คีย์บอร์ดและเมาส์ไร้สาย

WIRELESS MOUSE AND KEYBOARD

โดย

นางสาวธิดารัตน์ ศรีสมรทอง

นายธีร์รัตติ อินทรโชติ

นายนิติ ฉัยยากูล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ผ่านการตรวจรูปเล่มแล้ว
(ลงชื่อ)....................ผู้ตรวจ

ผ่านการตรวจชิ้นงานแล้ว
(ลงชื่อ)....................ผู้ตรวจ

คีย์บอร์ดและเมาส์ไร้สาย

WIRELESS MOUSE AND KEYBOARD

โดย

นางสาวชิดารัตน์ ศรีสมรทอง 46010308

นายธีร์รัตติ อินทรโชติ 46010310

นาย นิติ ฉัยยากูล 46010362

อาจารย์ที่ปรึกษา

ผศ.ดร.พิพัฒน์ พรหมมี

ผศ.ดร. สมเกียรติ ฤกษ์วรัญญู

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **เมาส์และคีย์บอร์ดไร้สาย**

WIRELESS MOUSE AND KEYBOARD

ผู้จัดทำ

1. นางสาวธิดารัตน์ ศรีสมรทอง 46010308

2. นายธีร์รัตติ อินทรโชติ 46010310

3. นายนิติ ฉัยยากูล 46010362

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. พิพัฒน์ พรหมมี)

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. สมเกียรติ ฤกษ์วรัญญู)

เมาส์และคีย์บอร์ดไร้สาย

WIRELESS MOUSE AND KEYBOARD

โดย	นางสาวธิดารัตน์ ศรีสมรทอง	46010308
	นายธีร์รัตติ อินทรโชติ	46010310
	นายนิติ ฉัยยากูล	46010362

อาจารย์ที่ปรึกษา	ผศ.ดร.พิพัฒน์ พรหมมี	
และ	ผศ.ดร.สมเกียรติ ฤกษ์วรัญญู	

บทคัดย่อ

โครงงานนี้นำเสนอชิ้นงานเมาส์และคีย์บอร์ดแบบไร้สาย ซึ่งใช้วิธีการส่งสัญญาณแบบ FSK โดยให้สัญญาณของเมาส์และคีย์บอร์ดส่งไปในช่องสัญญาณเดียวกัน โดยชุดเชื่อมต่อคีย์บอร์ดและเมาส์แบบไร้สายซึ่งจะใช้ไมโครคอนโทรลเลอร์ ส่วนอีกด้านหนึ่งใช้ไมโครคอนโทรลเลอร์เชื่อมต่อกับคอมพิวเตอร์ผ่านพอร์ตเมาส์และคีย์บอร์ดและเชื่อมต่อกับด้านเมาส์และคีย์บอร์ดต้นทาง รวมทั้งควบคุมการสื่อสารระหว่างชุดเชื่อมต่อโดยใช้โปรโตคอลที่กำหนดขึ้น ส่งผ่านตัวกลางที่เป็นความถี่วิทยุ

Abstract

The mouse and keyboard are able to multiplexed in same channel of its radio frequency. The microcontroller is used to interface with the mouse and keyboard. Others microcontrollers one used to interface with a computer. The proprietary protocol is used for communicated between both site of microcontroller based on radio frequency.

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

หน้า

สารบัญรูปภาพ(ต่อ)

