาริตี้จะต้องเป็น "1" เพื่อให้ข้อมูล 1 ไบต์ รวมทั้งบิตพาริตี้มีจำนวนบิตที่เป็นลอจิก "1" มีจำนวนรวมกันเป็นเลขคี่ ตารางที่ 2.2 แสดงตัวอย่างของบิตพาริตี้ในการรับส่งข้อมูลอนุกรม

ตารางที่ 2.2 แสดงบิตพาริตี้ของข้อมูล

ข้อมูล	บิตพาริตี้คู่	บิตพาริตี้คี่
00000000	0	1
00000001	1	0
00000010	1	0
00000011	0	1
00000100	1	0
11111110	1	0
11111111	0	1

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART ซึ่งทางภาครับจะต้องทำการกำหนดคุณสมบัติการตรวจสอบพาริตี้ให้ตรงกันว่าจะตรวจสอบพาริตี้คี่หรือพาริตี้คู่ จากนั้นภาครับของ UART จะทำการตรวจสอบค่าพาริตี้ที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก "1" ทั้งหมดรวมทั้งบิตพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแสดง

ข้อผิดพลาดออกมาให้ผู้ใช้ทราบ นับเป็นการตรวจสอบความผิดพลาดที่เกิดขึ้นในการถ่ายทอดข้อมูลที่ง่ายที่สุด แต่จะเชื่อถือได้เมื่อมีบิตข้อมูลที่ทำการส่งผิดพลาดแค่เพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิตผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่งจะไม่มีการตรวจสอบพาริตี้

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ UART หมายเลข 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT ใช้ UART หมายเลข 8250 UART ชิปเหล่านี้มีระดับแรงดันเป็นแบบทีทีแอล (0 และ +5V) แต่เพื่อให้มีแรงดันเป็นไปตามมาตรฐาน RS-232 และเพื่อให้การรับส่งข้อมูลสามารถทำได้ที่ระยะทางไกลขึ้นระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้น โดยลอจิก "0" มีระดับแรงดัน +3V ถึง +12V ในขณะที่ลอจิก "1" มีระดับแรงดัน -3V จนถึง -12V

2.2 การสื่อสารข้อมูลอนุกรม

การสื่อสารข้อมูลอนุกรมเป็นการรับหรือส่งข้อมูลในลักษณะกลุ่มของบิตคราวละหนึ่งบิตเรียงลำดับเรื่อยไปจนสิ้นสุดการสื่อสารแบบนี้จะมีข้อแตกต่างจากการสื่อสารแบบขนานเป็นอย่างมาก เนื่องจากข้อมูลมีการโอนย้ายมาพร้อมกันจึงมีความจำเป็นต้องใช้จำนวนเส้นสัญญาณมากขึ้นตามจำนวนบิตของข้อมูลด้วย ในขณะที่การสื่อสารแบบอนุกรมนั้นต้องการเส้นสัญญาณเพียงสองหรือสามเส้นเท่านั้น ดังนั้นในการสื่อสารแบบขนานจึงไม่เหมาะสมในการสื่อสารกับอุปกรณ์ภายนอกเป็นระยะไกลๆ เพราะจะทำให้สิ้นเปลืองค่าใช้จ่ายมาก

ความเร็วของการสื่อสารแบบอนุกรม

เนื่องจากการสื่อสารข้อมูลแบบอนุกรมเป็นการรับส่งข้อมูลในลักษณะกลุ่มของบิตข้อมูล (Bit Stream) ดงนั้นจึงต้องให้ความสนใจในการพิจารณาถึงเรื่องอัตราความเร็วในการรับส่งบิตเหล่านี้เป็นลำดับแรก โดยทั่วไปมักจะระบุกันในหน่วยของจำนวนบิตข้อมูลภายในเวลาหนึ่งวินาทีเรียกว่า อัตราบอด ตามค่ามาตรฐานเหล่านี้ ได้แก่ 110,150,300,1200,2400,4800,9600,19200 บอด ข้อมูลทั้ง 8 บิตนี้ หากถูกส่งออกมาด้วยอัตรา 2400บอด จะใช้เวลาในการส่งข้อมูลหนึ่งบิตมีค่าเท่ากับ 1/ 2400 หรือ 416 μS และเวลาในการส่งข้อมูลทั้ง 8 บิตมีค่าเท่ากับ (8*416) หรือ 3328μS

รูปแบบของการส่งข้อมูลอนุกรม

การสื่อสารอนุกรมแบบอะซิงโครนัสจะใช้การแปลงข้อมูลขนานให้เป็นอนุกรมแล้วเพิ่มบิตบางอย่างร่วมไปกับการส่งข้อมูลจริงซึ่งได้แก่

บิตเริ่มต้น (Start Bit)

บิตเริ่มต้นมีหน้าที่สำหรับการบ่งบอกให้ทราบถึงตำแหน่งจุดเริ่มต้นก่อนบิตข้อมูล ตามปกติแล้วค่าของบิตเริ่มต้นจะเป็นระดับลอจิกต่ำ

บิตแสดงภาวะความเป็นเลขคู่หรือเลขคี่ (Parity Bit)

บิตนี้มีหน้าทีเพื่อการตรวจสอบความถูกต้องของข้อมูลโดยทั่วไปมักเรียกว่า บิตพาริตี้และจะนำไปต่อท้ายบิตข้อมูล ค่าของบิตนี้จะขึ้นอยู่กับจำนวนค่าของบิตที่เป็น 1 ซึ่งจะเป็นได้สองลักษณะคือ พา

ริตี้คู่ (Even Parity) หรือพาริตี้คี่ (Odd Parity) ตัวอย่างเช่นระบบที่ติดต่อกัน โดยระบุว่าจะใช้พาริตี้คี่ ทางด้านส่งจะนำข้อมูลที่จะส่งมาพิจารณา จำนวนของบิตที่มีค่า 1 เป็นเลขจำนวนคู่อยู่แล้ว ค่าของบิตพาริตี้จะมีค่าเป็น 0 แต่หากว่าจำนวนของบิตที่มีค่าเป็น 1 เป็นจำนวนเลขคี่ ค่าของบิตพาริตี้จะมีค่าเป็น 1 การพิจารณาทางด้านรับเป็นการตรวจสอบจำนวนบิตที่มีค่า 1 ของข้อมูลที่ได้รับมาทั้งหมดรวมทั้งบิตพาริตี้ ถ้ามีค่าเป็นเลขจำนวนคู่ แสดงว่าข้อมูลที่รับเข้ามานี้ถูกต้อง หากไม่เป็นเลขจำนวนคู่แสดงว่าเกิดการผิดพลาดของข้อมูลขึ้น เป็นต้น

บิตสุดท้าย (Stop Bit) เป็นบิตที่เพิ่มขึ้นเพื่อระบุถึงขอบเขตการสิ้นสุดของกลุ่มบิตสุดท้ายนี้สามารถโปรแกรมได้คือ1 บิต 1 ½ บิตและ 2 บิต ดังนั้นกรณีของการส่งข้อมูล 8 บิตหากข้อมูลถูกส่งออกไปด้วยอัตราเร็ว 2400 บอด เวลาโดยรวมในการส่งข้อมูลหนึ่งไบต์จะมีค่าเป็น (12*416) μS หรือ 4.99 mS

การส่งข้อมูลอนุกรมของ 8051

พอร์ตอนุกรมของ 8051 มีโครงสร้างการทำงานในแบบที่เรียกว่า ฟูลดูเพล็กซ์ (Full Duplex) ในการรับและส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน โดยทางด้านวงจรของตัวส่ง (Transmitter) ประกอบด้วยข้อมูลออกไปยังพอร์ตอนุกรมทางขาสัญญาณ TxD (พอร์ต 3.1) ส่วนวงจรด้านตัวรับ (Receiver) ประกอบด้วย SUBF เช่นเดียวกับสัญญาณข้อมูลอนุกรมที่รับเข้ามาทางขาสัญญาณ RxD (พอร์ต 3.0)

พอร์ตอนุกรมของ 8051

สามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิต SM 0 และ SM 1 ซึ่งอยู่ในรีจิสเตอร์ควบคุม SCON การทำงานทั้ง 4 โหมดของพอร์ตอนุกรม มีดังนี้

โหมด 0 : ใช้รับส่งข้อมูล 8 บิต โดยการส่งจะเลื่อนออกทีละบิต โดยส่งบิต 0 ออกไปก่อนทางขา RxD และไม่มีการส่ง start bit แต่จะส่ง shift clock ทางขา TxD ความเร็ว 1/12 เท่าของ CPU CLOCK

โหมด1: ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART (Universal Asynchronous Receiver/Transmitter) โดยส่งแบบ 10 บิต ข้อมูล 8 บิต 1 start bit และ 1 stop bit และสามารถเปลี่ยนแปลงอัตราเร็วในการส่งข้อมูลได้ โดยขึ้นกับบิต SMOD ใน PCON และอัตราโอเวอร์โฟลว์ของ Timer 1

โหมด 2 : ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิต และกำหนดอัตราความเร็วในการส่งข้อมูลเท่ากับ 1/32 และ 1/64 ของ CPU CLOCK โดยโปรแกรมที่บิต SMOD ใน PCON

โหมด 3 : ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิต และสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้ โดยควบคุมที่บิต SMOD และอัตราโอเวอร์โฟลว์ของ Timer 1 นอกจากนี้ โหมด 2 และ 3 ยังมีการดำเนินการอีกแบบหนึ่ง โดยสามารถนำมาใช้ประโยชน์ในการสื่อสารข้อมูลแบบที่มีไมโครโปรเซสเซอร์หลายตัวทำงานร่วมกันได้ซึ่งมีชื่อเรียกว่า Multiprocessor Mode

ตารางที่ 2.3 การเลือกโหมดการทำงาน

SM 0	SM 1	SM 2	REN	TB 8	RB 8	TI	RI

รายละเอียดใน SCON

SM 1	SM 0	โหมด	การทำงาน
0	0	0	ทำงานเป็น shift register อัตราเร็วในการรับหรือส่งข้อมูลเท่ากับ 1/12 ของความถี่ออสซิลเลเตอร์
0	1	1	8 bit UART อัตราเร็วในการรับหรือส่งข้อมูลกำหนดเองได้
1	0	2	9 bit UART อัตราเร็วในการรับหรือส่งข้อมูล 1/32 หรือ 1/64 ของความถี่ออสซิลเลเตอร์
1	1	3	9 bit UARTอัตราเร็วในการรับหรือส่งข้อมูลกำหนดเองได้

SM 0, SM 1 บิตเลือกการทำงานแบบ
1 : เลือก Multiprocessor Mode ใช้ได้กับโหมด 2, 3
2: เลือก Single Processor Mode ใช้ได้กับทุกโหมด

REN บิตควบคุมให้รับหรือไม่รับข้อมูล
1 : ให้รับข้อมูลได้
2 : ห้ามรับข้อมูล

TB 8 (Transmit Bit D8) ข้อมูลบิตที่ 9 ที่ส่งออกไปในโหมด 2,3 ให้ใส่ในบิต TB 8

RB 8 (Receiver Bit D8) ข้อมูลบิตที่9 ที่รับเข้ามาจะเก็บในบิตนี้ ข้อมูลบิตที่ 9 ก็คือค่าใน TB 8 ทางด้านส่งนั้นเอง

TI บิต TI จะเป็น 1 เมื่อสิ้นสุดการส่งข้อมูล 1ไบต์

RI บิต RI จะเป็น 1 เมื่อรับข้อมูลเสร็จ 1ไบต์

2.2.1 กระบวนการรับและส่งข้อมูลอนุกรมของ 8051

การส่งข้อมูลออกทางพอร์ตอนุกรมของ 8051 จะเริ่มต้นขึ้นภายหลังเมื่อมีการเขียนข้อมูลลงใน SBUF ข้อมูลนี้จะถูกเลื่อนทีละบิตและส่งสัญญาณออกไปภายนอกโดยอัตโนมัติ เมื่อข้อมูลเหล่านี้ได้ส่งออกครบถ้วนแล้วจะทำให้ค่าของแฟล็ก TI ให้เป็น 1 เพื่อแจ้งให้ทราบว่าขณะนี้ SBUF ว่างและพร้อมที่จะส่งข้อมูลไบต์ต่อไป

แล้วในกรณีที่ใช้เขียนข้อมูลใหม่ลงในรีจิสเตอร์ SBUF โดยไม่รอให้แฟล็ก TI มีค่าเป็น 1 ก่อนจะมีผลทำให้ข้อมูลที่ส่งไปผิดพลาดได้ สำหรับการรับข้อมูลจากพอร์ตอนุกรมจะต้องเริ่มต้นโดยการกำหนดเซ็ทค่า

ดังนี้ REN (Receiver Enable) ให้มีค่าเป็น 1ก่อน หลังจากนั้นเมื่อข้อมูลภายนอกถูกส่งเข้ามายัง 8051 ทีละบิตจนครบ และเมื่อบิตสุดท้ายถูกเลื่อนเข้ามาเรียบร้อยแล้วข้อมูลนั้นจะถูกย้ายมาเก็บไว้ยังรีจิสเตอร์ SBUF และแฟล็ก RI ก็จะมีค่าเป็น 1 หลังจากนั้นก็จะเกิดการอินเตอร์รัพต์ขึ้น

2.2.2 การสื่อสารพอร์ตอนุกรม RS-232

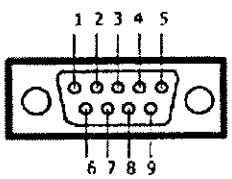
เป็น IC เป็นวงจรที่ทำหน้าที่เปลี่ยนแรงดันที่เข้ามาจาก Serial Port ไปเป็นแรงดันตามมาตรฐานของ RS-232 โดยเปลี่ยนระดับแรงดัน TTL เพื่อให้ใช้ได้กับไมโครคอนโทรลเลอร์

ลักษณะของการส่งข้อมูลแบบอนุกรมนั้น ข้อมูลจะส่งออกมาทีละบิตจากตัวส่งไปตัวรับข้อมูล ช่องสัญญาณในการส่งข้อมูลอาจใช้เพียง 1 หรือ 2 ช่องสัญญาณเท่านั้น ทำให้ค่าใช้จ่ายในการสื่อสารจะถูกกว่าแบบขนาน แต่อัตราการรับ-ส่งข้อมูลจะช้ากว่าแบบขนาน ในการส่งข้อมูลแบบอนุกรมข้อมูลที่ต้องการส่งจะอยู่ในลักษณะเป็นไบต์จะทยอยส่งทีละบิต และทางตัวรับจะต้องรับข้อมูลเข้ามาทีละบิตแล้วมารวมกันเป็นไบต์ซึ่งทางตัวรับต้องคอยตรวจสอบว่าบิตใดเป็นบิตเริ่มต้นหรือบิตสุดท้ายของข้อมูล การตรวจสอบนั้นจะขึ้นอยู่กับรูปแบบของรหัสของบิตข้อมูลที่ใช้ ซึ่งในการรับส่งข้อมูลแบบอนุกรมระหว่างไมโครคอมพิวเตอร์กับอุปกรณ์ภายนอกนั้น จำเป็นจะต้องมีมาตรฐานในการรับส่งข้อมูล ซึ่งมาตรฐานที่นิยมมากที่สุดก็คือมาตรฐาน RS-232

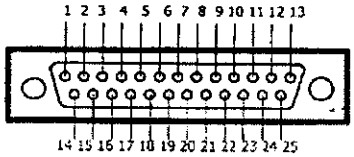
มาตรฐาน RS-232

เพื่อที่จะทำให้อุปกรณ์จากผู้ผลิตต่างกันทำงานร่วมกันได้ มาตรฐานหลายชนิดจึงได้รับการออกแบบขึ้น มาตรฐานที่ใช้กันอย่างกว้างขวางที่สุดคือ RS-232 ซึ่งโดยปรกติไมโครคอมพิวเตอร์จะมีพอร์ตที่เป็นแบบอนุกรมอยู่ในตัวแล้ว และจะทำหน้าที่รับส่งข้อมูลในแบบอนุกรม

ตามจุดประสงค์ของมาตรฐาน RS-232 นั้นเพื่อจะสามารถเชื่อมต่อกันระหว่างอุปกรณ์รับส่งปลายทาง (Data Terminal Equipment: DTE) เช่น พอร์ตของคอมพิวเตอร์หลักหรืออุปกรณ์ปลายทางกับอุปกรณ์สื่อสาร RS-232 เป็นข้อกำหนดของการอินเตอร์เฟสมาตรฐาน และสามารถและสามารถใช้เพื่อจุดประสงอื่นต่างกันไป เช่นการสื่อสารแบบซิงโครนัส (synchronous communication) และรูปแบบการสื่อสารที่ต้องการสัญญาณนาฬิกา และสัญญาณกำหนดจังหวะเพิ่มเติมขึ้นมา ในความเป็นจริงแล้วเราสามารถทำให้มีการสนทนากันระหว่าง DTE และ DCE โดยการใช้สายสัญญาณเพียง 3 เส้นเท่านั้น คือ ใช้สาย TD สาย RD และสายกราวด์เท่านั้น



รูปที่ 2.10 คอนเน็กเตอร์ 9 ขาหรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์)



รูปที่ 2.11 คอนเน็กเตอร์อนุกรม 25 ขาหรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์)

ตารางที่ 2.4 ข้อกำหนดของขาสัญญาณต่างๆ

หมายเลขขาสัญญาณ	ชื่อสายสัญญาณ
1	Protective Ground
2	Transmitted Data
3	Received Data
4	Request To Send
5	Clear To Send
6	Data Set Ready
7	Signal Common
8	Received Line Signal Detect
9	Reserved For Testing
10	Reserved For Testing
11	Unassigned
12	Secondary Received Line Signal Detect
13	Secondary Clear To Send
14	Secondary Transmitted Data
15	Transmission Signal Element Timing
16	Secondary Received Data
17	Receiver Signal Element Timing
18	Unassigned
19	Secondary Request To Send
20	Data Terminal Ready
21	Signal Quality Detector
22	Ring Indicator
23	Data Signal Rate Detector
24	Transmitter Signal Element Timing
25	Unassigned

ตารางที่ 2.5 แสดงรายละเอียดของสัญญาณที่ต่อจาก DTE ไปยัง DCE โดยใช้คอนเน็กเตอร์แบบ DB-25

หมายเลขขาสัญญาณ	ชื่อสายสัญญาณ
1	Protective Ground
2	Transmitted Data
3	Received Data
4	Request To Send
5	Clear To Send
6	Data Set Ready
7	Signal Common
8	Data Carrier Detect
20	Data Terminal Ready
22	Ring Indicator
23	Data Signal Rate Detector

ตารางที่ 2.6 รายละเอียดการต่อคอนเน็กเตอร์แบบ DB9 มาตรฐาน RS-232

หมายเลขขาสัญญาณ	ชื่อสายสัญญาณ
1	Data Carrier Detect
2	Received Data
3	Transmitted Data
4	Data Terminal Ready
5	Signal Common
6	Data Set Ready
7	Request To Send
8	Clear To Send
9	Ring Indicator

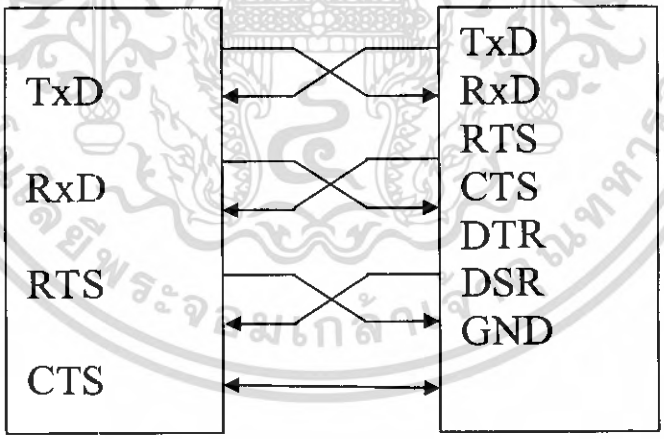
- ขา 1 (Protective Ground Circuit, AA) ขานี้จะต่อเข้ากับตัวถังของอุปกรณ์ และสามารถต่อเข้ากับกราวด์ภายนอกถ้าอุปกรณ์ต้องใช้งานขานี้
- ขา 2 (Transmitted Data Circuit BA, TD) เป็นขาสัญญาณข้อมูลที่ออกจากอุปกรณ์ DTE กระแสบิตอนุกรมจากขานี้ คือข้อมูลที่ถูกถ่ายทอดไปโดยโมเด็ม หรือถูกถอดรหัสโดยอุปกรณ์ DCE ที่มี

- ขา 3 (Received Data Circuit BB, RD) สัญญาณที่ขานี้จะถูกสร้างจากอุปกรณ์ DCE กระแสบิตอนุกรมนี้จะเกิดขึ้นที่อุปกรณ์ DTE ปลายทางและเป็นผลผลิตของวงจรรับข้อมูลของอุปกรณ์ DCE สัญญาณนี้มักเป็นข้อมูลที่ถูกสร้างขึ้นโดยอุปกรณ์ DCE
- ขา 4 (Request To Send Circuit CA,RTS) สัญญาณนี้จะเตรียมพร้อมอุปกรณ์ DCE สำหรับการทำงานส่งข้อมูลเมื่อสัญญาณ RTS นี้อยู่ในสภาวะ "ON" จะทำให้อุปกรณ์ DCE อยู่ในโหมดส่งข้อมูล (Transmit mode) ในขณะที่สัญญาณนี้อยู่ในสภาวะ "OFF" ทำให้อุปกรณ์ DCE อยู่ในโหมดรับข้อมูล (Receive mode) อุปกรณ์ DCE ควรจะตอบสนองต่อสัญญาณ RST ON โดยการทำให้สัญญาณ "OFF" เสียก่อน สัญญาณนี้จะถูกใช้ร่วมกับสัญญาณ DTR DSR และ DCD ขาสัญญาณ RTS จะถูกใช้อย่างมากในการควบคุมการไหลของข้อมูล
- ขา 5 (Clear To Send Circuit CB,CTS) สัญญาณนี้จะตอบรับกลับไปยังอุปกรณ์ DTE เมื่อได้รับสัญญาณ RTS และข้อมูลสามารถถูกส่งออกไปได้ ข้อมูลจะถูกส่งออกไปตามตัวกลางที่ใช้สื่อสารได้ ก็ต่อเมื่อสัญญาณ CTS นี้อยู่ในสภาวะ "ON" เท่านั้น สัญญาณนี้จะใช้ร่วมกับสัญญาณ DTR DSR และ DCD ขาสัญญาณนี้จะใช้ร่วมกับขา RTS สำหรับควบคุมการไหลของข้อมูล
- ขา 6 (Data Set Ready Circuit CC,DSR) สัญญาณ DSR จะบอกต่ออุปกรณ์ DTE ว่าอุปกรณ์ DCE ได้ต่อกับตัวกลางการสื่อสารที่ถูกต้องแล้ว และในบางกรณีจะบ่งชี้ว่าสายโทรศัพท์อยู่ในสภาวะ "OFF HOOK" สภาวะ "OFF HOOK" จะเป็นตัวบ่งชี้ว่าอุปกรณ์ DCE กำลังอยู่ในโหมด dialing หรือกำลังติดต่อกับอุปกรณ์ DCE อีกตัวหนึ่งอยู่ เมื่อสัญญาณ DSR อยู่ในสภาวะ "OFF" อุปกรณ์ DTE ก็ควรจะถูกกำหนดให้ไม่สนใจสัญญาณอื่นๆทั้งหมดจากอุปกรณ์ DCE ถ้าสัญญาณนี้ถูกทำให้อยู่ในสภาวะ "OFF" ก่ออุปกรณ์ DTR แล้วอุปกรณ์ DCE ก็จะสรุปว่าการสื่อสารนั้นสิ้นสุดลง
- ขา 7 (Signal Common Circuit AB) สายนี้จะให้สัญญาณอ้างอิงของกราวด์ร่วมกันสำหรับวงจรการแลกเปลี่ยนข้อมูลทั้งหมด ยกเว้นวงจร AA หรือ Protective Ground ข้อกำหนด RS-232B จะอนุญาตให้วงจรนี้ถูกตัดต่อเพิ่มเติมกับ Protective Ground ภายในอุปกรณ์ DCE ได้ ถ้าจำเป็น
- ขา 8 (Data Carrier Detect Circuit CF, DCD) ขานี้รู้จักกันในนามของ Receive Line Signal Detect (RLSD) หรือขา Carrier Detect (CD) สัญญาณนี้จะเกิด Active เมื่อเกิดสัญญาณพาหะที่เหมาะสมระหว่างอุปกรณ์ DCE ที่สถานีกับที่อยู่ระยะไกล เมื่อสัญญาณนี้อยู่ในสภาวะ "OFF" สัญญาณที่ขา RD ควรจะถูกทำให้ค้างอยู่ในสภาวะ "Mark" (สถานะ "1"ในเลขฐานสอง)
- ขา 20 (Data Terminal Ready Circuit CD, DTR) สัญญาณ DTR ถูกใช้ในการควบคุมสวิตช์อุปกรณ์ DCE เข้ากับตัวกลางในการสื่อสารสัญญาณ DTR ON บ่งชี้ว่าอุปกรณ์ DCE ที่กำลังเชื่อมต่อกันอยู่ก็ยังต่อร่วมกัน และถ้ายังไม่มีการเชื่อมต่อกันก็สามารถทำให้มีการเชื่อมต่อกันครั้งใหม่ได้ ปกติแล้วสัญญาณ DTR จะอยู่ในสภาวะ "OFF" เพื่อกระตุ้นให้เกิดสภาวะ "ON HOOK" (วางสาย) อุปกรณ์ DCE โดยการทำให้สัญญาณ DSR แอกทีฟ

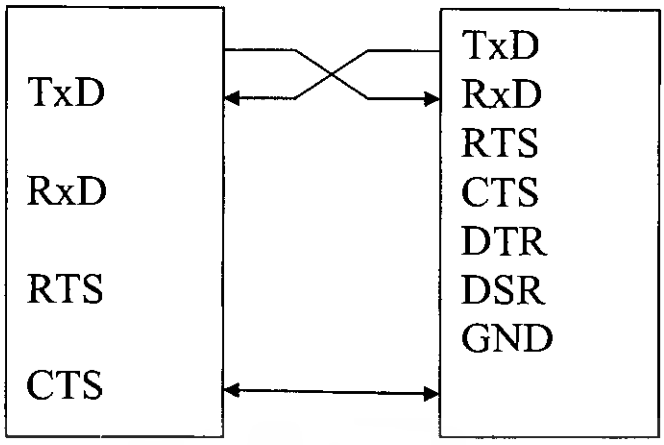
- ขา 22 (Ring Indicator Circuit CE, RI) สภาวะ "ON" ของขานี้จะบ่งชี้ว่า ได้รับสัญญาณเรียกสายโทรศัพท์จากตัวกลางในการสื่อสาร โดยปกติแล้วจะขึ้นอยู่กับโปรแกรมควบคุม ในการที่จะทำให้เกิดสัญญาณนี้ขึ้นหรือไม่
- ขา 23 (Data Signal Rate Detector Circuit CH/CI DSRD) วงจร CH เป็นส่วนประกอบของ DTE และวงจร CI เป็นส่วนประกอบของ DCE สัญญาณนี้ถูกใช้ในการเลือกใช้ค่าอัตราในการส่งข้อมูลค่าใดค่าหนึ่งในสองค่า ในกรณีที่ใช้โมเด็มที่มีอัตราการส่งข้อมูลได้ 2 ค่า (Dual Rate Modems) ถ้าสัญญาณนี้เป็น "ON" ก็จะเป็นการเลือกอัตราส่วนการส่งข้อมูลที่มีค่าสูงสุดในสองค่านั้น

2.2.3 ขั้นตอนการติดต่อระหว่างอุปกรณ์ DTE และ DCE

1. เมื่อจ่ายกำลังงานให้กับ DTE และอุปกรณ์ก็จะส่งสัญญาณ DTR ออกมา
2. อุปกรณ์ DCE ถูกเปิดขึ้นและรับรู้สัญญาณ DTR ที่ส่งมาจากอุปกรณ์ DTE
3. อุปกรณ์ DCE ส่งสัญญาณ DSR ออกมา และโมเด็มก็กระทำกระบวนการ OFF HOOK
4. ถ้าสายสัญญาณอยู่ในสภาพดีและปลายทางอีกด้านหนึ่งก็พร้อมจะรับข้อมูลแล้ว จะมีการตรวจจับสัญญาณพาหะแล้วอุปกรณ์ DCE ส่งสัญญาณ DCD ออกมา
5. อุปกรณ์ DCE จะตอบสนองด้วยการ ส่งสัญญาณ CTS ออกมา
6. การติดต่อสื่อสารก็เริ่มขึ้น โปรแกรมควบคุมจะทำการส่งหรือรับข้อมูล



(ก) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์ แบบ Null modem



(ข) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์ แบบ RS-232 โดยใช้สายสัญญาณเพียง 3 เส้น

รูปที่ 2.12 การต่ออุปกรณ์ภายนอกกับพอร์ตอนุกรมของคอมพิวเตอร์ในลักษณะต่างๆ

ส่วนลำดับขั้นในการตอบรับก็เป็นในทำนองนี้

1. อุปกรณ์ DTE จะส่งสัญญาณ DTR ออกมา
2. อุปกรณ์ DCE จะอยู่ในโหมดตอบรับอัตโนมัติ (auto answer mode) โดยมีสัญญาณออกมา
3. สถานีปลายทางส่งสัญญาณเรียกอุปกรณ์ DCE และอุปกรณ์ DCE ส่งสัญญาณ RI ออกมา
4. อุปกรณ์ DTE รับรู้ถึงอุปกรณ์ RI ที่ส่งมาจากเครื่องปลายทาง และอุปกรณ์ DCE ก็เข้าสู่สภาวะ OFF HOOK
5. อุปกรณ์ DCE ทำการแลกเปลี่ยนข้อมูลกับอุปกรณ์ DCE ที่อีกปลายทางหนึ่ง และมีการส่งสัญญาณ DCD ออกมา
6. อุปกรณ์ DTE จะส่งสัญญาณ RTS ออกมาหรืออาจจะรอข้อมูลก็ได้ ขึ้นอยู่กับโปรแกรมควบคุม
7. อุปกรณ์ DCE จะตอบสนองด้วยการส่งสัญญาณ DTS กลับออกมา
8. การติดต่อสื่อสารก็จะเริ่มขึ้น

2.3 Single chip 2.4 GHz Transceiver nRF2401

Mode of operation (โหมดดำเนินการ)

2.3.1 Overview

ระบบย่อยของ nRF2401 สามารถจัดอยู่ในโหมดหลักได้ดังต่อไปนี้ โดยจะขึ้นกับ 3 pins ควบคุม

ตารางที่ 2.7 nRF2401 subsystem main modes

Mode	PWR_UP	CE	CS
Active	1	1	0
Configuration	1	0	1
Stand by	1	0	0
Power down	0	X	X

2.3.2 Active modes

ระบบย่อย nRF2401 มี 2 Active โหมด (TX / Rx) :

- Shock Burst
- Direct Mode (not supported by nRF24E1)

ฟังก์ชันการทำงานของอุปกรณ์ในโหมดนี้เลือกโดย เนื้อหาของ Configuration word โดย Configuration word จะแสดงในส่วนของ Configuration

2.3.3 Shock Burst

เทคโนโลยี Shock Burst ใช้บนชิป FIFO เพื่อจับเวลาข้อมูลที่อัตราข้อมูลต่ำและส่งต่อที่อัตราข้อมูลสูงมาก ด้วยเหตุนี้จะสามารถทำให้เกิดการลดทอนของกำลังสูงสุด เมื่อดำเนินการระบบย่อย nRF2401 ใน Shock Burst เราจะได้อัตราข้อมูลสูงสุด (1 Mbps) โดย band 2.4 GHz โดยไม่ต้องใช้ไมโครคอนโทรลเลอร์ความเร็วสูงและราคาสูง (MCU) สำหรับการดำเนินการทางข้อมูล

โดยการใส่การประมวลผลข้อมูลความเร็วสูง สัมพันธ์กับ RF Protocol บนชิป nRF24E1 มีประโยชน์ ดังนี้

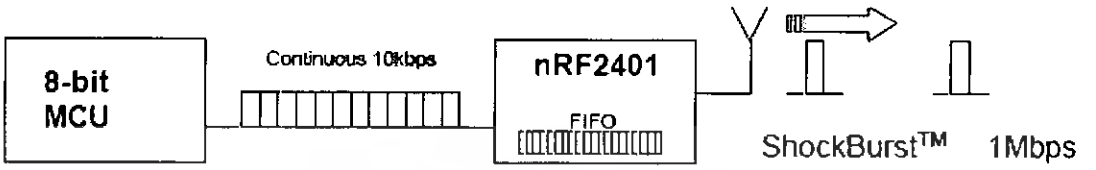
- ลดการใช้กระแส
- ระบบที่มีราคาถูก (ใช้งานได้ง่ายสำหรับ microcontroller ราคาไม่แพง)
- ลดอัตราการเสี่ยงของการชนกันในอากาศ เนื่องจากการส่งสัญญาณในช่วงเวลาสั้นๆได้ดีมาก

ระบบย่อย nRF 2401 สามารถโปรแกรมโดยใช้ สาย interface พื้นฐาน 3 เส้น ซึ่งอัตราของข้อมูลจะเลือกโดยความเร็ว CPU โดยยอมรับส่วนดิจิตอลของ application เพื่อจะ run ที่ความเร็วต่ำขณะที่อัตราข้อมูลบน RF Link มีค่าเพิ่มสูงสุด โหมด Shock Burst จะลดการใช้กระแสเฉลี่ยใน application

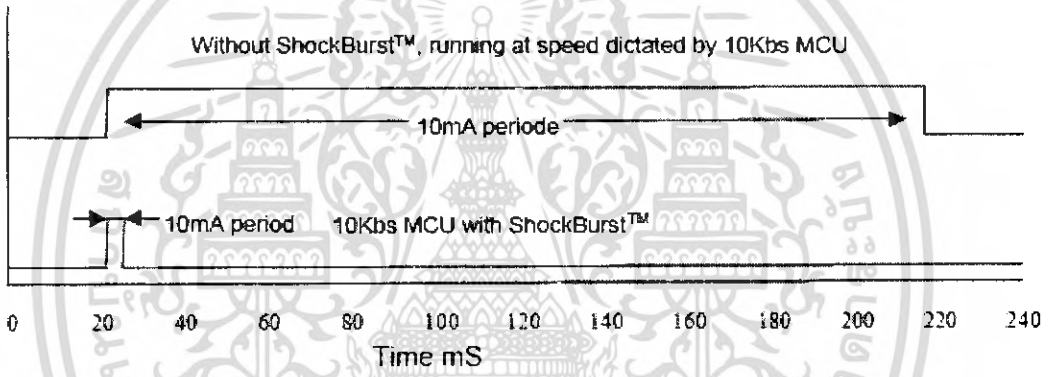
2.3.3.1 หลักการของ Shock Burst

เมื่อระบบย่อย nRF 2401 มีโครงร่างใน Shock Burst กระบวนการ Tx จะดำเนินการดังต่อไปนี้

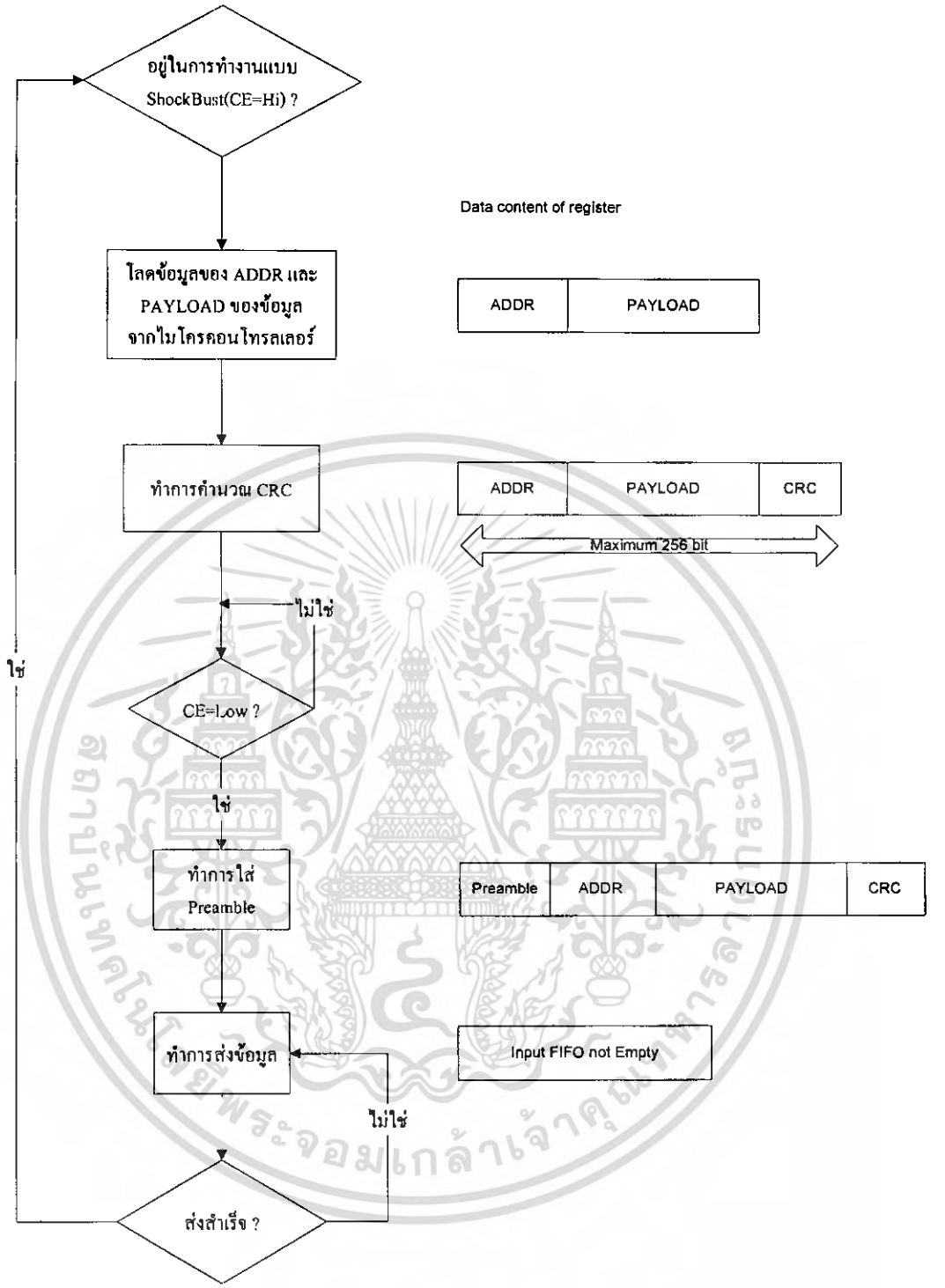
ตัวอย่าง 10 kbps



รูปที่ 2.13 จับเวลาข้อมูลโดย CPU ส่งโดยเทคโนโลยี Shock Burst



รูปที่ 2.14 การใช้กระแส RF โดยใช้และไม่ใช้เทคโนโลยี Shock Burst

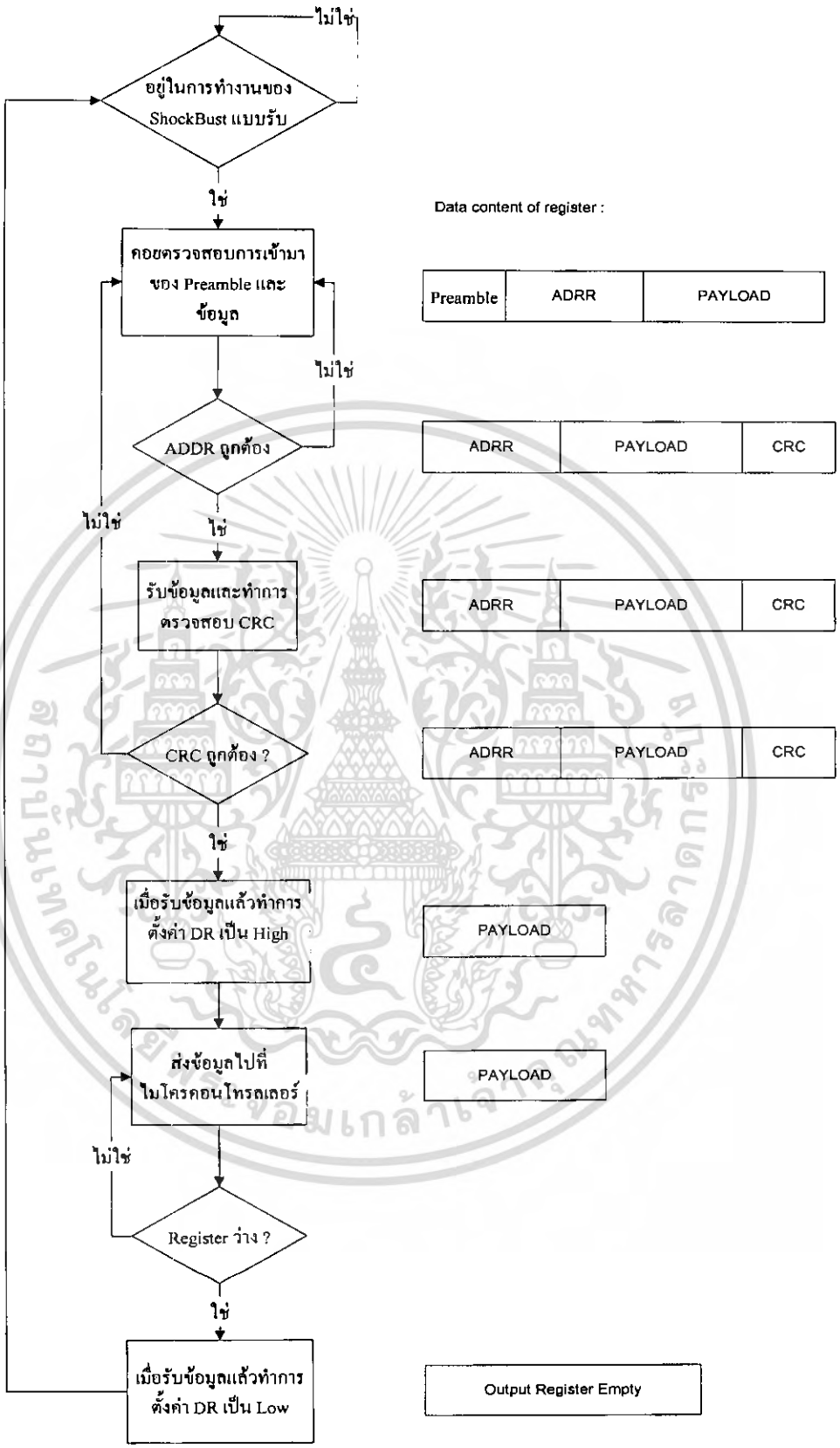


รูปที่ 2.15 แสดงแผนผังงาน(Flow Chart) ShockBurst การส่งของระบบย่อย nRF2401

2.3.3.2 การส่ง Shock Burst

CPU interface pins : CE, CLK1, DATA

1. เมื่อ CPU มีข้อมูลที่จะส่ง set CE high คือ กระตุ้น nRF2401 ให้ประมวลผลบนบอร์ด
2. แอดเดรสของโหมดรับ (Rx) และ payload ข้อมูลจะจับเวลาเข้าสู่ช่วงเวลาระบบย่อย nRF2401 protocol หรือ CPU ตั้งความเร็วน้อยกว่า 1 Mbps
3. CPU ตั้ง CE low คือ กระตุ้นการส่ง Shock Burst
4. Shock Burst
   - RF front end จะมีกำลังเพิ่มขึ้น
   - RF Package จะสมบูรณ์
   - ข้อมูลจะถูกส่งที่ความเร็วสูง (250 bps หรือ 1Mbps แล้วแต่ผู้ใช้)



รูปที่ 2.16 แสดงแผนผังงาน(Flow Chart) ShockBurst การรับระบบย่อย nRF2401

2.3.3.3 การรับ Shock Burst

CPU interface pins : CE, DRT, CLK1, DATA (1 ช่องรับ Rx)

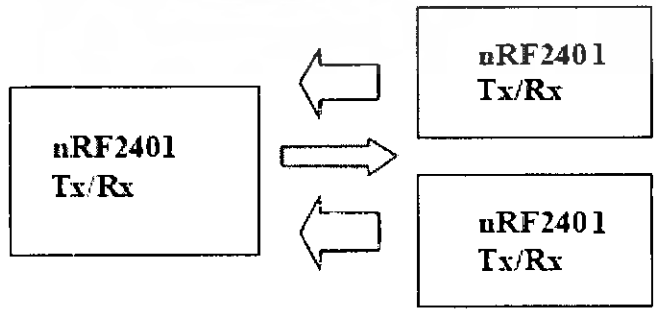
1. แอดเดรสถูกต้องและขนาดของ payload ของ RF package ที่เข้ามาจะ set เมื่อระบบย่อย nRF
2. 2401 เป็นโครงร่างของ Shock Burst Rx
3. เมื่อกระตุ้น Rx ; set CE high
4. หลังจากตั้ง 200us nRF 2401 จะตรวจสอบอากาศสำหรับการสื่อสารที่จะเข้ามา เมื่อ package ที่ถูกต้องถูกเรียกมา (แอดเดรสถูกต้องและพบ CRC) nRF 2401 จะทำการย้ายบทนำแอดเดรสและบิต CRC
5. จากนั้นระบบย่อย nRF 2401 จะ interrupt CPU โดยการตั้ง DR1 high
6. CPU จะ set CE low เพื่อป้องกัน RF front end (โหมดที่กระแสต่ำ)
7. CPU จะจับเวลาที่หยุด payload data ที่อัตราที่เหมาะสม
8. เมื่อ payload data ทั้งหมดถูกเรียกมา nRF 2401 set DR1 low อีกครั้ง และพร้อมสำหรับข้อมูลที่จะเข้ามา ถ้า CE ยัง high ระหว่างที่ download ข้อมูล ถ้า CE set low การลำดับ start up ใหม่จะสามารถเริ่มได้

2.3.4 DUOCiever Simultaneous two channel Receive mode

ในโหมด Shock Burst ของ nRF 24E1 สามารถรองรับความถี่ 2channel ที่ขนานกันละเป็นอิระต่อกัน ที่อัตราข้อมูลสูงสุด ได้สะดวกในเวลาเดียวกัน ซึ่งหมายความว่า

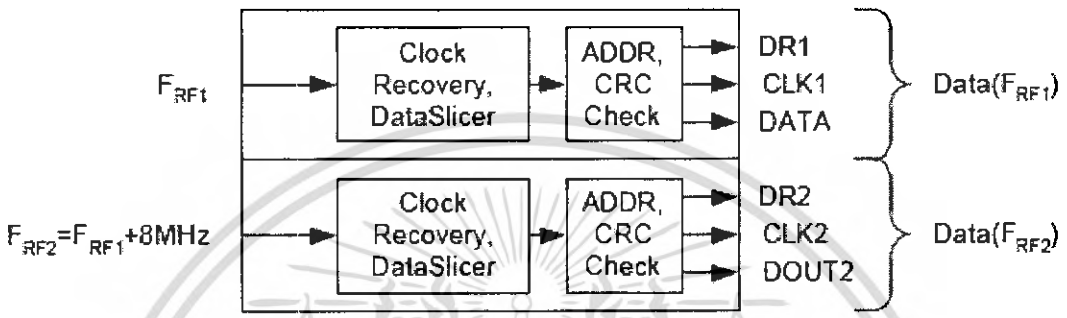
- RF 2401 สามารถรับข้อมูลจากเครื่องส่ง 2 เครื่อง ที่1 Mbps และ 8 MHz ผ่าน 1 เสาอากาศ
- Output จาก 2 ช่องสัญญาณข้อมูล จะป้อนให้กับ 2 interface pins ที่แยกจากกัน
- Data Channel 1: CLK1, DATA, DR1
- Data Channel 2: CLK2, DATA2, DR2

เทคโนโลยี DUOCiever ให้ 2 ช่องสัญญาณข้อมูลที่แยกกันอย่างละเอียดสำหรับ Rx และตอบสนองความต้องการต่อ 2 channel โดยมีระบบรับเพียง 1



รูปที่ 2.17 Simultaneous 2 channel receiver on nRF24E1

สำหรับ nRF24E1 จะสามารถรับที่ช่องสัญญาณข้อมูลช่องที่ 2 ได้นั้น ความถี่ช่องสัญญาณต้องอยู่สูงกว่าอยู่ 8 MHz ความถี่ของช่องสัญญาณข้อมูล nRF2401 ต้อง program เพื่อรับข้อมูลที่ความถี่ของช่องสัญญาณที่1 ไม่มีการใช้เวลา multiplexing เพื่อทำฟังก์ชันนี้ให้สำเร็จ ใน Direct Mode ถ้ามันไม่รับส่งได้พร้อมกันระหว่าง 2 ช่องสัญญาณข้อมูล CPU ต้องสามารถจัดการข้อมูลที่เข้ามาพร้อมกันได้ ส่วนใน Shock Burst สามารถทำให้ CPU หยุดจับเวลาข้อมูลช่วงนึง ขณะที่ไม่มีเกิดการสูญเสีย package ข้อมูลและไม่ลดประสิทธิภาพของ CPU



รูปที่ 2.18 DUOCiever โดยมี 2 ช่องสัญญาณรับข้อมูลที่เป็นอิสระต่อกัน

2.3.5 Device configuration

Configuration ทั้งหมดของระบบย่อย nRF2401 จะสามารถทำผ่านทาง 3-wire interface ของการติดต่อสื่อสาร (CS, CLK1, DATA) กับ register เดี่ยว โดย configuration word สามารถมีได้ถึง 18 bytes ส่วนconfiguration bit (DATA) จะต้องจับเวลาโดย clocked (CLK1) โดย msb มาก่อนขณะที่ CS = 1 ไม่เกิน 18 bytes อาจถูกดาวน์โหลด

2.3.5.1 Configuration for ShockBurst operation (Configuration สำหรับ การดำเนินการ ShockBurst)

Configuration word ใน Shock Burst ทำให้ระบบย่อย nRF2401 สามารถจัดการกับ RF protocol ได้ หาก protocol สมบูรณ์ และโหลดไประบบย่อย nRF2401 เพียง 1 byte :bit[7:0] จำเป็นที่จะต้อง update ตลอดช่วงที่ดำเนินการ

Configuration block โดยละเอียด สำหรับ Shock Burst ดังนี้

- ความกว้างของ payload: ระบุจำนวนของบิต payload ใน RF package ซึ่งมันทำให้ระบบย่อย nRF2401 สามารถแบ่งแยกระหว่างข้อมูล payload และ CRC bytes ในตัวรับ package ได้
- ความกว้าง address: set จำนวนบิตที่แอดเดรสใช้ใน RF package ซึ่งมันทำให้ระบบย่อย nRF2401 สามารถแบ่งแยกระหว่างแอดเดรสและข้อมูล package ได้
- แอดเดรส (Rx ช่อง 1 และ 2) : กำหนดแอดเดรสสำหรับข้อมูล
- CRC : สามารถทำให้ on-chip CRC generate และถอดรหัสได้

Configuration Block นี้ โดยการยกเว้น CRC จะละเอียดสำหรับ package ซึ่งระบบ nRF2401 ที่จะรับในโหมด Tx CPU ต้อง generate แอดเดรส และส่วน payload ให้เหมาะกับระบบย่อย nRF2401 เพื่อจะรับข้อมูลเมื่อใช้ระบบย่อย nRF2401 on-chip รูปแบบบ CRC ต้องแน่ใจว่า CRC นั้นใช้ได้และใช้ที่ความยาวเดียวกันทั้งอุปกรณ์ Tx และ Rx

PRE-AMBLE	ADDRESS	PAYLOAD	CRC

รูปที่ 2.19 DATA package set-up

2.3.5.2 Configuration for Direct Mode operation (Configuration สำหรับการดำเนินการใน Direct Mode)

สำหรับใน Direct mode จะดำเนินการเพียงแค่ใน 2 ไบต์แรกของ Configuration word ที่เกี่ยวข้อง

ตารางที่ 2.8 ตารางของ Configuration word Configuration word จะ shift MSB ก่อนที่ clock ขอบขาขึ้น configuration ใหม่จะเกิดที่ขอบขาลงของCS

	Bit position	Number of bits	Name	Function
ShockBurst™ configuration	143:120	24	TEST	Reserved for testing
	119:112	8	DATA2_W	Length of data payload section RX channel 2
	111:104	8	DATA1_W	Length of data payload section RX channel 1
	103:64	40	ADDR2	Up to 5 byte address for RX channel 2
	63:24	40	ADDR1	Up to 5 byte address for RX channel 1
	23:18	6	ADDR_W	Number of address bits (both RX channels).
	17	1	CRC_L	8 or 16 bit CRC
	16	1	CRC_EN	Enable on-chip CRC generation/checking.
General device configuration	15	1	RX2_EN	Enable two channel receive mode
	14	1	CM	Communication mode (Direct or ShockBurst™)
	13	1	RFDR_SB	RF data rate (1Mbps requires 16MHz crystal)
	12:10	3	XO_F	Crystal frequency
	9:8	2	RF_PWR	RF output power
	7:1	7	RF_CH#	Frequency channel
	0	1	RXEN	RX or TX operation

2.3.6 Configuration Word Detailed Description (อธิบายรายละเอียดของ configuration word) อธิบายฟังก์ชันของ 144 บิต (บิต 143 = MSB) ซึ่งใช้ configure ระบบ nRF2401

Configuration อุปกรณ์ทั่วไป: bit [15:0]

Configuration Shock Burst: bit [119:0]

Test Configuration: bit [143:120]

ตารางที่ 2.9 Configuration data word

MSB				TEST			
D143	D142	D141	D140	D139	D138	D137	D136
Reserved for testing							
1	0	0	0	1	1	1	0	Default

MSB								TEST							
D135	D134	D133	D132	D131	D130	D129	D128	D127	D126	D125	D124	D123	D122	D121	D120
Reserved for testing														Close PLL in TX	
0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	Default

DATA2_W							
D119	D118	D117	D116	D115	D114	D113	D112
Data width channel#2 in # of bits excluding addr/crc							
0	0	1	0	0	0	0	0	Default

DATA1_W							
D111	D110	D109	D108	D107	D106	D105	D104
Data width channel#1 in # of bits excluding addr/crc							
0	0	1	0	0	0	0	0	Default

ADDR2												
D103	D102	D101	....	D71	D70	D69	D68	D67	D66	D65	D64	
Channel#2 Address RX (up to 40bit)												
0	0	0	....	1	1	1	0	0	1	1	1	Default

ADDR1												
D63	D62	D61	....	D31	D30	D29	D28	D27	D26	D25	D24	
Channel#1 Address RX (up to 40bit)												
0	0	0	....	1	1	1	0	0	1	1	1	Default

ADDR_W						
D23	D22	D21	D20	D19	D18	
Address width in # of bits (both channels)						
0	0	1	0	0	0	Default

CRC		
D17	D16	
CRC Mode 1 = 16bit, 0 = 8bit	CRC 1 = enable; 0 = disable	
0	1	Default

RF-Programming															LSB	
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
Two Ch	BUF	OD	XO Frequency			RF Power		Channel selection							RXEN	
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0	Default

โดย MSB bit ควรจะถูกโหลดสู่ configuration register เป็นอันดับแรก

Default Configuration word: h8E08.1C20.2000.0000.00E7.0000.0000.E721.0F04.

2.3.6.1 Shock Burst configuration:

ช่วงบิต [119:16] ประกอบด้วย segments อย่างละเอียดของ configuration register ไปยัง protocol การทำงานของ Shock Burst บิตรวมของ 120 บิตต้อง shift เข้าเพื่อจะ switch ระหว่าง Rx และ Tx

2.3.6.1.1 PLL_CTRL

ตารางที่ 2.10 PLL setting

PLL_CTRL		
D121	**D120**	**PLL**
0	0	Open TX/Closed RX
0	1	Open TX/Open RX
1	0	Closed TX/Closed RX
1	1	Closed TX/Open RX

Bit 121-120:

PLL_CTRL ควบคุมการ set ของ PLL ใน Tx จะไม่เกิดการเบี่ยงเบน สำหรับโหมดดำเนินการปกติ 2 บิตนี้ต้องเป็น bit low

2.3.6.1.2 DATAx_W

ตารางที่ 2.11 จำนวนของบิตใน payload

DATA2_W							
119	118	117	116	115	114	113	112

DATA1_W							
111	110	109	108	107	106	105	104

Bit 119-112:

DATA2_W ความยาว package ของ RF payload สำหรับสัญญาณช่อง 2

Bit 111-104:

DATA1_W ความยาว package ของ RF payload สำหรับสัญญาณช่อง 1

NOTE: จำนวนบิตทั้งหมดใน Shock Burst RF package อาจไม่เกิน 256! ความยาวสูงสุดของส่วน payload หาได้จาก DATAx_W (bits) = 256 – ADDR_W – CRC โดย

ADD_W: ความยาวแอดเดรสของ Rx ที่ set ใน configuration word Bit[23:18]

CRC: Check sum 8 หรือ 16 บิต ที่เซ็ตใน configuration word Bit[17]

PRE: Preamble 8 บิตซึ่งรวมอัตโนมัติ

แอดเดรส และ CRC ที่สั้นขึ้น จะทำให้มีที่สำหรับ payload data มากขึ้น

2.3.6.1.3 ADDRx

ตารางที่ 2.12 แอดเดรสของตัวรับ 1 และ 2

ADDR2											
103	102	101	....	71	70	69	68	67	66	65	64

ADDR1											
63	62	61	....	31	30	29	28	27	26	25	24

Bit 103-64:

ตัวรับ ADDR2 แอดเดรส channel2 มีได้ถึง 40 บิต

Bit 63-24:

ตัวรับ ADDR1 แอดเดรส channel1 มีได้ถึง 40 บิต

NOTE: บิตใน ADDRx มากกว่าความกว้างที่ set ใน ADDR_W ซึ่งยืดยาวเกินไปและสามารถ set เป็น logic 0 ได้

2.3.6.1.4 ADDR_W & CRC

ตารางที่ 2.13 จำนวนบิตที่ต้องคงไว้สำหรับ RX address + CRC setting

			ADDR_W			CRC_L	CRC_EN
23	22	21	20	19	18	17	16

Bit 23-18:

ADDR_W :จำนวนบิตที่ต้องสงวนไว้สำหรับแอดเดรส Rx ใน Shock Burst

NOTE: จำนวนบิตแอดเดรสสูงสุดคือ 40 บิต (5 ไบต์) ค่าที่มากกว่า 40 บิต ADDR_W จะไม่สามารถใช้ได้

Bit 17:

CRC_L: ความยาว CRC ที่จะคำนวณใน Shock Burst

Logic 0: 8 bit CRC

Logic 1: 16 bit CRC

Bit 16:

CRC_EN: ทำให้ CRC on-chip Tx และ Rx ใช้ได้

Logic 0: on-chip CRC generation / checking disable

Logic 1: on-chip CRC generation / checking enable

NOTE:

บิด 8 CRC อาจเพิ่มจำนวน payload ใน Shock Burst ได้แต่จะลดความสมบูรณ์ของระบบลง

2.3.6.2 General RF configuration

General RF configuration ของ configuration word จะควบคุม RF และ พารามิเตอร์ที่เกี่ยวกับอุปกรณ์

2.3.6.2.1 Modes

ตารางที่ 2.14 RF operational setting

RX2_EN	CM	RFDR_SB	XO_F			RF_PWR	
15	14	13	12	11	10	9	8

Bit 15: Rx2_EN

Logic 0: Channel 1 รับ

Logic 1: Channel 2 รับ

NOTE: ในตัวรับทั้ง 2 channel nRF 24E1 รับทั้ง 2 channel แยกความถี่ของช่องสัญญาณทันที ความถี่ของตัวรับ channel 1 จะ set ในบิต [7-1] ของ configuration word ตัวรับ channel 2 จะเหนือตัวรับ channel 1 อยู่ 8 channel

(8 MHz)

Bit 14:

Communication Mode:

Logic 0: ระบบย่อย nRF 2401 ดำเนินการใน Direct Mode

Logic 1: ระบบย่อย nRF 2401 ดำเนินการใน Shock Burst Mode

Bit 13:

RF Data Rate:

Logic 0: 250 kbps

Logic 1: 1 Mbps

NOTE: ใช้ 250 kbps แทน 1 Mbps จะช่วยปรับปรุง sensitivity ของตัวรับ 10 db 1Mbps ต้องใช้ crystal 16 MHz

Bit 12-10:

XO_F: เลือกความถี่ crystal ที่จะใช้

ตารางที่ 2.15 crystal frequency setting

XO FREQUENCY SELECTION			
D12	D11	D10	Crystal Frequency [MHz]
0	0	0	4
0	0	1	8
0	1	0	12
0	1	1	16
1	0	0	20

Bit 9-8:

RF_PWR set nRF 24E1 RF กำลังของเอาท์พุทในโหมดการส่ง

ตารางที่ 2.16 RF output power setting

RF OUTPUT POWER		
D9	D8	P [dBm]
0	0	-20
0	1	-10
1	0	-5
1	1	0

2.3.6.2.2 RF channel & direction

ตารางที่ 2.17 ช่องความถี่ และ การ set Rx / Tx

RF_CH#							RXEN
7	6	5	4	3	2	1	0

Bit 7-1:

RF_CH# ตั้งความถี่ช่องสัญญาณของ nRF 24E1 ให้ดำเนินการความถี่ของช่องสัญญาณในการส่งได้โดย

Channel RF = 2400 MHz + RF_CH#*1.0 MHz

RF_CH#: อาจตั้งไว้ระหว่าง 2400 MHz และ 2527 MHz

NOTE: ช่วงที่มากกว่า 83 สามารถใช้ได้เพียงแค่ในอาณาเขตที่แน่ใจได้

ความถี่ช่วงสัญญาณในช่องข้อมูล2 หาได้โดย

Channel RF = 2400 MHz + RF_CH#*1.0 MHz + 8 MHz (รับที่ PIN#4)

RF_CH#: อาจตั้งไว้ระหว่าง 2400 MHz และ 2527 MHz

Bit 0:

Active Mode

Logic 0: transmit mode โหมดส่ง

Logic 1: receive mode โหมดรับ

2.3.7 Data package Description

PRE-AMBLE	ADDRESS	PAYLOAD	CRC

รูปที่ 2.20 Data package Diagram

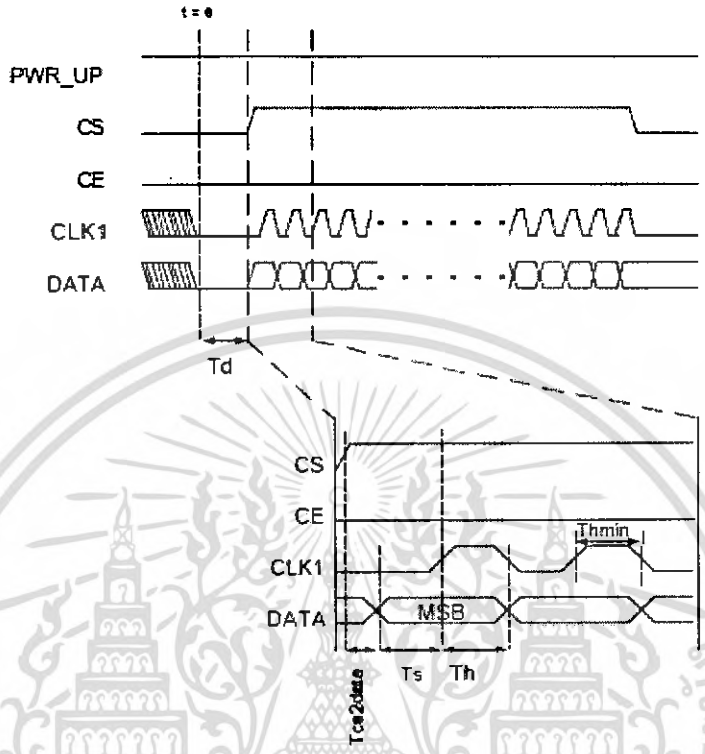
ชุดข้อมูลสำหรับการสื่อสารใน Shock Burst mode และ Direct mode จะแบ่งเป็น 4 ส่วน ดังนี้

ตารางที่ 2.18 รายละเอียดของ Data package

1. PREMBLE	- ส่วน preamble ต้องมีใน Shock Burst mode และ Direct mode - Preamble มีความยาว 8 บิตและ ขึ้นกับบิตแรกของแอดเดรส PREAMBLE 1st ADDR_BIT 01010101 0 10101010 1 - Preamble จะใส่ไปในชุดข้อมูลโดยอัตโนมัติและดังนั้นจึงมีที่ว่างพิเศษสำหรับ payload ใน Direct mode MCU ต้องควบคุม preamble ใน ShockBurst mode Rx preamble จะถูกย้ายจากรับข้อมูลเอาท์พุท ใน Direct mode preamble จะเห็นได้ชัดเจนสำหรับข้อมูลเอาท์พุท
2. ADDRESS	- ส่วนของแอดเดรสต้องมีใน ShockBurst โหมด และ Direct โหมด - ความยาว 80 – 40 บิต - แอ่ดเดรสจะถูกย้ายโดยอัตโนมัติจากชุดรับใน ShockBurst mode ใน Direct mode MCU ต้องควบคุมแอดเดรส
3. PAYLOAD	- ข้อมูลถูกส่ง - ใน ShockBurst โหมด payload มีขนาดน้อยที่สุด 256 บิต (แอดเดรส 8-40 บิต +CRC 8หรือ16 บิต - ใน Direct mode ความยาวมากสุดสำหรับ 1 Mbps คือ 4000 บิต
4. CRC	- CRC จะเลือกได้ใน ShockBurst โหมด และไม่ใช้ใน Direct โหมด - ความยาว 8 หรือ 16 บิต - ShockBurst Rx CRC จะถูกย้ายจากรับข้อมูลเอาท์พุท

2.3.8 Configuration mode timing

เมื่อ 1 บิต หรือ มากกว่า 1 บิต ใน configuration word จำเป็นที่จะต้องเปลี่ยนแปลง เราจะไปดูได้จาก timing

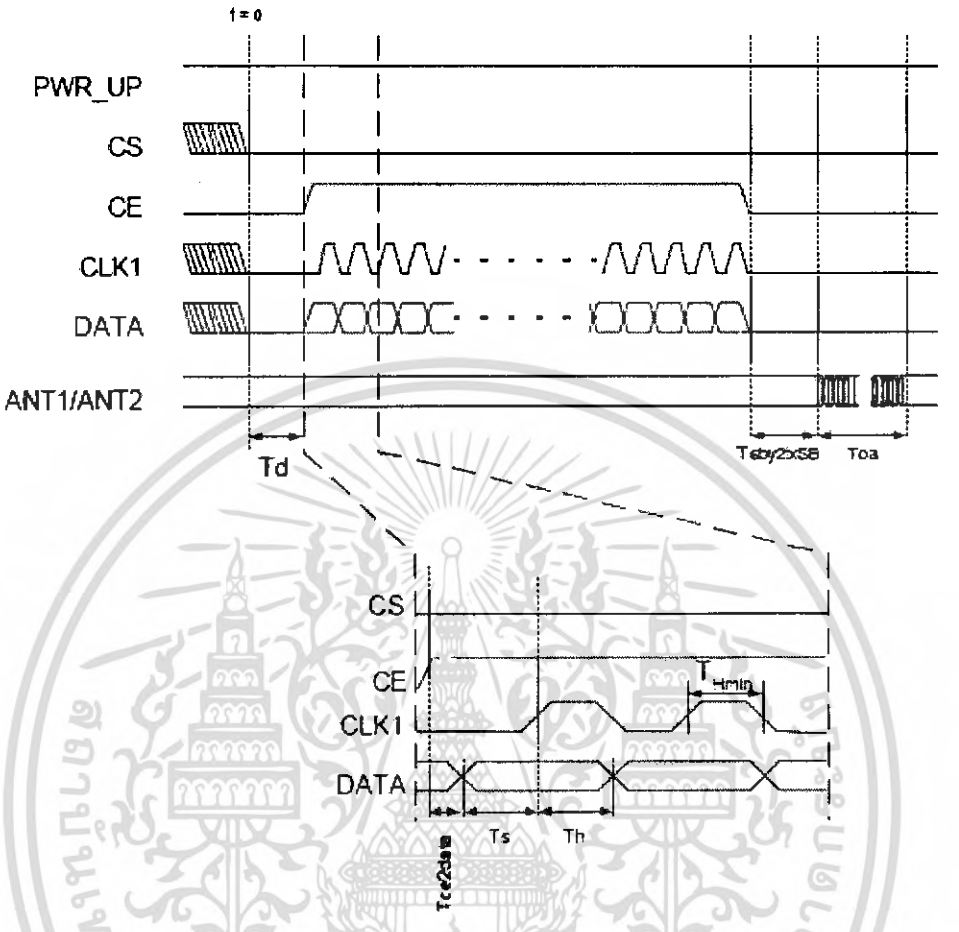


รูปที่ 2.21 Timing Diagram สำหรับ configuration ระบบย่อย nRF2401

ถ้า configuration mode เข้ามาจาก power down CS สามารถ set high หลังจาก Tpd2sby โดยแสดงให้เห็นในรูปที่ 2.20

2.3.9 ShockBurst Mode Timing

ShockBurst TX:

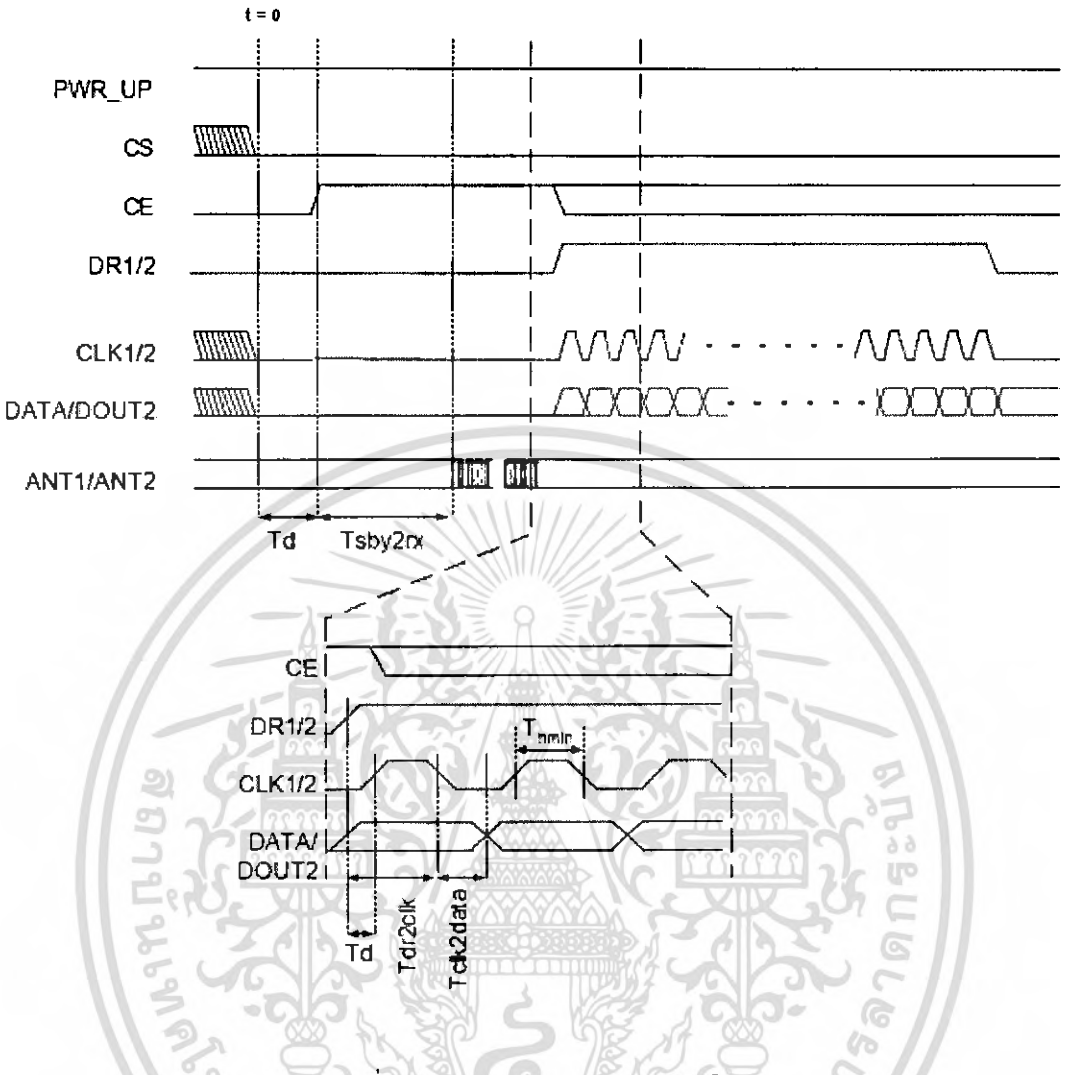


รูปที่ 2.22 Timing ของ ShockBurst ใน TX

ความยาวชุดข้อมูลและอัตราข้อมูลทำให้ delay Toa (Time on air : เวลาในอากาศ) ซึ่งแสดงในสมการ บิตข้อมูลจะเป็นผลรวมของบิตที่รวมทุกๆบิด CRC และบิต preamble ซึ่งอาจเพิ่มขึ้น

Toa = 1/data rate*(#databits+1)

ShockBurst Rx:



รูปที่ 2.23 Timing ของ ShockBurst ใน Rx

CE อาจยังคงสถานะ high ช่วงที่ดาวน์โหลดข้อมูล แต่จะทำให้ค่าการใช้กระแสสูงกว่า (19mA) และผลประโยชน์คือ เวลา start-up น้อย (200us) เมื่อ DR1 ขาลง

2.4 การเชื่อมต่อคีย์แพดเข้ากับไมโครคอนโทรลเลอร์ MCS-51

จากโครงงานจะทำการต่อคีย์แพดเข้ากับไมโครคอนโทรลเลอร์ที่พอร์ต 1 โดยใช้สายทั้งหมด 7 เส้น เป็นสายของคอลัมน์ 3 เส้นและสายของโรว 4 เส้น โดยไมโครคอนโทรลเลอร์จะทำการส่งข้อมูล "0" ไปยัง P1.4 – P1.6 ตามลำดับ ในทุกครั้งที่มีการส่งข้อมูลไปยังสายคอลัมน์ของคีย์แพด ไมโครคอนโทรลเลอร์จะทาการอ่านค่าที่ P1.0 – P1.3 เข้ามาด้วย หากไม่มีการกด ค่าของ P1.0 – P1.3 ก็จะเป็น"1" ทั้งหมด ถ้าหากมีการกดคีย์ ค่าของ P1.0 – P1.3 ก็จะไม่เป็น 1111 อีกต่อไป เป็นการแจ้งให้ทราบว่า มีการกดคีย์แพดขึ้นแล้ว จากนั้นไมโครคอนโทรลเลอร์ก็จะทำการค้นหาตำแหน่งต่อไป โดยการค้นหา

ตำแหน่งนั้น สิ่งที่จะได้มาอย่างแรกคือ ค่าตำแหน่งของคีย์นั้น จากนั้นก็จะนำค่าตำแหน่งนั้นไปเปิดตารางข้อมูล เพื่อที่จะได้ค่าที่ต้องการนำไปแสดงผลที่แท้จริง

2.5 การเชื่อมต่อกับโมดูลแอลซีดี(LCD Module)

ในโมดูลแอลซีดีนั้นจะมีส่วนประกอบ 3 ส่วนหลักดังนี้

- ตัวแสดงผล(Display) ภายในเป็นผลึกเหลวที่สามารถแสดงผลให้เห็นได้โดยอาศัยแสงจากภายนอก ดังนั้นจึงต้องมีมุมในการมองข้อมูลที่แสดงบนจอแอลซีดี
- ตัวควบคุม(Controller) เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอก มาควบคุมการทำงานของโมดูลแอลซีดี เช่น ลบจอภาพ แสดงตัวอักษรหรือเลื่อนเคอร์เซอร์เป็นต้น ตัวควบคุมนี้ใช้ชิบควบคุมโดยเฉพาะชิบ ที่นิยมใช้คือเบอร์ HD44780 และ HD61830 โดย HD44780 จะใช้ควบคุม LCD แบบอักขระ และ HD61830 ใช้ควบคุม LCD แบบกราฟฟิก
- ตัวขับ(Driver) เป็นตัวรับสัญญาณจากตัวควบคุมมาขับให้ตัวแสดงผลแสดงข้อมูลตามที่กำหนด ชิบทีใช้ทำงานเป็นตัวขับนี้ได้แก่ เบอร์ HD44100H และ MSM5259 เป็นต้น

2.5.1 โครงสร้างภายในของตัวควบคุมโมดูล LCD

ในการใช้งานโมดูล LCD จำเป็นต้องทำความเข้าใจเกี่ยวกับโครงสร้างและคำสั่งในการควบคุมให้ดีเสียก่อน โดยในที่นี้จะทำการแสดงตัวอย่าง โมดูล LCD แบบอักขระ เพราะสามารถเข้าใจง่าย โดยบล็อกไดอะแกรมภายในของชิบควบคุม LCD เบอร์ HD44780H ซึ่งใช้ในโมดูล LCD แบบอักขระประกอบด้วย

บัฟเฟอร์อินพุตเอาท์พุต เป็นส่วนที่ใช้ในการติดต่อรับส่งข้อมูลกับอุปกรณ์ภายนอก เพื่อที่จะถ่ายทอดข้อมูลเข้าออกภายในตัวควบคุม

รีจิสเตอร์คำสั่ง (Instruction Register:IR) เป็นรีจิสเตอร์ที่ใช้รับข้อมูลจากอุปกรณ์ภายนอก เพื่อถ่ายทอดต่อไปยังหน่วยความจำที่ทำหน้าที่เก็บข้อมูลแสดงผล หรือนำข้อมูลไปสร้างตัวอักษรเพิ่มเติมในแรมเก็บตัวอักษร

แรมเก็บข้อมูลแสดงผล(Display Data RAM:DDRAM) เป็นหน่วยความจำแบบแรมทำหน้าที่เก็บข้อมูลที่มาจากรีจิสเตอร์ DR ตัวควบคุมจะนำข้อมูลใน DDRAM นี้ไปเปิดตาราง (Look up table) ของตัวอักษรที่เก็บไว้ในหน่วยความจำรอมและแรมเก็บตัวอักษร เพื่อนำไปแสดงที่ตัวแสดงผล

รอมเก็บตัวอักษร (Character Generator ROM:CGROM)เป็นหน่วยความจำแบบรอมที่ใช้เก็บข้อมูลตัวอักษรหรือสัญลักษณ์ทีสามารถอ่านออกไปแสดงที่ตัวแสดงผลได้ มีขนาด 7200 บิต โดยจะถูกอ่านด้วยของข้อมูลใน DRAM

แรมเก็บตัวอักษร(Character Generator RAM:CGRAM) เป็นหน่วยความจำแรมที่ใช้เก็บอักษรที่มีการสร้างเพิ่มเติมขึ้นใหม่ ในกรณีที่ตัวอักษรใน CGROM ไม่เพียงพอ มีขนาด 512 บิต การเขียนและอ่านค่าไปใช้นั้นทำได้เช่นเดียวกับ CGROM คือ เขียนข้อมูลลงใน DDRAM แล้วตัวควบคุมจะมาอ่านค่าจาก CGROM เอง

แฟลกบิซซี(Busy Flag) เป็นส่วนที่ทำหน้าที่แจ้งสถานะการทำงานของตัวควบคุมให้อุปกรณ์ภายนอกทราบว่าตัวควบคุมพร้อมที่จะรับข้อมูลหรือคำสั่งหรือไม่ ดังนั้นก่อนที่จะส่งข้อมูลหรือคำสั่งมายังโมดูล LCD ขนาด 20 ตัวอักษร 4 บรรทัด

สำหรับโมดูล LCD ที่ใช้ในโครงงาน เป็นขนาด 20 ตัวอักษร 4 บรรทัด เนื่องจากเป็นขนาดที่เหมาะสมที่สุดในการใช้การเป็นจอแสดงผลสำหรับการสั่งอาหารซึ่งพิจารณาจากประโยชน์ใช้สอยและความประหยัด ซึ่งประกอบด้วยขาทั้งหมด 14 ขา มีการจัดวางดามรูป

รูปที่ แสดงการจัดขาของโมดูลซีดีแบบอักขระ

Vss(ขา1) : ต่อกราวน์

Vdd(ขา2) : ต่อไฟเลี้ยง +5 v

Vo(ขา4) : เป็นอินพุตรับแรงดันเพื่อปรับความเข้มของการแสดงผล

RS(ขา4) : เป็นขาอินพุตใช้ในการแยกชนิดของข้อมูลที่ทำการประมวลผลในขณะนั้นว่าเป็นคำสั่งสำหรับรีจิสเตอร์ IR หรือเป็นข้อมูลสำหรับรีจิสเตอร์ DR โดยขานี้เป็น “0” ข้อมูลที่ส่งมาจะเป็นคำสั่งแต่ถ้าขานี้เป็น “1” ข้อมูลที่ส่งมาจะเป็นข้อมูลสำหรับการแสดงผล

R/W(ขา5) : เป็นขาที่ใช้เลือกการอ่านหรือเขียนข้อมูลจากโมดูล LCD ถ้าเป็น “0” เป็นการกำหนดให้เขียนข้อมูล แต่ถ้าเป็น “1” จะเป็นการอ่านข้อมูล

E(ขา6) : เป็นขาสำหรับรับสัญญาณพัลส์เอ็นเอเบิลโมดูล LCD ให้ทำงาน

D0-D7 : เป็นขาที่ใช้เป็นทางผ่านของข้อมูลระหว่าง LCD กับอุปกรณ์ภายนอกขนาด 8 บิต

(ขา 7-14) : อนึ่งขา RS,R/W และ E จะทำงานร่วมกัน โดยมีความสัมพันธ์แสดงดังตาราง

2.5.2 คำสั่งควบคุมโมดูล LCD

ในการเขียนคำสั่งลงในตัวควบคุม แน่นอนว่าต้องกำหนดให้ขา RS และ R/W เป็น “0” แล้วเขียนคำสั่งตามไป คำสั่งควบคุมโมดูล LCD ของชิปควบคุม HD44780 ที่สำคัญมี 10 คำสั่งดังนี้

1. คำสั่งเคลียร์ตัวแสดงผล(Clear display) มีข้อมูลคำสั่งเป็น 01H เป็นคำสั่งที่ใช้เขียนข้อมูลช่องว่าง หรือ space เข้าไปใน DDRAM ทั้งหมด เมื่อตัวควบคุมเอ็กซิคิวต์คำสั่งนี้ จะทำการกำหนดแอดเดรสของ DDRAM เป็น 0 เคอร์เซอร์จะกลับไปอยู่ที่ตำแหน่งซ้ายมือสุด ของจอแสดงผล แล้วเซ็ทบิต I/D ให้เป็น “1”
2. คำสั่ง(Return home) ต้องกำหนดให้บิต 1 ของข้อมูลเป็น “1” เป็นคำสั่งให้เคอร์เซอร์เคลื่อนที่กลับไปยังตำแหน่งซ้ายสุดของจอแสดงผล แต่ข้อมูลบนจอแสดงผลไม่เปลี่ยนแปลง นั่นคือข้อมูลคำสั่งของคำสั่งนี้จะเป็น 02H หรือ 03H ก็ได้
3. คำสั่งเลือกโหมดการป้อนข้อมูล (Entry mode set) มีรายละเอียดของรูปแบบข้อมูลคำสั่ง ดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	0	0	0	1	I/D	S

- บิต S เป็นบิตที่ใช้ในการกำหนดลักษณะของการแสดงผล เมื่อมีการป้อนข้อมูล ถ้าหากบิต S เป็น "1" เมื่อเกิดข้อมูลใหม่บนจอแสดงผล ตัวเคอร์เซอร์จะอยู่กับที่ แต่ตัวอักษรข้อมูลเดิมจะถูกดันไปทางซ้าย แต่ถ้าหากบิตนี้เป็น "0" เมื่อเกิดข้อมูลใหม่ตัวเคอร์เซอร์จะเลื่อนไปทางขวามือ
- บิต I/D เป็นบิตที่ใช้กำหนดว่า เมื่อเขียนหรืออ่านข้อมูลแล้ว แอดเดรสของ DDRAM เพิ่มขึ้น หรือลดลงหนึ่งแอดเดรส โดยถ้าบิตนี้เป็น "1" แอดเดรสของ DDRAM จะเพิ่มขึ้น แต่ถ้าเป็น "0" แอดเดรสจะลดลง ดังนั้นข้อมูลคำสั่งที่เกิดขึ้นสำหรับคำสั่งนี้ได้แก่ 04H-07H (4 ข้อมูลคำสั่งและที่ใช้บ่อยคือ 06H หมายถึง กำหนดให้เมื่อเกิดข้อมูลใหม่ เคอร์เซอร์จะเลื่อนไปทางขวามือ และแอดเดรสของ DDRAM เพิ่มขึ้น

2.5.3 คำสั่งควบคุมการแสดงผล

มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	0	0	0	D	C	B

- บิต D ใช้ควบคุมการเปิดปิดจอแสดงผล ถ้าบิตนี้เป็น "1" จะเป็นการเปิดจอแสดงผล ถ้าเป็น "0" จะเป็นการปิดจอแสดงผล
- บิต C ใชควบคุมการแสดงตัวเคอร์เซอร์บนจอแสดงผล ถ้าต้องการให้มีเคอร์เซอร์แสดงผลบนจอแสดงผล ต้องกำหนดให้บิตนี้เป็น "1" ถ้ากำหนดให้เป็น "0" จะเป็นการปิดเคอร์เซอร์ หรือไม่แสดงเคอร์เซอร์
- บิต B ใช้ควบคุมการกระพริบของเคอร์เซอร์ ถ้าบิตนี้เป็น "1" เคอร์เซอร์จะกระพริบดังนั้นจะมีข้อมูลคำสั่งได้ตั้งแต่ 08H-0FH ที่ใช้บ่อยคือ 0CH เป็นการสั่งให้เปิดจอแสดงผล แต่ไม่แสดงเคอร์เซอร์ และ 0FH เป็นการสั่งให้เปิดจอแสดงผล แสดงเคอร์เซอร์ และสั่งให้เคอร์เซอร์กระพริบ

2.5.4 คำสั่งควบคุมการเลื่อนเคอร์เซอร์และข้อมูลตัวอักษร

มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	0	1	S/C	R/L	*	*

การควบคุมการเลื่อนเคอร์เซอร์และตัวอักษรบนจอแสดงผล ขึ้นอยู่กับการกำหนดบิต S/C และ R/L ซึ่งสามารถสรุปได้ดังนี้

S/C	R/L	ลักษณะการเลื่อน	ข้อมูลคำสั่ง
0	0	เลื่อนเคอร์เซอร์ไปทางซ้าย	10H-13H
0	1	เลื่อนเคอร์เซอร์ไปทางขวา	14H-17H
1	0	เลื่อนตัวอักษรใหม่ไปทางซ้าย	18H-1BH
1	1	เลื่อนตัวอักษรใหม่ไปทางขวา	1CH-1FH

2.5.5 คำสั่งกำหนดฟังก์ชั่นการทำงาน

มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	1	DL	N	F	*	*

- บิต DL ใช้กำหนดจำนวนบิตที่ใช้ติดต่อส่งผ่านข้อมูล ถ้าบิตนี้เป็น "0" จะเป็นการติดต่อแบบ 4 บิต แต่ถ้าบิตนี้เป็น "1" จะเป็นแบบ 8 บิต
- บิต N ใช้กำหนดจำนวนบรรทัดของการแสดงผล ถ้าเป็น "0" จะแสดงผล 1 บรรทัด ถ้าเป็น "1" จะแสดงผล 2 บรรทัด ในกรณี ในกรณีที่จอแสดงผลสามารถแสดงได้มากกว่า 2 บรรทัด และต้องการให้แสดงผลมากกว่า 2 บรรทัด ก็กำหนดบิต N นี้ให้เป็น "1" จุดที่น่าสังเกตคือ โมดูล LCD แบบ 16 ตัวอักษร 1 บรรทัด แม้จะมีบรรทัดการแสดงผลเพียง 1 บรรทัด แต่จะต้องกำหนด N ให้เป็น "1" เนื่องจากแอดเดรสของ DDRAM แบ่งเป็น 2 ช่อง คือ 00H และ 40H
- บิต F ใช้เลือกความละเอียดของตัวอักษรในการแสดงผล ถ้าบิตนี้เป็น "0" จะเป็นการแสดงผลแบบ 5x7 และถ้าเป็น "1" จะแสดงเป็นแบบ 5x10 จุด ข้อมูลคำสั่งที่ใช้บ่อยคือ 38H เป็นการกำหนดให้โมดูล LCD ทำงานในแบบ 8 บิต แสดงผล 4 บรรทัด และเลือกความละเอียดเป็น 5x7 จุด