หน้า

สารบัญตาราง

บทที่1
บทนำ

ในปัจจุบันคอมพิวเตอร์กลายเป็นส่วนหนึ่งสำหรับการดำเนินชีวิตประจำวันของมนุษย์ จนอาจกล่าวได้ว่าเป็นปัจจัย 5 ที่เพิ่มเข้ามาอย่างหลีกเลี่ยงไม่ได้ ในการใช้งานคอมพิวเตอร์ต้องมีการป้อนข้อมูลให้กับคอมพิวเตอร์โดยใช้อุปกรณ์อินพุต ได้แก่ เมาส์และคีย์บอร์ด ทั้งนี้เพื่อความสะดวกในการติดต่อสื่อสารกับคอมพิวเตอร์ในระยะทางไกล เช่น การประชุม การสัมมนา การเสนอชิ้นงาน โครงงาน ตลอดจนงานวิจัยต่างๆ เพื่อทำการนำเสนองานจากการควบคุมเมาส์และคีย์บอร์ดในระยะทางไกล โครงงานนี้จึงได้มีการนำระบบการสื่อสารไร้สายที่เรียกว่า wireless มาประยุกต์ใช้ ประกอบกับการนำความรู้ด้านเทคนิคการมอดูเลต (modulate) สัญญาณ เพื่อให้สามารถส่งข้อมูลไปในอากาศในระยะทางไกลได้ อย่างไรก็ตามในการรับ-ส่งข้อมูลอาจมีการผิดพลาด (error) ของสัญญาณเกิดขึ้นจากปัจจัยต่าง ๆ ทั้งภายในและภายนอกระบบ ดังนั้นการที่จะตรวจสอบว่าข้อมูลที่ได้รับมานั้นถูกต้องหรือไม่ จำเป็นที่จะต้องมีระบบที่สามารถตรวจสอบการทำงานได้ จึงนำโปรโตคอลในระบบการสื่อสารข้อมูล(data communication) มาใช้ ซึ่งโปรโตคอลนี้สามารถที่จะใช้จัดการกับข้อมูลให้เป็นระบบแบบแผนและสามารถที่จะตรวจสอบความผิดพลาดของข้อมูลได้เป็นอย่างดี

โครงงานนี้ได้นำเสนอการป้อนข้อมูลผ่านเมาส์และ คีย์บอร์ดในรูปแบบของการสื่อสารไร้สาย โดยใช้คลื่นวิทยุ(RF wireless) และไมโครคอนโทรลเลอร์ (microcontroller) ในการประมวลผลข้อมูลต่างๆ ทางโปรโตคอล นอกจากนี้ยังมีวงจรคลื่นวิทยุในการรับ-ส่งข้อมูลระหว่างอุปกรณ์อินพุต (เมาส์และคีย์บอร์ด) กับ คอมพิวเตอร์

ขั้นตอนดำเนินงานในภาคการศึกษาที่ 1

- ศึกษาการสื่อสารระหว่างคีย์บอร์ดกับคอมพิวเตอร์และระหว่างเมาส์กับคอมพิวเตอร์
- ศึกษาการทำงานและเขียนโปรแกรมควบคุมเอฟเอสเคโมดูล(FSK module)
- เขียนโปรแกรมส่งค่าสแกนโค้ดของคีย์บอร์ดผ่านทาง FSK module

ขั้นตอนดำเนินงานในภาคการศึกษาที่ 2

- เขียนโปรแกรมการติดต่อระหว่างคีย์บอร์ดกับคอมพิวเตอร์ให้ทำการติดต่อกันได้อย่างสมบูรณ์
- เขียนโปรแกรมการติดต่อระหว่างเมาส์กับคอมพิวเตอร์ให้ทำการติดต่อกันได้อย่างสมบูรณ์
- ทำชิ้นงานเมาส์และคีย์บอร์ดไร้สายให้สมบูรณ์

บทที่ 2
ทฤษฎี และ หลักการ

ในบทนี้จะมี 2 ส่วนใหญ่ ๆ คือ ส่วนของการอินเตอร์เฟสของเมาส์-คีย์บอร์ด และส่วนของเครื่องรับ-ส่งวิทยุ

ส่วนของการอินเตอร์เฟสของเมาส์-คีย์บอร์ด

2.1 ลักษณะทางกายภาพของพอร์ต PS/2

พอร์ต PS/2 มีลักษณะทางกายภาพอยู่ 2 รูปแบบคือ 5 - pin DIN และ 6 - pin mini DIN ทั้งคู่มีรูปแบบที่เหมือนกันทางด้านไฟฟ้า (Electrically) แต่ในทางปฏิบัติเราสามารถที่จะหาตัวแปลง (adaptor) มาทำการแปลงจาก 5 - pin DIN มาเป็น 6 - pin mini DIN และแปลงจาก 6 - pin mini DIN มาเป็น 5 - pin DIN โดยทั่วไปแล้วคอมพิวเตอร์ในปัจจุบัน พอร์ต PS/2 จะเป็นแบบ 6 - pin mini DIN ซึ่งมีลักษณะทางกายภาพดังนี้