คำสั่งเลือกแอดเดรสของ CGRAM ต้องกำหนดให้บิต 7 เป็น "0" บิต 6 เป็น "1" ส่วนอีก 6 บิตที่เหลือจะแทนด้วยค่าแอดเดรสของ CGRAM จะต้องทากรกำหนดแอดเดรสด้วยคำสั่งนี้ ก่อนจะอ่านหรือเขียนข้อมูลให้ CGRAM โดยแอดเดรสของ CGRAM อยู่ระหว่าง 00H-3FH

2.5.6 คำสั่งเลือกแอดเดรสของ DDRAM

ใช้ในการเลือกแอดเดรสของ DDRAM ก่อนที่จะทำการอ่านหรือเขียนข้อมูล โดยบิต 7 ต้องเป็น "1" และข้อมูลอีก 7 บิตที่เหลือจะเป็นคู่แอดเดรสของ DDRAM ซึ่งแอดเดรสของ DDRAM จะอยู่ระหว่าง

8CH-0FFH ทั้งนี้จำนวนแอดเดรสยังขึ้นกับการกำหนดสถานะบิตที่ N ด้วย เป็น "0" แอดเดรสของ DDRAM จะอยู่ระหว่าง 80H-0CFH และถ้าบิต N เป็น "1" แอดเดรสของ DDRAM จะมีสองช่วงคือ 8CH-87H และ 0C0H-0C7H

2.5.7 คำสั่งอ่านบิตซีแฟลกและแอดเดรส

มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
BF	A	A	A	A	A	A	A

เป็นคำสั่งที่ใช้อ่านบิซซีแฟลก(BF) โดยแฟลกนี้จะเป็นตัวบอกสถานะของตัวควบคุม LCD ว่าพร้อมจะรับข้อมูลอยู่หรือไม่ ถ้าหากบิต BF เป็น "0" แสดงว่าพร้อมรับข้อมูลหรือคำสั่ง แต่ถ้าเป็น "1" แสดงว่าขณะนี้ตัวควบคุม LCD ยังอยู่ในกระบวนการทำงานภายในหรือกำลังประมวลผลข้อมูลอยู่ ยังไม่พร้อมรับข้อมูลหรือคำสั่ง เมื่อต้องการอ่านค่าแฟลกต้องกำหนดให้ขา R/W เป็น "1" ด้วย แต่สัญญาณที่ RS ยังต้องเป็น "0" อยู่เพราะข้อมูลนี้เป็นข้อมูลคำสั่ง นอกจากนี้ ยังใช้เป็นคำสั่งอ่านข้อมูล แอดเดรสของ CGRAM และ DDRAM ด้วย

2.6 Delphi 7

Delphi 7 คือ ซอฟต์แวร์ที่นำมาใช้ในการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชัน หรือซอฟต์แวร์อีกที โดยมันจะประกอบไปด้วยเครื่องมือชนิดต่างๆที่ใช้ให้การเขียนโปรแกรมทำได้อย่างสะดวก

Delphi 7 จัดเป็นเครื่องมือเขียนโปรแกรมชนิด Visual Programming เช่นเดียวกับ Visual Basic หรือ Visual C++ โดยมีข้อดีคือ สามารถเขียนโปรแกรมได้ง่าย และให้ผลงานออกมาอย่างรวดเร็ว ซึ่งจะแตกต่างจากเครื่องมือเขียนโปรแกรมรุ่นเดิมๆ เช่น Turbo Pascal หรือ Borland C ที่มีความยุ่งยากในการใช้งานและการเรียนรู้ในการเขียนโปรแกรม ดังนั้นจึงจัดให้ Delphi 7 เป็นซอฟต์แวร์ประเภท RAD หรือ Rapid Application Development ซึ่งแปลว่าสามารถสร้างแอพพลิเคชันได้อย่างรวดเร็ว

Delphi 7 นั้นผ่านการพัฒนามาเกือบ 10 ปี ตั้งแต่เวอร์ชัน 1.0 ที่ทำงานบน Windows 3.1X โดยมีจุดเด่นมากๆ ตั้งแต่สมัยนั้นคือ โปรแกรมที่ได้จากการเขียนโปรแกรมมีขนาดเล็ก ทำงานได้รวดเร็ว ซึ่งมักจะถูกนำไปเปรียบเทียบกับ Visual Basic 3.0 ในสมัยนั้น อีกประการหนึ่ง Delphi ใช้ภาษาออบเจ็กต์ปาสคาล จึงเคยถูกเปรียบว่าเป็น Visual Pascal มาแล้ว

เวอร์ชันปัจจุบันของ Delphi นั้นได้รับการพัฒนาให้สามารถสร้างแอพพลิเคชันที่ทำงานบน Windows ได้ดีเหมือนเดิม โดยมีการปรับปรุงให้สามารถพัฒนาแอพพลิเคชันตามแนวความคิดของ .NET ซึ่งจะช่วยให้สามารถเขียนโปรแกรมครั้งเดียว แล้วนำไปใช้งานบนอุปกรณ์ต่างๆไม่ว่าจะเป็น PDA , โทรศัพท์มือถือ และ บนเว็บได้

2.6.1 ความสามารถของ Delphi 7

Delphi 7 นั้นมีความสามารถมากมาย ได้รับการต้อนรับเป็นอย่างดีจากนักพัฒนาแอพพลิเคชันทั่วโลก รวมทั้งเมืองไทยด้วย ซึ่งจะเห็นได้จากการนำ Delphi ไปประกอบการเรียนการสอน การฝึกอบรม ตลอดจนการนำไปสร้างเป็นซอฟต์แวร์เชิงพาณิชย์จำนวนมาก

2.6.1.1 สร้างแอพพลิเคชันสำหรับ Windows

Delphi ได้ชื่อว่าเป็นเครื่องมือสำหรับพัฒนาแอพพลิเคชันที่ทำงานบน Windows ที่ใช้งานกันแพร่หลายมาก สามารถสร้างงานได้อย่างรวดเร็ว นิยมนำไปสร้างแอพพลิเคชันทั้งเพื่อการศึกษา และแอพพลิเคชันที่ใช้ในระบบงานจริงๆ ในโลกธุรกิจ

Delphi สามารถสร้างแอพพลิเคชันบน Windows ได้หลายเวอร์ชัน ซึ่งแอพพลิเคชันที่สร้างจาก Delphi ได้รับการยกย่องเป็นอย่างมากในด้านขนาดโปรแกรมที่เล็กกะทัดรัด และทำงานได้อย่างรวดเร็ว เมื่อเทียบกับแอพพลิเคชันที่สร้างจากเครื่องมือตัวอื่นๆ ทำให้ Delphi แต่ละเวอร์ชันได้รับรางวัลเกียรติยศจากสถาบันต่างๆทั่วโลกมาโดยตลอด

2.6.1.2 สร้างระบบงานด้านฐานข้อมูล

ถ้าจะกล่าวว่า Delphi ถูกนำไปสร้างผลงานอะไรมากที่สุดสำหรับเมืองไทยแล้ว หนีไม่พ้นเรื่องของการพัฒนาระบบงานด้านฐานข้อมูล ซึ่งถูกนำไปใช้งานอย่างแพร่หลายตั้งแต่องค์กรขนาดเล็กไปจนถึงองค์กรระดับประเทศ โดย Delphi มีจุดเด่นในการสร้างแอพพลิเคชันที่ติดต่อกับฐานข้อมูลได้หลายรูปแบบ มีวิธีการและเครื่องไม้เครื่องมือในการสร้างระบบงาน ระบบบริหารงานบุคคล ระบบคลังสินค้า หรือแม้แต่ระบบจองตั๋วในโรงภาพยนตร์ชั้นนำ

แอพพลิเคชันที่เกี่ยวกับฐานข้อมูลที่สร้างจาก Delphi สามารถนำไปใช้งานกับระบบฐานข้อมูลชั้นนำแทบทุกชนิดทั่วโลก นับตั้งแต่ระบบฐานข้อมูลส่วนบุคคลทั้ง Access , Paradox , Foxpro ไปจนถึงระบบฐานข้อมูลขนาดใหญ่ทั้ง Oracle , Sybase , SQL Server รวมถึงระบบไฟล์ชนิดต่างๆได้ด้วย

2.6.1.3 สร้างแอพพลิเคชันรองรับ .NET Web Service

แนวโน้มของการพัฒนาแอพพลิเคชันในยุคหน้าที่นี้ต้องเตรียมตัวให้พร้อมก็คือ การพัฒนาแอพพลิเคชันด้วยเทคโนโลยี .NET ซึ่งจะช่วยให้แอพพลิเคชันชนิดต่างๆไม่ว่าจะทำงานบนพีซี โน๊ตบุ๊ค PDA หรือแม้แต่โทรศัพท์มือถือ สามารถเชื่อมโยงข้อมูล และคุยกันได้

ใน Delphi 7 นั้นรองรับการพัฒนาแอพพลิเคชันที่เรียกว่า .NET Web Service แล้ว โดยสามารถใช้เครื่องมือชนิดต่างๆพัฒนาแอพพลิเคชันตามแนวความคิดของ .NET เป็นอย่างดี

2.6.1.4 ขั้นตอนการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชันกับ Delphi7

ขั้นตอนในการสร้างแอพพลิเคชันด้วย Delphi 7 มีขั้นตอนดังต่อไปนี้

ขั้นที่ 1 : ออกแบบโปรแกรม

นับเป็นขั้นตอนแรกของการเขียนโปรแกรมที่ควรต้องทำ โดยผู้เขียนโปรแกรมจะต้องออกแบบเสียก่อนว่าจะให้โปรแกรมมีหน้าตาอย่างไร มีขั้นตอนการทำงานเป็นอย่างไร ซึ่งบ่อยครั้งที่ผู้เริ่มต้นเขียนโปรแกรมมือใหม่หลายรายละเลย ซึ่งจะส่งผลเสียในอนาคตคือ ถ้าออกแบบโปรแกรมไม่ดี (ขาดการทำงานที่ดี , ขาดความยืดหยุ่นในการปรับแต่ง , ขาดความสวยงามน่าใช้งาน) ก็จะทำให้เกิดความยุ่งยากในการปรับปรุงโปรแกรมนี้ในอนาคต

ในการพัฒนาระบบงานขนาดใหญ่นั้น งานในการออกแบบโปรแกรมกับการเขียนโปรแกรมจะแยกจากกัน อาจจะใช้ทีมงานคนละทีมงานกันทำให้ผู้ออกแบบโปรแกรมต้องมีเครื่องมือที่จะใช้สื่อสารและบอกรายละเอียดของการออกแบบให้กับผู้เขียนโปรแกรมได้ทราบ ซึ่งจะใช้สิ่งที่เรียกว่า โฟลว์ชาร์ต (Flow Chart) เป็นเครื่องมือในการสื่อสารนั้น

โฟลว์ชาร์ตนั้นจะบอกถึงขั้นตอนการทำงาน การตัดสินใจ และข้อมูลที่จะใช้ให้ผู้เขียนโปรแกรมทราบอย่างครบถ้วน เพราะฉะนั้นนอกจากจะใช้สื่อสารระหว่างกัน ยังใช้เป็นเครื่องมือสำคัญในการควบคุมการทำงานของผู้เขียนโปรแกรมได้ด้วย

ขั้นที่ 2 : วางคอมโพเนนต์ต่างๆลงใน Form Designer

ผู้เขียนโปรแกรมจะนำแบบที่คิดไว้จากขั้นที่ 1 มาสร้างเป็นแอพพลิเคชันไว้ใช้งาน

ขั้นที่ 3 : กำหนดค่าพร็อพเพอร์ตี้ให้กับคอมโพเนนต์ต่างๆ

ในขั้นตอนนี้ผู้เขียนโปรแกรมจะกำหนดค่าพร็อพเพอร์ตี้ หรือคุณสมบัติให้กับคอมโพเนนต์ต่างๆที่วางอยู่ใน Form Designer

ขั้นที่ 4 : เขียนคำสั่งกำกับการทำงานให้คอมโพเนนต์

หลังจากกำหนดค่าพร็อพเพอร์ตี้ ซึ่งก็เป็นการกำหนดหน้าตา ลักษณะการทำงานต่างๆแล้ว ต่อไปจะต้องเขียนคำสั่งเพื่อกำกับการทำงานของคอมโพเนนต์โดยเขียนคำสั่งผ่าน Code Editor

ขั้นที่ 5 : คอมไพล์โปรแกรมที่เขียนขึ้น

เมื่อเขียนคำสั่งเพื่อดักการทำงานในรูปแบบต่างๆจนครบแล้ว ต่อไปผู้เขียนโปรแกรมก็จะคอมไพล์โปรแกรมที่ได้เขียนขึ้น ซึ่งการคอมไพล์ (Compile) ก็คือ การแปรจากคำสั่งที่ได้เขียนทั้งภาษาไทย ภาษาอังกฤษที่ผู้เขียนโปรแกรมอ่านรู้เรื่องนำมาแปลเป็นภาษาที่คอมพิวเตอร์เข้าใจ และทำงานตามที่ผู้เขียนโปรแกรมตั้งใจไว้ได้

ในการคอมไพล์โปรแกรมที่เขียนขึ้นโดย Delphi 7 นั้นให้คลิกเมนู Run > Run หรือง่ายกว่านั้นให้กดปุ่ม<F9> ทันที

Delphi 7จะทำการคอมไพล์คำสั่งที่ได้เขียนขึ้นทั้งหมด แล้วเปลี่ยนสภาวะ (Mode) การทำงานจากสภาวะการออกแบบเขียนโปรแกรม (Design Mode) ไปสู่สภาวะการรันโปรแกรม (Run Mode)

ขั้นที่ 6 : ทดสอบการทำงานของแอพพลิเคชัน

เมื่อคอมไพล์ผ่านแล้ว ผู้เขียนโปรแกรมจะทดสอบการทำงานว่าสิ่งที่สร้างนั้นทำงานได้ถูกต้องหรือไม่ เพราะบางครั้งอาจจะเขียนคำสั่งได้ถูกต้องตามหลักภาษาโปรแกรมทุกอย่าง แต่อาจจะทำงานผิดพลาดก็ได้ จึงต้องตรวจสอบการทำงานให้แน่ใจเสียก่อน

ขั้นที่ 7 : บันทึกผลการทำงาน

เมื่อทดสอบจนเป็นที่น่าพอใจแล้ว ผู้เขียนโปรแกรมก็สามารถบันทึกสิ่งที่ได้สร้างขึ้นทั้งหมดเอาไว้ ซึ่งจะสามารถเรียกขึ้นมาแก้ไขปรับปรุงภายหลังได้ โดยจะบันทึกไว้ 2 ส่วนคือ ไฟล์ยูนิต (.pas) และโปรเจ็กต์ (.dpr)

2.6.1.5 ภาษาออบเจ็กต์ปาสคาล

ภาษาออบเจ็กต์ปาสคาลในการเขียนโปรแกรมกับ Delphi 7 นั้นจะเก็บซอร์สโค้ดที่ได้เขียนขึ้นไว้ในไฟล์ที่เรียกว่า ยูนิต (Unit) ซึ่งจะถูกบันทึกไว้ในไฟล์ .pas และเมื่อไฟล์ยูนิตถูกคอมไพล์ก็จะสร้างไฟล์ที่เป็นผลจากการคอมไพล์มีนามสกุลเป็น .dcu

ปกติเมื่อเริ่มสร้างแอพพลิเคชันใน Delphi 7 (ก็คือการสร้างโปรเจ็กต์ใหม่ขึ้นมา) จะมีการสร้างไฟล์ชื่อ Unit1.pas ขึ้นมาให้เขียนโค้ดเข้าไป เมื่อตรวจสอบจาก Code Editor จะพบว่า Delphi ได้เขียนโค้ดบางส่วนไว้ให้ผู้เขียนโปรแกรมอยู่แล้ว

ไฟล์ยูนิตหนึ่งๆ จะใช้กับการเขียนโค้ดสำหรับฟอร์ม 1 ฟอร์ม ดังนั้นถ้าโปรเจ็กต์ที่ประกอบด้วย ฟอร์มมากกว่าหนึ่งฟอร์มสามารถจะมีไฟล์ยูนิตที่มีนามสกุล .pas ได้มากกว่า 1 ไฟล์ได้

เมื่อผู้เขียนโปรแกรมมีการนำคอมโพเนนต์เข้ายังฟอร์มก็จะพบว่า Delphi ได้เขียนโค้ดให้แล้วในตอนต้นของไฟล์ยูนิต และเมื่อเขียนโปรแกรมเพื่อเลือกจัดการกับอีเวนต์ที่สนใจ Delphi ก็จะเตรียมโค้ดที่เป็นโครงเอาไว้ให้ผู้เขียนโปรแกรมเขียนเพิ่มเติม

2.6.1.6 โครงสร้างของยูนิต

เมื่อกลับมามองไฟล์ Unit1.pas ที่ Delphi สร้างไว้ให้นั้น สามารถแบ่งโค้ดออกเป็นส่วนต่างๆ ดังนี้

- ส่วน interface ส่วนนี้เป็นส่วนที่ใช้ประกาศชนิดข้อมูล , ตัวแปร , ค่าคงที่ , ออบเจ็กต์ , โพรซีเจอร์ และฟังก์ชัน ซึ่งทุกสิ่งที่บรรจุไว้ในส่วนนี้สามารถเข้าถึง และใช้งานได้จากยูนิตอ่นๆ ต้องไม่ลืมว่าโปรเจ็กต์หนึ่งๆสามารถมีไฟล์ยูนิตได้หลายตัว ซึ่งพื้นที่ส่วน interfage นี้จะเริ่มจากคำว่า interface ไปจนถึง implementation
- ส่วน implementation ส่วนนี้จะทำหน้าที่เหมือนกับส่วน Interface ต่างกันตรงที่ขอบเขตการเข้าถึงข้อมูลคือ จะเข้าถึงข้อมูล , ตัวแปร , ค่าคงที่ , ออบเจ็กต์ , โพรซีเจอร์ และฟังก์ชัน ได้จากสิ่งที่อยู่ขอบเขตเฉพาะภายในยูนิตนี้เท่านั้น ยูนิตอื่นๆ ไม่มีสิทธิเข้าถึง ซึ่งพื้นที่ส่วน implementation ไปจนถึง initialization
- ส่วน initialization ส่วนนี้จะใช้เก็บคำสั่งที่ถูกเรียกใช้งานก่อนการทำงานของแอพพลิเคชันโดยปกติจะทำงานก่อนที่จะมีการสร้างออบเจ็กต์ หรือฟอร์มขึ้นมา ดังนั้นผู้เขียนโปรแกรมจึงมักใช้พื้นที่ส่วนนี้กำหนดค่าให้กับตัวแปรบางตัว ซึ่งพื้นที่ของส่วย initialization นี้จะเริ่มต้นจากคำว่า initialization ไปจนถึง finalization (ถ้าไม่มีคำว่า finalization ก็จะถึงคำว่า end.) สำหรับส่วนนี้จะมีก็ได้ ไม่มีก็ได้แล้วแต่โปรแกรม

- ส่วน finalization ส่วนนี้จะทำหน้าที่ตรงกันข้ามกับ initialization ซึ่งพื้นที่ส่วน finalization นี้จะเริ่มตั้งแต่คำว่า finalization ไปจนถึง end.

2.6.1.7 ไฟล์โปรเจ็กต์

ปกติการเขียนโปรแกรมจะเขียนที่ไฟล์ยูนิตเท่านั้น แต่ในการทำงานจริงๆของแอพพลิเคชันนั้นจะต้องเริ่มจากการทำงานของไฟล์โปรเจ็กต์ก่อน ซึ่งรายละเอียดการทำงานของไฟล์โปรเจ็กต์นี้ผู้เขียนโปรแกรมไม่ต้องจัดการเอง แต่สามารถดูการทำงานภายในได้โดยเลือกเมนู Project > View Source ซึ่งใน Code Editor จะเพิ่มแท็บที่เป็นรายละเอียดของไฟล์โปรเจ็กต์ดังนี้

สำหรับโค้ดของไฟล์โปรเจ็กต์นั้นประกอบไปด้วยส่วนสำคัญ 3 ส่วนคือ

- Program heading
- Uses clause
- การทำงานของโปรแกรม

สำหรับส่วนที่เป็น program heading นั้นจะบอกถึงชื่อโปรแกรม ในที่นี้ก็คือชื่อที่ผู้เขียนโปรแกรมตั้งให้ในตอนบันทึกชื่อไฟล์โปรเจ็กต์นั่นเอง และชื่อนี้จะถูกสร้างเป็นไฟล์ .EXE ด้วย

ส่วน uses clause จะเป็นส่วนที่ใช้ประกาศยูนิตที่ใช้งาน ถ้ามีมากกว่าหนึ่งยูนิตในนี้ก็จะเก็บรายการของยูนิตทั้งหมดที่ใช้เอาไว้

ส่วนสุดท้ายจะเก็บรายละเอียดของคำสั่งในการทำงานของแอพพลิเคชัน ซึ่งทุกๆแอพพลิเคชันที่เป็นชนิดเดียวกันจะมีส่วนนี้เหมือนกันหมดคือ มีส่วนที่กำหนดค่าเริ่มต้น สร้างฟอร์มขึ้นมา แล้วเริ่มการทำงานให้กับแอพพลิเคชัน (ซึ่งถ้าเป็นแอพพลิเคชันทั่วไป ฟอร์มที่ถูกสร้างขึ้นมาก็จะรอการโต้ตอบจากผู้ใช้โปรแกรมงานนั่นเอง)

2.6.1.8 การเขียนโปรแกรมแบบ OOP

การเขียนโปรแกรม OOP เป็นการเลียนแบบแนวคิดของสิ่งที่ผู้เขียนโปรแกรมพบเห็นในสิ่งต่างๆที่อยู่รอบตัวทั้งนั้น

- คลาสกับออบเจ็กต์

แนวคิดของ OOP นั้นจะกำหนดให้สิ่งต่างๆเป็นวัตถุ หรือออบเจ็กต์ (Object) ซึ่งออบเจ็กต์แต่ละตัว จะถูกสร้างขึ้นมาจากแม่พิมพ์ที่เรียกว่า คลาส (Class) ดังนั้นออบเจ็กต์ที่สร้างจากคลาสจะถือว่าเป็นคนละตัวกัน แต่มีชนิดเดียวกัน เพราะสร้างจากคลาสเดียวกัน

เมื่อมามองถึงออบเจ็กต์ตัวหนึ่งๆ จะเห็นว่ามันจะต้องมีข้อมูลที่ให้กำหนดลักษณะเฉพาะของออบเจ็กต์ ซึ่งจะถูกเรียกว่า พร็อพเพอร์ตี้ (Property)

เมื่อมีพร็อพเพอร์ตี้แล้วสิ่งหนึ่งที่มักจะขาดไม่ได้ในออบเจ็กต์ก็คือ ความสามารถในการทำงานของออบเจ็กต์ ซึ่งจะถูกเรียกว่า เมธอด (Method)

เมื่อผู้เขียนโปรแกรมสร้างออบเจ็กต์หนึ่งๆ ขึ้นมาจากคลาส ในภาษาของการเขียนโปรแกรมจะกล่าวได้ว่าเป็น "ออบเจ็กต์ ก็คือ อินสแตนซ์ (Instance) ของคลาส"

เมื่อมีพร็อพเพอร์ตี้แล้วสิ่งหนึ่งที่มักจะขาดไม่ได้ออบเจ็กต์ก็คือ ความสามารถในการทำงานของออบเจ็กต์ ซึ่งจะถูกเรียกว่า เมธอด (Method)

เมื่อผู้เขียนโปรแกรมสร้างออบเจ็กต์หนึ่งๆ ขึ้นมาจากคลาส ในภาษาของการเขียนโปรแกรมจะกล่าวได้ว่าเป็น "ออบเจ็กต์ ก็คือ อินสแตนซ์ (Instance) ของคลาส"

2.6.1.9 การเข้าถึงข้อมูลที่เก็บภายในคลาส

ในคลาสจะประกอบด้วยข้อมูลต่างๆนั่นคือ พร็อพเพอร์ตี้ และเมธอด ซึ่งศัพท์ของนักเขียนโปรแกรม จะเรียกทั้งสองอย่างรวมๆว่า เมมเบอร์ (Member) ของคลาส

จุดดีอย่างหนึ่งของคลาสคือ การที่ไม่อนุญาตให้เข้าถึงข้อมูลภายในของคลาสได้โดยตรง จะต้องกระทำผ่านเมมเบอร์ของคลาสซึ่งก็สอดคล้องกับชีวิต ที่ผู้เขียนโปรแกรมใช้งานรถยนต์ได้จากสิ่งที่เตรียมไว้ให้ สำหรับการเข้าถึงข้อมูลภายในคลาส Delphi นั้นแบ่งได้เป็น 3 ระดับ ได้แก่

- Private เป็นระดับการเข้าถึงข้อมูลที่เข้าถึงได้เฉพาะข้อมูลที่อยู่ภายใจตัวคลาส หรือ อินสแตนซ์ของคลาสเท่านั้น ไม่อนุญาตให้ผู้ใช้โปรแกรม หรือคลาสอื่นใดเข้าถึงได้
- Pubilc เป็นระดับการเข้าถึงข้อมูลที่เข้าถึงได้จากผู้ใช้โปรแกรมคลาสทุกคน หรือคลาสอื่นๆ โดยไม่จำกัด
- Protected เป็นระดับการเข้าถึงข้อมูลเฉพาะคลาสที่ถูกสืบทอดมาจากคลาสนี้เท่านั้น

2.6.1.10 Constructor และ Destructor

นอกเหนือจากเมธอดที่ได้กำหนดให้คลาสแล้ว ในการเขียนโปรแกรมแบบ OOP นั้นยังต้องมีเมธอดที่จำเป็นอยู่อีก 2 ตัว ซึ่งจะต้องทำหน้าที่ทุกครั้งเมื่อออบเจ็กต์นั้นเริ่มสร้างขึ้นมา หรือ กำลังจะเลิกใช้งานไป

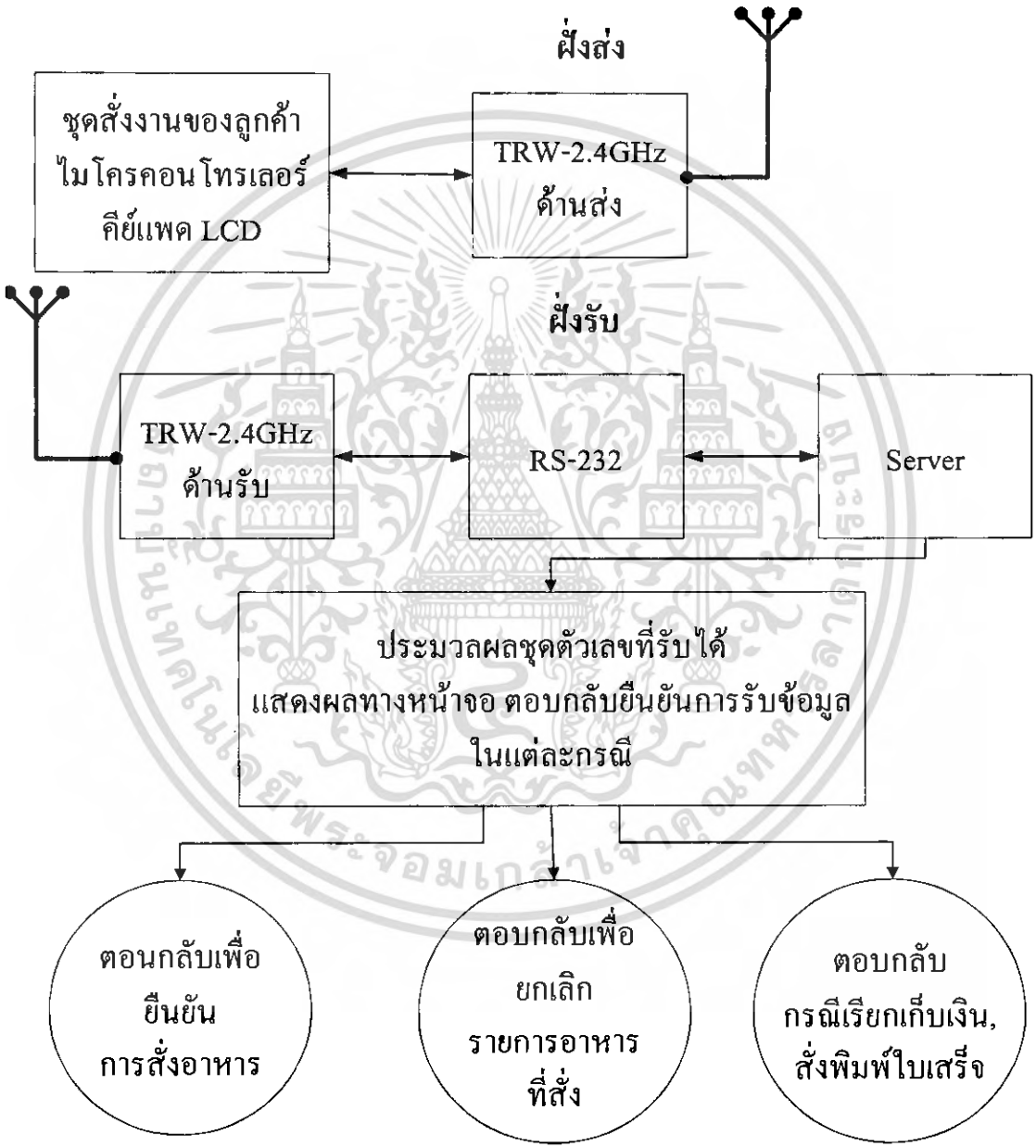
Constructor เป็นเมธอดที่ทำงานตอนที่ผู้เขียนโปรแกรมเริ่มสร้างออบเจ็กต์ขึ้นมาจากคลาส โดยจะทำหน้าที่กำหนดค่าเริ่มต้นที่จำเป็นให้กับพร็อพเพอร์ตี้ต่างๆ จับจองทรัพยากรของระบบที่จำเป็นต่อการทำงานของออบเจ็กต์

ส่วน Destructor เป็นเมธอดที่ทำงานในลักษณะตรงกันข้ามคือ ทำหน้าที่ ตอนที่ออบเจ็กต์นั้นเลิกใช้งาน โดยจะคืนทรัพยากรให้แก่ระบบปฏิบัติการ

บทที่3

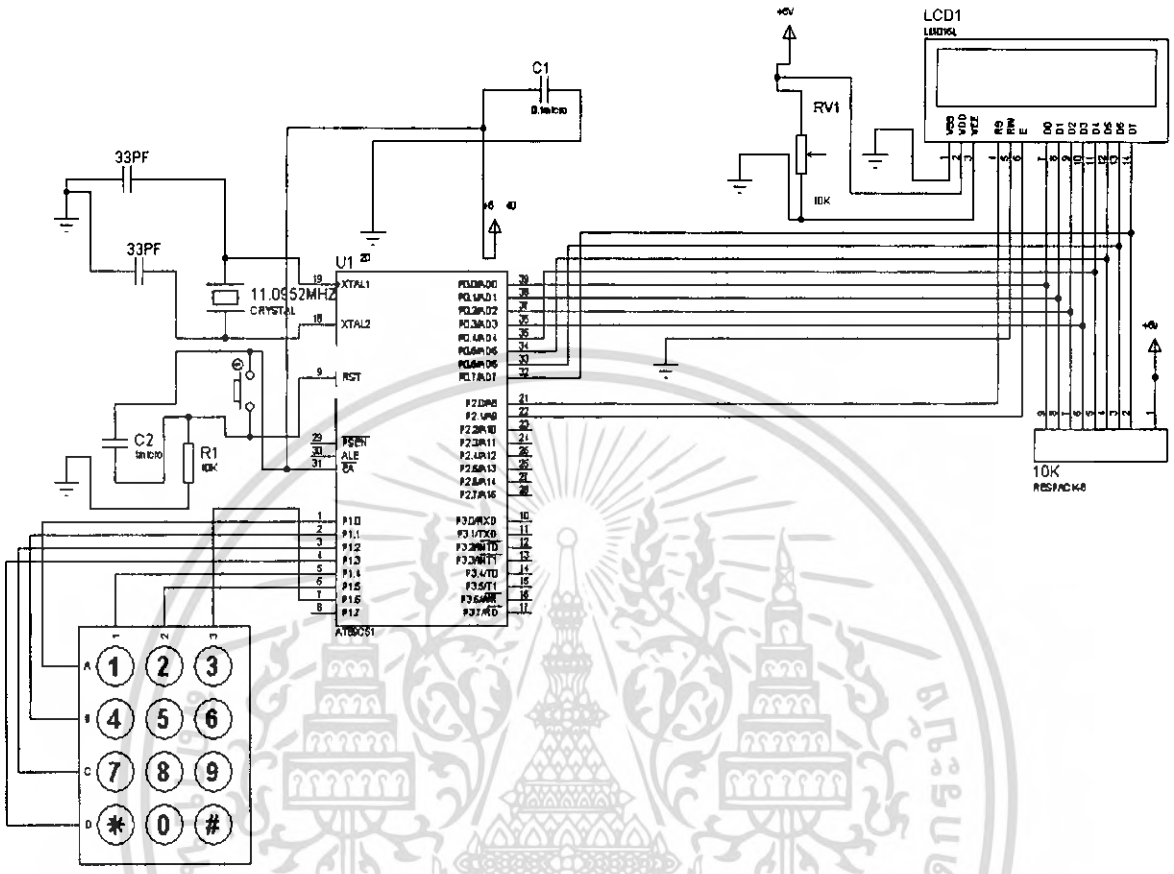
การคำนวณและการสร้าง

องค์ประกอบของเครื่องสั่งอาหารไร้สายประกอบด้วยส่วนต่างๆดังนี้



รูปที่ 3.1 แสดงบล๊อกไดอะแกรมของเครื่องสั่งอาหารแบบไร้สาย

3.1 ชุดสั่งงานของลูกค้า

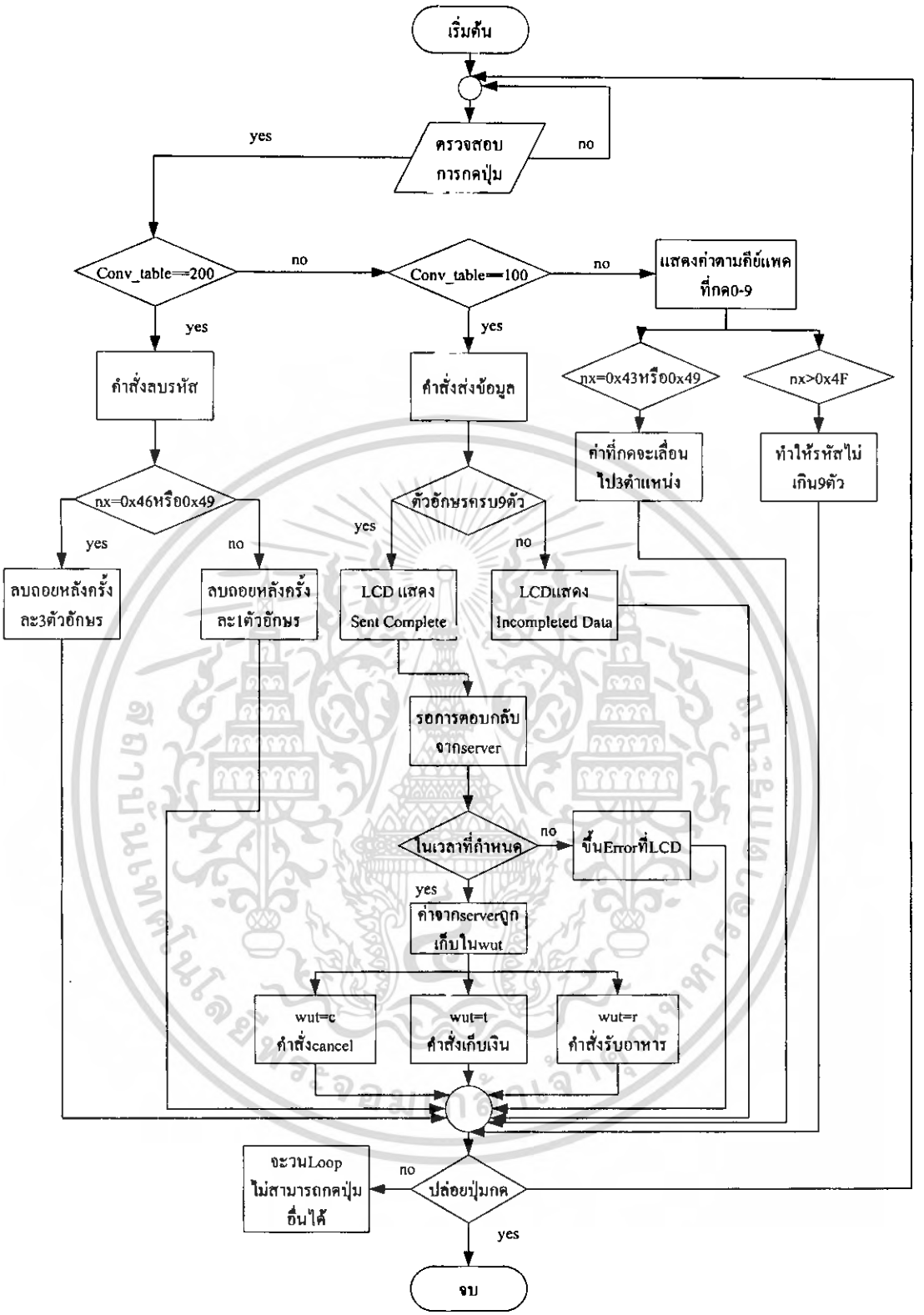


รูปที่ 3.2 แสดงวงจรชุดสั่งงานของลูกค้า

ประกอบไปด้วยจอ LCD คีย์แพดและ ไมโครคอนโทรเลอร์ เมื่อลูกค้ามาใช้บริการก็จะทำการกดรหัสโดยดูจากเมนูอาหาร แล้วทำการกดรหัสผ่านคีย์แพด ซึ่งตัวคีย์แพดจะทำการตรวจสอบที่แถวและหลักของคีย์แพดที่เชื่อมกับไมโครคอนโทรเลอร์ เพื่อนำค่าหมายเลขรหัสที่กดนั้นไปแสดงผลที่หน้าจอ LCD และเก็บค่าไว้ที่หน่วยความจำ เพื่อทำการส่งต่อไปยัง วงจรTRW-2.4GHz

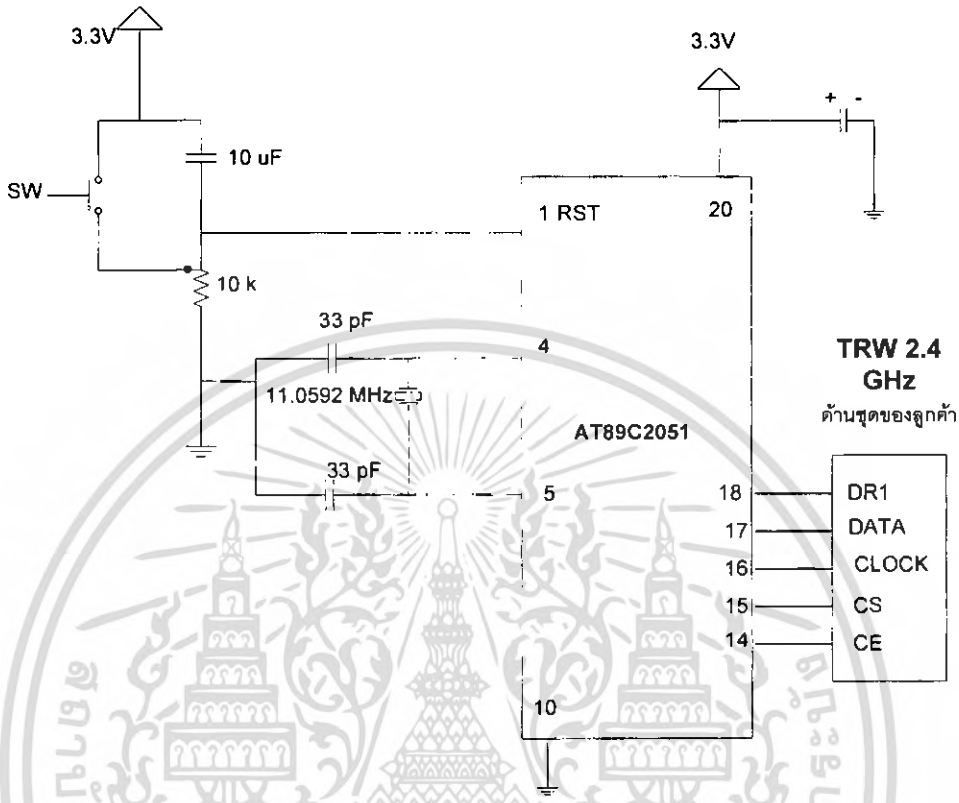
ในหน้าปัดของคีย์แพดนั้นนอกจากจะมีปุ่มกดหมายเลข 0-9 แล้วยังมีปุ่ม * ที่ทำหน้าที่ในการลบทีละตัวอักษร และปุ่ม # สำหรับใช้ส่งข้อมูล ส่วนในหน้าจอ LCD การแสดงผลมีรหัส 9 ตัว ซึ่งมี3ช่วงๆละ3ตัว โดยที่ รหัส 3 ตัวแรกจะเป็นรหัสของอาหาร อีก 3 ตัวต่อมาจะเป็นรหัสของจำนวนอาหารที่ลูกค้าสั่ง ส่วนรหัส 3 ตัวสุดท้ายจะเป็นหมายเลขโต๊ะ ซึ่งการทำงานทั้งหมดนั้น จะถูกควบคุมโดยไมโครคอนโทรเลอร์

ในการที่จะส่งข้อมูลรหัสอาหารออกไปนั้นจะต้องทำการกดปุ่ม# แล้วข้อมูลจะถูกส่งออกจากขา Tx ไปยังขาRx ของวงTRW-2.4GHz นอกจากนี้วงจรชุดสั่งงานของลูกค้ายังสามารถแสดงผลการตอบกลับที่มาจาก Server แล้วแสดงผลที่จอ LCD ได้ด้วยโดยข้อมูลจะเข้ามาทางขา Rx ของไมโครคอนโทรลเลอร์ในชุดสั่งงานของลูกค้า



รูปที่ 3.3 แสดงแผนผังงาน(Flow Chart) การทำงานในไมโครคอนโทรเลอร์ของชุดสั่งงานของลูกค้า

3.2 วงจร TRW-2.4GHz ฝั่งลูกค้า

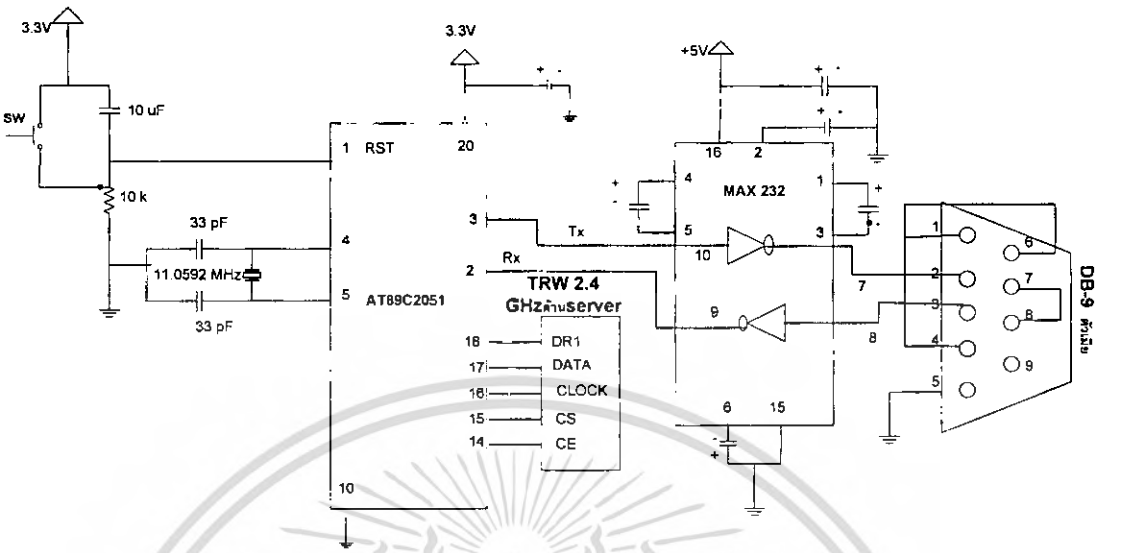


รูปที่ 3.4 แสดงวงจร TRW-2.4GHz ด้านส่ง

ในวงจรนี้จะใช้ไมโครคอนโทรเลอร์ชนิด 20 ขา และ TRW-2.4GHz เนื่องจาก TRW-2.4GHz จะใช้ไฟได้ไม่เกิน 3.3 โวลท์ ดังนั้นจึงต้องมี regulator แปลงไฟให้ไม่เกิน 3.3 โวลท์ ถ้าไม่เป็นเช่นนั้น TRW-2.4GHz อาจเสียหายได้