รูปที่ 2.1 ลักษณะทางกายภาพของพอร์ต PS/2 แบบ 6 - pin mini DIN

6 - Pin mini DIN (PS/2):

1) Data
2) Not Implemented
3) Ground
4) Vcc (+5V)
5) Clock
6) Not Implemented

จากรูปขาต่าง ๆ ของอินเตอร์เฟส PS/2 จะมีขา Vcc / Ground สำหรับจ่ายไฟเลี้ยงแก่เมาส์และคีย์บอร์ดโดยที่กระแสที่อุปกรณ์ (Device) ไม่ควรเกิน 100 mA และควรหลีกเลี่ยงการถอดอุปกรณ์ออกขณะที่โฮสต์กำลังทำงานอยู่ในเมนบอร์ด (mainboard) รุ่นเก่านั้นจะมีฟิวส์ต่ออยู่ที่พอร์ตของตัวอุปกรณ์ (Device Port) ถ้าฟิวส์นี้ขาดเมนบอร์ดนั้นก็จะใช้การไม่ได้ แต่เมนบอร์ดรุ่นใหม่ ๆ ได้ทำการแก้ปัญหานี้แล้วแต่ก็ควรหลีกเลี่ยงเพื่อความปลอดภัย

2.2 ทฤษฎีของ คีย์บอร์ด

สแกนโค้ด

ตัวประมวลผล (Processor) ของคีย์บอร์ดนั้น เสียเวลาส่วนใหญ่ไปกับการสแกน (Scanning) หรือการ "Monitoring" ถ้ามันสแกนหาทุกคีย์ที่ถูกกด, ถูกปล่อย หรือกดค้างไว้ คีย์บอร์ดจะส่งข้อมูลในรูปแพคเก็ต (Data Packet) ที่เรียกว่า สแกนโค้ด (Scan Code) ไปยังคอมพิวเตอร์ซึ่ง สแกนโค้ด นั้นมี 2 ชนิด คือเมคโค้ด (Make Code) และ เบรคโค้ด (Break Code) โดยเมคโค้ดจะส่งไปยังคอมพิวเตอร์เมื่อปุ่ม (Key) ถูกปล่อย (Released) ซึ่งทุก ๆ ปุ่มจะมีเมคโค้ดและ เบรคโค้ดเฉพาะของแต่ละปุ่ม ซึ่งคอมพิวเตอร์จะรับไปคำนวณ ประมวลผลได้ว่า เรากดปุ่มใด โดยมีตารางของเมคโค้ดและเบรคโค้ดของทุกปุ่มบนคีย์บอร์ดประกอบกันเป็นตารางเรียกว่า สแกนโค้ดเซต (Scan Code Set) ซึ่งมีอยู่ 3 มาตรฐาน คือ Set1,Set2,Set3 ซึ่งคีย์บอร์ดในปัจจุบันจะถูกตั้งไว้เป็น Set2