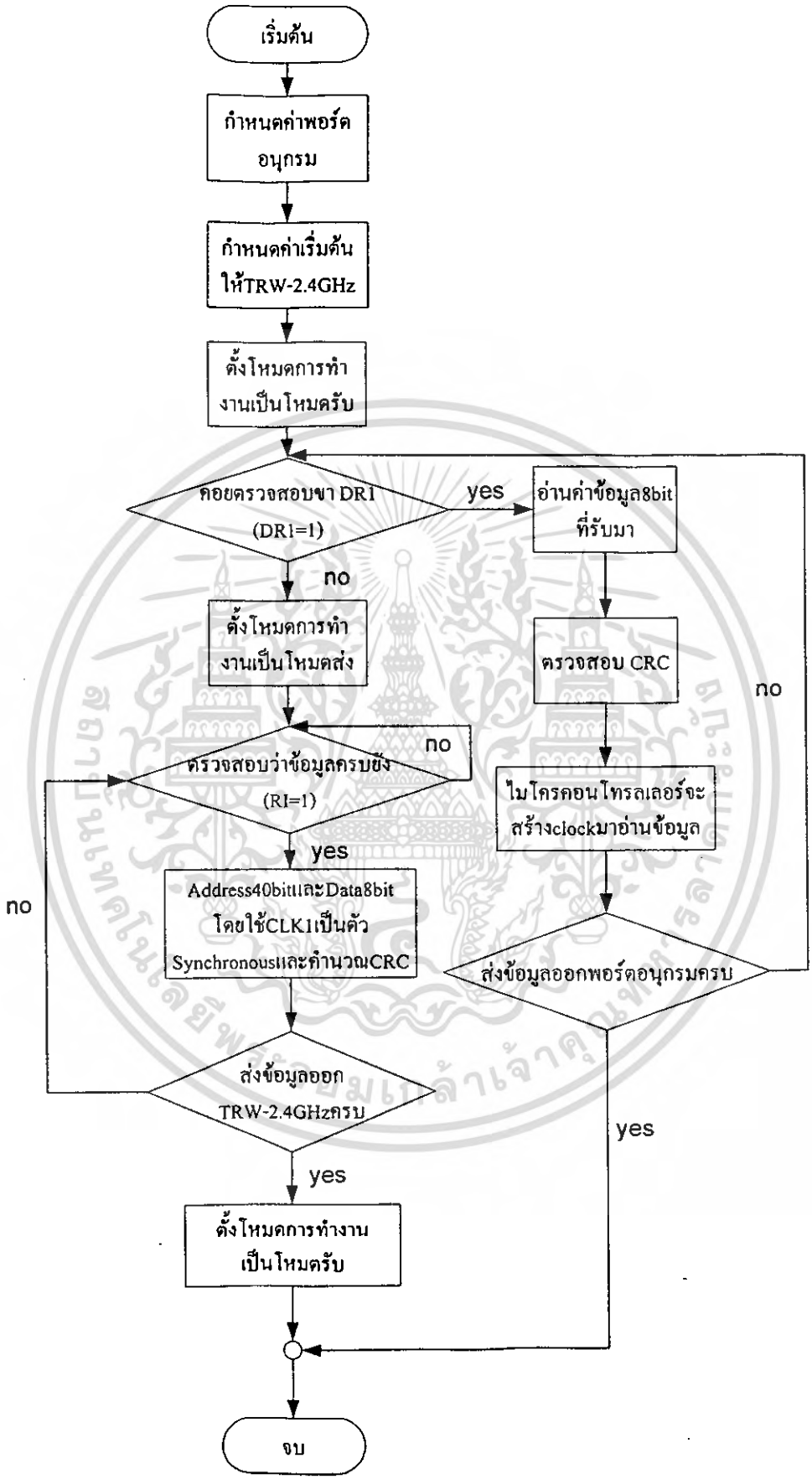
ในการทำงานนั้นไมโครคอนโทรเลอร์จะรับข้อมูลที่ส่งจากชุดสั่งงานของลูกค้า เข้าทางขา Rx แล้วทำการส่งต่อไปแบบไร้สาย ไปยัง TRW-2.4GHz ด้านรับต่อไป จากนั้นจะคอยการตอบกลับจากฝั่ง Server แล้วส่งไปยังวงจรชุดสั่งงานของลูกค้าเพื่อแสดงผล

3.3 วงจร TRW-2.4GHz ฝั่งเครื่องServer



รูปที่ 3.5 แสดงวงจร TRW-2.4GHz ด้านรับ

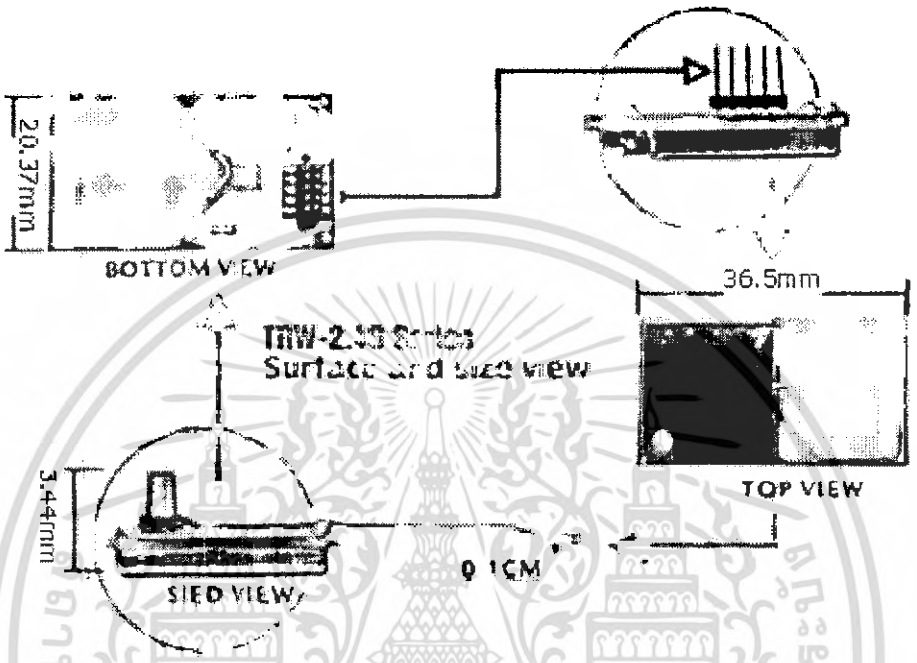
ในส่วนวงจรนี้จะทำการรับข้อมูลที่ส่งมาจาก TRW-2.4GHz ด้านส่ง ซึ่งจะทำการส่งออกด้วยขา Tx ของไมโครคอนโทรเลอร์ไปยังวงจรส่งผ่านพอร์ตอนุกรม(Max-232) ซึ่งไอซีMax-232 ทำหน้าที่แปลงระดับสัญญาณไฟฟ้า แล้วส่งข้อมูลผ่านDB-9 เข้าสู่คอมพิวเตอร์เพื่อทำการตรวจสอบรหัสที่ลูกค้ากดมาต่อไป จากนั้นเมื่อ Server ทำการประมวลผลแล้วจะส่งข้อมูลที่แสดงการยืนยันสิ่งที่ลูกค้าต้องการกลับมาแล้วส่งต่อไปยังฝั่งลูกค้าต่อไป



รูปที่ 3.6 แสดงแผนผังงาน(Flow chart) การรับ-ส่งของTRW-2.4GHz

3.4 การใช้งาน RF Module (TRW-2.4GHz)

TRW-2.4GHz เป็น RF Module ที่ใช้ความถี่ 2.4-2.524 GHz สื่อสาร RF ด้วยเทคนิค GFSK มีความเร็วในการส่งข้อมูล 250 kbps ในช่วง 280 m แต่มีความเร็วสูงขึ้นเป็น 1 Mbps ในระยะทำงาน 150 m ใช้ไฟล์เลี้ยง Module ≈ 3V สามารถตั้งค่าให้เป็น transmitter หรือ Receiver ก็ได้และมี mode การทำงานแบบ ShochBurst แบบ Duo Ceiver มี Antenna ภายใน Module



รูปที่ 3.7 แสดง RF Module (TRW-2.4GHz)

3.4.1 การทำงานของ Mode ShockBurst

เมื่อต้องการส่งข้อมูล (ใช้ Pin CE, CLK1, DATA)

- สั่งPin CE = 1 (เปิด ShockBurst Mode)
- สั่งข้อมูล(ADDRESS และ PAYLOAD) ทาง pin DATA โดยใช้ CLK1 เป็นตัว Synchronous Data ที่ส่งให้ TRW-2.4G
- TRW-2.4GHz จะคำนวณ CRC ให้(ขนาดตามที่เรา config ไว้)
- สั่ง pin CE = 0 (ปิด ShockBurst Mode) TRW-2.4GHz ใส่ Preamble และเริ่มส่ง RF จนกว่าจะเสร็จแล้วกลับมา Standby

เมื่อข้อมูลเข้ามาที่ตัวรับ (ใช้ Pin CE, DR1, CLK1, DATA)

- เมื่อมีข้อมูลเข้ามาที่ตัวรับ TRW-2.4GHz จะทำการตรวจสอบ ADDR ที่ส่งมาค่าตรงกับของตัวมันที่เรา Config ไว้หรือไม่ หากตรงจะทำงานต่อ หากไม่ตรงก็จะเพิกเฉยรอ package ใหม่
- ทำการ Check CRC ที่ส่งมาว่าถูกต้องหรือไม่ ถ้าถูกทำงานต่อหากผิดจะไม่มีการทำงานตอบกลับว่า Error ใดๆ ทั้งสิ้น
- TRW-2.4GHz จะ set ขา DR1 เป็น 1 ดังนั้น microcontroller จึงคอยวน check ขา DR1 ว่าเป็น 1 เมื่อใดแสดงว่า Dataพร้อมแล้ว
- Microcontroller ต้องสร้าง clock มาที่ CLK1 เพื่ออ่านข้อมูลจากทาง DATA ออกไป ถ้าครบแล้ว DR1 จะกลายเป็น 0 อีกครั้ง

ถ้าหากใช้ Duo Ceiver (คือ สามารถกำหนดให้ TRW-2.4GHz ตัวนี้มีช่องรับ 2 ทาง) การแบ่งการทำงานออกเป็น 2 channel แต่ละ channel มี ADDR ของมันเอง โดยที่ใช้ขาสัญญาณไม่ตรงกัน

Rx1 ใช้ pin (DR1, CLK1, DATA)

Rx2 ใช้ pin (DR2, CLK2, DOUT2)

3.4.2 การ Config TRW-2.4GHz

- สั่งให้ขา CS = 1 (เริ่มการ config) แต่ขา CE ต้องเป็น 0 ด้วยเพราะจะไม่อนุญาตให้ config และรับส่งข้อมูลพร้อมกันได้
- สั่งขา DATA ที่ใช้ config 144bit เข้าไปทาง DATA pin เริ่มจาก MSB และใช้ CLK1 ในการ Synchronous
- เมื่อเสร็จสิ้นให้ set CS = 0 เหมือนเดิม

ตารางที่ 3.1 วิธีการ Config TRW-2.4GHz

ตำแหน่ง bit	จำนวน bit	ชื่อ	คำอธิบาย
143:120	24	Test	สงวนใช้Test
119:112	8	DATA2_1	ความยาวของ payload คำนวนจาก
111:104	8	DATA1_W	Payload = 256-ADDR_W-CRC
103:64	40	ADDR2	ค่าของADDR ของตัวนรับ ตั้งค่าได้ทั้ง
63:24	40	ADDR1	channel 1 และ 2 มีขนาดสูงสุดได้ 5 byte
23:18	6	ADDR_W	กำหนดความยาวของ ADDR ได้
17	1	CRC_L	ความยาวของ CRC '0' = 8 bit '1' = 16 bit
16	1	CRC_EN	กำหนดใช้หรือไม่ใช้ CRC
15	1	RX2_EN	กำหนดใช้กี่ channel(รับ)
14	1	CM	เปิดปิด Shockburst Mode
13	1	RFDR_SB	กำหนด Data Rate '0' = 250 kbps '1' = 1 Mbps
12:10	3	XO_F	กำหนดเกี่ยวกับ CRYSTAL แต่กำหนดตายตัวเป็น 011
9:8	2	RF_PWR	กำหนด Power ในการส่ง '00' (-20dBm) '01' (-1dBm) '10' (-5dBm) '11' (0dBm)
7:1	7	RF_CH	กำหนดความถี่ที่ใช้ในการส่งคำนวณดังนี้ $RF_{ส่ง}$ = 2400+RF_CH MHz
0	1	RE_EN	กำหนดให้ TRW เป็น Tx-Rx '0'(Tx) '1'(RX)

ดังนั้นในวงจรTRW-2.4GHzที่ใช้นั้นจึงได้กำหนดค่าต่างๆไว้ดังนี้

1. ความยาวข้อมูลใน channel 1 และ channel 2 คือ 8 bit
2. Address Channel 2 ทั้ง 5 bytes คือ (0xD1),(0xAA),(0x55),(0xAA),(0x55)
   Address Channel 1 ทั้ง 5 bytes คือ (0xB5),(0x55),(0xAA),(0x55),(0xAA)
3. กำหนดให้ใช้ CRC ขนาด 16 bit
4. ใช้ 1 channel ในการรับ
5. เปิด ShockBurst mode
6. Data Rate คือ 250 kbps
7. RF Power คือ 0 dbm
8. ความถี่ในการส่งคือ 2,410 MHz

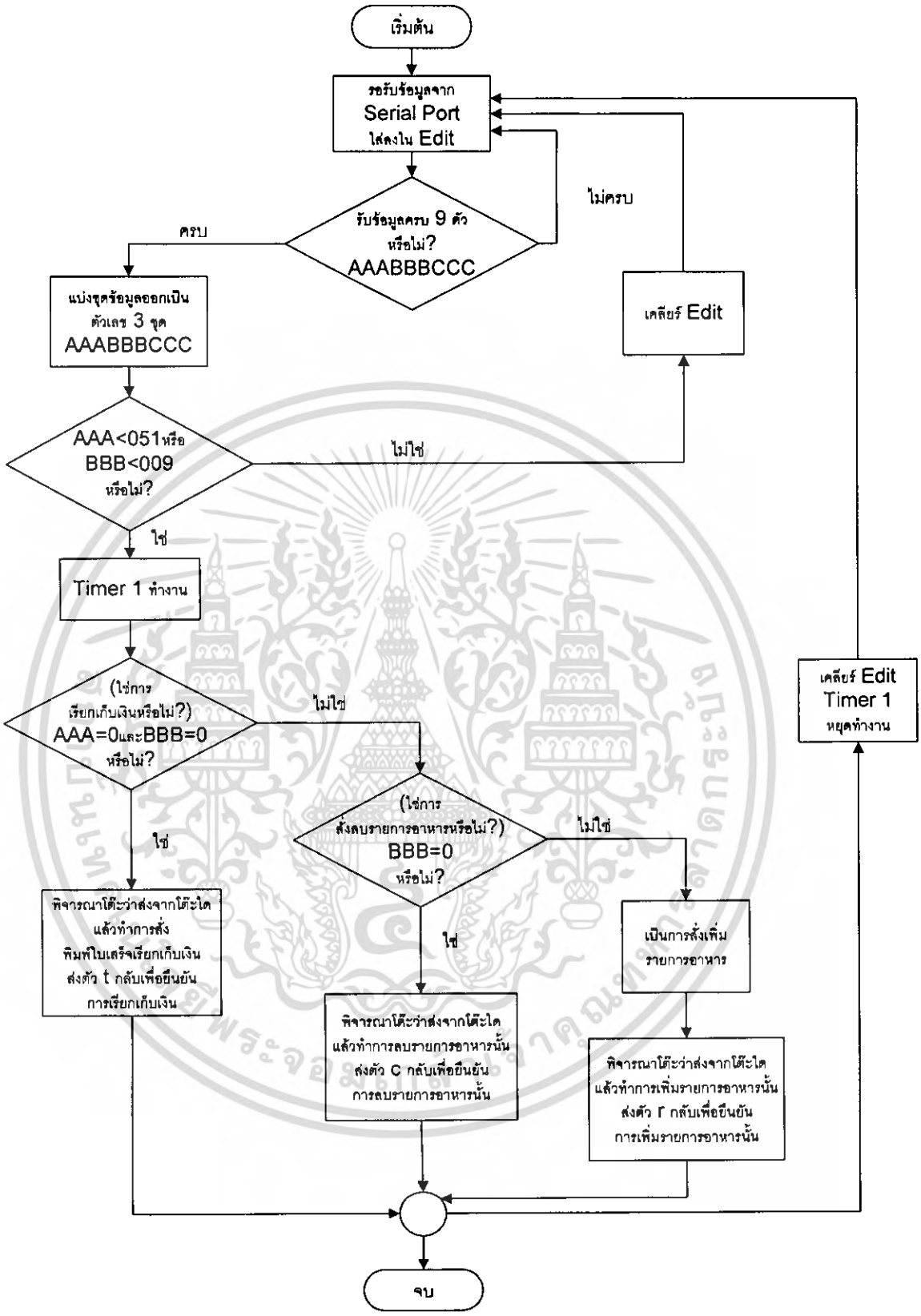
3.5 ส่วนของเครื่อง Server

หน้าที่ของเครื่อง Server

1. รับข้อมูลจากลูกค้าแต่ละโต๊ะ โดยรับค่าผ่านทาง Serial Port
2. แสดงรายละเอียดของข้อมูลที่ลูกค้าแต่ละโต๊ะสั่ง โดยแสดงผ่านหน้าจอแสดงผลที่สร้างขึ้นจากโปรแกรมเดลไฟล์
3. ส่งค่ากลับเพื่อยืนยันการส่งข้อมูลของลูกค้าแต่ละโต๊ะ ว่าข้อมูลที่ลูกค้าส่งมานั้นเป็นข้อมูลแบบใด
4. สามารถพิมพ์ใบเสร็จเพื่อใช้สำหรับการเรียกเก็บเงินจากลูกค้าได้

โปรแกรมของเครื่อง Server มีลักษณะการทำงาน ดังนี้

เมื่อโปรแกรมของเครื่อง Server อยู่ในสถานะพร้อมทำงาน โปรแกรมจะทำการรอรับข้อมูลที่ถูกส่งเข้ามาทาง Serial Port โดยลักษณะของข้อมูลที่ต้องการของโปรแกรมจะเป็นชุดตัวเลข 9 ตัวเท่านั้น โปรแกรมจะรอค่าจนกว่ารับค่าตัวเลขครบ 9 ตัว แล้วจึงนำชุดตัวเลข 9 ตัวนั้น ไปประมวลผล แล้วแสดงผลของข้อมูลที่รับมาได้ออกทางหน้าจอแสดงผลที่สร้างขึ้นจากโปรแกรม Delphi และมีการตอบกลับเพื่อใช้สำหรับการยืนยันว่าเครื่อง Server นั้น ได้รับข้อมูลจากลูกค้าแล้ว โดยการส่งการยืนยันไปยังโต๊ะของลูกค้าที่ส่งข้อมูลมา การทำงานของโปรแกรมเครื่อง Server นั้น จะเป็นการทำงานแบบอัตโนมัติ โดยใช้ Timer ควบคุมในการรับค่าและประมวลผลทั้งหมด

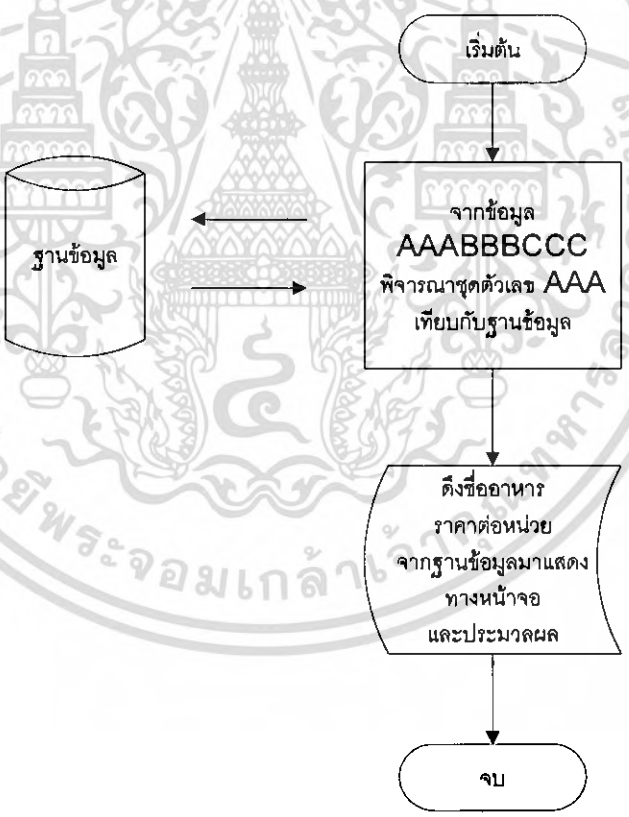


รูปที่ 3.8 แสดงแผนผังงาน(Flow Chart) การทำงานในส่วนของเครื่อง Server

อธิบายได้ดังนี้

1. เมื่อโปรแกรมอยู่ในสถานะพร้อมทำงาน โปรแกรมจะทำการรอรับข้อมูลจาก Serial Port โดยรับข้อมูลมาใส่ในช่อง Edit ข้อมูลที่เข้ามานั้น จะต้องเป็นข้อมูลตัวเลข 9 ตัว โดยโปรแกรมจะต้องรอรับข้อมูลให้ครบ 9 ตัวก่อน โปรแกรมถึงจะดำเนินการทำในขั้นตอนต่อไป เพื่อกันไม่ให้เกินการผิดพลาดในการนำข้อมูลไปประมวลผลโดยที่ข้อมูลยังไม่ครบ 9 ตัว
2. เมื่อข้อมูลตัวเลขใน Edit ครบ 9 ตัวแล้ว โปรแกรมจะทำการแบ่ง ชุดตัวเลข 9 ตัวที่รับมานั้นออกเป็นชุดตัวเลข 3 ชุด ชุดละ 3 ตัวเลข โดยหลักการแบ่งชุดตัวเลขมีดังนี้

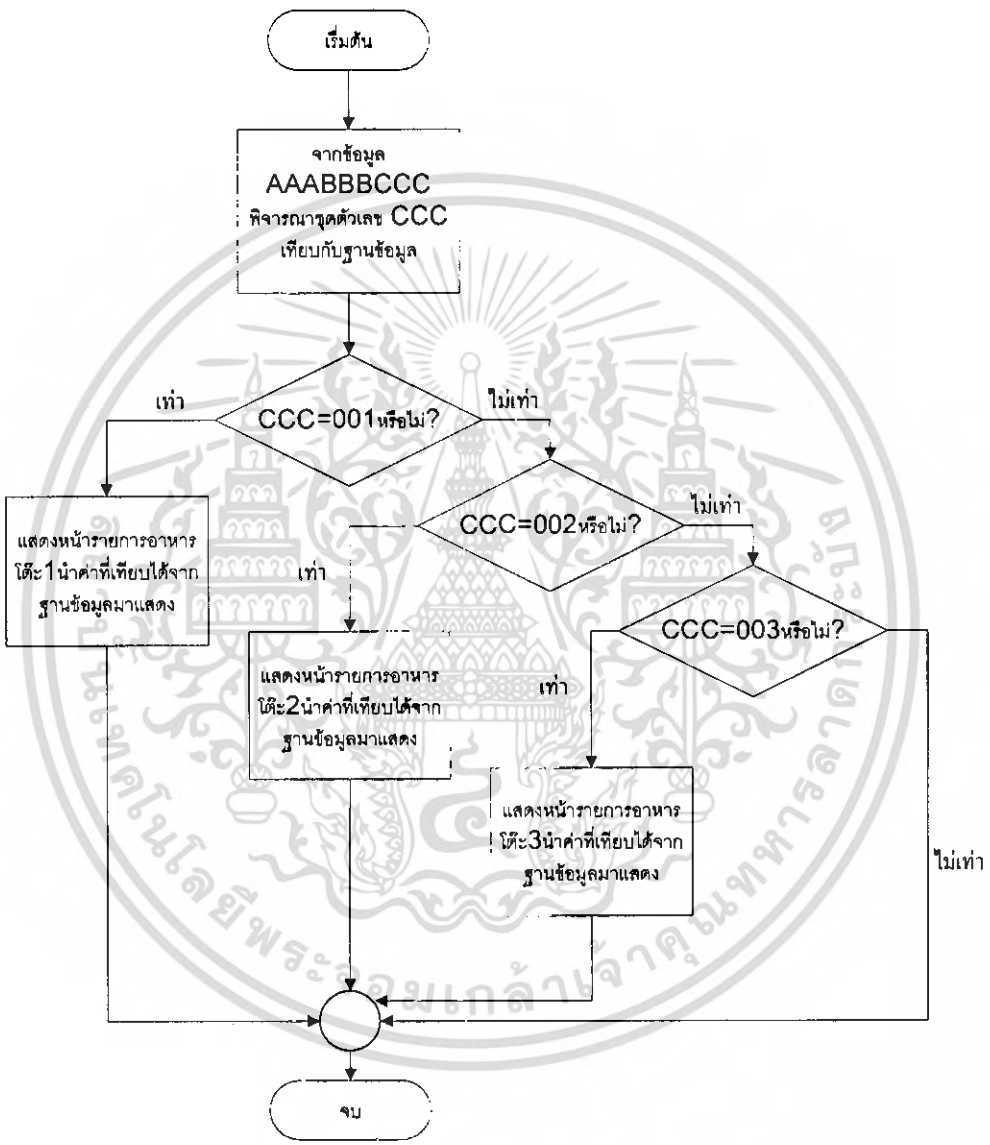
- ชุดที่หนึ่ง ใช้สำหรับแสดงรหัสอาหารที่ลูกค้าสั่ง ชุดตัวเลขนี้จะถูกนำไปเปรียบเทียบกับฐานข้อมูล แล้วดึงค่า ชื่อรายการอาหาร และ ราคาต่อหน่วย จากฐานข้อมูลมาแสดงทางจอ และนำไปใช้สำหรับการคิดราคาอาหารของโต๊ะนั้น ตัวอย่างเช่น รหัสอาหาร 002 เมื่อเทียบกับฐานข้อมูลแล้ว จะได้ชื่ออาหารคือ ข้าวผัดรวมมิตร ราคาต่อหน่วยเท่ากับ 30 บาท



รูปที่ 3.9 แสดงแผนผังงาน(Flow Chart) การทำงานในส่วนของเครื่อง Server ในการพิจารณาตัวเลขชุดที่หนึ่ง(รหัสอาหาร)

- ชุดที่สอง ใช้สำหรับแสดงจำนวนอาหารต่อหนึ่งรายการอาหารที่ลูกค้าสั่ง โดยชุดตัวเลขที่สองนี้จะถูกนำไปคูณกับราคาต่อหน่วยของรายการอาหารที่ลูกค้าสั่งมา เพื่อนำไปใช้สำหรับการคิดราคาอาหารของโต๊ะนั้น

- ชุดที่สาม ใช้สำหรับแสดงหมายเลขของโต๊ะลูกค้าที่ส่งข้อมูลตัวเลขเข้ามา เพื่อนำไปพิจารณาว่าข้อมูลถูกส่งมาจากลูกค้าโต๊ะใด เช่น 002001003 นั่นคือ ข้อมูลถูกส่งมาจากลูกค้าโต๊ะที่ 3



รูปที่ 3.10 แสดงแผนผังงาน (Flow Chart) การทำงานในส่วนของเครื่อง Server ในการพิจารณาตัวเลขชุดที่สาม(โต๊ะที่)

เช่น สมมติให้ข้อมูลที่รับได้มีค่าเท่ากับ 002001003 (AAA=002,BBB=001,CCC=003)

ชุดที่หนึ่ง	ชุดที่สอง	ชุดที่สาม
002	001	003
รหัสอาหาร	จำนวน	โต๊ะที่

รูปที่ 3.11 แสดงข้อมูลที่รับได้มีค่าเท่ากับ 002001003

นั่นคือ เป็นข้อมูลที่ลูกค้าจากโต๊ะที่ 3 ส่งมาเพื่อ สั่งอาหารที่มีรหัสเท่ากับ 002 = ข้าวผัดรวมมิตร ราคาต่อหน่วย 30 บาท สั่งเป็นจำนวน 1 หน่วย(จาน)

3. โปรแกรมจะทำการพิจารณาว่า รหัสอาหารนั้นเกินในฐานข้อมูลหรือไม่ และหมายเลขโต๊ะมีอยู่ในโปรแกรมหรือไม่ ถ้ารหัสอาหารเกิน หรือหมายเลขโต๊ะไม่มีอยู่ในโปรแกรมก็จะไม่มีการประมวลผลในขั้นตอนต่อไป

4. เมื่อพิจารณาแล้วว่าชุดตัวเลขแต่ละชุดเข้ากับโปรแกรม Timer 1 จะเริ่มทำงาน โปรแกรมจะทำการพิจารณาชุดตัวเลขทีละส่วน ว่าลูกค้าส่งข้อมูลตัวเลขมาเพื่อจุดประสงค์ใด

การพิจารณาจุดประสงค์ของชุดตัวเลขแบ่งออกได้เป็น 3 กรณี ดังนี้

(สมมติให้ รหัสอาหาร=002 ,จำนวน=001 และ โต๊ะที่=003)

กรณีที่ 1 เป็นการส่งข้อมูลจากลูกค้า เพื่อที่จะต้องการเรียกเก็บเงิน โดยชุดตัวเลขจะมีเลขชุดที่หนึ่งและชุดที่สอง มีค่าเป็น 000

ชุดที่หนึ่ง	ชุดที่สอง	ชุดที่สาม
000	000	003
รหัสอาหาร	จำนวน	โต๊ะที่

รูปที่ 3.12 แสดงข้อมูลจากลูกค้าเพื่อเรียกเก็บเงิน

จากตัวอย่าง หลังจากที่โปรแกรมพิจารณาแล้วว่า เป็นข้อมูลตัวเลขที่ส่งมาเพื่อที่ต้องการเรียกเก็บเงิน โปรแกรมจะทำการสั่งพิมพ์ใบเสร็จของโต๊ะ 3 พร้อมทั้งส่งตัว t ออกไปเพื่อยืนยันการเรียกเก็บเงินของโต๊ะ 3

กรณีที่ 2 เป็นการส่งข้อมูลจากลูกค้า เพื่อที่จะต้องการลบรายการอาหาร ที่ได้ทำการสั่งมาแล้ว โดยชุดตัวเลขจะมีเลขชุดที่สองมีค่าเป็น 000

ชุดที่หนึ่ง	ชุดที่สอง	ชุดที่สาม
002	000	003
รหัสอาหาร	จำนวน	โต๊ะที่

รูปที่ 3.13 แสดงข้อมูลจากลูกค้าเพื่อลบรายการอาหาร

จากตัวอย่าง หลังจากที่โปรแกรมพิจารณาแล้วว่า เป็นข้อมูลตัวเลขที่ส่งมาเพื่อที่ต้องการลบรายการอาหาร โปรแกรมจะทำการลบรายการอาหารที่มีรหัสเท่ากับ 002 ของโต๊ะ 3 และทำการคิดราคารวมของค่าอาหารของโต๊ะนั้นใหม่ แล้วทำการส่งตัว c ออกไป เพื่อยืนยันการลบรายการอาหารของโต๊ะ 3

กรณีที่ 3 เป็นการส่งข้อมูลจากลูกค้า เพื่อที่จะต้องการสั่งเพิ่มรายการอาหารตามรายละเอียดของชุดตัวเลขที่ส่งมา

ชุดที่หนึ่ง	ชุดที่สอง	ชุดที่สาม
002	001	003
รหัสอาหาร	จำนวน	โต๊ะที่

รูปที่ 3.14 แสดงข้อมูลจากลูกค้าเพื่อสั่งเพิ่มรายการอาหาร

จากตัวอย่าง หลังจากที่โปรแกรมพิจารณาแล้วว่า เป็นข้อมูลตัวเลขที่ต้องการสั่งเพิ่มรายการอาหาร โปรแกรมจะทำการเพิ่มรายการอาหารที่มีรหัสเท่ากับ 002 ของโต๊ะ 3 และทำการคิดราคารวมของค่าอาหารของโต๊ะนั้นใหม่ แล้วทำการส่งตัว r ออกไป เพื่อยืนยันการเพิ่มรายการอาหารของโต๊ะ 3

4. เมื่อทำการประมวลผลและแสดงผลทางหน้าจอแสดงผลในแต่ละกรณีเรียบร้อยแล้ว โปรแกรมจะทำการเคลียร์ค่าข้อมูลตัวเลข 9 ตัว ออกจาก Edit และ Timer 1 หยุดทำงาน โปรแกรมจะทำการรอรับค่าที่จะเข้ามาใหม่อีกครั้ง

บทที่4

การทดลองและผลการทดลอง

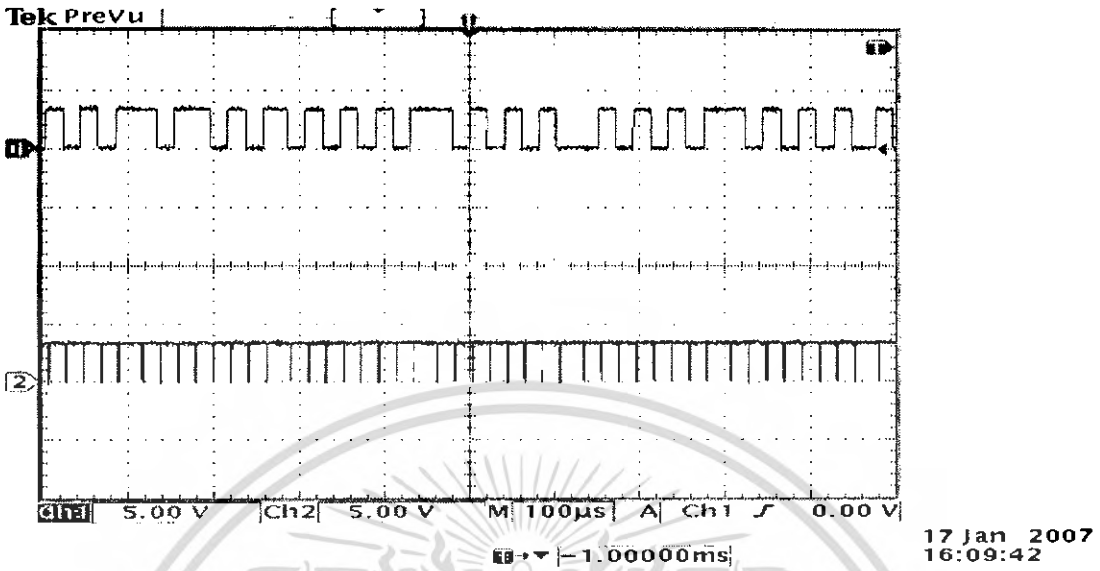
การทดลอง

4.1 การทดลองเรื่อง การวัดสัญญาณการรับ – ส่งระหว่าง TRW-2.4GHz ด้านส่งและด้านรับ

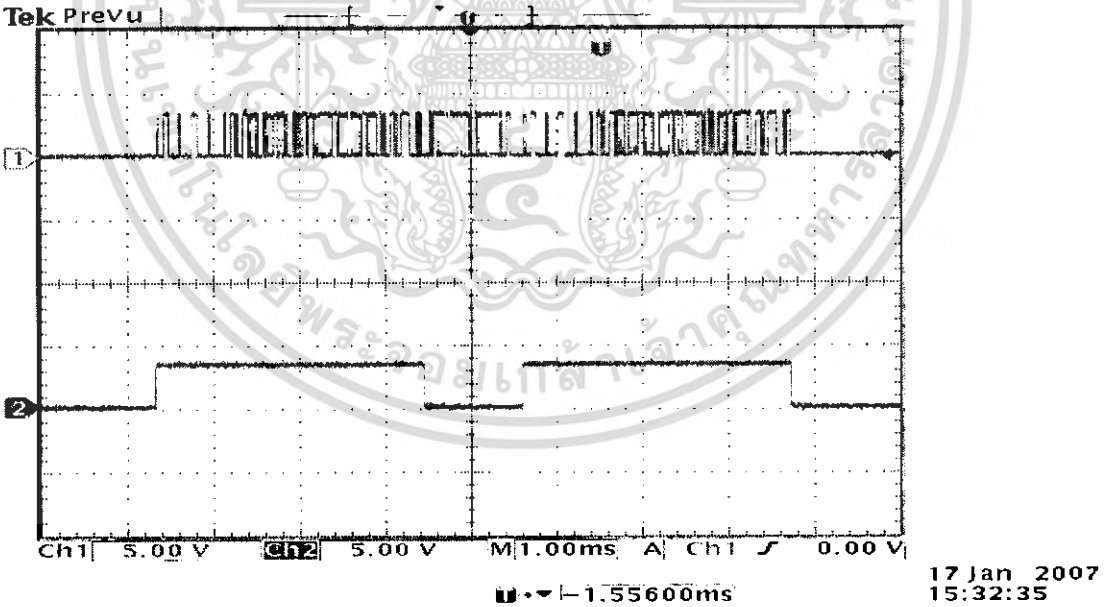
ขั้นตอนการทดลอง

1. ทำการวัดสัญญาณโดยออสซิโลสโคป ที่ขา Data เทียบกับ Clock ของ TRW-2.4GHz ฝั่งชุดสั่งงานของลูกค้า(ด้านส่ง)
2. ทำการวัดสัญญาณโดยออสซิโลสโคป ที่ขา Data เทียบกับ CS ของ TRW-2.4GHz ฝั่งชุดสั่งงานของลูกค้า(ด้านส่ง)
3. ทำการวัดสัญญาณโดยออสซิโลสโคป ที่ขา Data เทียบกับ CE ของ TRW-2.4GHz ฝั่งชุดสั่งงานของลูกค้า(ด้านส่ง)
4. ทำการวัดสัญญาณโดยออสซิโลสโคป ที่ขา Data เทียบกับ Clock ของ TRW-2.4GHz ฝั่ง Server (ด้านรับ)
5. ทำการวัดสัญญาณโดยออสซิโลสโคป ที่ขา DR1 เทียบกับ Data ของ TRW-2.4GHz ฝั่ง Server (ด้านรับ)
6. ทำการวัดสัญญาณโดยออสซิโลสโคป ที่ขา Data เทียบกับ CS ของ TRW-2.4GHz ฝั่ง Server (ด้านรับ)
7. ทำการวัดสัญญาณโดยออสซิโลสโคป ที่ขา Data เทียบกับ CE ของ TRW-2.4GHz ฝั่ง Server (ด้านรับ)

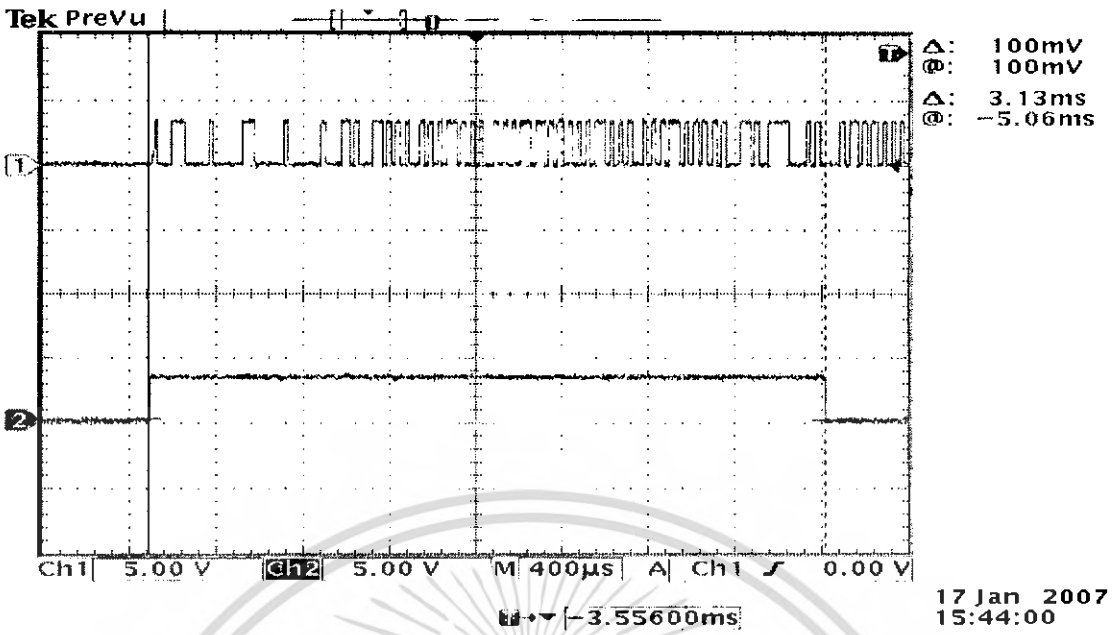
ผลการทดลอง



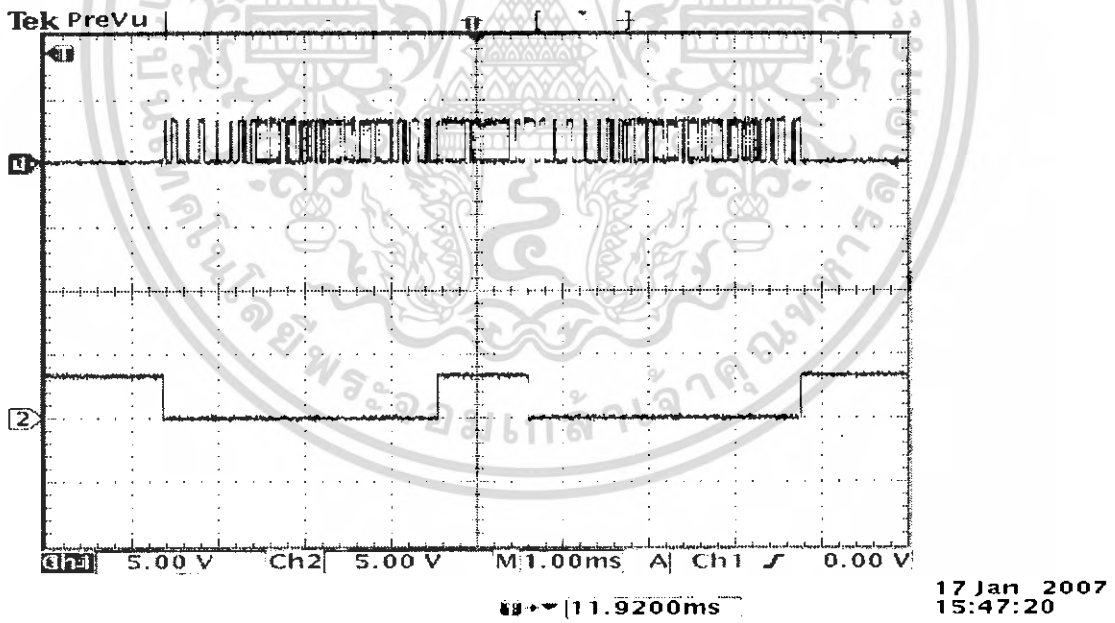
รูปที่ 4.1 การวัดสัญญาณที่ขาData (บน) เทียบกับสัญญาณนาฬิกาที่ขาCLK1 (ล่าง) ฝั่งชุดของลูกค้า



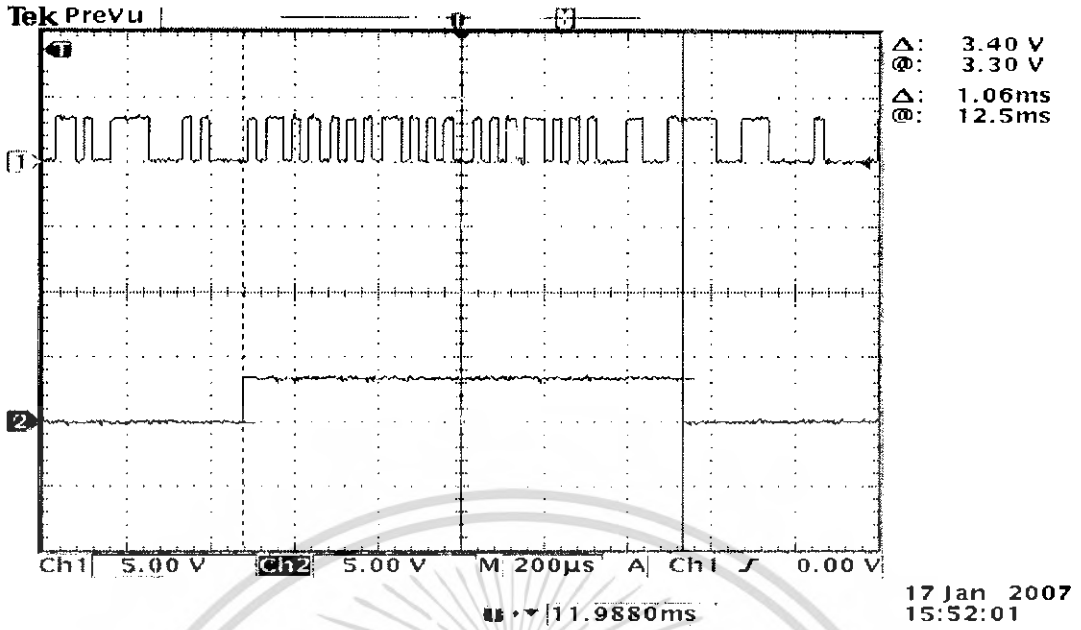
รูปที่ 4.2 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCS (ล่าง) ฝั่งชุดของลูกค้า



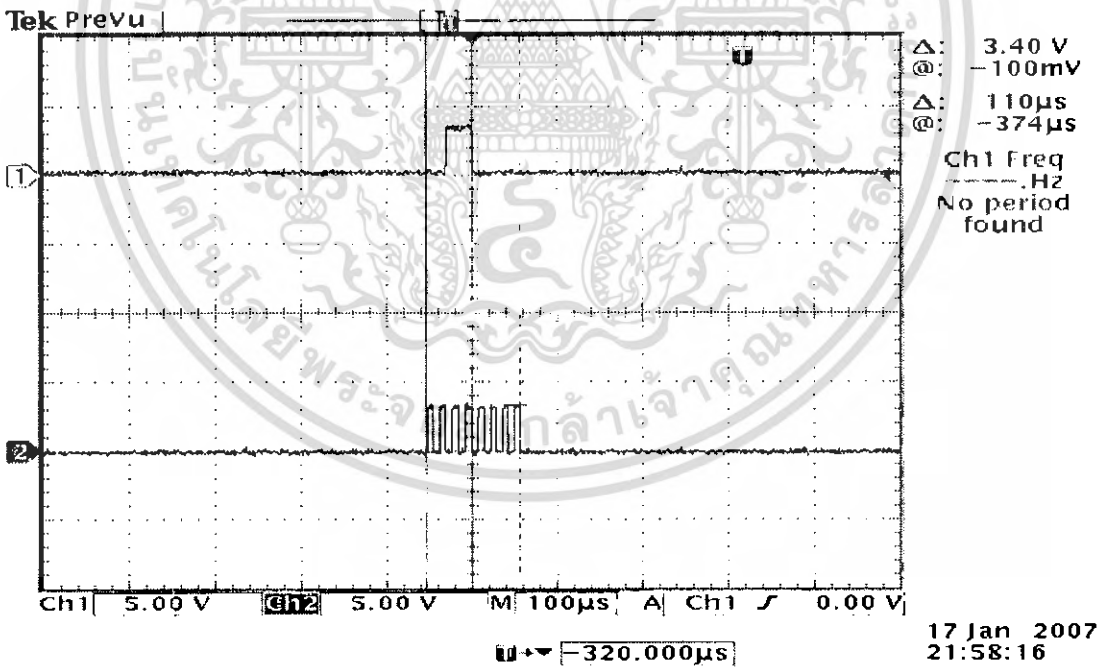
รูปที่ 4.3 ภาพขยายการวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCS (ล่าง)ฝั่งชุดของลูกค้า



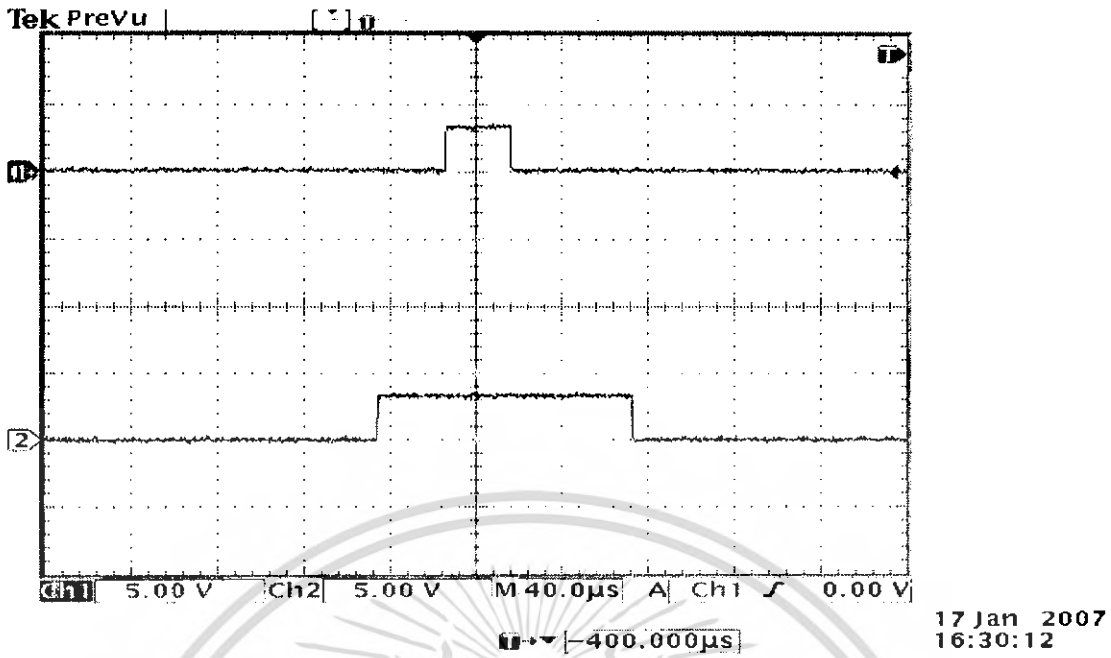
รูปที่ 4.4 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCE (ล่าง) ฝั่งชุดของลูกค้า



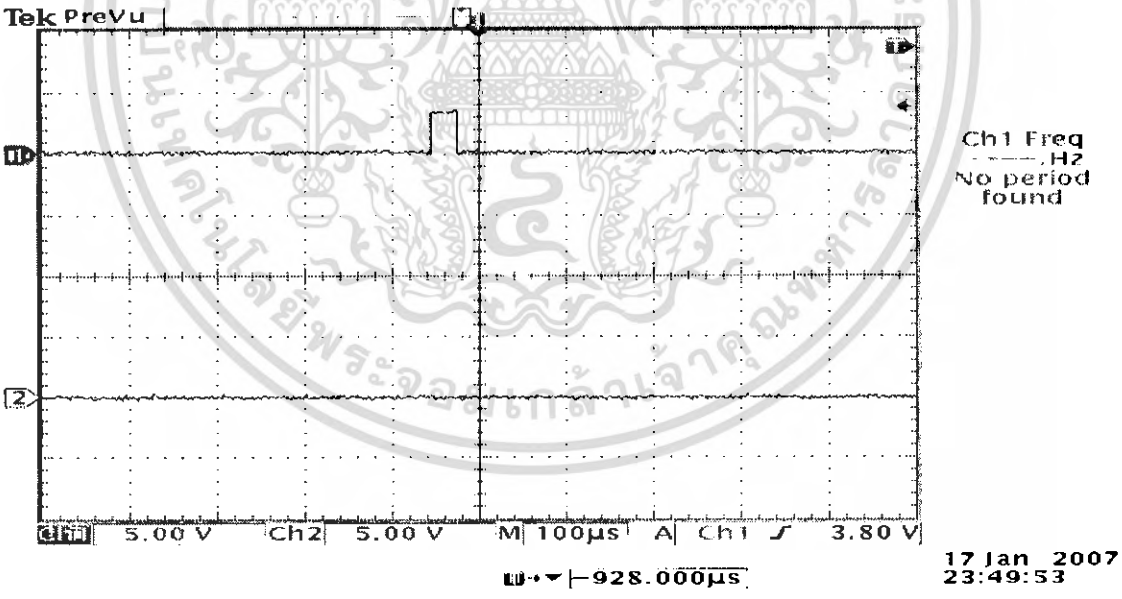
รูปที่ 4.5 ภาพขยายการวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCE (ล่าง) ฝั่งชุดของลูกค้า



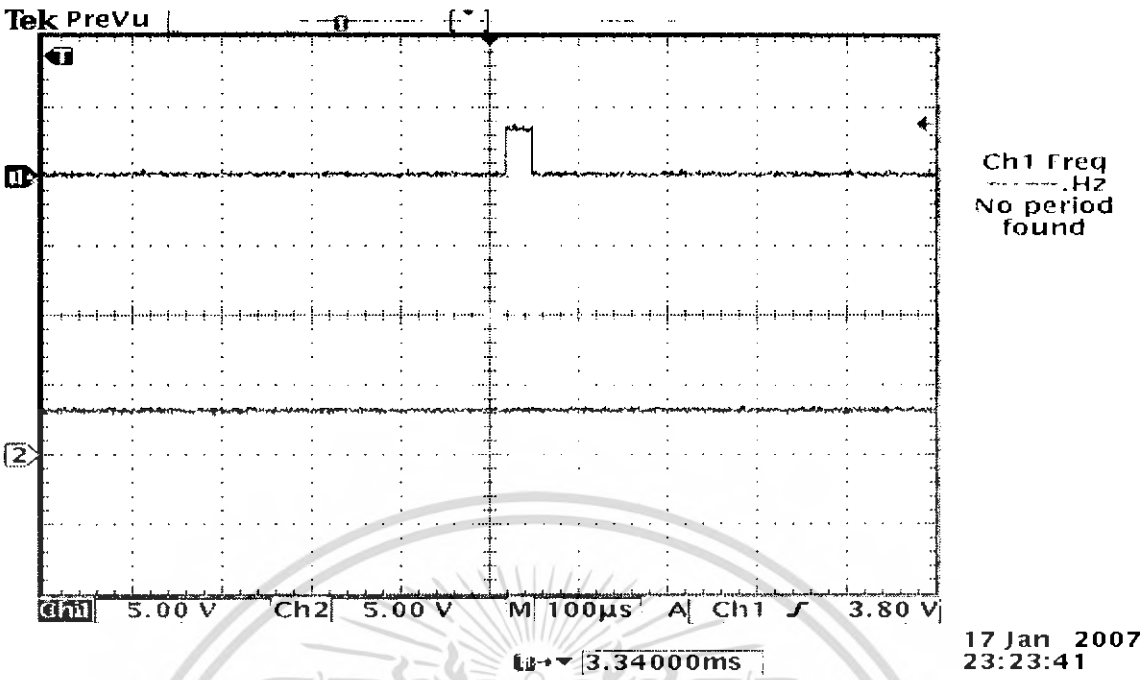
รูปที่ 4.6 การวัดสัญญาณที่ขาData (บน) กับสัญญาณนาฬิกาที่ขา CLK1 (ล่าง) ฝั่งเครื่อง Server



รูปที่ 4.7 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขา DR1 (ล่าง) ฝั่งเครื่อง Server



รูปที่ 4.8 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCS (ล่าง) ฝั่งเครื่อง Server



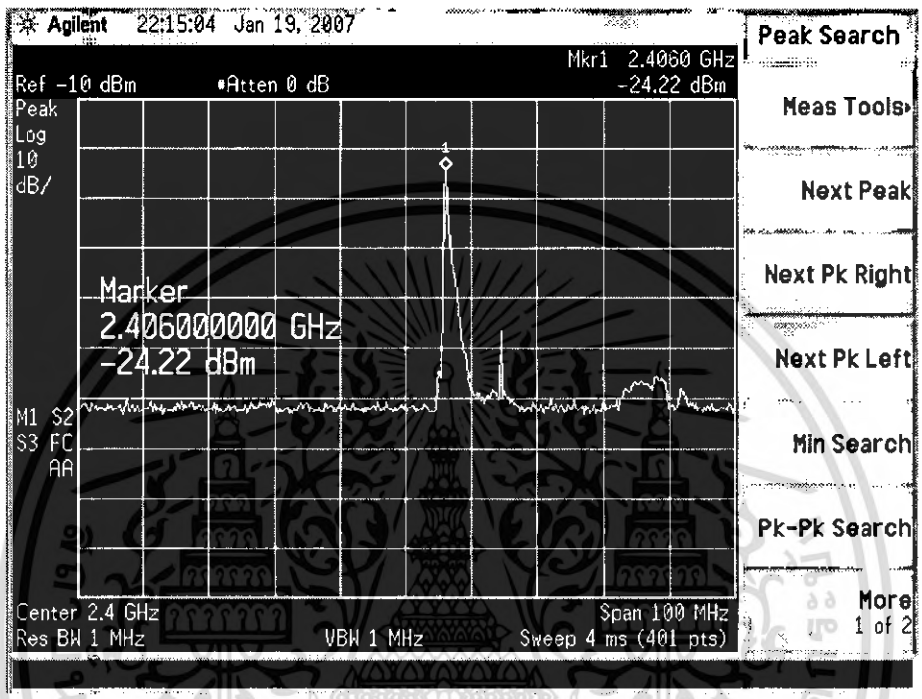
รูปที่ 4.9 การวัดสัญญาณที่ขาData (บน) กับสัญญาณที่ขาCE (ล่าง) ฝั่งเครื่อง Server

4.2 การทดลองเรื่อง การวัดความถี่ที่ใช้ส่งและกำลังของสัญญาณของTRW-2.4GHz

ขั้นตอนการทดลอง

1. ทำการวัดความถี่และกำลังส่งของสัญญาณโดยใช้เครื่อง Spectrum Analyzer
2. ทำการสังเกตและบันทึกผลการทดลอง

ผลการทดลอง



รูปที่ 4.10 แสดงผลการวัดความถี่และกำลังของสัญญาณที่ใช้ส่งของ TRW-2.4GHz

จากรูปที่ 4.10 จะเห็นได้ว่าความถี่ที่วัดได้คือ 2.4060 GHz หรือ 2,406 MHz และวัดค่ากำลังสัญญาณได้ -24.22 dBm

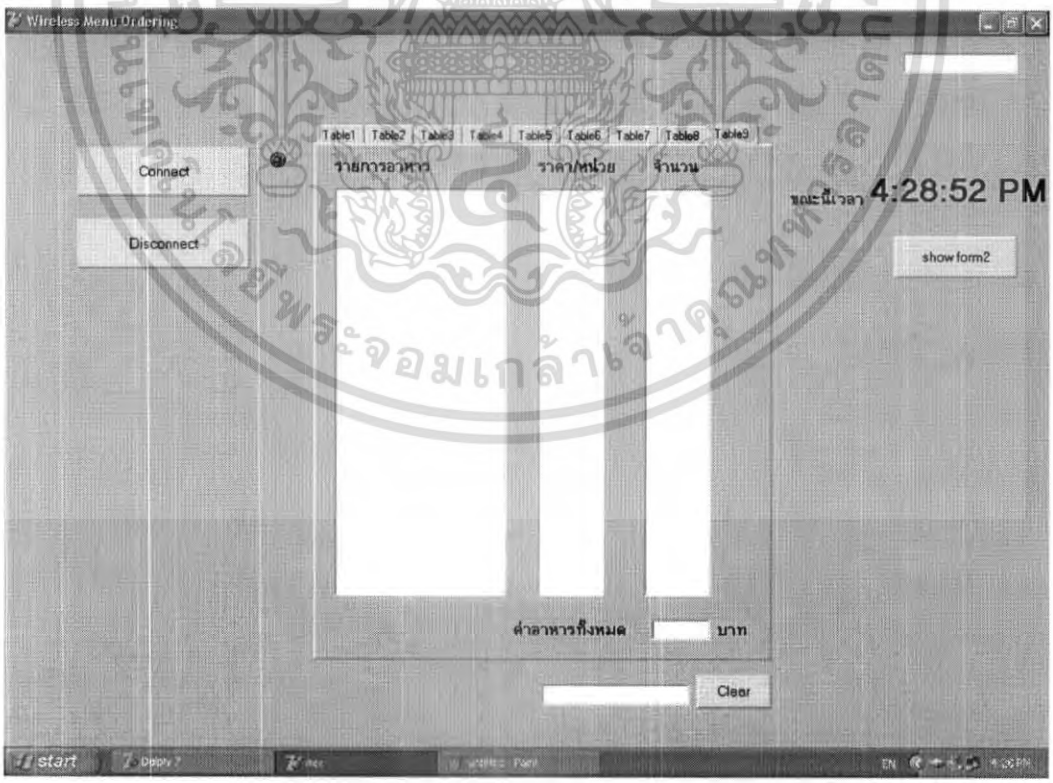
4.3 การทดลองเรื่อง การแสดงผลของเครื่อง Server และการแสดงผลตอบรับที่จอ LCD กรณีที่ข้อมูลที่ส่งมาจากชุดสั่งงานของลูกค้าถูกต้อง (รหัสอาหาร < 050 , หมายเลขโต๊ะ < 009)

ขั้นตอนการทดลอง

1. เปิดเครื่องทางด้านชุดสั่งงานของลูกค้าและเครื่องทางด้าน Server ให้อยู่ในสภาวะพร้อมทำงาน และเชื่อมต่อเครื่องทางด้าน Server เข้ากับเครื่อง Server โดยเชื่อมต่อผ่านทาง Serial Port

รูปที่ 4.11 แสดงหน้าจอ LCD เมื่อชุดสั่งงานของลูกค้าอยู่ในสถานะพร้อมทำงาน

2. Run โปรแกรมเดลไฟล์ หน้าฟอร์ม Wireless Food Ordering จะปรากฏขึ้น ทำการคลิกปุ่ม Connect เพื่อทำการเชื่อมต่อโปรแกรมเข้ากับ Serial Port เครื่อง Server จะอยู่ในสถานะพร้อมรอรับข้อมูล

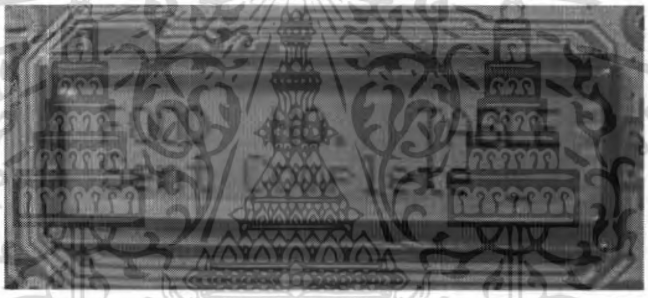


รูปที่ 4.12 แสดงหน้าจอแสดงผลของเครื่อง Server ในสถานะพร้อมทำงาน

3. ทำการกดคีย์แพดทางด้านชุดสั่งงานของลูกค้า ในการทดลองที่ 4.3 ให้ส่งข้อมูลที่มีค่าเท่ากับ 002001003

รูปที่ 4.13 แสดงหน้าจอ LCD พร้อมที่จะส่งข้อมูลที่มีค่าเท่ากับ 002001003

กดปุ่ม # เพื่อทำการส่งข้อมูล หน้าจอ LCD จะปรากฏคำว่า Sent complete ดังรูปที่ 4.14

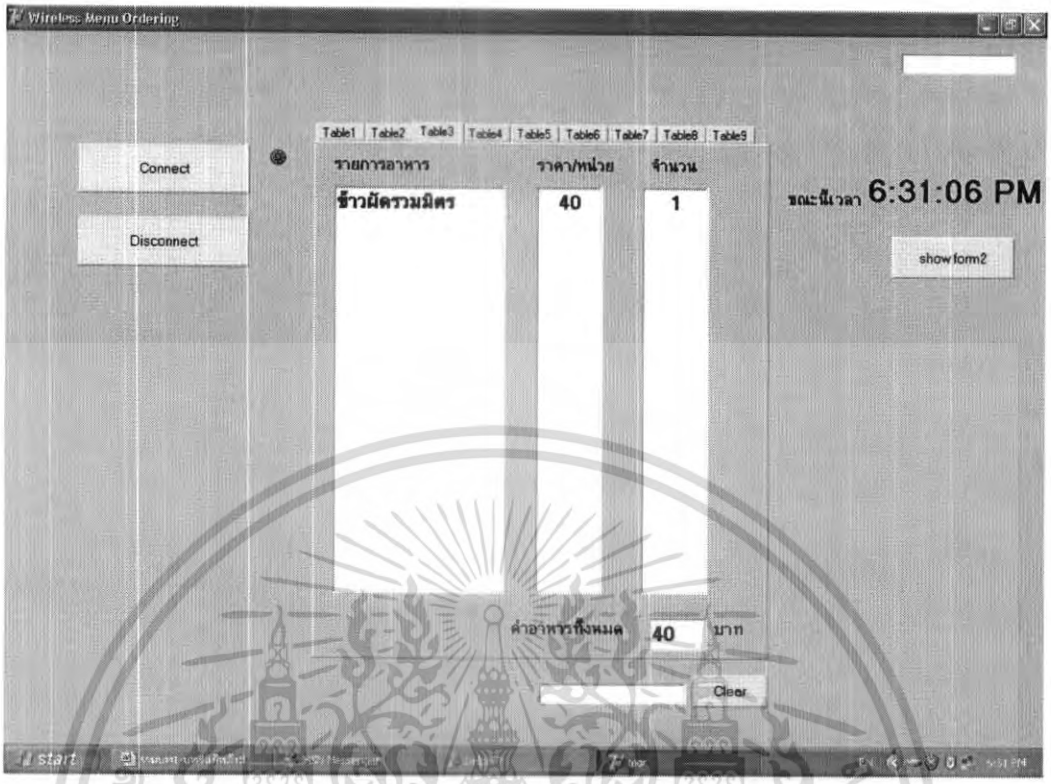


รูปที่ 4.14 แสดงหน้าจอ LCD หลังจากการกดปุ่ม #

ถ้ากดตัวเลขไม่ครบ 9 หลัก แล้วกดปุ่ม # หน้าจอ LCD จะปรากฏคำว่า Incompleted data ดังรูปที่ 4.15

รูปที่ 4.15 แสดงหน้าจอ LCD เมื่อมีกดตัวเลขไม่ครบ 9 ตัว

ผลการทดลอง



รูปที่ 4.16 แสดงหน้าจอแสดงผลของเครื่อง Server หลังจากรับข้อมูลที่มีค่าเท่ากับ 002001003

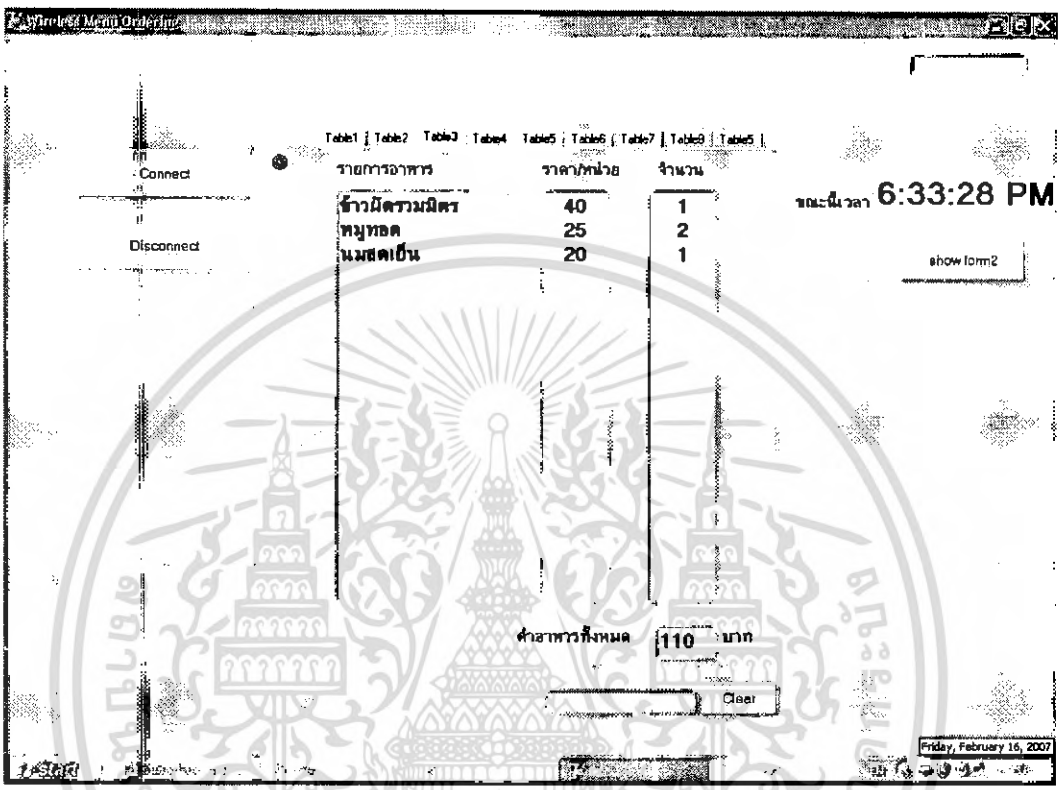
หน้าจอแสดงผลของเครื่อง Server จะแสดงหน้ารายการอาหารของโต๊ะที่ 3 และทำการเพิ่มรายการอาหารที่มีรหัสเท่ากับ 002 = ข้าวผัดรวมมิตร และราคาต่อหน่วยเท่ากับ 30 บาท และสั่งเป็นจำนวน 1 หน่วย ค่าอาหารทั้งหมด 30 บาท

ส่วนหน้าจอ LCD ของเครื่องชุดสั่งงานฝั่งลูกค้า จะปรากฏคำว่า Receive 002 เป็นการยืนยันว่าเครื่อง Server นั้น ได้รับข้อมูลการสั่งอาหารรหัส 002 เรียบร้อยแล้ว ดังรูปที่ 4.17

รูปที่ 4.17 แสดงหน้าจอ LCD เมื่อได้รับการยืนยันการสั่งเพิ่มรายการอาหารที่ 002 จากเครื่อง Server

4. ทำการส่งข้อมูลจากชุดสั่งงานฝั่งลูกค้าอีก 2 ครั้ง โดยให้ข้อมูลมีค่าเท่ากับ 021002003 และ 045001003 ทำการส่งตามลำดับ

ผลการทดลอง

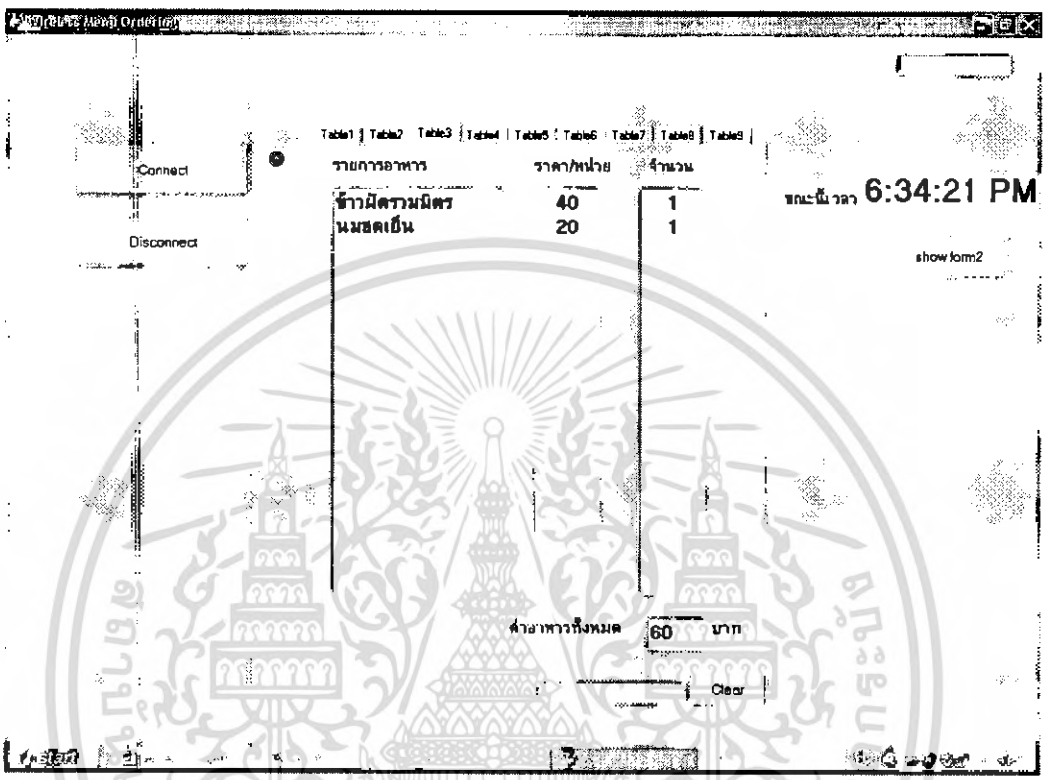


รูปที่ 4.18 แสดงหน้าจอแสดงผลของเครื่อง Server หลังจากรับข้อมูลที่มีค่าเท่ากับ 021002003 , 045001003 ตามลำดับ

จากรูปที่ 4.18 หน้ารายการอาหารของโต๊ะที่ 3 จะถูกเพิ่มรายการอาหารเข้าไปอีก 2 รายการ ตามลำดับที่ชุดสั่งงานฝั่งลูกค้าส่งมา พร้อมทั้งคิดราคาค่าอาหารรวมของโต๊ะที่ 3 ใหม่

5. ทำการส่งข้อมูลจากชุดสั่งงานของลูกค้าอีกครั้งโดยให้ข้อมูลมีค่าเท่ากับ 021000003

ผลการทดลอง



รูปที่ 4.19 แสดงหน้าจอแสดงผลของเครื่อง Server หลังจากรับข้อมูลที่มีค่าเท่ากับ 021000003

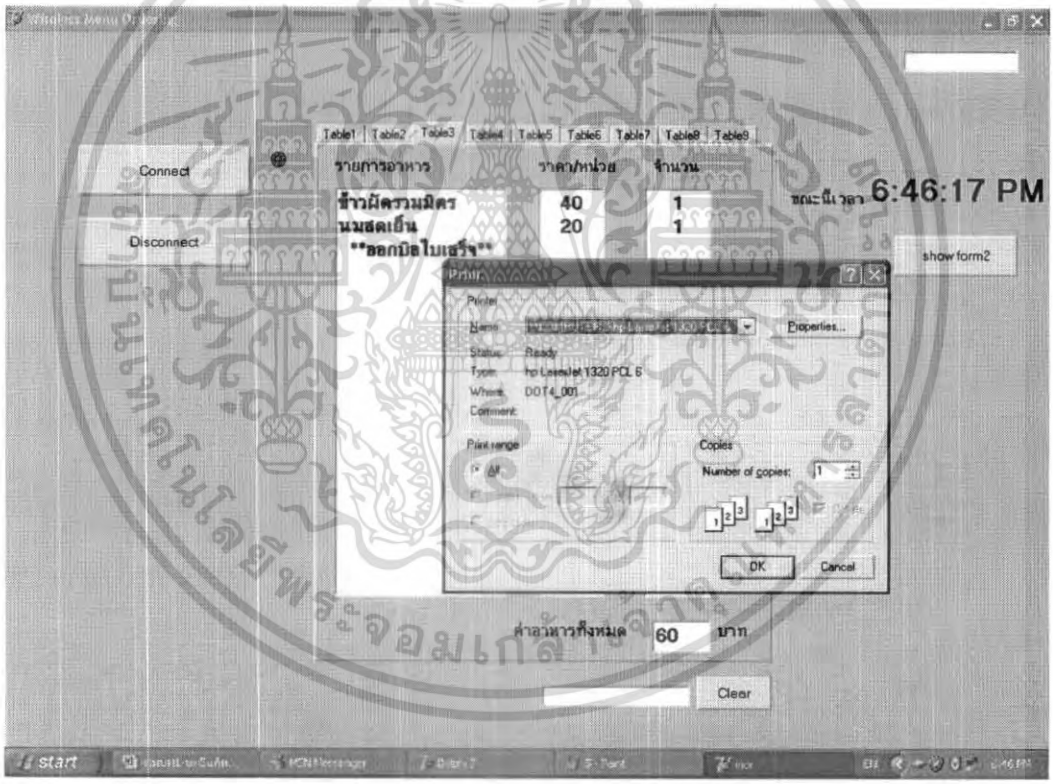
จากรูปที่ 4.19 หน้ารายการอาหารของโต๊ะที่ 3 รายการอาหารรหัส 021= หมูทอด จะถูกลบไป โปรแกรมจะทำการคิดราคาค่าอาหารรวมใหม่

ส่วนหน้าจอ LCD ของเครื่องชุดสั่งงานฝั่งลูกค้า จะปรากฏคำว่า Cancel 021 เป็นการยืนยันว่าเครื่อง Server นั้น ได้รับข้อมูลการลบอาหารรหัส 012 เรียบร้อยแล้ว ดังรูปที่ 4.20

รูปที่ 4.20 แสดงหน้าจอ LCD เมื่อได้รับการยืนยันการสั่งลบรายการอาหารที่ 021 จากเครื่อง Server

6. ทำการส่งข้อมูลจากชุดสั่งงานของลูกค้าอีกครั้งโดยให้ข้อมูลมีค่าเท่ากับ 000000003

ผลการทดลอง



รูปที่ 4.21 แสดงหน้าจอแสดงผลของเครื่อง Server หลังจากรับข้อมูลที่มีค่าเท่ากับ 000000003

จากรูปที่ 4.21 หน้ารายการอาหารของโต๊ะที่ 3 จะปรากฏ ******** ออกบิลใบเสร็จ ******** แล้วหน้าต่าง Print จะปรากฏขึ้น แล้วทำการคลิกปุ่ม OK เพื่อทำการพิมพ์ใบเสร็จสำหรับใช้ในการคิดเงินกับลูกค้า

ใบเสร็จค่าอาหารโต๊ะ 3

ราคา/หน่วย	จำนวนที่สั่ง	รายการอาหาร
40 บาท	1	ข้าวผัดรวมมิตร
25 บาท	2	หมูทอด
20 บาท	1	นมสดเย็น

ยกเลิกรายการ หมูทอด

********** ค่าอาหารทั้งหมด 60 บาท **********

6:46:17 PM

รูปที่ 4.22 แสดงใบเสร็จที่ใช้สำหรับการเรียกเก็บเงินกับลูกค้าโต๊ะ 3

ส่วนหน้าจอ LCD ของเครื่องชุดสั่งงานฝั่งลูกค้า จะปรากฏคำว่า Thank you 003 เป็นการยืนยันว่าเครื่อง Server นั้น ได้รับข้อมูลการเรียกเก็บเงินของโต๊ะที่ 3 เรียบร้อยแล้ว ดังรูปที่ 4.23

รูปที่ 4.23 แสดงหน้าจอ LCD เมื่อได้รับการยืนยันการเรียกเก็บเงินของลูกค้าโต๊ะที่ 3 จากเครื่อง Server

4.4 การทดลองเรื่อง การแสดงผลของเครื่อง Server และการแสดงผลตอบรับที่จอLCD กรณีที่ข้อมูลที่ส่งมาจากชุดสั่งงานของลูกค้าไม่ถูกต้อง (รหัสอาหาร > 050 , หมายเลขโต๊ะ > 009)

ขั้นตอนการทดลอง

1. ทำการทดลองต่อจากการทดลองที่ 4.4 โดยทำการส่งข้อมูลจากชุดสั่งงานของลูกค้าอีกครั้งโดยให้ข้อมูลมีค่าเท่ากับ 085001003

ผลการทดลอง

รูปที่ 4.24 แสดงหน้าจอ LCD เมื่อไม่ได้รับการยืนยันการรับข้อมูลจากเครื่อง Server เนื่องจากรหัสอาหารไม่มีในฐานข้อมูล

หน้าจอ LCD ของเครื่องชุดสั่งงานฝั่งลูกค้า ก็จะปรากฏคำว่า Error เนื่องจากไม่มีการตอบยืนยันจากเครื่อง Server เพราะข้อมูลที่เครื่อง Server รับได้นั้น เป็นข้อมูลที่มีรหัสอาหารเกิน 050

ส่วนของหน้าจอแสดงผลของเครื่อง Server นั้น จะไม่มีการเปลี่ยนแปลงใดๆ

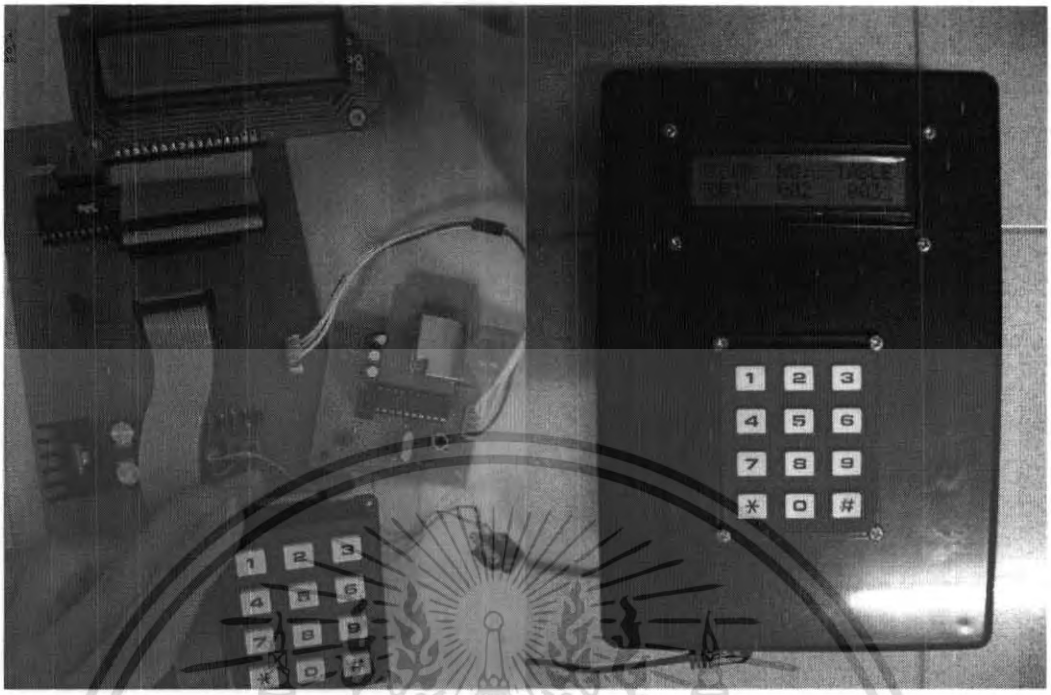
2. ทำการส่งข้อมูลจากชุดสั่งงานของลูกค้าอีกครั้งโดยให้ข้อมูลมีค่าเท่ากับ 002001090

ผลการทดลอง

รูปที่ 4.25 แสดงหน้าจอ LCD เมื่อไม่ได้รับการยืนยันการรับข้อมูลจากเครื่อง Server เนื่องจากหมายเลขโต๊ะที่ไม่มีในโปรแกรม

จากรูปจะเห็นได้ว่า ผลการทดลองนั้น จะเหมือนผลการทดลองในข้อที่ 1 คือ หน้าจอ LCD ของเครื่องชุดสั่งงานฝั่งลูกค้า จะปรากฏคำว่า Error เนื่องจากไม่มีการตอบยืนยันจากเครื่อง Server เพราะข้อมูลที่เครื่อง Server รับได้นั้น เป็นข้อมูลที่มีรหัสโต๊ะที่เกิน 009

ในส่วนของหน้าจอแสดงผลของเครื่อง Server ให้ผลเหมือนการทดลองในข้อที่ 1 คือไม่มีการเปลี่ยนแปลงใดๆ



รูปที่ 4.26 แสดงชุดสั่งงานของลูกค้า

รูปที่ 4.27 แสดงภาครับทางฝั่ง Server

บทที่5

บทวิจารณ์และสรุป

- ชุดสั่งงานของลูกค้า

จากการทำงานของชุดสั่งงานของลูกค้า ซึ่งมีการสั่งงานผ่านคีย์แพดและแสดงผลที่ LCD นั้น ทำให้ลูกค้าสามารถระบุชนิดอาหาร จำนวนที่สั่ง และโต๊ะที่สั่งได้โดยผ่านรหัส 9 ตัวที่จะส่งออกไป นอกจากนี้ยังรอการตอบกลับจาก Server โดยการตอบกลับนั้นจะบ่งบอกว่า Server ได้รับทราบความต้องการของลูกค้าโดยจะแสดงผลที่ LCD เช่นกัน ซึ่งมีดังนี้ คือ

1. Server รับทราบรายการอาหารที่ลูกค้าสั่ง ซึ่งจะตอบกลับว่า Received ตามด้วยรหัสอาหาร
2. กรณีลูกค้าต้องการลบรายการอาหารที่ส่งไปแล้ว จะส่ง รหัสอาหารที่จะลบ ตามด้วยจำนวนอาหาร 000 และหมายเลขโต๊ะ แล้ว Server จะตอบกลับว่า Cancel ตามด้วยรหัสอาหาร
3. กรณีลูกค้าจะเรียกเก็บเงิน จะส่งรหัสอาหาร000 จำนวนอาหาร000 และตามด้วยหมายเลขโต๊ะ แล้ว Server จะตอบกลับว่า Thank you ตามด้วยหมายเลขโต๊ะ
4. กรณีมีความผิดพลาดในการส่งข้อมูล หรือ ข้อมูลที่ลูกค้าส่งนั้นไม่มีในฐานข้อมูลจะแสดงคำว่า Error ที่ LCD แล้วให้ลูกค้าทำการส่งข้อมูลใหม่

ปัญหาของชุดสั่งงานของลูกค้า

1. ความสว่างของจอ LCD บางครั้งจออาจมืดหรือสว่างไป ซึ่งในการใช้งานจริงลูกค้าอาจไม่ทราบว่าต้องทำอย่างไร
2. การที่ใช้ LCD ชนิด 2 บรรทัดแสดงผลนั้น ทำให้การที่จะเขียนโปรแกรมเพิ่มรายละเอียดนั้นเช่น อาหารไม่ใส่ผัก หรือไม่เผ็ดมาก มีข้อจำกัด ซึ่งการที่ใช้ LCD ชนิดนี้ก็เพราะราคาถูกกว่า

วิธีแก้ไขปัญหา

1. ก่อนที่จะให้บริการทุกครั้งควรตรวจสอบความสว่างของหน้าจอโดยการปรับที่ตัวต้านทานปรับค่าได้
2. ถ้าหากมีงบประมาณสูงขึ้น การใช้ LCD ที่แสดงผลได้มากขึ้นจะทำให้ สามารถระบุความละเอียดของข้อมูลอาหารได้มากขึ้น

- ส่วนรับ-ส่งข้อมูล

การส่งข้อมูลนั้น จะมีการนำผลของรหัสที่ลูกค้ากดเข้ามา 9 ตัวนั้น ส่งออกที่ขา Tx ของไมโครคอนโทรเลอร์ของชุดสั่งงานของลูกค้า เข้ามาที่ขา Rx ของวงจร TRW-24GHz แล้วจะทำการส่งแบบไร้สายไปยังวงจร TRW-24 GHz ทางด้าน Server จากนั้นวงจร TRW-2.4GHz ด้านฝั่งของลูกค้าก็จะทำการเปลี่ยนโหมดเป็นโหมดรับเพื่อรอการตอบกลับจาก Server

เมื่อวงจรTRW-2.4GHz ฝั่ง Server ได้รับข้อมูลนั้นก็จะทำการส่งผ่าน Max-232 ไปยัง Server โดยผ่านขา Tx Server จะทำการประมวลผลข้อมูลนั้นแล้วส่งค่าค่าหนึ่งกลับมาเข้าทางขา Rx วงจร TRW-2.4GHz ทางฝั่ง Server จะเปลี่ยนเป็นโหมดส่ง ส่งค่านั้นไปยังวงจร TRW-2.4GHz ด้านฝั่งลูกค้า แล้วจะส่งต่อทางขา Tx ของไมโครคอนโทรลเลอร์ชนิด 20 ขา เข้าทาง ขา Rx ของไมโครคอนโทรลเลอร์ที่ควบคุม LCD ให้ขึ้นแสดงข้อความต่างๆให้ลูกค้าทราบ

จากการทำการส่งข้อมูลจากชุดสั่งงานของลูกค้าไปยังเครื่องฝั่ง Server จริงๆพบว่าระยะทางการรับส่งข้อมูลจะประมาณ 80 เมตร ในที่โล่ง แต่ในการทำงานจริงในร้านอาหารนั้นอาจมีสิ่งกีดขวางมากมายที่เป็นอุปสรรคในการส่งข้อมูลทำให้การใช้งานได้ระยะไม่ถึง

ปัญหาของส่วนรับ-ส่งข้อมูล

1. ในการส่งข้อมูลอาจมีสัญญาณรบกวนจากที่อื่นทำให้ข้อมูลผิดพลาดได้
2. สิ่งกีดขวางต่างๆเช่น ตึก บ้าน ต้นไม้ เป็นส่วนสำคัญที่มีผลต่อการทำให้ประสิทธิภาพการรับส่งลดลง
3. ในส่วนของการส่งแบบไร้สายโดยใช้ TRW-2.4GHz ของโครงงานนี้นั้นจากอุปสรรคที่กล่าวมาแล้วข้างต้น จึงทำให้ส่งได้ระยะไม่มากนัก ซึ่งรูปแบบร้านอาหารที่ใช้การสั่งอาหารแบบไร้สายนั้น ส่วนมากสร้างมาเพื่ออำนวยความสะดวกภายในร้านอาหารที่มีพื้นที่กว้างๆ

วิธีแก้ไขปัญหา

1.ในการใช้งานจริง หากมีงบประมาณควรใช้ตัวรับส่งที่มีประสิทธิภาพดีกว่านี้

- ส่วนของเครื่อง Server

จากผลการทดลองในส่วนของโปรแกรมแสดงผลทางหน้าจอ ซึ่งรับข้อมูลตัวเลข 9 ตัว จาก Serial Port แล้วนำไปประมวลผลออกทางหน้าจอคอมพิวเตอร์ จะเห็นได้ว่าโปรแกรมทำงานได้ตรงตามจุดประสงค์ของโครงงาน โดยสามารถแยกโต๊ะที่สั่งอาหาร ระบุรายการอาหารตามรหัสอาหารที่สั่งมา แสดงราคาของรายการอาหารแต่ละชนิด สามารถเพิ่ม ลบ รายการอาหาร คิดราคาอาหารรวมของแต่ละโต๊ะ พร้อมทั้งสั่งพิมพ์ใบเสร็จสำหรับการคิดเงินกับลูกค้าได้ และยังสามารถส่งผลยืนยันการรับข้อมูลของแต่ละโต๊ะได้

ปัญหาของส่วนของเครื่อง Server

ในการเขียนโปรแกรมสำหรับร้านอาหารนั้น ต้องคำนึงถึงสถานการณ์ที่จะเกิดขึ้นจริงกับร้านอาหาร ซึ่งเกิดได้หลายกรณีมาก ซึ่งโปรแกรมในส่วนของ Server นั้น ยังไม่สามารถที่จะครอบคลุมในทุกๆสถานการณ์ได้ อย่างเช่น ลูกค้าบางคนต้องการสั่งต้มยำรวมมิตร และไม่ต้องเผ็ดมาก จึงยากที่จะเขียนระบบฐานข้อมูลให้ครบทุกกรณีได้

และในกรณีที่ลูกค้าส่งข้อมูลเพื่อเรียกเก็บเงิน การพิมพ์ใบเสร็จของโปรแกรมเครื่อง Server นั้นยังมีการขึ้นหน้าต่างของ Print Setup อยู่ ซึ่งทำให้โปรแกรมทำงานไม่เป็นอัตโนมัติตามที่ต้องการของผู้เขียนโปรแกรม

วิธีแก้ไขปัญหา

ต้องพิจารณาความละเอียดในแต่ละส่วนของโครงงานนี้ตามความเป็นจริงให้มากที่สุด ทั้งในส่วนของชุดสั่งงานของลูกค้า, ระบบฐานข้อมูล และ ส่วนการแสดงผลทางหน้าจอคอมพิวเตอร์ จึงจะสามารถตอบสนองความต้องการของลูกค้าได้อย่างน่าพอใจ

- ข้อจำกัดของโครงงาน

1. ในส่วนของการส่งแบบไร้สายโดยใช้ TRW-2.4GHz ของโครงงานนี้นั้น จากอุปสรรคที่กล่าวมาแล้วข้างต้น จึงทำให้ส่งได้ระยะไม่มากนัก ซึ่งรูปแบบร้านอาหารที่ใช้การสั่งอาหารแบบไร้สายนั้น ส่วนมากสร้างมาเพื่ออำนวยความสะดวกภายในร้านอาหารที่มีพื้นที่กว้างๆ
2. ความสามารถในการระบุความละเอียดของอาหารมีจำกัด

หนังสืออ้างอิง

1. ธีรบูลย์ หล่อวิเชียรรุ่ง , นคร ภักดีชาติ , ชัยวัฒน์ ลิ้มพรจิตรวิไล "ปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 ด้วยโปรแกรมภาษาซี " , บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด ,กรุงเทพ, 2521 .
2. รศ.สมยศ อุณณปิยะ , "การประยุกต์ใช้งาน ไมโครคอนโทรลเลอร์ตระกูล MCS-51 " คณะวิศวกรรมศาสตร์ , สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2546.
3. อุดม รานอก , "ภาษาซีสำหรับงานควบคุมไมโครคอนโทรลเลอร์ MCS-51 ", นนทบุรี , ไอดีซีฯ, 2548.
4. สัจจะ จรัสรุ่งรวีวร , จักรพงษ์ สุขประเสริฐ "เริ่มต้นอย่างมืออาชีพด้วย Delphi 7 ฉบับสมบูรณ์" , บริษัท เอช เอ็น กรุ๊ป จำกัด , กรุงเทพ ,2546.