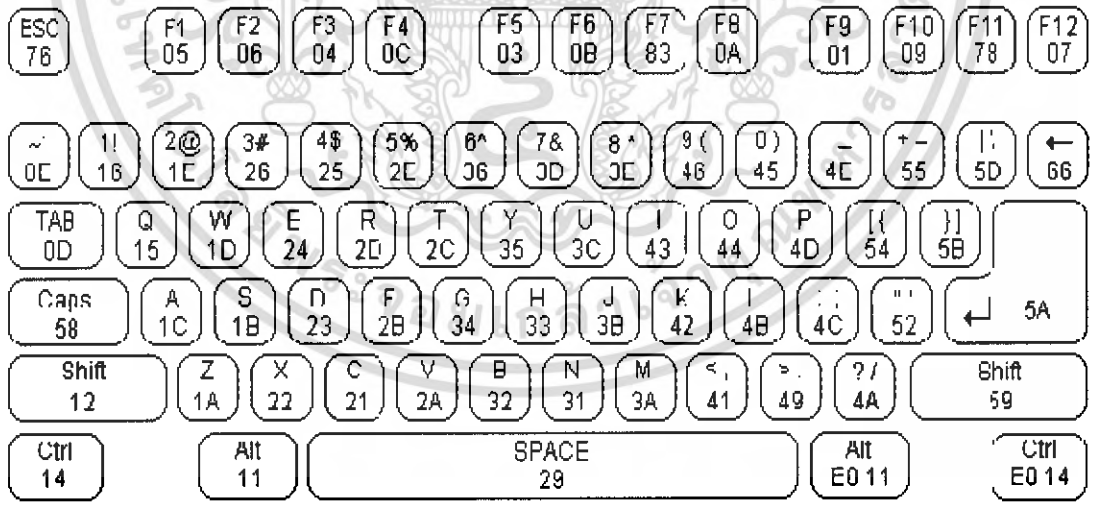
สแกนโค้ด Set 1 : เป็นมาตรฐานคีย์บอร์ดแบบ XT

สแกนโค้ด Set 2 : คีย์บอร์ดในปัจจุบันใช้มาตรฐานนี้

สแกนโค้ด Set 3 : เป็นตารางเพิ่มเติมของคีย์บอร์ดแบบ PS/2 (ไม่ค่อยถูกใช้งาน)

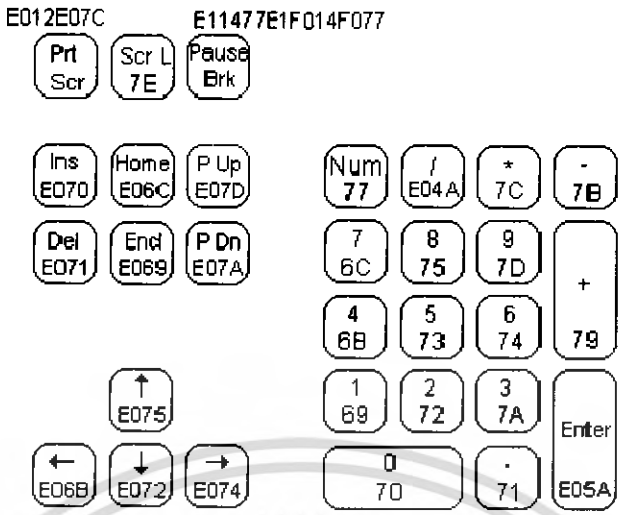
Make Codes, Break Codes and Typematic Repeat:

เมื่อใดก็ตามที่ปุ่มบนคีย์บอร์ดถูกกด เมคโค้ดจะถูกส่งไปยังคอมพิวเตอร์ ซึ่งเมคโค้ดของแต่ละปุ่มจะต่างกัน และมีลักษณะเฉพาะสำหรับแต่ละปุ่มและสแกนโค้ดจะไม่สัมพันธ์กับรหัส ASCII เลย มันเป็นหน้าที่ของคอมพิวเตอร์ที่ทำการแปลงสแกนโค้ด ที่ได้รับให้เป็นตัวอักษร (Character) หรือ คำสั่ง (Command)



รูปที่ 2.2 โค้ดตัวอักษร

ดังนั้น เมคโค้ดส่วนมากจะมีขนาด 1 ไบต์ ในสแกนโค้ด Set 2 มีอยู่บางกลุ่มเท่านั้นที่ เมคโค้ด จะมีขนาด 2 ไบต์หรือ 4 ไบต์พวกนี้ถูกเรียกว่า เอ็กซ์เทนด์คีย์ (Extended Keys) ซึ่งเมคโค้ดของเอ็กซ์เทนด์โค้ด (Extended Code) นั้นจะขึ้นต้นด้วย E0H เสมอ