ภาคผนวก

**** เมนูอาหาร ****

รหัสอาหาร	รายการอาหาร	ราคา
001	ข้าวผัดหมู	30 บาท
002	ข้าวผัดรวมมิตร	40 บาท
003	ข้าวผัดปู	30 บาท
004	ข้าวกระเพราไก่	30 บาท
005	ราดหน้าหมูเส้นหมี่	30 บาท
006	ราดหน้าหมูหมี่กรอบ	30 บาท
007	ผัดซีอิ๊วหมูเส้นใหญ่	30 บาท
008	ผัดซีอิ๊วไก่เส้นหมี่	30 บาท
009	ข้าวผัดรวมมิตร	35 บาท
010	ต้มยำรวมมิตร	45 บาท
011	ต้มยำทะเล	50 บาท
012	ปลาเก๋าราดพริก	50 บาท
013	แกงส้มปลาทอด	50 บาท
014	ไข่เจียวหมูสับ	25 บาท
015	ไข่เจียวหอยนางรม	50 บาท
016	ไข่ดาว	10 บาท
017	กระเพราไข่เยี่ยวม้า	30 บาท
018	กระเทียมรวมมิตร	30 บาท
019	ต้มโคล้งปลาย่าง	45 บาท
020	ผัดผักรวมมิตร	30 บาท
021	หมูทอด	25 บาท
022	แกงส้มรวมมิตร	50 บาท
023	แกงส้มปลาเก๋าทอด	60 บาท
024	ข้าวกระเพราหมู	30 บาท
025	ปูผัดผงกระหรี่	70 บาท
026	ยำสามกรอบ	50 บาท
027	แกงขี้เหล็ก	40 บาท
028	แกงจืดรวมมิตร	50 บาท

029	ปลาช่อนลุยสวน	90 บาท
030	ต้มยำกุ้ง	50 บาท
031	ปลาไหลผัดเผ็ด	35 บาท
032	ผักบุ้งไฟแดง	30 บาท
033	ข้าวเปล่า	10 บาท

**** เมนูเครื่องดื่ม ****

รหัสอาหาร	รายการอาหาร	ราคา
034	น้ำส้ม	20 บาท
035	น้ำแอปเปิ้ล	20 บาท
036	น้ำชามะนาว	20 บาท
037	น้ำมะนาว	20 บาท
038	น้ำฝรั่ง	20 บาท
039	น้ำกระเจี๊ยบ	15 บาท
040	น้ำเก๊กฮวย	15 บาท
041	แตงโมปั่น	20 บาท
042	สตอเบอรี่ปั่น	20 บาท
043	กาแฟเย็น	30 บาท
044	กาแฟร้อน	30 บาท
045	นมสดเย็น	20 บาท
046	นมสดร้อน	20 บาท
047	น้ำโค้ก	15 บาท
048	กระเทียมรวมมิตร	30 บาท
049	น้ำโซดา	15 บาท
050	น้ำแข็งเปล่า	10 บาท

คู่มือการใช้งานเครื่องสั่งอาหารไร้สาย

1. เลือกเมนูอาหารที่ท่านต้องการ แล้วทำการกดรหัสอาหารเป็นตัวเลข 3 หลัก ตามรายการอาหารที่ท่านเลือก ตามด้วยรหัสจำนวนอาหารที่ท่านต้องการสั่งเป็นตัวเลข 3 หลัก และตามด้วยรหัสของโต๊ะท่านเป็นตัวเลข 3 หลัก
2. ในกรณีที่ท่านใส่ตัวเลขผิด ปุ่ม * ใช้ในการลบรหัสที่กดผิดทีละตัว
3. เมื่อท่านกดตัวเลขครบ 9 ตัวแล้ว ให้กดปุ่ม # เพื่อทำการส่งข้อมูล หน้าจอจะขึ้นคำว่า Sent Complete
4. ในกรณีที่ท่านใส่ตัวเลขไม่ครบ 9 ตัว หน้าจอจะขึ้นคำว่า Incompleted Data
5. เมื่อท่านได้กดส่งไปแล้วจะมีการตอบยืนยันกลับจาก Server โดยหน้าจอจะขึ้นคำว่า received ตามด้วยรหัสอาหารที่ท่านสั่งไป
6. เมื่อท่านต้องการลบรายการอาหารที่ส่งไปแล้ว ให้ท่านทำการกดรหัสอาหารที่ต้องการลบ ตามด้วยรหัสจำนวนอาหาร 000 และ ตามด้วยรหัสโต๊ะของท่าน กดปุ่ม # หน้าจอจะขึ้นคำว่า cancel ตามด้วยรหัสอาหาร ที่ท่านสั่งลบไป
7. เมื่อท่านต้องการเรียกเก็บเงิน ให้ท่านกดรหัสอาหาร 000 รหัสจำนวนอาหาร 000 ตามด้วยรหัสโต๊ะของท่าน กดปุ่ม # หน้าจอจะขึ้นคำว่า Thank you ตามด้วยรหัสโต๊ะของท่าน
8. ในกรณีที่ท่านกดปุ่ม # แล้ว หน้าจอขึ้นคำว่า Error แสดงว่าท่านได้ใส่รหัสอาหาร หรือรหัสโต๊ะผิด ท่านต้องทำการกรอกตัวเลข และทำการส่งข้อมูลใหม่

ชุดสั่งงานของลูกค้า

```
#pragma CODE
#include <reg51.h>
#include <stdio.h>
#define KEYPAD P1
sbit RS = P2^0;
sbit E = P2^1;

char str[]="000000000";
unsigned char nx2,ch,count,q;
/***************Function keep data******************/
void putcharracter() {
      unsigned char ny;
      ny=nx2;
      if (ny==0x40){
            str[0]=ch;
      }
      else if (ny==0x41){
            str[1]=ch;
      }
      else if (ny==0x42){
            str[2]=ch;
      }
      else if (ny==0x46){
            str[3]=ch;
      }
      else if (ny==0x47){
            str[4]=ch;
      }
      else if (ny==0x48){
            str[5]=ch;
      }
      else if (ny==0x4C){
            str[6]=ch;
      }
      else if (ny==0x4D){
            str[7]=ch;
      }
      else if (ny==0x4E){
            str[8]=ch;
      }

}
/******************Function delayLCD*****************/

void delayLCD() {
  unsigned int i;
  for (i=0;i<255;i++);
}
/******************Function  delay******************/
void delay() {
      unsigned char i,j;
      for(i=0;i<100;i++)
```

```
        for(j=0;j<255;j++);
}

/******************Function initLCD*************************/

void InitLCD() {

  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();
  delayLCD();


  P0 = 0x01; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();

  P0 = 0x03; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();

  P0 = 0x03; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();

  P0 = 0x03; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();

  P0 = 0x01; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();

  P0 = 0x04; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();

  P0 = 0x38; RS = 0;
```

```
  E = 1; delayLCD(); E = 0; delayLCD();

  P0 = 0x0F; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();

  P0 = 0x01; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();

}
/**********************Function Clearscreen for
LCD**********************/
void ClearScreen(){
  P0 = 0x01; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();
}
/**********************Function  display charracter 1 byte to LCD
************************/

void PutLCD(unsigned char ch) {
  P0 = ch; RS = 1;
  E = 1; delayLCD(); E = 0; delayLCD();
}

/******************Function display string to LCD***************/
void PrintLCD(unsigned char stt[]) {
  unsigned char i;
  for (i=0;stt[i]!=0;i++)
    PutLCD(stt[i]);
}
/*****************Function go to adress LCD*****************/

void GotoLCD(unsigned char addr) {
  P0 = addr | 0x80; RS = 0;
  E = 1; delayLCD(); E = 0; delayLCD();
}
/***************Fuction init keypad*************/
void Keypad_Init(void)
{
      KEYPAD = 0xFF;
}


unsigned char x_scan() {
  KEYPAD = ~0x01;
    if ((~KEYPAD)&0x10) return 0;
    if ((~KEYPAD)&0x20) return 1;
    if ((~KEYPAD)&0x40) return 2;
  KEYPAD = ~0x02;
    if ((~KEYPAD)&0x10) return 3;
    if ((~KEYPAD)&0x20) return 4;
    if ((~KEYPAD)&0x40) return 5;
  KEYPAD = ~0x04;
    if ((~KEYPAD)&0x10) return 6;
    if ((~KEYPAD)&0x20) return 7;
    if ((~KEYPAD)&0x40) return 8;
  KEYPAD = ~0x08;
    if ((~KEYPAD)&0x10) return 9;
    if ((~KEYPAD)&0x20) return 10;
```

```
    if ((~KEYPAD)&0x40) return 11;
  return 255;    // if press  return 255
}


/************************Main Loop*****************/
void main() {


  unsigned char yy,nx,i;
  unsigned int timeout;
  char wut;
  char stt[16];

  char conv_table[] = {

            1,2,3,
            4,5,6,
            7,8,9,
          200,0,100
  };
      PCON=0x00;
      TMOD=0x20;
      TH1=0xF4;       // Baud rate 2400
      SCON=0x50;
      RI=0;
      TI=0;
      TR1=1;
  InitLCD();


  GotoLCD(0x00);
  PrintLCD("FOOD");
  GotoLCD(0x06);
  PrintLCD("NO.");
  GotoLCD(0x0B);
  PrintLCD("TABLE");

  Keypad_Init();
  GotoLCD(0x40);

  nx = 0x40 ; // where start
  while (1) {
    yy = x_scan();
      if (yy!=255) {
        if (conv_table[yy]==200) {
          if (nx>0x40) {

            if (nx==0x4C){
            nx=nx-3;
            }
            if (nx==0x46){
            nx=nx-3;

            }
             nx--;
            GotoLCD(nx);
            PrintLCD(" ");
```

```
            GotoLCD(nx);

          }
        } else
        if (conv_table[yy]==100) {
          if(nx >= 0x4F){//data 9 bytes
          GotoLCD(0x40);
          PrintLCD("                ");
          GotoLCD(0x40);
          PrintLCD("Sent Complete ");
          q=0;
          count=0;
          while (count<=9){
          SBUF=str[q];
            TI=0;
            while (TI==0)
                  {
                  }
            q=q++;
            count=count++;
            delay();
          }


          delay();
          delay();
          delay();
          delay();
          delay();
          delay();
          GotoLCD(0x40);
          PrintLCD("                ");
          nx = 0x40;
          GotoLCD(nx);


      ////////////////////////////////////////////////////////////////
////////////////
            // Wait for feed back

      ////////////////////////////////////////////////////////////////
////////////////

              timeout = 50000;
              while ((RI==0)&&(timeout)) {
                timeout--;
                _nop_();
                _nop_();
              }

              if (timeout) {
                wut = SBUF;
                RI = 0;
                if (wut=='c') {//for cancel menu
                  GotoLCD(0x40);
                     PrintLCD("Cancel          ");
                  GotoLCD(0x47);
```

```
            PutLCD(str[0]);
            PutLCD(str[1]);
            PutLCD(str[2]);
        }

        if (wut=='t') {//for check bill
          GotoLCD(0x40);
            PrintLCD("Thank you       ");
          GotoLCD(0x49);
            PutLCD(str[6]);
            PutLCD(str[7]);
            PutLCD(str[8]);
        }

        if (wut=='r') {//for receive menu
          GotoLCD(0x40);
            PrintLCD("Received        ");
          GotoLCD(0x49);
            PutLCD(str[0]);
            PutLCD(str[1]);
            PutLCD(str[2]);
        }
      } else {//no menu in database or data failed
          GotoLCD(0x40);
            PrintLCD("Error           ");
      }
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();
    delay();

    GotoLCD(0x40);
    PrintLCD("                ");
    nx = 0x40;
    GotoLCD(nx);

  }

  else {//data is not 9 bytes
  GotoLCD(0x40);
  PrintLCD("Incompleted data");
  delay();
  delay();
  delay();
  delay();
  delay();
```

```
            delay();
            nx = 0x40;
            GotoLCD(0x40);
            PrintLCD("                ");
            GotoLCD(nx);
            }

          } else {//show data on LCD

            sprintf(stt, "%d", conv_table[yy]);
            PrintLCD(stt);
            ch=stt[0];
            nx2=nx;
            putcharracter();
            nx++;
            if (nx==0x43){
            nx=nx+3;
            GotoLCD(nx);
            }
            if (nx==0x49){
            nx=nx+3;
            GotoLCD(nx);
            }
            if (nx>0x4F){
              nx=nx--;
              GotoLCD(nx);
              PrintLCD("  ");

            }
            }
          while (x_scan()!=255) ;  //check finish presss

        }
    }
}
```

ด้านฝั่งลูกค้า

```
#include <regx52.h>

#define CE   P1_2
#define CS   P1_3
#define DAT  P1_5
#define CLK  P1_4
#define DR1  P1_6

#define MODE_TX    0
#define      MODE_RX 1

void Wait(unsigned char n)
{
      while (n--) ;
}
```

```
void Init_TRW24G(void)
{
      CE  = 0;
      CS  = 0;
      CLK = 0;
      DAT = 0;
      DR1 = 0;
      Wait(1);
}

void CLK_TRW24(void)
{
      CLK = 0;
      CLK = 1;
}

void Write_TRW24(unsigned char Data)
{
      unsigned char i;
      bit Out;
      for (i=0;i<8;i++)
      {
            Out = Data & 0x80;
            DAT = Out;
            CLK_TRW24();
            Data = Data << 1;
      }
}
unsigned char Read_TRW24(void)
{
      unsigned char i,Temp;
      bit Out;
      DAT = 1;
      for (i=0;i<8;i++)
      {
            Temp = Temp << 1;
            CLK = 1; Wait(1);
            Out = DAT;
            if (Out) { Temp = Temp + 0x01;}
            CLK = 0; Wait(1);
      }
      return(Temp);
}
void SetMode_TRW24(unsigned char Mode)
{
      Wait(1);
      CE = 0;
      CS = 1;
      Write_TRW24(0x8E);
      Write_TRW24(0x08);
      Write_TRW24(0x1C);
```

```
        Write_TRW24(0X08);
        Write_TRW24(0X08);

        Write_TRW24(0xD1);
        Write_TRW24(0xAA);
        Write_TRW24(0X55);
        Write_TRW24(0xAA);
        Write_TRW24(0X55);

        Write_TRW24(0xB5);
        Write_TRW24(0X55);
        Write_TRW24(0xAA);
        Write_TRW24(0X55);
        Write_TRW24(0xAA);

        Write_TRW24(0xA3);
        Write_TRW24(0X4F);
        Write_TRW24(0X0A+Mode);

        DAT = Mode;
        DR1 = Mode;
        CE = Mode;

        CS = 0;
        Wait(1);
}

void Send_TRW24(unsigned char Data)
{
        Wait(1);
        CS = 0;
        CE = 1;
        Write_TRW24(0xB5);
        Write_TRW24(0X55);
        Write_TRW24(0xAA);
        Write_TRW24(0X55);
        Write_TRW24(0xAA);

        Write_TRW24(Data);
        Wait(1);
        CLK = 0;
        CE = 0;
        Wait(1);
}

void InitSerial() {
        PCON=0X00;
        TMOD=0X20;
        TH1=0xF4;
        SCON=0X50;
        RI=0;
        TI=0;
```

```
        TR1=1;
}

unsigned char GetChar() {
        unsigned char ch;
        while (RI==0);
        ch = SBUF;
        RI = 0;
        return ch;
}

void SendChar(unsigned char ch) {
        SBUF = ch;
        while (TI==0);
        TI = 0;
}

void main()
{
        unsigned char ch;

        P1=0xFF;

        InitSerial();

        Init_TRW24G();
        SetMode_TRW24(MODE_RX);

    while (1){

         while (!DR1){
           if (RI==1){
        //Init_TRW24G();
              SetMode_TRW24(MODE_TX);
              ch = GetChar();
              RI = 0;
              Send_TRW24(ch);
            //Init_TRW24G();
          SetMode_TRW24(MODE_RX);
            }
        }

          ch = Read_TRW24();
          SendChar(ch);
        }

}
```

ด้านฝั่ง Server

```
#include <regx52.h>

#define CE  P1_2
```

```
#define CS  P1_3
#define DAT P1_5
#define CLK P1_4
#define DR1 P1_6

#define MODE_TX   0
#define     MODE_RX 1


void Wait(unsigned char n)
{
      while (n--) {

      }
}

void Init_TRW24G(void)
{
      CE  = 0;
      CS  = 0;
      CLK = 0;
      DAT = 0;
      DR1 = 0;
      Wait(1);
}

void CLK_TRW24(void)
{
      CLK = 0;
      CLK = 1;
}

void Write_TRW24(unsigned char Data)
{
      unsigned char i;
      bit Out;
      for (i=0;i<8;i++)
      {
            Out = Data & 0x80;
            DAT = Out;
            CLK_TRW24();
            Data = Data << 1;
      }
}
unsigned char Read_TRW24(void)
{
      unsigned char i,Temp;
      bit Out;
      DAT = 1;
      for (i=0;i<8;i++)
      {
            Temp = Temp << 1;
            CLK = 1;
            Out = DAT;
            if (Out) { Temp = Temp + 0x01;}
            CLK = 0;
      }
      return(Temp);
```

```
}
void SetMode_TRW24( unsigned char Mode)
{
      Wait(1);
      CE = 0;
      CS = 1;
      Write_TRW24(0x8E);
      Write_TRW24(0x08);
      Write_TRW24(0x1C);

      Write_TRW24(0x08);
      Write_TRW24(0x08);

      Write_TRW24(0xD1);
      Write_TRW24(0xAA);
      Write_TRW24(0x55);
      Write_TRW24(0xAA);
      Write_TRW24(0x55);

      Write_TRW24(0xB5);
      Write_TRW24(0x55);
      Write_TRW24(0xAA);
      Write_TRW24(0x55);
      Write_TRW24(0xAA);

      Write_TRW24(0xA3);
      Write_TRW24(0x4F);
      Write_TRW24(0x0A+Mode);

      DAT = Mode;
      DR1 = Mode;
      CE = Mode;

      CS = 0;
      Wait(1);
}

void Send_TRW24(unsigned char Data)
{
      Wait(1);
      CS = 0;
      CE = 1;
      Write_TRW24(0xB5);
      Write_TRW24(0x55);
      Write_TRW24(0xAA);
      Write_TRW24(0x55);
      Write_TRW24(0xAA);

      Write_TRW24(Data);
      Wait(1);
      CLK = 0;
      CE = 0;
      Wait(1);
}

void InitSerial() {
      PCON=0x00;
      TMOD=0x20;
```

```
        TH1=0xFD;
        SCON=0x50;
        RI=0;
        TI=0;
        TR1=1;
}

unsigned char GetChar() {
        unsigned char ch;
        while (RI==0) ;
        ch = SBUF;
        RI = 0;
        return ch;
}

void SendChar(unsigned char ch) {
        SBUF = ch;
        while (TI==0) ;
        TI = 0;
}

void main()
{
        unsigned char ch;

        P1 = 0xFF;

        InitSerial();

        Init_TRW24G();
        SetMode_TRW24(MODE_RX);

    while (1) {

          while (!DR1) {
            if (RI==1) {
              Init_TRW24G();
                SetMode_TRW24(MODE_TX);
                ch = GetChar();
                RI = 0;
                Send_TRW24(ch);
                Init_TRW24G();
                SetMode_TRW24(MODE_RX);
              }
        }

          ch = Read_TRW24();
          SendChar(ch);
        }

}
```

โปรแกรมเครื่อง Sever

- โปรแกรมในหน้าForm1

```
unit mor1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, {CPortCtl, CPort,} StdCtrls, Mask, DBCtrls, DB, DBTables,
ComCtrls,
  ExtCtrls, CPortCtl, CPort;

type
  TForm1 = class(TForm)
    DataSource1: TDataSource;
    Query1: TQuery;
    DBfood: TDBEdit;
    DBprize: TDBEdit;
    btnconnect: TButton;
    btndisconnect: TButton;
    t1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    TabSheet5: TTabSheet;
    TabSheet6: TTabSheet;
    TabSheet7: TTabSheet;
    TabSheet8: TTabSheet;
    TabSheet9: TTabSheet;
    MmoFood1: TMemo;
    MmoPrize1: TMemo;
    MmoFood2: TMemo;
    MmoPrize2: TMemo;
    MmoFood3: TMemo;
    MmoPrize3: TMemo;
    MmoFood4: TMemo;
    MmoPrize4: TMemo;
    Timer1: TTimer;
    edtshow: TEdit;
    MmoFood5: TMemo;
    MmoPrize5: TMemo;
    MmoFood6: TMemo;
    MmoPrize6: TMemo;
    MmoFood7: TMemo;
    MmoPrize7: TMemo;
    MmoFood8: TMemo;
    MmoPrize8: TMemo;
    MmoFood9: TMemo;
    MmoPrize9: TMemo;
    Label10: TLabel;
    Label18: TLabel;
    Mmomany1: TMemo;
    Mmomany2: TMemo;
    Mmomany3: TMemo;
    Mmomany4: TMemo;
    Mmomany5: TMemo;
```

```
Mmomany6: TMemo;
Mmomany7: TMemo;
Mmomany8: TMemo;
Mmomany9: TMemo;
Label28: TLabel;
Edtpiz1: TEdit;
Edtpiz2: TEdit;
Edtpiz3: TEdit;
Edtpiz4: TEdit;
Edtpiz5: TEdit;
Edtpiz6: TEdit;
Edtpiz7: TEdit;
Edtpiz8: TEdit;
Edtpiz9: TEdit;
Label37: TLabel;
Label45: TLabel;
ComPort1: TComPort;
ComLed1: TComLed;
btnshowform2: TButton;
PrintDialog1: TPrintDialog;
Timer2: TTimer;
Label46: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label1: TLabel;
Label17: TLabel;
Label16: TLabel;
Label15: TLabel;
Label14: TLabel;
Label13: TLabel;
Label12: TLabel;
Label11: TLabel;
Label2: TLabel;
Label27: TLabel;
Label26: TLabel;
Label25: TLabel;
Label24: TLabel;
Label23: TLabel;
Label22: TLabel;
Label20: TLabel;
Label21: TLabel;
Label36: TLabel;
Label35: TLabel;
Label34: TLabel;
Label33: TLabel;
Label32: TLabel;
Label31: TLabel;
Label30: TLabel;
Label29: TLabel;
Label44: TLabel;
Label43: TLabel;
Label42: TLabel;
Label41: TLabel;
Label40: TLabel;
Label39: TLabel;
Label38: TLabel;
```

```
    Label19: TLabel;
    Label47: TLabel;
    Button1: TButton;
    edtclear: TEdit;
    Timer3: TTimer;

    procedure btnconnectClick(Sender: TObject);
    procedure btndisconnectClick(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure btnClick(Sender: TObject);
    procedure btnshowform2Click(Sender: TObject);
    procedure Timer2Timer(Sender: TObject);
    procedure ComPort1RxChar(Sender: TObject; Count: Integer);
    procedure Button1Click(Sender: TObject);
    procedure Timer3Timer(Sender: TObject);


  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  piz1,piz2,piz3,piz4,piz5,piz6,piz7,piz8,piz9:integer;
implementation
uses mor2;
{$R *.dfm}

procedure TForm1.btnconnectClick(Sender: TObject);
begin
    comport1.Open;
    comport1.Connected:=true;
end;

procedure TForm1.btndisconnectClick(Sender: TObject);
begin
  comport1.Close;
  comport1.Connected:=False;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var No,mm,Table,Food,m:string; i,j,k:integer;
  begin
  begin
    mm:= edtshow.Text;
    Table:=copy(mm,7,3);
    Food:=copy(mm,1,3);
    No:= copy(mm,4,3);
begin
    with Query1 do
     begin
     Close;
     ParamByName('mor').AsString:=Food;
     open;
      end;
      end;
if (Food='000') and (No='000') then
  begin
```

```
   if table='001' then
      begin
          tabsheet1.Show;
          edtshow.Clear;
          comport1.WriteStr('t');
          form2.RichEdit1.Lines.Add(' ');
          form2.RichEdit1.Lines.Add('********** ค่าอาหารทั้งหมด ' ' +
IntToStr(piz1) + '  บาท **********');
          form2.RichEdit1.Lines.Add(TimeToStr(Time));
          Mmofood1.Lines.Add( '******** ออกบิลใบเสร็จ ********');
          Mmomany1.Lines.Add('  ');
          Mmoprize1.Lines.Add('  ');
          PrintDialog1.Copies :=1;
              Form2.RichEdit1.Lines.SaveToFile('bill1.rtf');
              if PrintDialog1.Execute() = true then
              begin
                  Form2.RichEdit1.Print('bill1.rtf');
                end;
        end
      else
        if table='002' then
        begin
          tabsheet2.Show;
          edtshow.Clear;
          comport1.WriteStr('t');
          form2.RichEdit2.Lines.Add(' ');
          form2.RichEdit2.Lines.Add('********** ค่าอาหารทั้งหมด ' ' +
IntToStr(piz2) + '  บาท **********');
          form2.RichEdit2.Lines.Add(TimeToStr(Time));
          Mmofood2.Lines.Add( '******** ออกบิลใบเสร็จ ********');
          Mmomany2.Lines.Add('  ');
          Mmoprize2.Lines.Add('  ');
          PrintDialog1.Copies :-1;
              Form2.RichEdit2.Lines.SaveToFile('bill2.rtf');
              if PrintDialog1.Execute() = true then
              begin
                  Form2.RichEdit2.Print('bill2.rtf');
                end;
        end
       else
        if table='003' then
          begin
          tabsheet3.Show;
          edtshow.Clear;
          comport1.WriteStr('t');
          form2.RichEdit3.Lines.Add(' ');
          form2.RichEdit3.Lines.Add('********** ค่าอาหารทั้งหมด ' +
IntToStr(piz3) + '  บาท **********');
          form2.RichEdit3.Lines.Add(TimeToStr(Time));
          Mmofood3.Lines.Add( '******** ออกบิลใบเสร็จ ********');
          Mmomany3.Lines.Add('  ');
          Mmoprize3.Lines.Add('  ');
          PrintDialog1.Copies :=1;
          Form2.RichEdit3.Lines.SaveToFile('bill3.rtf');
              if PrintDialog1.Execute() = true then
```

```
            begin
                Form2.RichEdit3.Print('bill3.rtf');
              end;
        end
        else
    if table='004' then
        begin
        tabsheet4.Show;
        edtshow.Clear;
        comport1.WriteStr('t');
        form2.RichEdit4.Lines.Add(' ');
        form2.RichEdit4.Lines.Add('********** ค่าอาหารทั้งหมด ' +
IntToStr(piz4) + '  บาท **********');
        form2.RichEdit4.Lines.Add(TimeToStr(Time));
        Mmofood4.Lines.Add( '******** ออกบิลใบเสร็จ ********');
        Mmomany4.Lines.Add('  ');
        Mmoprize4.Lines.Add('  ');
        PrintDialog1.Copies := 1;
        Form2.RichEdit4.Lines.SaveToFile('bill4.rtf');
            if PrintDialog1.Execute() = true then
            begin
                Form2.RichEdit4.Print('bill4.rtf');
              end;
         end
        else
    if table='005' then
        begin
        tabsheet5.Show;
        edtshow.Clear;
        comport1.WriteStr('t');
        form2.RichEdit5.Lines.Add(' ');
        form2.RichEdit5.Lines.Add('********** ค่าอาหารทั้งหมด ' +
IntToStr(piz5) + '  บาท **********');
        form2.RichEdit5.Lines.Add(TimeToStr(Time));
        Mmofood5.Lines.Add( '******** ออกบิลใบเสร็จ ********');
        Mmomany5.Lines.Add('  ');
        Mmoprize5.Lines.Add('  ');
        PrintDialog1.Copies := 1;
        Form2.RichEdit5.Lines.SaveToFile('bill5.rtf');
            if PrintDialog1.Execute() = true then
            begin
                Form2.RichEdit5.Print('bill5.rtf');
              end;
         end
        else
    if table='006' then
        begin
        tabsheet6.Show;
        edtshow.Clear;
        comport1.WriteStr('t');
        form2.RichEdit6.Lines.Add(' ');
        form2.RichEdit6.Lines.Add('********** ค่าอาหารทั้งหมด ' +
IntToStr(piz6) + '  บาท **********');
        form2.RichEdit6.Lines.Add(TimeToStr(Time));
        Mmofood6.Lines.Add( '******** ออกบิลใบเสร็จ ********');
```

```
            Mmomany6.Lines.Add('  ');
            Mmoprize6.Lines.Add('  ');
            PrintDialog1.Copies :=1;
            Form2.RichEdit6.Lines.SaveToFile('bill6.rtf');
                if PrintDialog1.Execute() = true then
                begin
                    Form2.RichEdit6.Print('bill6.rtf');
                  end;
            end
            else
        if table='007' then
            begin
            tabsheet7.Show;
            edtshow.Clear;
            comport1.WriteStr('t');
            form2.RichEdit7.Lines.Add(' ');
            form2.RichEdit7.Lines.Add('********** ค่าอาหารทั้งหมด ' +
IntToStr(piz7) + '  บาท **********');
            form2.RichEdit7.Lines.Add(TimeToStr(Time));
            PrintDialog1.Copies :=1;
            Mmofood7.Lines.Add( '******** ออกบิลใบเสร็จ ********');
            Mmomany7.Lines.Add('  ');
            Mmoprize7.Lines.Add('  ');
            Form2.RichEdit7.Lines.SaveToFile('bill7.rtf');
                if PrintDialog1.Execute() = true then
                begin
                    Form2.RichEdit7.Print('bill7.rtf');
                  end;
             end
            else
        if table='008' then
            begin
            tabsheet8.Show;
            edtshow.Clear;
            comport1.WriteStr('t');
            form2.RichEdit8.Lines.Add(' ');
            form2.RichEdit8.Lines.Add('********** ค่าอาหารทั้งหมด ' +
IntToStr(piz8) + '  บาท **********');
            form2.RichEdit8.Lines.Add(TimeToStr(Time));
            Mmofood8.Lines.Add( '******** ออกบิลใบเสร็จ ********');
            Mmomany8.Lines.Add('  ');
            Mmoprize8.Lines.Add('  ');
            PrintDialog1.Copies :=1;
            Form2.RichEdit8.Lines.SaveToFile('bill8.rtf');
                if PrintDialog1.Execute() = true then
                begin
                    Form2.RichEdit8.Print('bill8.rtf');
                  end;
               end
            else
        if table='009' then
            begin
            tabsheet9.Show;
            edtshow.Clear;
            comport1.WriteStr('t');
            form2.RichEdit9.Lines.Add(' ');
```

```
          form2.RichEdit9.Lines.Add('********** ค่าอาหารทั้งหมด ' +
IntToStr(piz9) + '  บาท **********');
          form2.RichEdit9.Lines.Add(TimeToStr(Time));
          Mmofood9.Lines.Add( '******** ออกบิลใบเสร็จ ********');
          Mmomany9.Lines.Add('  ');
          Mmoprize9.Lines.Add('  ');
          PrintDialog1.Copies :=1;
          Form2.RichEdit9.Lines.SaveToFile('bill9.rtf');
              if PrintDialog1.Execute() = true then
              begin
                  Form2.RichEdit9.Print('bill9.rtf');
               end;
             end;
     end
  else
  if No='000' then
  begin
   if Table='001' then
       begin
        tabsheet1.Show;
        for i:= 0 to 50 do
         begin
           if MmoFood1.Lines[i]=DBfood.Text then
           begin
              MmoFood1.Lines.Delete(i);
              MmoPrize1.Lines.Delete(i);
              piz1 := piz1 -
StrToInt(copy(MmoMany1.Lines[i],1,50))*StrToInt(DBprize.Text);
              edtpiz1.Text:=IntToStr(piz1);
              MmoMany1.Lines.Delete(i);
              form2.RichEdit1.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
              comport1.WriteStr('c');
         end;
         end;
         end
       else
  if Table='002' then
  begin
        tabsheet2.Show;
        for i:= 0 to 50 do
         begin
           if MmoFood2.Lines[i]=DBfood.Text then
           begin
              MmoFood2.Lines.Delete(i);
              MmoPrize2.Lines.Delete(i);
              piz2 := piz2 -
StrToInt(copy(MmoMany2.Lines[i],1,50))*StrToInt(DBprize.Text);
              edtpiz2.Text:=IntToStr(piz2);
              MmoMany2.Lines.Delete(i);
              form2.RichEdit2.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
              comport1.WriteStr('c');
              end;
         end;
         end
    else
    if Table='003' then
```

```
    begin
         tabsheet3.Show;
         for i:= 0 to 50 do
          begin
            if MmoFood3.Lines[i]=DBfood.Text then
            begin
               MmoFood3.Lines.Delete(i);
               MmoPrize3.Lines.Delete(i);
               piz3 := piz3 -
StrToInt(copy(MmoMany3.Lines[i],1,50))*StrToInt(DBprize.Text);
               edtpiz3.Text:=IntToStr(piz3);
               MmoMany3.Lines.Delete(i);
               form2.RichEdit3.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
               comport1.WriteStr('c')
          end;
          end;
          end
    else
       if Table='004' then
       begin
         tabsheet4.Show;
         for i:= 0 to 50 do
          begin
            if MmoFood4.Lines[i]=DBfood.Text then
            begin
               MmoFood4.Lines.Delete(i);
               MmoPrize4.Lines.Delete(i);
               piz4 := piz4 -
StrToInt(copy(MmoMany7.Lines[i],1,50))*StrToInt(DBprize.Text);
               edtpiz4.Text:=IntToStr(piz4);
               MmoMany4.Lines.Delete(i);
               form2.RichEdit4.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
               comport1.WriteStr('c');
          end;
          end;
          end
    else
       if Table='005' then
       begin
         tabsheet5.Show;
        for i:= 0 to 50 do
          begin
            if MmoFood5.Lines[i]=DBfood.Text then
            begin
               MmoFood5.Lines.Delete(i);
               MmoPrize5.Lines.Delete(i);
               piz5 := piz5 -
StrToInt(copy(MmoMany5.Lines[i],1,50))*StrToInt(DBprize.Text);
               edtpiz5.Text:=IntToStr(piz5);
               MmoMany5.Lines.Delete(i);
               form2.RichEdit5.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
               comport1.WriteStr('c');
          end;
          end;
          end
    else
```

```
       if Table='006' then
       begin
         tabsheet6.Show;
         for i:= 0 to 50 do
          begin
            if MmoFood6.Lines[i]=DBfood.Text then
            begin
               MmoFood6.Lines.Delete(i);
               MmoPrize6.Lines.Delete(i);
               piz6 := piz6 -
StrToInt(copy(MmoMany6.Lines[i],1,50))*StrToInt(DBprize.Text);
               edtpiz6.Text:=IntToStr(piz6);
               MmoMany6.Lines.Delete(i);
               form2.RichEdit6.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
               comport1.WriteStr('c');
          end;
          end;
          end
    else
       if Table='007' then
       begin
         tabsheet7.Show;
       for i:= 0 to 50 do
          begin
            if MmoFood7.Lines[i]=DBfood.Text then
            begin
               MmoFood7.Lines.Delete(i);
               MmoPrize7.Lines.Delete(i);
               piz7 := piz7 -
StrToInt(copy(MmoMany7.Lines[i],1,50))*StrToInt(DBprize.Text);
               edtpiz7.Text:=IntToStr(piz7);
               MmoMany7.Lines.Delete(i);
               form2.RichEdit7.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
               comport1.WriteStr('c');
          end;
          end;
          end
       else
        if Table='008' then
       begin
         tabsheet8.Show;
        for i:= 0 to 50 do
          begin
            if MmoFood8.Lines[i]=DBfood.Text then
            begin
               MmoFood8.Lines.Delete(i);
               MmoPrize8.Lines.Delete(i);
               piz8 := piz8 -
StrToInt(copy(MmoMany8.Lines[i],1,50))*StrToInt(DBprize.Text);
               edtpiz8.Text:=IntToStr(piz8);
               MmoMany8.Lines.Delete(i);
               form2.RichEdit8.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
               comport1.WriteStr('c');
          end;
          end;
          end
```

```
         else
        if Table='009' then
        begin
          tabsheet9.Show;
          for i:= 0 to 50 do
           begin
             if MmoFood9.Lines[i]=DBfood.Text then
             begin
                MmoFood9.Lines.Delete(i);
                MmoPrize9.Lines.Delete(i);
                piz9 := piz9 -
StrToInt(copy(MmoMany9.Lines[i],1,50))*StrToInt(DBprize.Text);
                edtpiz9.Text:=IntToStr(piz9);
                MmoMany9.Lines.Delete(i);
                form2.RichEdit9.Lines.Add('                ยกเลิกรายการ' + '
' + DBfood.Text  );
                comport1.WriteStr('c');
            end;
            end;
            end;
            end
          else
begin
if Table='001' then
   begin
        tabsheet1.Show;
        MmoFood1.Lines.Add(DBfood.Text);
        MmoPrize1.Lines.Add(DBprize.Text);
        form2.RichEdit1.Lines.Add(' '      ' + DBprize.Text + '  บาท'
+ '              ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany1.Lines.Add((IntToStr(StrToInt(No))));
        piz1:= piz1 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz1.Text:=IntToStr(piz1);
        comport1.WriteStr('r');
       end
   else
        if Table='002' then
   begin
        tabsheet2.Show;
        MmoFood2.Lines.Add(DBfood.Text);
        MmoPrize2.Lines.Add(DBprize.Text);
        form2.RichEdit2.Lines.Add(' '      ' + DBprize.Text + '  บาท'
+ '              ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany2.Lines.Add((IntToStr(StrToInt(No))));
        piz2:= piz2 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz2.Text:=IntToStr(piz2);
        comport1.WriteStr('r');
   end
     else
        if Table='003' then
   begin
        tabsheet3.Show;
        MmoFood3.Lines.Add(DBfood.Text);
        MmoPrize3.Lines.Add(DBprize.Text);
```

```
        form2.RichEdit3.Lines.Add( '        ' + DBprize.Text + '  บาท'
+ '             ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany3.Lines.Add((IntToStr(StrToInt(No))));
        piz3:= piz3 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz3.Text:=IntToStr(piz3);
        comport1.WriteStr('r');
    end
    else
       if Table='004' then
    begin
        tabsheet4.Show;
        MmoFood4.Lines.Add(DBfood.Text);
        MmoPrize4.Lines.Add(DBprize.Text);
        form2.RichEdit4.Lines.Add( '        ' + DBprize.Text + '  บาท'
+ '             ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany4.Lines.Add((IntToStr(StrToInt(No))));
        piz4:= piz4 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz4.Text:=IntToStr(piz4);
        comport1.WriteStr('r');
          end
      else
         if Table='005' then
    begin
        tabsheet5.Show;
        MmoFood5.Lines.Add(DBfood.Text);
        MmoPrize5.Lines.Add(DBprize.Text);
        form2.RichEdit5.Lines.Add( '        ' + DBprize.Text + '  บาท'
+ '             ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany5.Lines.Add((IntToStr(StrToInt(No))));
        piz5:= piz5 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz5.Text:=IntToStr(piz5);
        comport1.WriteStr('r');
        end
      else
         if Table='006' then
    begin
        tabsheet6.Show;
        MmoFood6.Lines.Add(DBfood.Text);
        MmoPrize6.Lines.Add(DBprize.Text);
        form2.RichEdit6.Lines.Add( '        ' + DBprize.Text + '  บาท'
+ '             ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany6.Lines.Add((IntToStr(StrToInt(No))));
        piz6:= piz6 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz6.Text:=IntToStr(piz6);
        comport1.WriteStr('r');
    end
      else
         if Table='007' then
    begin
        tabsheet7.Show;
        MmoFood7.Lines.Add(DBfood.Text);
        MmoPrize7.Lines.Add(DBprize.Text);
```

```
        form2.RichEdit7.Lines.Add( '        ' + DBprize.Text + '  บาท'
+ '              ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany7.Lines.Add((IntToStr(StrToInt(No))));
        piz7:= piz7 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz7.Text:=IntToStr(piz7);
        comport1.WriteStr('r');
    end
       else
       if Table='008' then
    begin
        tabsheet8.Show;
        MmoFood8.Lines.Add(DBfood.Text);
        MmoPrize8.Lines.Add(DBprize.Text);
        form2.RichEdit8.Lines.Add( '        ' + DBprize.Text + '  บาท'
+ '              ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany8.Lines.Add((IntToStr(StrToInt(No))));
        piz8:= piz8 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz8.Text:=IntToStr(piz8);
        comport1.WriteStr('r');
        end
       else
        if Table='009' then
    begin
        tabsheet9.Show;
        MmoFood9.Lines.Add(DBfood.Text);
        MmoPrize9.Lines.Add(DBprize.Text);
        form2.RichEdit9.Lines.Add( '        ' + DBprize.Text + '  บาท'
+ '              ' + (inttostr(strtoint(No))) + '
' +DBfood.Text  ) ;
        Mmomany9.Lines.Add((IntToStr(StrToInt(No))));
        piz9:= piz9 + (StrToInt(No)*StrToInt(DBprize.Text));
        edtpiz9.Text:=IntToStr(piz9);
        comport1.WriteStr('r')
    end;
    end;
    end;
     edtshow.Clear;
     Timer1.Enabled:=False;
    end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    timer1.Enabled:=false;
    dbfood.Enabled:=false;
    dbprize.Enabled:=false;
    MmoFood1.Enabled:=False;
    MmoFood2.Enabled:=False;
    MmoFood3.Enabled:=False;
    MmoFood4.Enabled:=False;
    MmoFood5.Enabled:=False;
    MmoFood6.Enabled:=False;
    MmoFood7.Enabled:=False;
    MmoFood8.Enabled:=False;
    MmoFood9.Enabled:=False;
    Mmomany1.Enabled:=False;
    Mmomany2.Enabled:=False;
    Mmomany3.Enabled:=False;
```

```
    Mmomany4.Enabled:=False;
    Mmomany5.Enabled:=False;
    Mmomany6.Enabled:=False;
    Mmomany7.Enabled:=False;
    Mmomany8.Enabled:=False;
    Mmomany9.Enabled:=False;
    MmoPrize1.Enabled:=False;
    MmoPrize2.Enabled:=False;
    MmoPrize3.Enabled:=False;
    MmoPrize4.Enabled:=False;
    MmoPrize5.Enabled:=False;
    MmoPrize6.Enabled:=False;
    MmoPrize7.Enabled:=False;
    MmoPrize8.Enabled:=False;
    MmoPrize9.Enabled:=False;
    edtpiz1.Enabled:=False;
    edtpiz2.Enabled:=False;
    edtpiz3.Enabled:=False;
    edtpiz4.Enabled:=False;
    edtpiz5.Enabled:=False;
    edtpiz6.Enabled:=False;
    edtpiz7.Enabled:=False;
    edtpiz8.Enabled:=False;
    edtpiz9.Enabled:=False;
end;

procedure TForm1.btnClick(Sender: TObject);
begin
    timer1.Enabled:=True;
end;

procedure TForm1.btnshowform2Click(Sender: TObject);
begin
    form2.ShowModal;
end;

procedure TForm1.Timer2Timer(Sender: TObject);
begin
    Label46.Caption:=TimeToStr(Time);
end;

procedure TForm1.ComPort1RxChar(Sender: TObject; Count: Integer);
var data:string;
begin
     comport1.ReadStr(data,count);
     edtshow.Text:=edtshow.Text+data;
    if Edtshow.GetTextLen=9 then
    begin
        timer3.Enabled:=true;
    end;
    end;

procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
    if edtclear.Text='001' then
       begin
         tabsheet1.Show;
         Mmofood1.Clear;
         Mmomany1.Clear;
         Mmoprize1.Clear;
```

```
        edtpiz1.Clear;
        form2.RichEdit1.Clear;
        form2.RichEdit1.Lines.Add('  ');
        form2.RichEdit1.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 1');
        form2.RichEdit1.Lines.Add('  ');
        form2.RichEdit1.Lines.Add('  ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz1:=0;
      end
   else
    if edtclear.Text='002' then
      begin
        tabsheet2.Show;
        Mmofood2.Clear;
        Mmomany2.Clear;
        Mmoprize2.Clear;
        edtpiz2.Clear;
        form2.RichEdit2.Clear;
        form2.RichEdit2.Lines.Add('  ');
        form2.RichEdit2.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 2');
        form2.RichEdit2.Lines.Add('  ');
        form2.RichEdit2.Lines.Add('  ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz2:=0;
      end
    else
      if edtclear.Text='003' then
      begin
        tabsheet3.Show;
        Mmofood3.Clear;
        Mmomany3.Clear;
        Mmoprize3.Clear;
        edtpiz3.Clear;
        form2.RichEdit3.Clear;
        form2.RichEdit3.Lines.Add('  ');
        form2.RichEdit3.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 3');
        form2.RichEdit3.Lines.Add('  ');
        form2.RichEdit3.Lines.Add('  ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz3:=0;
      end
      else
      if edtclear.Text='004' then
      begin
        tabsheet4.Show;
        Mmofood4.Clear;
        Mmomany4.Clear;
        Mmoprize4.Clear;
        edtpiz4.Clear;
        form2.RichEdit4.Clear;
```

```
        form2.RichEdit4.Lines.Add('   ');
        form2.RichEdit4.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 4');
        form2.RichEdit4.Lines.Add('   ');
        form2.RichEdit4.Lines.Add('   ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz4:=0;
      end
      else
      if edtclear.Text='005' then
      begin
        tabsheet5.Show;
        Mmofood5.Clear;
        Mmomany5.Clear;
        Mmoprize5.Clear;
        edtpiz5.Clear;
        form2.RichEdit5.Clear;
        form2.RichEdit5.Lines.Add('   ');
        form2.RichEdit5.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 5');
        form2.RichEdit5.Lines.Add('   ');
        form2.RichEdit5.Lines.Add('   ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz5:=5;
      end
      else
      if edtclear.Text='006' then
      begin
        tabsheet6.Show;
        Mmofood6.Clear;
        Mmomany6.Clear;
        Mmoprize6.Clear;
        edtpiz6.Clear;
        form2.RichEdit6.Clear;
        form2.RichEdit6.Lines.Add('   ');
        form2.RichEdit6.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 6');
        form2.RichEdit6.Lines.Add('   ');
        form2.RichEdit6.Lines.Add('   ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz6:=6;
      end
      else
      if edtclear.Text='007' then
      begin
        tabsheet7.Show;
        Mmofood7.Clear;
        Mmomany7.Clear;
        Mmoprize7.Clear;
        edtpiz7.Clear;
        form2.RichEdit7.Clear;
        form2.RichEdit7.Lines.Add('   ');
```

```
        form2.RichEdit7.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 7');
        form2.RichEdit7.Lines.Add('  ');
        form2.RichEdit7.Lines.Add('  ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz7:=0;
      end
      else
      if edtclear.Text='008' then
      begin
        tabsheet8.Show;
        Mmofood8.Clear;
        Mmomany8.Clear;
        Mmoprize8.Clear;
        edtpiz8.Clear;
        form2.RichEdit8.Clear;
        form2.RichEdit8.Lines.Add('  ');
        form2.RichEdit8.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 8');
        form2.RichEdit8.Lines.Add('  ');
        form2.RichEdit8.Lines.Add('  ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz8:=0;
      end
      else
      if edtclear.Text='009' then
      begin
        tabsheet9.Show;
        Mmofood9.Clear;
        Mmomany9.Clear;
        Mmoprize9.Clear;
        edtpiz9.Clear;
        form2.RichEdit9.Clear;
        form2.RichEdit9.Lines.Add('  ');
        form2.RichEdit9.Lines.Add('                                    ใบเสร็จค่าอาหาร
โต๊ะ 9');
        form2.RichEdit9.Lines.Add('  ');
        form2.RichEdit9.Lines.Add('  ราคาต่อหน่วย      จำนวนที่สั่ง
รายการอาหาร');
        piz9:=0;
      end;
      edtclear.Clear;
end;

procedure TForm1.Timer3Timer(Sender: TObject);
var food:string;
begin
      if Edtshow.GetTextLen=9 then
        begin
        Food:=copy(edtshow.Text,1,3);
        if strtoint(food) > 50 then
        begin
          edtshow.Clear;
```

```
            end;
          timer1.Enabled:=true;
          end;
      end;
end.
```