รูปที่ 2.3 เอ็กซ์เทนด์โค้ด

เมื่อเมคโค้ดถูกส่งไปยังคอมพิวเตอร์ เมื่อใดก็ตามที่กดปุ่ม เบรคโค้ดถูกส่งไป เมื่อใดก็ตามที่ปุ่มถูกปล่อย ซึ่งทุกปุ่มจะมีเมคโค้ดและเบรคโค้ดเฉพาะตัวของใครของมัน โดยที่เมคโค้ดกับเบรคโค้ดสัมพันธ์กัน โดยส่วนใหญ่ในสแกนโค้ด Set 2 นั้น เมคโค้ดจะประกอบด้วยความยาว 1 ไบต์ แล้วเบรคโค้ดประกอบด้วย 2 ไบต์ โดยไบต์แรกของ เบรคโค้ด จะเป็น F0 แล้วไบต์ที่ 2 จะเป็นเมคโค้ดของปุ่มนั้น

ตัวอย่าง

ตารางที่ 2.1 ตัวอย่างโค้ด

Key	Make code	Break Code
"A"	1C	F0,1C
"5"	2E	F0,2E
"F10"	09	F0,09
"Right Arrow"	E0,74	E0,F0,74
"Ctrl" ขวา	E0,14	E0,F0,14

เช่น

ถ้าเราต้องการอักษร "A" ให้ไปปรากฏในไมโครซอฟท์เวิร์ด (Microsoft Word) จะส่งข้อมูลไปยังคอมพิวเตอร์อย่างไร

ในกรณีนี้ เราต้องทำการกดปุ่ม "Shift" ค้างไว้ แล้วกดปุ่ม "A" จากนั้นปล่อยปุ่ม "A" หลังจากทำการปล่อยปุ่ม "A" แล้วทำการปล่อยปุ่ม "Shift" สแกนโค้ดเกี่ยวข้องกับเหตุการณ์ข้างต้นนี้ตามนี้

เมคโค้ด ของปุ่ม "Shift" (12H) ถูกส่งไปยังคอมพิวเตอร์

เมคโค้ด ของปุ่ม "A" (1CH) ถูกส่งไปยังคอมพิวเตอร์

เบรคโค้ด ของปุ่ม "A" (F0H, 1CH) ถูกส่งไปยังคอมพิวเตอร์

เบรคโค้ด ของปุ่ม "Shift" (F0H, 12H) ถูกส่งไปยังคอมพิวเตอร์

ดังนั้นสามารถสรุปข้อมูลที่ถูกส่งไปยังคอมพิวเตอร์ในกรณีนี้คือ 12H, 1CH, F0H, 1CH, F0H, 12H ถ้าเรากดปุ่มแล้ว เมคโค้ดของปุ่มนั้นก็จะส่งไปยังคอมพิวเตอร์ เมื่อกดปุ่มค้างไว้ปุ่มนั้นก็จะกลายเป็น "Typematic" หมายความว่าคีย์บอร์ดจะเก็บเมคโค้ดนั้นไว้จนกว่าปุ่มจะถูกปล่อย หรือปุ่มอื่นจะถูกกด เราตรวจสอบได้โดยเปิดโปรแกรมประเภท Text Editor และกดปุ่ม "A" ค้างไว้ เมื่อเรากดครั้งแรกตัวอักษร "A" จะปรากฏบนหน้าจอ หลังจากช่วงเวลาดีเลย์ (Delay) สั้น ๆ ตัวอักษร "A" ตัวอื่นก็ปรากฏตามมาจนกระทั่งเราจะปล่อยปุ่ม "A" ในเหตุการณ์นี้มี 2 ตัวแปร ที่สำคัญ คือ

1. Typematic Delay ซึ่งเป็นระยะหน่วงเวลาสั้นๆ (Short Delay) ระหว่างอักษร "A" ตัวที่ 1 และอักษร "A" ตัวที่ 2

2. Typematic Rate ซึ่งเป็นค่าอัตราตัวอักษรต่อนาที (Character per second) ว่ามีจำนวนเท่าใดจะปรากฏบนจอหลังจาก Typematic Delay

Typematic Delay อยู่ในช่วง 0.25 นาที – 1.00 นาที และ Typematic Rate อยู่ในช่วง 2.0cps – 30.0 cps (cps: Character per second) เราอาจสามารถเปลี่ยน Typematic Rate และ Typematic Delay โดยใช้คำสั่ง "Set Typematic Rate/Delay"

Typematic data จะไม่ถูกเก็บไว้ในคีย์บอร์ด ในกรณีที่มีมากกว่า 1 ปุ่มถูกกดค้างไว้ ซึ่งจะเป็นปุ่มสุดท้ายที่ถูกกดเท่านั้นที่จะกลายเป็น Typematic โดย Typematic เกิดขึ้นช้าๆ แล้วหยุดเมื่อปุ่มนั้นถูกปล่อย

หมายเหตุ ปุ่ม "Pause/Break" จะไม่มีเบรคโค้ดในสแกนโค้ด Set 1, 2 เมื่อปุ่มนี้ถูกกดจะส่งเมคโค้ดของมันไปยังคอมพิวเตอร์ เมื่อปล่อยปุ่มนี้มันจะไม่ส่งอะไรไปยังคอมพิวเตอร์

รีเซต(Reset)

เมื่อเปิดเครื่องคอมพิวเตอร์หรือ Software Reset คีย์บอร์ดจะโหลด BAT (Basic Assurance Text) และโหลดส่งมาดังนี้

Typematic Delay = 500 ms

Typematic Rate = 10.9 cps

สแกนโค้ดเซต 2 (Set 2)

เมื่อเข้าสู่กระบวนการ BAT คีย์บอร์ดจะเปิดการทำงานของ LED ทั้ง 3 (Num Lock, Caps Lock, Scroll lock) และปิดมันเมื่อกระบวนการ BAT เสร็จสิ้น ณ เวลานี้จะส่ง Code 0xAA แทนการทำกระบวนการ BAT สำเร็จ และจะส่ง code 0xFC เมื่อเกิด error ไปยังคอมพิวเตอร์ โดยที่ code 0xAA (BAT successful) ต้องส่งไปในเวลา 500-700 ms หลังจาก power-on

คีย์บอร์ดหลายแบบที่ละทิ้งการตรวจสอบคล็อก(Clock) และดาต้าไลน์ (Data line) ของมันจนกระทั่งหลังจาก กระบวนการ BAT เสร็จสิ้น (code 0xAA ถูกส่งไปยังคอมพิวเตอร์) ดังนั้นเงื่อนไขการยืนยัน (Inhibit) (Clock line เป็น Low) อาจจะไม่ขัดขวางจากการที่กำลังส่ง code 0xAA (BAT successful) ไป

การสื่อสารข้อมูล

การสื่อสารข้อมูลประกอบด้วย

- 1 บิตเริ่มต้น (Start bit) ซึ่งเป็น 0 เสมอ
- 8 บิตข้อมูล (Data bits)
- 1 พาริตี้บิตคี่ (Parity bit odd)
- 1 สต็อพบิต (Stop bit) ซึ่งเป็น 1 เสมอ