- โปรแกรมในหน้าForm2

```
unit mor2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls;

type
  TForm2 = class(TForm)
    RichEdit1: TRichEdit;
    RichEdit2: TRichEdit;
    RichEdit3: TRichEdit;
    RichEdit4: TRichEdit;
    RichEdit5: TRichEdit;
    RichEdit6: TRichEdit;
    RichEdit7: TRichEdit;
    RichEdit8: TRichEdit;
    RichEdit9: TRichEdit;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;
```

```
implementation
 uses mor1;
{$R *.dfm}

procedure TForm2.FormCreate(Sender: TObject);
begin
form2.RichEdit1.Enabled:=False;
  form2.RichEdit2.Enabled:=False;
  form2.RichEdit3.Enabled:=False;
  form2.RichEdit4.Enabled:=False;
  form2.RichEdit5.Enabled:=False;
  form2.RichEdit6.Enabled:=False;
  form2.RichEdit7.Enabled:=False;
  form2.RichEdit8.Enabled:=False;
  form2.RichEdit9.Enabled:=False;
end;

end.
```

Conditions: VDD = +3V, VSS = 0V, T_A = -40°C to +85°C

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
	Operating conditions					
VDD	Supply voltage		1.9	3.0	3.6	V
TEMP	Operating Temperature		-40	+27	+85	°C
	Digital input pin					
V_{IH}	HIGH level input voltage		VDD-0.3		VDD	V
V_{IL}	LOW level input voltage		Vss		0.3	V
	Digital output pin					
V_{OH}	HIGH level output voltage (I_{OH}=-0.5mA)		VDD-0.3		VDD	V
V_{OL}	LOW level output voltage (I_{OL}=0.5mA)		Vss		0.3	V
	General RF conditions					
f_{OP}	Operating frequency	1)	2400		2524	MHz
Δf	Frequency deviation			±156		kHz
R_{GFSK}	Data rate ShockBurst™		250		1000	kbps
$F_{CHANNEL}$	Channel spacing			1		MHz
	Transmitter operation					
P_{RF}	Maximum Output Power	4)		0	+4	dBm
P_{RFC}	RF Power Control Range		16	20		dB
P_{RFCR}	RF Power Control Range Resolution				±3	dB
P_{BW}	20dB Bandwidth for Modulated Carrier				1000	kHz
P_{RF2}	2nd Adjacent Channel Transmit Power 2MHz				-20	dBm
P_{RF3}	3rd Adjacent Channel Transmit Power 3MHz				-40	dBm
I_{VDD}	Supply current @ 0dBm output power	5)		13		mA
I_{VDD}	Supply current @ -20dBm output power	5)		8.8		mA
I_{VDD}	Average Supply current @ -5dBm output power, ShockBurst™	6)		0.8		mA
I_{VDD}	Average Supply current in stand-by mode	7)		12		uA
I_{VDD}	Average Supply current in power down			1		uA
	Receiver operation					
I_{VDD}	Supply current one channel 250kbps			18		mA
I_{VDD}	Supply current one channel 1000kbps			19		mA
I_{VDD}	Supply current two channels 250kbps			23		mA
I_{VDD}	Supply current two channels 1000kbps			25		mA
RX_{SENS}	Sensitivity at 0.1%BER (@250kbps)			-90		dBm
RX_{SENS}	Sensitivity at 0.1%BER (@1000kbps)			-80		dBm
C/I_{CO}	C/I Co-channel			6		dB
C/I_{1ST}	1st Adjacent Channel Selectivity C/I 1MHz			-1		dB
C/I_{2ND}	2nd Adjacent Channel Selectivity C/I 2MHz			-16		dB
C/I_{3RD}	3rd Adjacent Channel Selectivity C/I 3MHz			-26		dB
RX_B	Blocking Data Channel 2			-41		dB

ShockBurst™

The ShockBurst™ technology uses on-chip FIFO to clock in data at a low data rate and transmit at a very high rate thus enabling extremely power reduction.
When operating the TRW-2.4G in ShockBurst™, you gain access to the high data rates (1 Mbps) offered by the 2.4 GHz band without the need of a costly, high-speed micro controller (MCU) for data processing.
By putting all high speed signal processing related to RF protocol on-chip, the TRW-2.4G offers the following benefits:

- Highly reduced current consumption
- Lower system cost (facilitates use of less expensive micro controller)
- Greatly reduced risk of 'on-air' collisions due to short transmission time

The TRW-2.4G can be programmed using a simple 3-wire interface where the data rate is decided by the speed of the micro controller.
By allowing the digital part of the application to run at low speed while maximizing the data rate on the RF link, the nRF ShockBurst™ mode reduces the average current consumption in applications considerably.

ShockBurst™ principle

When the TRW-2.4G is configured in ShockBurst™, TX or RX operation is conducted in the following way (10 kbps for the example only).

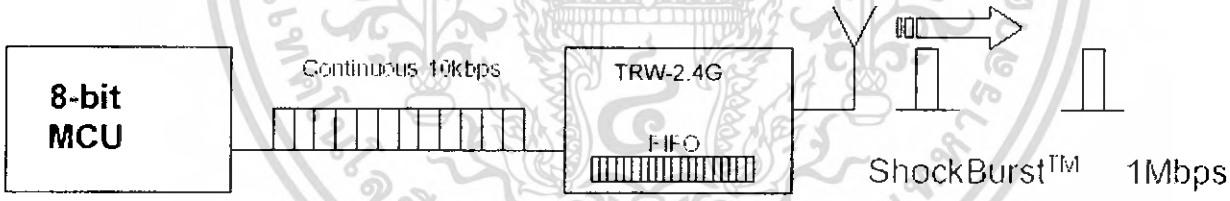


Figure 4 Clocking in data with MCU and sending with ShockBurst™ technology

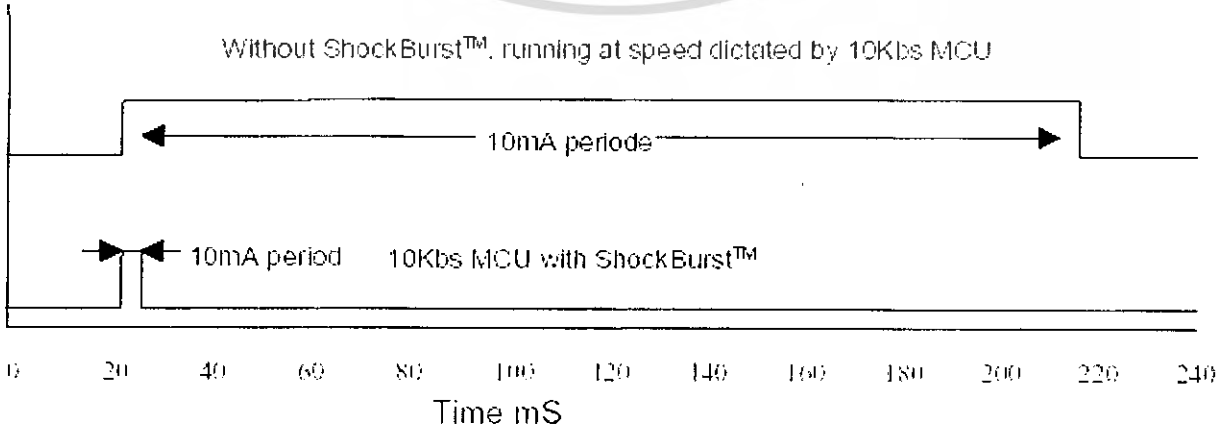


Figure 5 Current consumption with & without ShockBurst™ technology

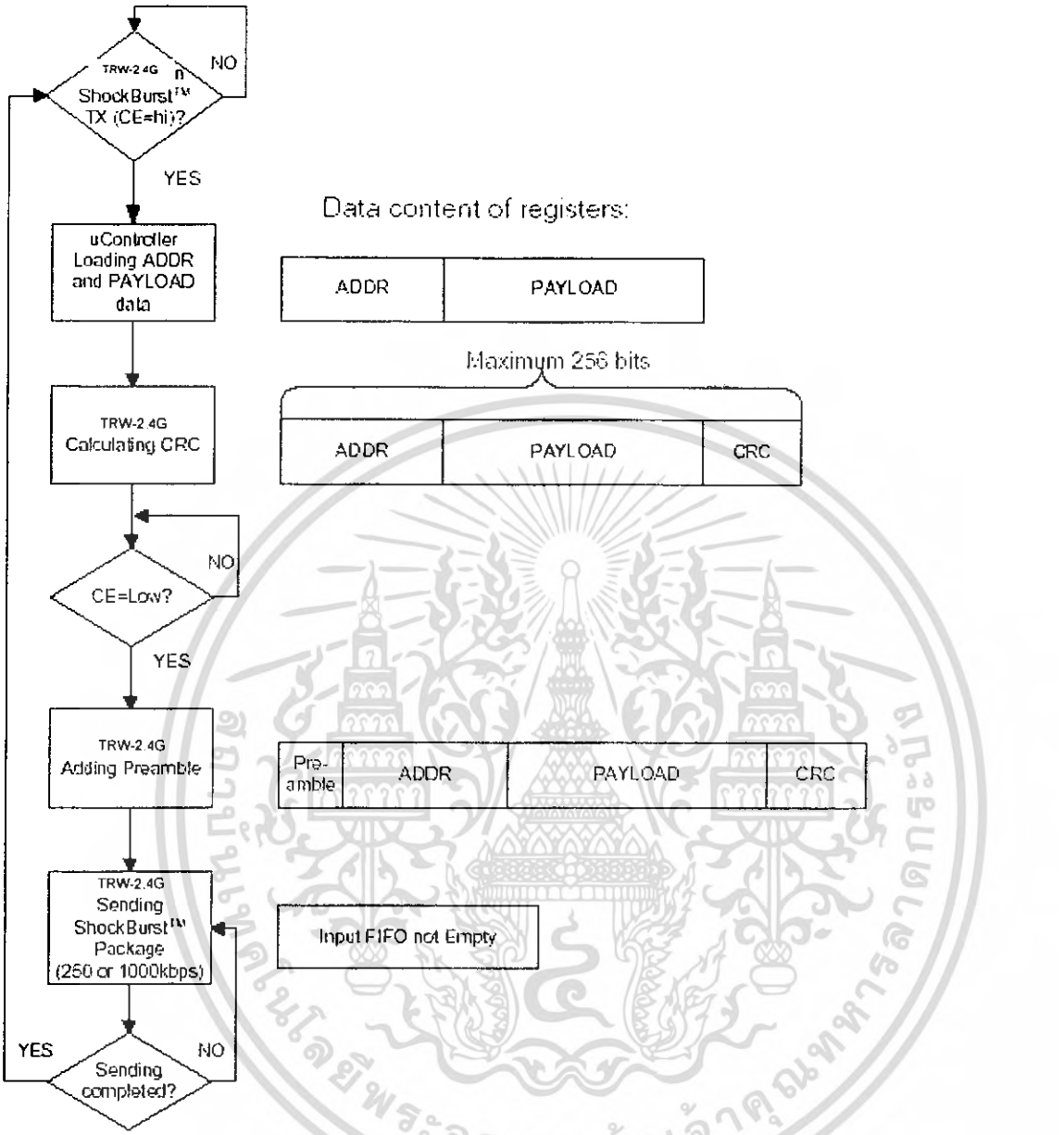


Figure 2 Flow Chart ShockBurst™ Transmit of TRW-2.4G

nRF2401 ShockBurst™ Transmit:

MCU interface pins: CE, CLK1, DATA

1. When the application MCU has data to send, set CE high. This activates TRW-2.4G on-board data processing.
2. The address of the receiving node (RX address) and payload data is clocked into the TRW-2.4G . The application protocol or MCU sets the speed <1Mbps (ex: 10kbps).
3. MCU sets CE low, this activates a TRW-2.4G ShockBurst™ transmission.
4. TRW-2.4G ShockBurst™:
   - RF front end is powered up
   - RF package is completed (preamble added, CRC calculated)
   - Data is transmitted at high speed (250 kbps or 1 Mbps configured by user).
   - TRW-2.4G return to stand-by when finished

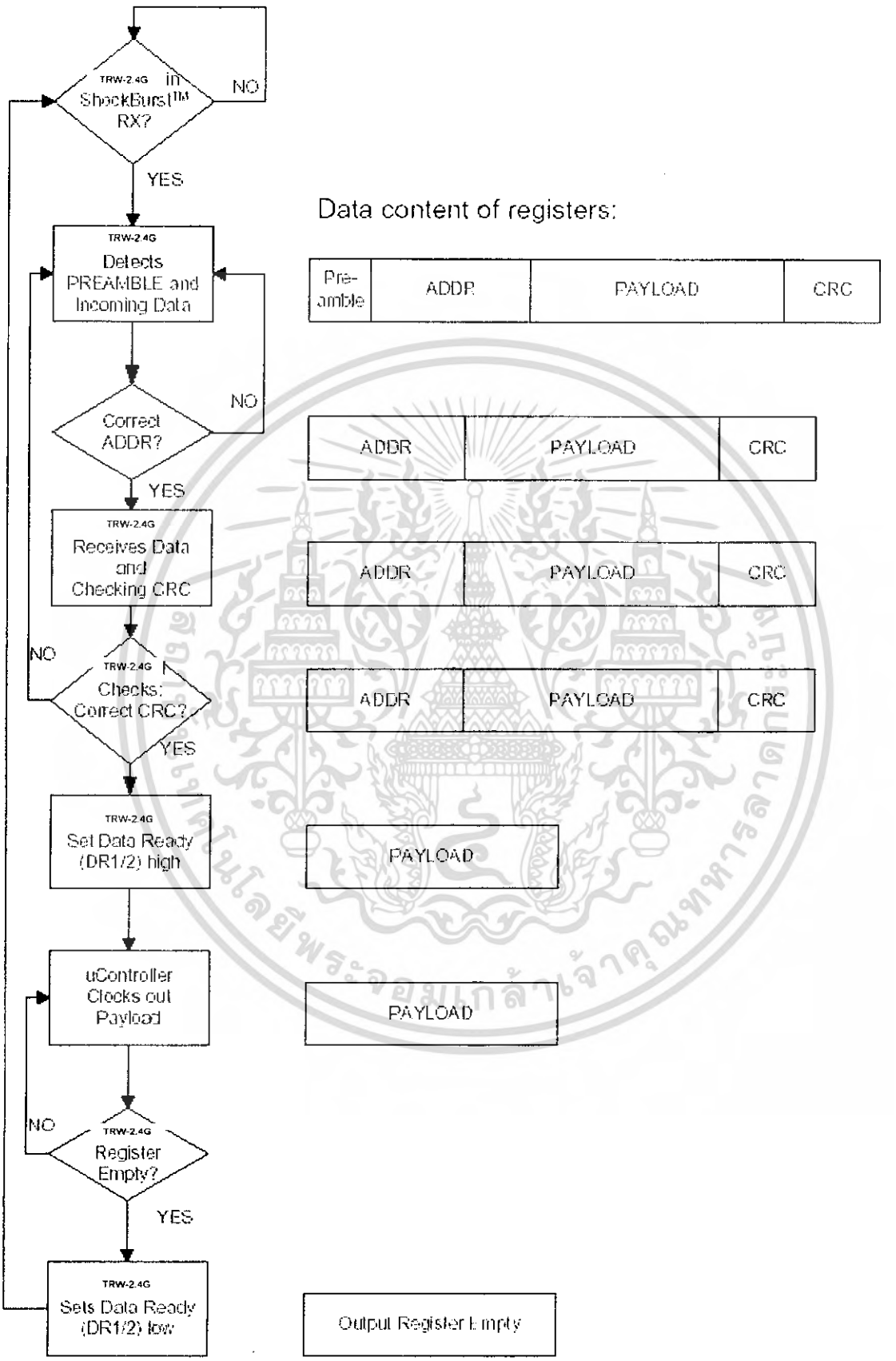


Figure 3 Flow Chart ShockBurst™ Receive of TRW-2.4G

TRW-2.4G ShockBurst™ Receive:

MCU interface pins: CE, DR1, CLK1 and DATA (one RX channel receive)

1. Correct address and size of payload of incoming RF packages are set when TRW-2.4G is configured to ShockBurst™ RX.
2. To activate RX, set CE high.
3. After 200 µs settling, TRW-2.4G is monitoring the air for incoming communication.
4. When a valid package has been received (correct address and CRC found), TRW-2.4G removes the preamble, address and CRC bits.
5. TRW-2.4G then notifies (interrupts) the MCU by setting the DR1 pin high.
6. MCU may (or may not) set the CE low to disable the RF front end (low current mode).
7. The MCU will clock out just the payload data at a suitable rate (ex. 10 kbps).
8. When all payload data is retrieved TRW-2.4G sets DR1 low again, and is ready for new incoming data package if CE is kept high during data download. If the CE was set low, a new start up sequence can begin, see Figure 12

DuoCeiver™ Simultaneous Two Channel Receive Mode

In both ShockBurst™ and direct modes the TRW-2.4G can facilitate simultaneous reception of two parallel independent frequency channels at the maximum data rate. This means:

- TRW-2.4G can receive data from two 1 Mbps transmitters (ex: TRW-2.4G or TRW-2.4G_) 8 MHz (8 frequency channels) apart through one antenna interface.
- The output from the two data channels is fed to two separate MCU interfaces.
  - Data channel 1: CLK1, DATA, and DR1
  - Data channel 2: CLK2, DOUT2, and DR2
  - DR1 and DR2 are available only in ShockBurst™.

The TRW-2.4G DuoCeiver™ technology provides 2 separate dedicated data channels for RX and replaces the need for two, stand alone receiver systems.

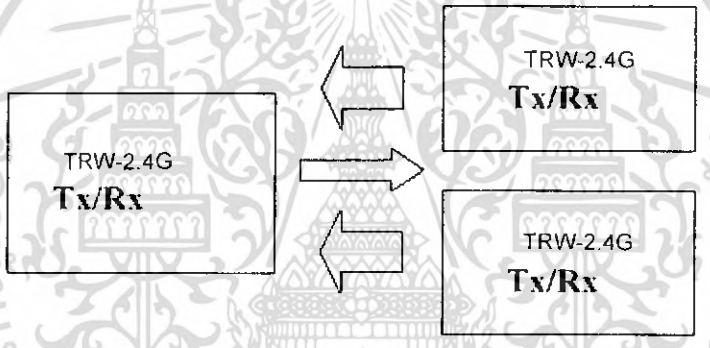


Figure 4 Simultaneous 2 channel receive on TRW-2.4G

There is one absolute requirement for using the second data channel. For the TRW-2.4G to be able to receive at the second data channel the frequency channel must be 8MHz higher than the frequency of data channel 1. The TRW-2.4G must be programmed to receive at the frequency of data channel 1. No time multiplexing is used in TRW-2.4G to fulfil this function. In direct mode the MCU must be able to handle two simultaneously incoming data packets if it is not multiplexing between the two data channels. In ShockBurst™ it is possible for the MCU to clock out one data channel at a time while data on the other data channel waits for MCU availability, without any lost data packets, and by doing so reduce the needed performance of the MCU.

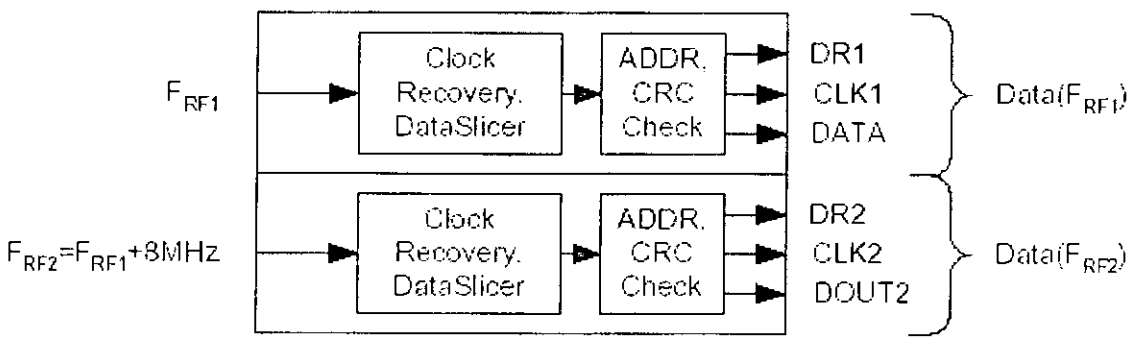


Figure 5 DuoCeiver™ with two simultaneously independent receive channels.

DEVICE CONFIGURATION

All configuration of the TRW-2.4G is done via a 3-wire interface to a single configuration register. The configuration word can be up to 15 bytes long for ShockBurst™

Configuration for ShockBurst™ operation

The configuration word in ShockBurst™ enables the TRW-2.4G to handle the RF protocol. Once the protocol is completed and loaded into TRW-2.4G only one byte, bit[7:0], needs to be updated during actual operation.

The configuration blocks dedicated to ShockBurst™ is as follows:

- Payload section width: Specifies the number of payload bits in a RF package. This enables the TRW-2.4G to distinguish between payload data and the CRC bytes in a received package.
- Address width: Sets the number of bits used for address in the RF package. This enables the TRW-2.4G to distinguish between address and payload data.
- Address (RX Channel 1 and 2): Destination address for received data.
- CRC: Enables nRF2401 on-chip CRC generation and de-coding.

NOTE:
These configuration blocks, with the exception of the CRC, are dedicated for the packages that a TRW-2.4G is to receive.
In TX mode, the MCU must generate an address and a payload section that fits the configuration of the TRW-2.4G that is to receive the data.
When using the TRW-2.4G on-chip CRC feature ensure that CRC is enabled and uses the same length for both the TX and RX devices.

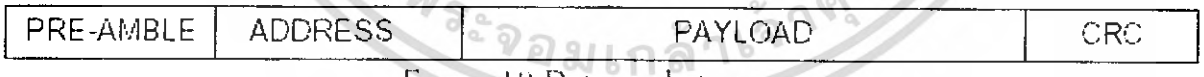


Figure 10 Data packet set-up

Configuration Word overview

	Bit position	Number of bits	Name	Function
ShockBurst™ configuration	143:120	24	TEST	Reserved for testing
	119:112	8	DATA2_W	Length of data payload section RX channel 2
	111:104	8	DATA1_W	Length of data payload section RX channel 1
	103:64	40	ADDR2	Up to 5 byte address for RX channel 2
	63:24	40	ADDR1	Up to 5 byte address for RX channel 1
	23:18	6	ADDR_W	Number of address bits (both RX channels).
	17	1	CRC_L	8 or 16 bit CRC
	16	1	CRC_EN	Enable on-chip CRC generation, checking.
General device configuration	15	1	RX2_EN	Enable two channel receive mode
	14	1	CM	Communication mode (Direct or ShockBurst™)
	13	1	RFDR_SB	RF data rate (1Mbps requires 16MHz crystal)
	12:10	3	XO_F	Crystal frequency
	9:8	2	RF_PWR	RF output power
	7:1	7	RF_CH#	Frequency channel
	0	1	RXEN	RX or TX operation

Table 1 Table of configuration words.

The configuration word is shifted in MSB first on positive CLK1 edges. New configuration is enabled on the falling edge of CS.

NOTE.
On the falling edge of CS, the TRW-2.4G updates the number of bits actually shifted in during the last configuration.
Ex:
If the TRW-2.4G is to be configured for 2 channel RX in ShockBurst™, a total of 120 bits must be shifted in during the first configuration after VDD is applied.
Once the wanted protocol, modus and RF channel are set, only one bit (RXEN) is shifted in to switch between RX and TX.

Configuration Word Detailed Description

The following describes the function of the 144 bits (bit 143 = MSB) that is used to configure the TRW-2.4G
General Device Configuration: bit[15:0]
ShockBurst™ Configuration: bit[119:0]
Test Configuration: bit[143:120]

MSB	TEST							
D143	D142	D141	D140	D139	D138	D137	D136	
Reserved for testing								
1	0	0	0	1	1	1	0	Default

MSB	TEST															
D135	D134	D133	D132	D131	D130	D129	D128	D127	D126	D125	D124	D123	D122	D121	D120	
Reserved for testing														Close PLL in TX		
0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	Default

DATA2_W								
D119	D118	D117	D116	D115	D114	D113	D112	
Data width channel#2 in # of bits excluding addr/crc								
0	0	1	0	0	0	0	0	Default

DATA1_W								
D111	D110	D109	D108	D107	D106	D105	D104	
Data width channel#1 in # of bits excluding addr/crc								
0	0	1	0	0	0	0	0	Default

ADDR2												
D103	D102	D101	....	D71	D70	D69	D68	D67	D66	D65	D64	
Channel#2 Address RX (up to 40bit)												
0	0	0	...	1	1	1	0	0	1	1	1	Default

ADDR1												
D63	D62	D61	....	D31	D30	D29	D28	D27	D26	D25	D24	
Channel#1 Address RX (up to 40bit)												
0	0	0	...	1	1	1	0	0	1	1	1	Default

ADDR_W						
D23	D22	D21	D20	D19	D18	
Address width in # of bits (both channels)						
0	0	1	0	0	0	Default

CRC		
D17	D16	
CRC Mode 1 = 16bit, 0 = 8bit	CRC 1 = enable; 0 = disable	
0	1	Default

RF-Programming															LSB	
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
Two Ch.	BUF	OD	XO			RF Power		Channel selection							RXEN	
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0	Default

Table 2Configuration data word

The MSB bit should be loaded first into the configuration register
Default configuration word: h8E08.1C20.2000.0000.00E7.0000.0000.E721.0F04.

ShockBurst™ configuration:

The section B[119:16] contains the segments of the configuration register dedicated to ShockBurst™ operational protocol. After VDD is turned on ShockBurst™ configuration is done once and remains set whilst VDD is present. During operation only the first byte for frequency channel and RX/TX switching need to be changed.

PLL_CTRL

PLL_CTRL		
D121	**D120**	**PLL**
0	0	Open TX Closed RX
0	1	Open TX Open RX
1	0	Closed TX Closed RX
1	1	Closed TX Open RX

Table 10 PLL setting.

Bit 121-120:

PLL_CTRL: Controls the setting of the PLL for test purposes. With closed PLL in TX no deviation will be present.

DATAx_W

DATA2_W							
119	118	117	116	115	114	113	112

DATA1_W							
111	110	109	108	107	106	105	104

Table 4 Number of bits in payload.

Bit 119 – 112:

DATA2_W: Length of RF package payload section for receive-channel 2.

Bit 111 – 104:

DATA1_W: Length of RF package payload section for receive-channel 1.

NOTE:

The total number of bits in a ShockBurst™ RF package may not exceed 256! Maximum length of payload section is hence given by:

$$DATAx\_W(bits) = 256 - ADDR\_W - CRC$$

Where:

ADDR_W: length of RX address set in configuration word B[23:18]
CRC: check sum, 8 or 16 bits set in configuration word B[17]
PRE: preamble, 4 or 8 bits are automatically included

Shorter address and CRC leaves more room for payload data in each package.

ADDRx

ADDR2											
103	102	101	....	71	70	69	68	67	66	65	64

ADDR1											
63	62	61	....	31	30	29	28	27	26	25	24

Table 5 Address of receiver #2 and receiver #1.

Bit 103 – 64:
ADDR2: Receiver address channel 2, up to 40 bit.

Bit 63 – 24: ADDR1
ADDR1: Receiver address channel 1, up to 40 bit.

NOTE!
Bits in ADDRx exceeding the address width set in ADDR_W are redundant and can be set to logic 0.

ADDR_W & CRC

		ADDR_W				CRC_L	CRC_EN
23	22	21	20	19	18	17	16

Table 6 Number of bits reserved for RX address + CRC setting.

Bit 23 – 18:
ADDR_W: Number of bits reserved for RX address in ShockBurst™ packages.

NOTE:
Maximum number of address bits is 40 (5 bytes). Values over 40 in ADDR_W are not valid.

Bit 17:
CRC_L: CRC length to be calculated by TRW-2.4G in ShockBurst™.
Logic 0: 8 bit CRC
Logic 1: 16 bit CRC

Bit: 16:
CRC_EN: Enables on-chip CRC generation (TX) and verification (RX).
Logic 0: On-chip CRC generation checking disabled
Logic 1: On-chip CRC generation checking enabled

NOTE:
An 8 bit CRC will increase the number of payload bits possible in each ShockBurst™ data packet, but will also reduce the system integrity.

General device configuration:

This section of the configuration word handles RF and device related parameters.

Modes:

RX2_EN	CM	RFDR_SB	XO_F			RF_PWR	
15	14	13	12	11	10	9	8

Table 7 RF operational settings.

Bit 15:

RX2_EN:

Logic 0: One channel receive
Logic 1: Two channels receive

NOTE:

In two channels receive, the TRW-2.4G receives on two separate frequency channels simultaneously. The frequency of receive channel 1 is set in the configuration word B[7-1], receive channel 2 is always 8 channels (8 MHz) above receive channel 1.

Bit 14:

Communication Mode:

Logic 1: nRF2401 operates in ShockBurst™ mode

Bit 13:

RF Data Rate:

Logic 0: 250 Kbps TRW-2.4G
Logic 1: 1 Mbps

NOTE:

Utilizing 250 kbps instead of 1Mbps will improve the receiver sensitivity by 10 dB. 1Mbps requires 16MHz crystal.

Bit 12-10:

D12	D11	D10
0	1	1

Table 8

Bit 9-8:

RF_PWR: Sets TRW-2.4G RF output power in transmit mode:

RF OUTPUT POWER		
D9	**D8**	**P [dBm]**
0	0	-20
0	1	-10
1	0	-5
1	1	0

Table 9 RF output power setting.

RF channel & direction

RF_CH#							RXEN
7	6	5	4	3	2	1	0

Table 10 Frequency channel + RX / TX setting.

Bit 7 – 1:

RF_CH#: Sets the frequency channel the nRF2401 operates on.

The channel frequency in ***transmit*** is given by:

$$Channel_{Hz} = 2400\,MHz + RF\_CH\# \cdot 1.0\,MHz$$

RF_CH #: between 2400MHz and 2527MHz may be set.

The channel frequency in ***data channel 1*** is given by:

$$Channel_{Hz} = 2400\,MHz + RF\_CH\# \cdot 1.0\,MHz$$ (Receive at PIN#8)

RF_CH #: between 2400MHz and 2524MHz may be set.

NOTE:
The channels above 83 can only be utilized in certain territories (ex: Japan)

The channel frequency in ***data channel 2*** is given by:

$$Channel_{Hz} = 2400\,MHz + RF\_CH\# \cdot 1.0\,MHz + 8MHz$$ (Receive at PIN#4)

RF_CH #: between 2408MHz and 2524MHz may be set.

Bit 0:

Set active mode:

Logic 0: transmit mode

Logic 1: receive mode

DATA PACKAGE DESCRIPTION

PRE-AMBLE	ADDRESS	PAYLOAD	CRC

Figure 7 Data Package Diagram

The data packet for both ShockBurst™ mode and direct mode communication is divided into 4 sections. These are:

1. PREAMBLE	· The preamble field is required in ShockBurst.
2. ADDRESS	· The address field is required in ShockBurst. mode. · 8 to 40 bits length. · Address automatically removed from received packet in ShockBurst.mode
3. PAYLOAD	· The data to be transmitted · In Shock-Burst mode payload size is 256 bits minus the following:(Address: 8 to 40 bits. + CRC 8 or 16 bits).
4. CRC	· 8 or 16 bits length · The CRC is stripped from the received output data.

IMPORTANT TIMING DATA

The following timing applies for operation [illegible] TRW-2.4G

TRW-2.4G Timing Information

nRF2401 timing	Max.	Min.	Name
VDD OFF → ST_BY mode	3ms		Tpd2sby
VDD OFF → Active mode (RX TX)	3ms		Tpd2a
ST_BY → TX ShockBurst™	195µs		Tsby2txSB
ST_BY → TX Direct Mode	202µs		Tsby2txDM
ST_BY → RX mode	202µs		Tsby2rx
Minimum delay from CS to data.		5µs	Tcs2data
Minimum delay from CE to data.		5µs	Tce2data
Minimum delay from DR1/2 to clk.		50ns	Tdr2clk
Maximum delay from clk to data.	50ns		Tclk2data
Delay between edges		50ns	Td
Setup time		500ns	Ts
Hold time		500ns	Th
Delay to finish internal GFSK data		1 data rate	Tfd
Minimum input clock high		500ns	Thmin
Set-up of data in Direct Mode	50ns		Tsdm
Minimum clock high in Direct Mode		300ns	Thdm
Minimum clock low in Direct Mode		230ns	Tldm

Table 11 Switching times for [illegible] TRW-2.4G

When the [illegible] TRW-2.4G 401 is in power down it must always settle in stand-by (Tpd2sby) before it can enter configuration or one of the active modes.

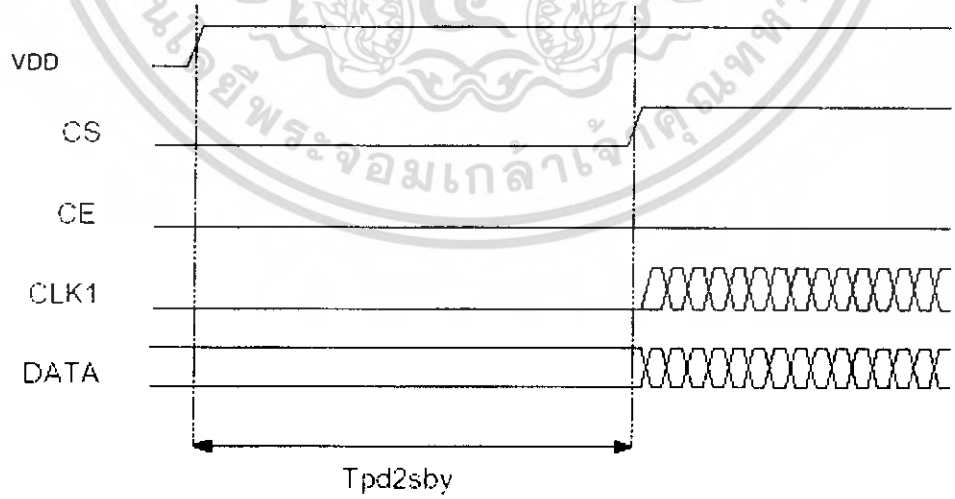


Figure 8 Timing diagram for [illegible] (or VDD off) to stand by mode for TRW-2.4G

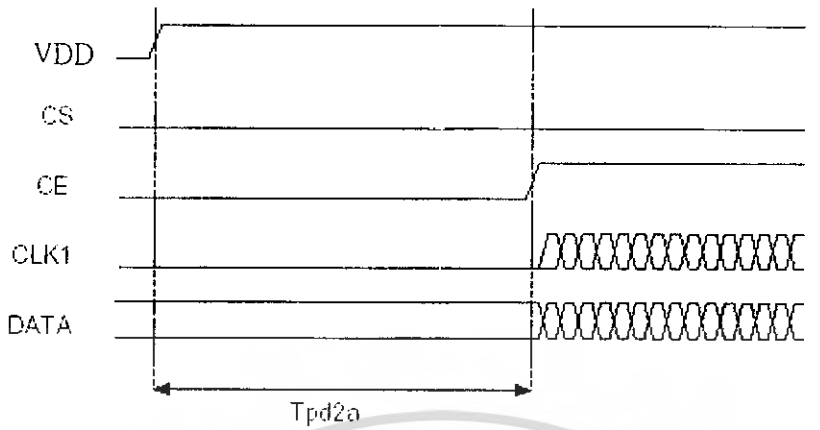


Figure 9 VDD off to active mode

Note that the configuration word will be lost when VDD is turned off and that the device then must be configured before going to one of the active modes. If the device is configured one can go directly from power down to the wanted active mode.

Note:

CE and CS may not be high at the same time. Setting one or the other decides whether configuration or active mode is entered.

Configuration mode timing

When one or more of the bits in the configuration word needs to be changed the following timing apply.

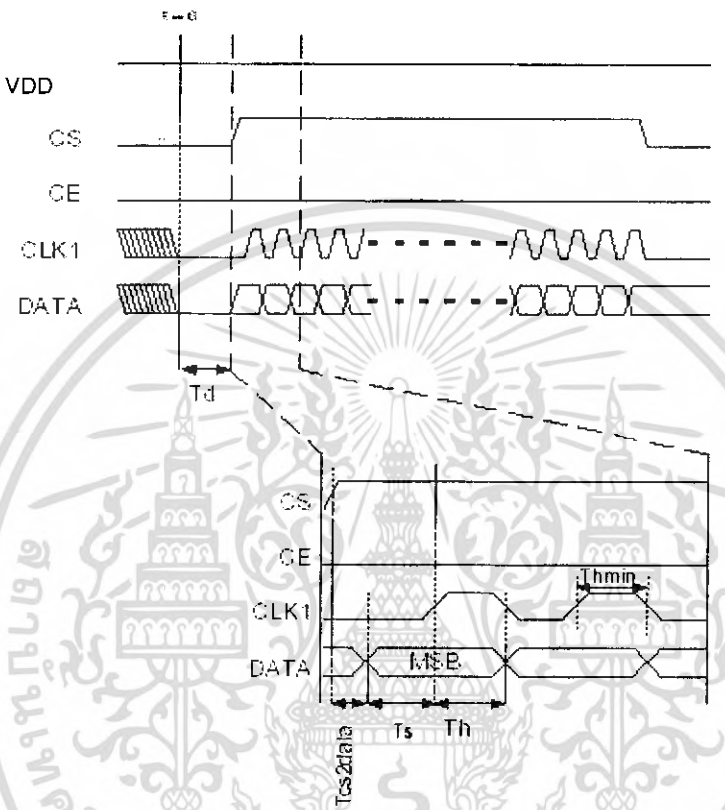


Figure 10 Timing diagram for configuration of TRW-2.4G

If configuration mode is entered from power down, CS can be set high after Tpd2sby as shown in Figure 8

ShockBurst™ Mode timing

ShockBurst™ TX:

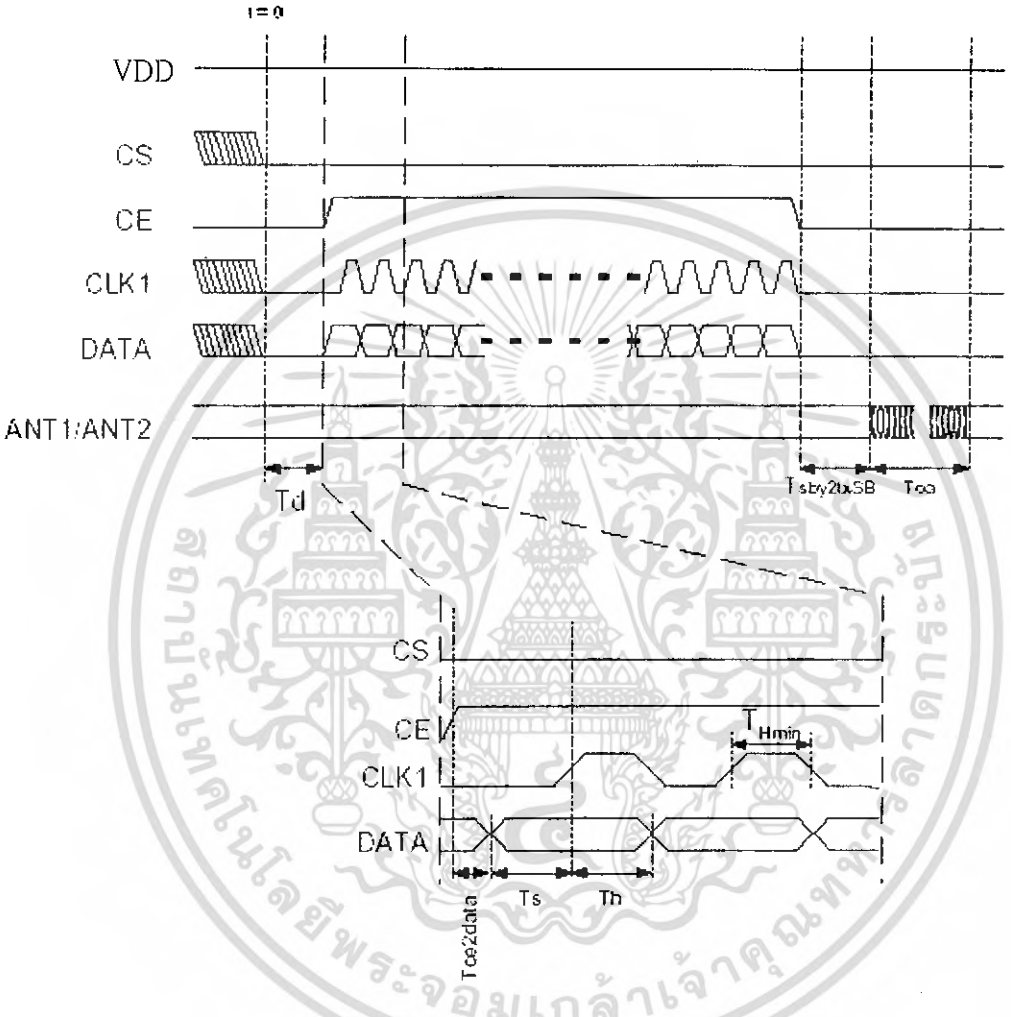
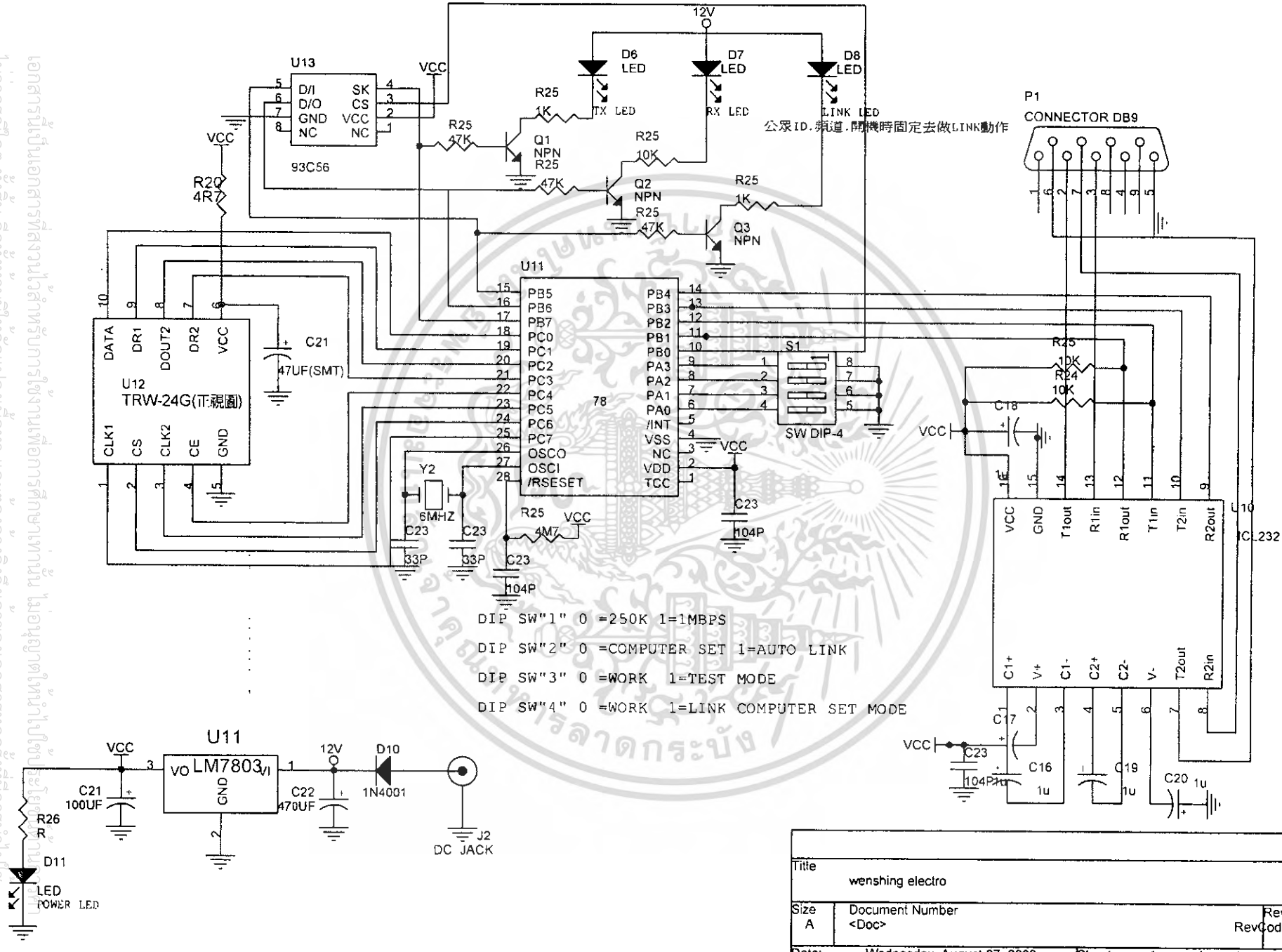


Figure 11 Timing of ShockBurst™ in TX

The package length and the data rate give the delay Toa (time on air), as shown in the equation.

$T_{oa} = 1/datarate \cdot (=databits + 1)$

2.4GHZ RS-232



Title	wenshing electro	
Size A	Document Number <Doc>	Rev RevCode = A
Date: Wednesday, August 27, 2003	Sheet 1 of 1	

ShockBurst™ RX:

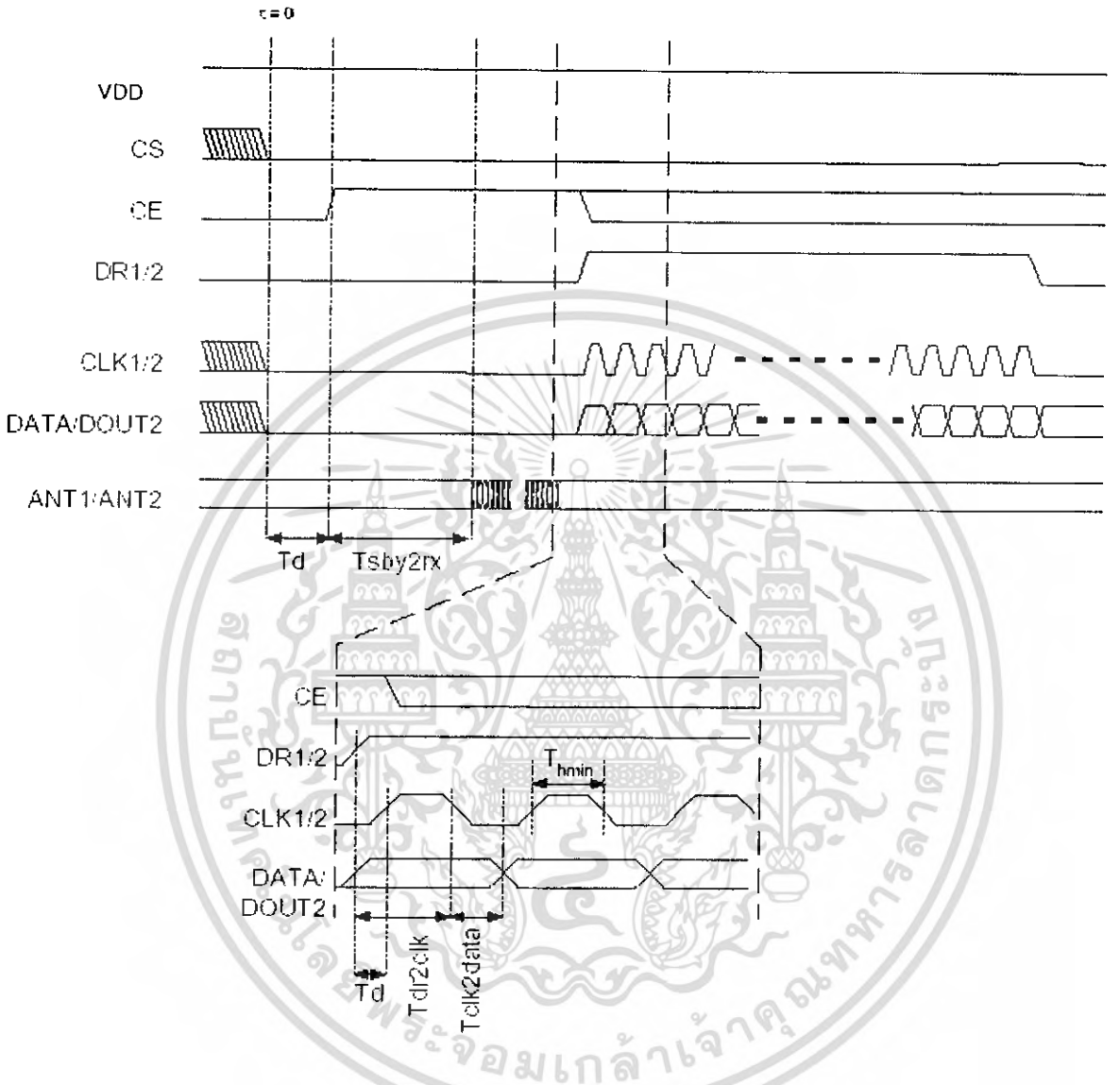


Figure 12 Timing of ShockBurst™ in RX

The CE may be kept high during downloading of data, but the cost is higher current consumption (18mA) and the benefit is no start-up time (200μs) after the DR1 goes low.

Output Power adjustment

Power setting bits of configuring word	RF output power	DC current consumption
11	0 dBm ±3dB	13.0 mA
10	-5 dBm ±3dB	10.5 mA
01	-10 dBm ±3dB	9.4 mA
00	-20 dBm ±3dB	8.8 mA

Conditions: VDD = 3.0V, VSS = 0V, T_A = 27°C, Load impedance = 400Ω.

Features

- Compatible with MCS-51™ Products
- 2K Bytes of Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- 2.7V to 6V Operating Range
- Fully Static Operation: 0 Hz to 24 MHz
- Two-Level Program Memory Lock
- 128 x 8-Bit Internal RAM
- 15 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial UART Channel
- Direct LED Drive Outputs
- On-Chip Analog Comparator
- Low Power Idle and Power Down Modes

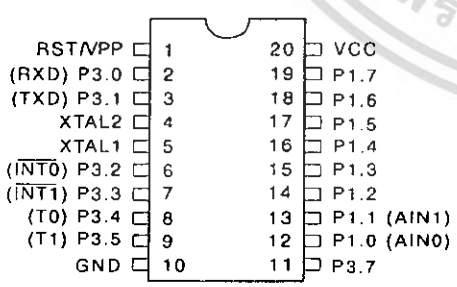
Description

The AT89C2051 is a low-voltage, high-performance CMOS 8-bit microcomputer with 2K Bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high density nonvolatile memory technology and is compatible with the industry standard MCS-51™ instruction set. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C2051 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications.

The AT89C2051 provides the following standard features: 2K Bytes of Flash, 128 bytes of RAM, 15 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, a precision analog comparator, on-chip oscillator and clock circuitry. In addition, the AT89C2051 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The Power Down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Pin Configuration

PDIP/SOIC

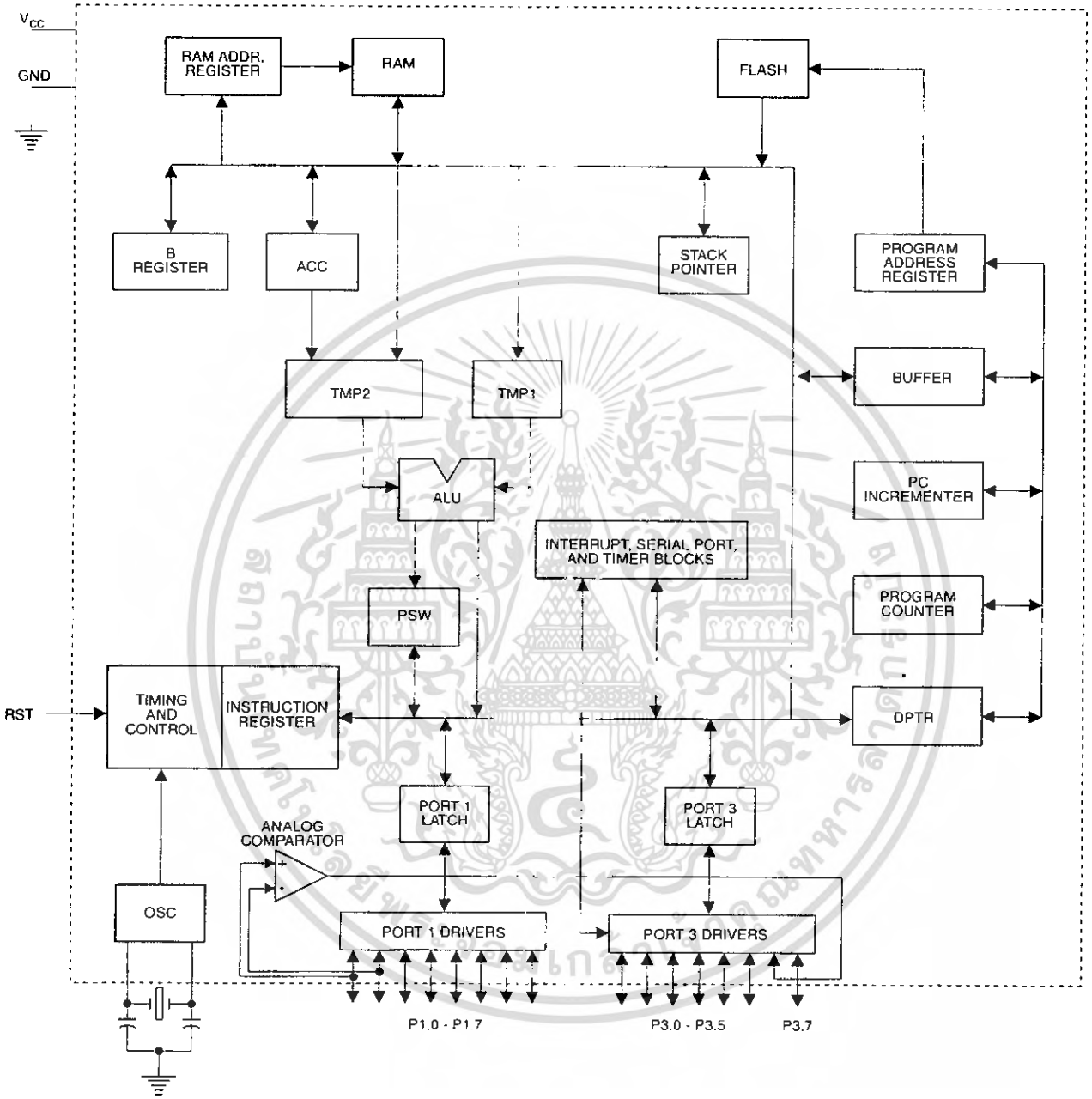


8-Bit Microcontroller with 2K Bytes Flash

AT89C2051

0368D-B-12/97

Block Diagram



Pin Description

V_{CC}
Supply voltage.

GND
Ground.

Port 1
Port 1 is an 8-bit bidirectional I/O port. Port pins P1.2 to P1.7 provide internal pullups. P1.0 and P1.1 require external pullups. P1.0 and P1.1 also serve as the positive input (AIN0) and the negative input (AIN1), respectively, of the on-chip precision analog comparator. The Port 1 output buffers can sink 20 mA and can drive LED displays directly. When 1s are written to Port 1 pins, they can be used as inputs. When pins P1.2 to P1.7 are used as inputs and are externally pulled low, they will source current (I_{IL}) because of the internal pullups.

Port 1 also receives code data during Flash programming and verification.

Port 3
Port 3 pins P3.0 to P3.5, P3.7 are seven bidirectional I/O pins with internal pullups. P3.6 is hard-wired as an input to the output of the on-chip comparator and is not accessible as a general purpose I/O pin. The Port 3 output buffers can sink 20 mA. When 1s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C2051 as listed below:

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)

Port 3 also receives some control signals for Flash programming and verification.

RST
Reset input. All I/O pins are reset to 1s as soon as RST goes high. Holding the RST pin high for two machine cycles while the oscillator is running resets the device.

Each machine cycle takes 12 oscillator or clock cycles.

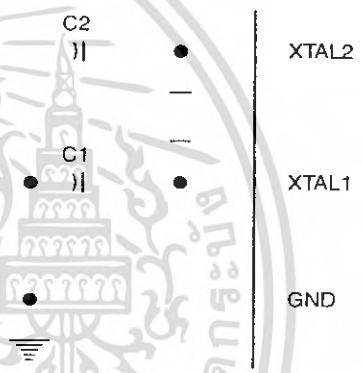
XTAL1
Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2
Output from the inverting oscillator amplifier.

Oscillator Characteristics

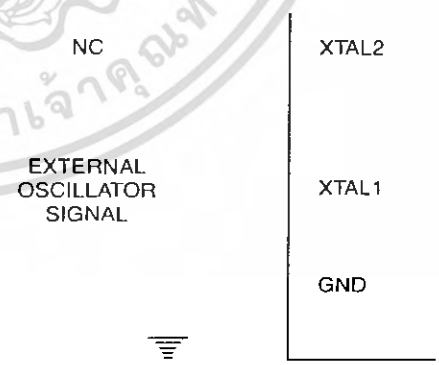
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Figure 1. Oscillator Connections



Note: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Figure 2. External Clock Drive Configuration



Special Function Registers

A map of the on-chip memory area called the Special Function Register (SFR) space is shown in the table below.

Note that not all of the addresses are occupied, and unoccupied addresses may not be implemented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have an indeterminate effect.

User software should not write 1s to these unlisted locations, since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.

Table 1. AT89C2051 SFR Map and Reset Values

0F8H								0FFH
0F0H	B 00000000							0F7H
0E8H								0EFH
0E0H	ACC 00000000							0E7H
0D8H								0DFH
0D0H	PSW 00000000							0D7H
0C8H								0CFH
0C0H								0C7H
0B8H	IP XXX00000							0BFH
0B0H	P3 11111111							0B7H
0A8H	IE 0XX00000							0AFH
0A0H								0A7H
98H	SCON 00000000	SBUF XXXXXXXX						9FH
90H	P1 11111111							97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000		8FH
80H		SP 00000111	DPL 00000000	DPH 00000000				PCON 0XXX0000 87H

Restrictions on Certain Instructions

The AT89C2051 and is an economical and cost-effective member of Atmel's growing family of microcontrollers. It contains 2K bytes of flash program memory. It is fully compatible with the MCS-51 architecture, and can be programmed using the MCS-51 instruction set. However, there are a few considerations one must keep in mind when utilizing certain instructions to program this device.

All the instructions related to jumping or branching should be restricted such that the destination address falls within the physical program memory space of the device, which is 2K for the AT89C2051. This should be the responsibility of the software programmer. For example, LJMP 7E0H would be a valid instruction for the AT89C2051 (with 2K of memory), whereas LJMP 900H would not.

1. Branching instructions:
LCALL, LJMP, ACALL, AJMP, SJMP, JMP @A+DPTR

These unconditional branching instructions will execute correctly as long as the programmer keeps in mind that the destination branching address must fall within the physical boundaries of the program memory size (locations 00H to 7FFH for the 89C2051). Violating the physical space limits may cause unknown program behavior.

CJNE [...], DJNZ [...], JB, JNB, JC, JNC, JBC, JZ, JNZ With these conditional branching instructions the same rule above applies. Again, violating the memory boundaries may cause erratic execution.

For applications involving interrupts the normal interrupt service routine address locations of the 80C51 family architecture have been preserved.

2. MOVX-related instructions, Data Memory:
The AT89C2051 contains 128 bytes of internal data memory. Thus, in the AT89C2051 the stack depth is limited to 128 bytes, the amount of available RAM. External DATA memory access is not supported in this device, nor is external PROGRAM memory execution. Therefore, no MOVX [...] instructions should be included in the program.

A typical 80C51 assembler will still assemble instructions, even if they are written in violation of the restrictions mentioned above. It is the responsibility of the controller user to know the physical features and limitations of the device being used and adjust the instructions used correspondingly.

Program Memory Lock Bits

On the chip are two lock bits which can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the table below:

Lock Bit Protection Modes[1]

Program Lock Bits			
	LB1	LB2	Protection Type
1	U	U	No program lock features.
2	P	U	Further programming of the Flash is disabled.
3	P	P	Same as mode 2, also verify is disabled.

Note: 1. The Lock Bits can only be erased with the Chip Erase operation.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

P1.0 and P1.1 should be set to '0' if no external pullups are used, or set to '1' if external pullups are used.

It should be noted that when idle is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

Power Down Mode

In the power down mode the oscillator is stopped, and the instruction that invokes power down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the power down mode is terminated. The only exit from power down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

P1.0 and P1.1 should be set to '0' if no external pullups are used, or set to '1' if external pullups are used.

Programming The Flash

The AT89C2051 is shipped with the 2K bytes of on-chip PEROM code memory array in the erased state (i.e., contents = FFH) and ready to be programmed. The code memory array is programmed one byte at a time. *Once the array is programmed, to re-program any non-blank byte, the entire memory array needs to be erased electrically.*

Internal Address Counter: The AT89C2051 contains an internal PEROM address counter which is always reset to 000H on the rising edge of RST and is advanced by applying a positive going pulse to pin XTAL1.

Programming Algorithm: To program the AT89C2051, the following sequence is recommended.

1. Power-up sequence:
   Apply power between V_{CC} and GND pins
   Set RST and XTAL1 to GND
2. Set pin RST to 'H'
   Set pin P3.2 to 'H'
3. Apply the appropriate combination of 'H' or 'L' logic levels to pins P3.3, P3.4, P3.5, P3.7 to select one of the programming operations shown in the PEROM Programming Modes table.

To Program and Verify the Array:

4. Apply data for Code byte at location 000H to P1.0 to P1.7.
5. Raise RST to 12V to enable programming.
6. Pulse P3.2 once to program a byte in the PEROM array or the lock bits. The byte-write cycle is self-timed and typically takes 1.2 ms.
7. To verify the programmed data, lower RST from 12V to logic 'H' level and set pins P3.3 to P3.7 to the appropiate levels. Output data can be read at the port P1 pins.
8. To program a byte at the next address location, pulse XTAL1 pin once to advance the internal address counter. Apply new data to the port P1 pins.
9. Repeat steps 5 through 8, changing data and advancing the address counter for the entire 2K bytes array or until the end of the object file is reached.
10. Power-off sequence:
    set XTAL1 to 'L'
    set RST to 'L'
    Turn V_{CC} power off

Data Polling: The AT89C2051 features Data Polling to indicate the end of a write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written data on P1.7. Once the write cycle has been completed, true data is valid on all outputs, and the next cycle may begin. Data Polling may begin any time after a write cycle has been initiated.

Ready/Busy: The Progress of byte programming can also be monitored by the RDY/BSY output signal. Pin P3.1 is pulled low after P3.2 goes High during programming to indicate BUSY. P3.1 is pulled High again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed code data can be read back via the data lines for verification:

1. Reset the internal address counter to 000H by bringing RST from 'L' to 'H'.
2. Apply the appropriate control signals for Read Code data and read the output data at the port P1 pins.
3. Pulse pin XTAL1 once to advance the internal address counter.
4. Read the next code data byte at the port P1 pins.
5. Repeat steps 3 and 4 until the entire array is read.

The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire PEROM array (2K bytes) and the two Lock Bits are erased electrically by using the proper combination of control signals and by holding P3.2 low for 10 ms. The code array is written with all "1"s in the Chip Erase operation and must be executed before any non-blank memory byte can be re-programmed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 000H, 001H, and 002H, except that P3.5 and P3.7 must be pulled to a logic low. The values returned are as follows.

(000H) = 1EH indicates manufactured by Atmel
(001H) = 21H indicates 89C2051

Programming Interface

Every code byte in the Flash array can be written and the entire array can be erased by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

Flash Programming Modes

Mode		RST/VPP	P3.2/PROG	P3.3	P3.4	P3.5	P3.7
Write Code Data[1][3]		12V		L	H	H	H
Read Code Data[1]		H	H	L	L	H	H
Write Lock	Bit - 1	12V		H	H	H	H
	Bit - 2	12V		H	H	L	L
Chip Erase		12V	(2)	H	L	L	L
Read Signature Byte		H	H	L	L	L	L

Notes:
1. The internal PEROM address counter is reset to 000H on the rising edge of RST and is advanced by a positive pulse at XTAL 1 pin.
2. Chip Erase requires a 10-ms PROG pulse.
3. P3.1 is pulled Low during programming to indicate RDY/BSY.

Figure 3. Programming the Flash Memory

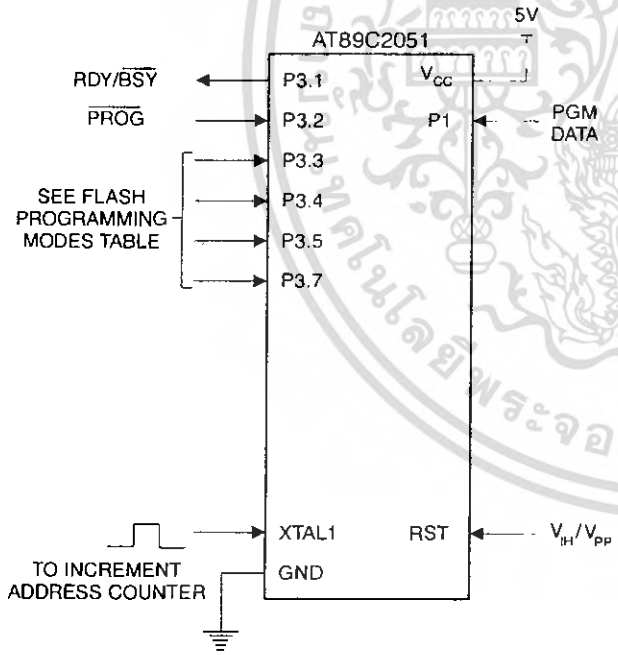
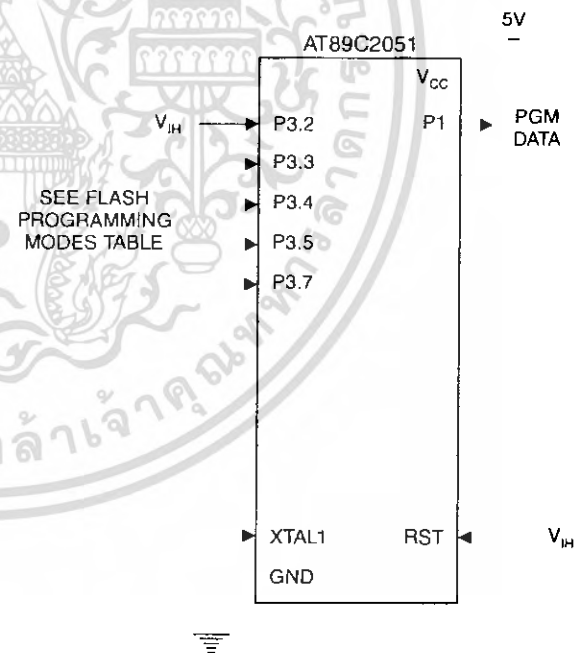


Figure 4. Verifying the Flash Memory



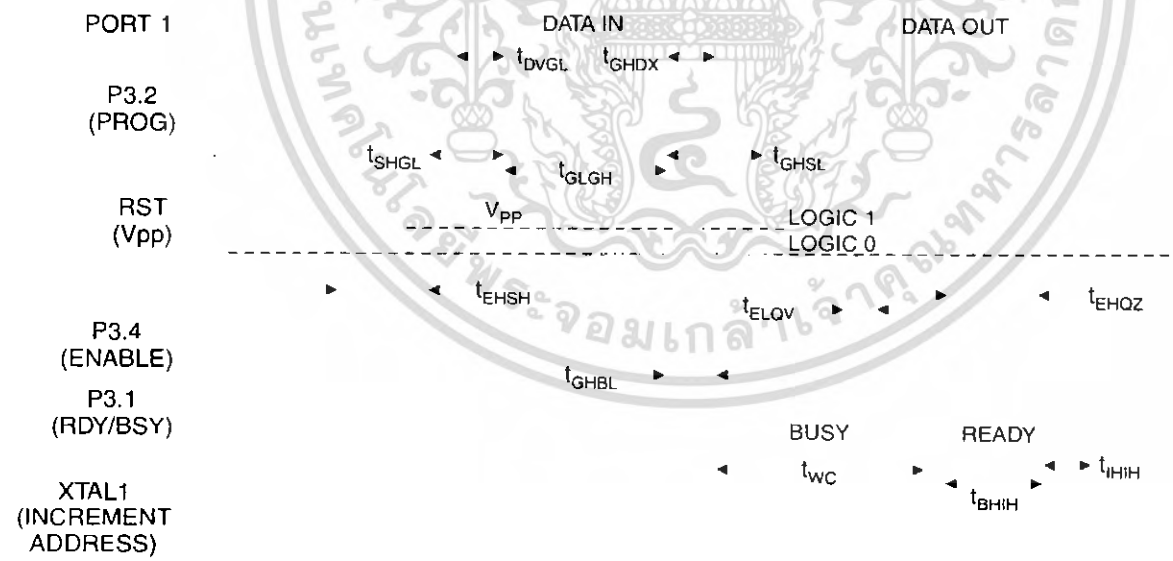
Flash Programming and Verification Characteristics

T_A = 0°C to 70°C, V_{CC} = 5.0 ± 10%

Symbol	Parameter	Min	Max	Units
V_{PP}	Programming Enable Voltage	11.5	12.5	V
I_{PP}	Programming Enable Current		250	µA
t_{DVGL}	Data Setup to $\overline{PROG}$ Low	1.0		µs
t_{GHDX}	Data Hold After $\overline{PROG}$	1.0		µs
t_{EHSH}	P3.4 ($\overline{ENABLE}$) High to V_{PP}	1.0		µs
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GHSL}	V_{PP} Hold After $\overline{PROG}$	10		µs
t_{GLGH}	$\overline{PROG}$ Width	1	110	µs
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		1.0	µs
t_{EHQZ}	Data Float After $\overline{ENABLE}$	0	1.0	µs
t_{GHBL}	$\overline{PROG}$ High to $\overline{BUSY}$ Low		50	ns
t_{WC}	Byte Write Cycle Time		2.0	ms
t_{BHIH}	RDY/$\overline{BSY}$\ to Increment Clock Delay	1.0		µs
t_{IHIL}	Increment Clock High	200		ns

Note: 1. Only used in 12-volt programming mode.

Flash Programming and Verification Waveforms



Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-1.0V to +7.0V
Maximum Operating Voltage	6.6V
DC Output Current	25.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

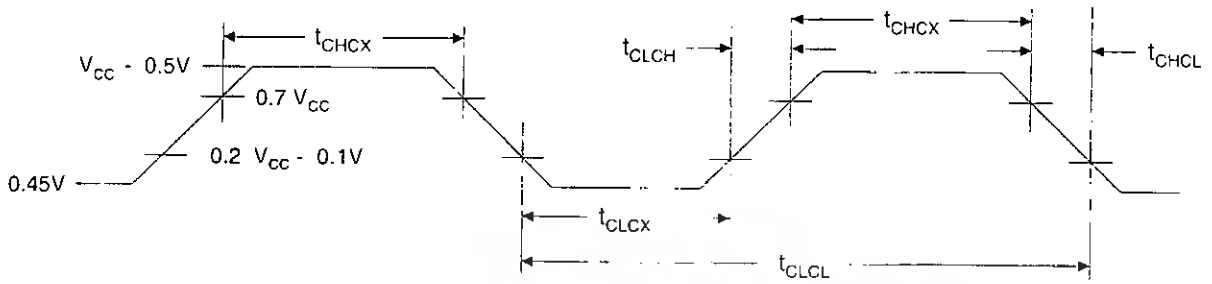
DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 2.0V to 6.0V (unless otherwise noted)

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low Voltage		-0.5	0.2 V_{CC} - 0.1	V
V_{IH}	Input High Voltage	(Except XTAL1, RST)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V
V_{IH1}	Input High Voltage	(XTAL1, RST)	0.7 V_{CC}	V_{CC} + 0.5	V
V_{OL}	Output Low Voltage[1] (Ports 1, 3)	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 2.7V		0.5	V
V_{OH}	Output High Voltage (Ports 1, 3)	I_{OH} = -80 µA, V_{CC} = 5V ± 10%	2.4		V
		I_{OH} = -30 µA	0.75 V_{CC}		V
		I_{OH} = -12 µA	0.9 V_{CC}		V
I_{IL}	Logical 0 Input Current (Ports 1, 3)	V_{IN} = 0.45V		-50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1, 3)	V_{IN} = 2V, V_{CC} = 5V ± 10%		-750	µA
I_{LI}	Input Leakage Current (Port P1.0, P1.1)	0 < V_{IN} < V_{CC}		±10	µA
V_{OS}	Comparator Input Offset Voltage	V_{CC} = 5V		20	mV
V_{CM}	Comparator Input Common Mode Voltage		0	V_{CC}	V
RRST	Reset Pulldown Resistor		50	300	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz, V_{CC} = 6V/3V		15/5.5	mA
		Idle Mode, 12 MHz, V_{CC} = 6V/3V P1.0 & P1.1 = 0V or V_{CC}		5/1	mA
	Power Down Mode[2]	V_{CC} = 6V P1.0 & P1.1 = 0V or V_{CC}		100	µA
		V_{CC} = 3V P1.0 & P1.1 = 0V or V_{CC}		20	µA

Notes:
1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
   Maximum I_{OL} per port pin: 20 mA
   Maximum total I_{OL} for all output pins: 80 mA
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
2. Minimum V_{CC} for Power Down is 2V.

External Clock Drive Waveforms



External Clock Drive

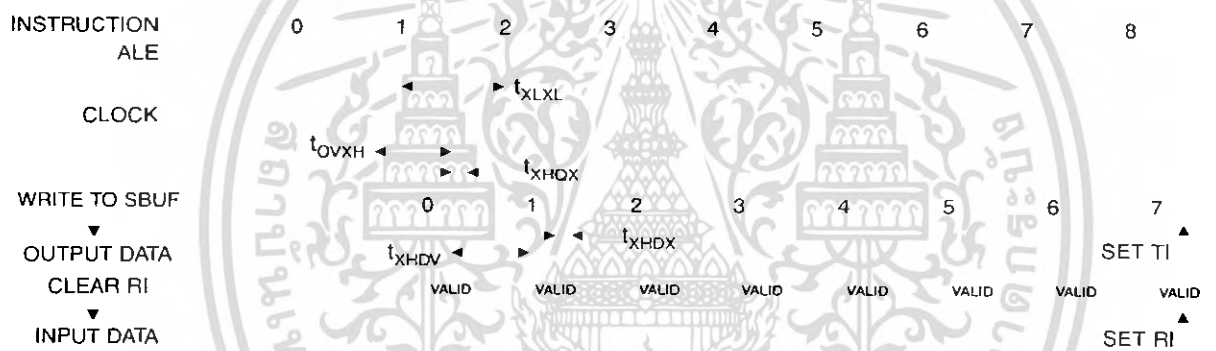
Symbol	Parameter	V_{CC} = 2.7V to 6.0V		V_{CC} = 4.0V to 6.0V		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency	0	12	0	24	MHz
t_{CLCL}	Clock Period	83.3		41.6		ns
t_{CHCX}	High Time	30		15		ns
t_{CLCX}	Low Time	30		15		ns
t_{CLCH}	Rise Time		20		20	ns
t_{CHCL}	Fall Time		20		20	ns

Serial Port Timing: Shift Register Mode Test Conditions

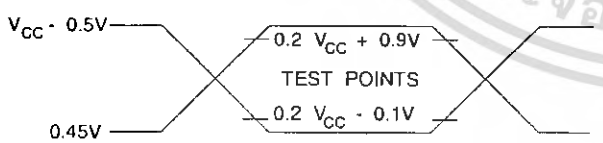
(V_{CC} = 5.0V ± 20%; Load Capacitance = 80 pF)

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
t_{XLXL}	Serial Port Clock Cycle Time	1.0		$12t_{CLCL}$		µs
t_{QVXH}	Output Data Setup to Clock Rising Edge	700		$10t_{CLCL}$-133		ns
t_{XHQX}	Output Data Hold After Clock Rising Edge	50		$2t_{CLCL}$-117		ns
t_{XHDX}	Input Data Hold After Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		$10t_{CLCL}$-133	ns

Shift Register Mode Timing Waveforms

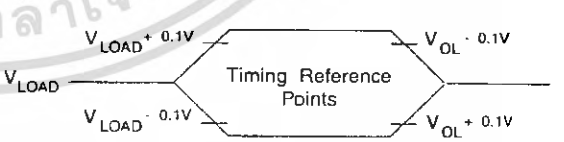


AC Testing Input/Output Waveforms[1]



Note: 1. AC Inputs during testing are driven at V_{CC} - 0.5V for a logic 1 and 0.45V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

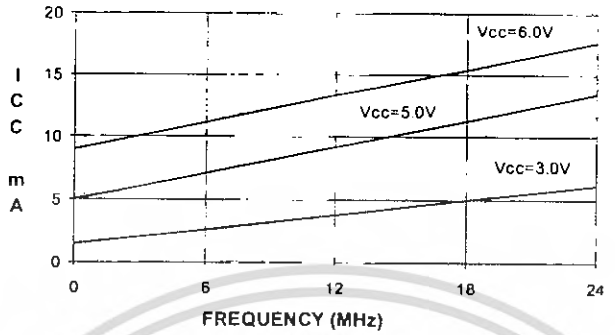
Float Waveforms[1]



Note: 1. For timing purposes, a port pin is no longer floating when a 100 mV change from load voltage occurs. A port pin begins to float when 100 mV change frothe loaded V_{OH}/V_{OL} level occurs.

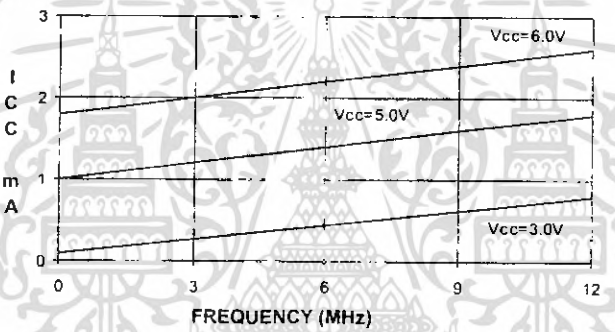
AT89C2051

TYPICAL ICC - ACTIVE (85°C)



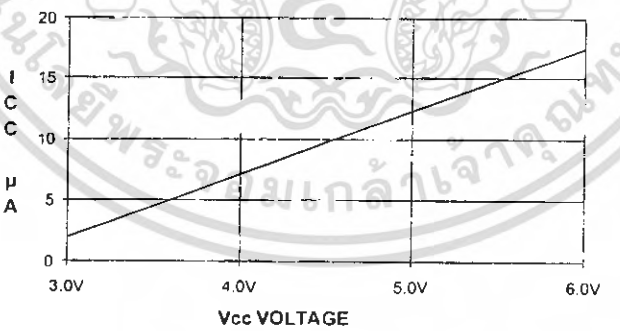
AT89C2051

TYPICAL ICC - IDLE (85°C)



AT89C2051

TYPICAL ICC vs. VOLTAGE- POWER DOWN (85°C)



Notes:
1. XTAL1 tied to GND for I_{CC} (power down)
2. P1.0 and P1.1 = V_{CC} or GND
3. Lock bits programmed

Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
12	2.7V to 6.0V	AT89C2051-12PC AT89C2051-12SC	20P3 20S	Commercial (0°C to 70°C)
		AT89C2051-12PI AT89C2051-12SI	20P3 20S	Industrial (-40°C to 85°C)
		AT89C2051-12PA AT89C2051-12SA	20P3 20S	Automotive (-40°C to 105°C)
24	4.0V to 6.0V	AT89C2051-24PC AT89C2051-24SC	20P3 20S	Commercial (0°C to 70°C)
		AT89C2051-24PI AT89C2051-24SI	20P3 20S	Industrial (-40°C to 85°C)

Package Type	
20P3	20 Lead, 0.300" Wide, Plastic Dual In-line Package (PDIP)
20S	20 Lead, 0.300" Wide, Plastic Gull Wing Small Outline (SOIC)