สัญญาณนาฬิกาและสัญญาณข้อมูล

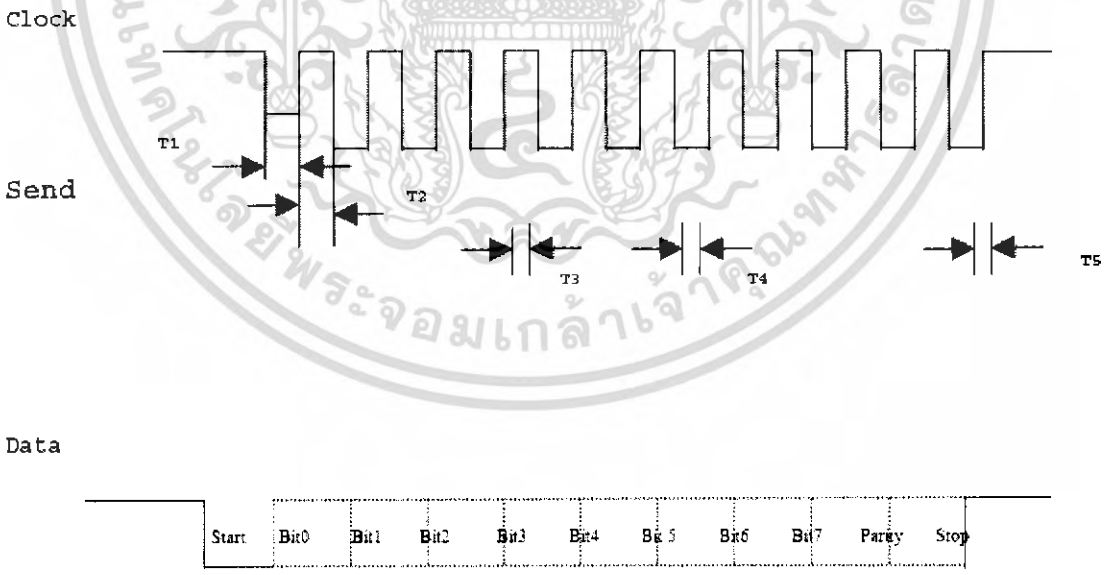
คีย์บอร์ดและระบบการสื่อสารบน Clock และ Data line แหล่งข้อมูลของแต่ละ line เป็นอุปกรณ์ open-drain บนคีย์บอร์ดหรือระบบทำให้อยู่ในสถานะ Low เมื่อไม่มีการติดต่อสื่อสารเกิดขึ้น Clock และ Data line จะเป็น High โดย pull-up resistors

เมื่อระบบส่งข้อมูลไปยังคีย์บอร์ด ทำให้ Data line เป็น Low จนกระทั่งคีย์บอร์ดเริ่มส่ง Clock data stream

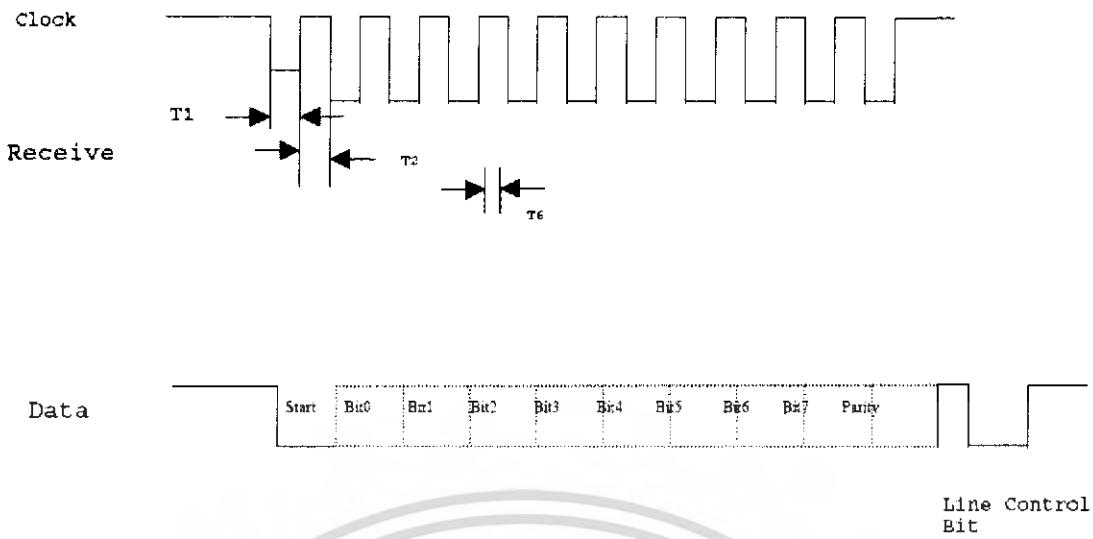
Clock line คีย์บอร์ดประกอบด้วยสัญญาณนาฬิกาใช้ข้อมูล Clock serial ทั้งขาเข้าและขาออกจากคีย์บอร์ด ถ้าระบบ Host ให้ Clock line เป็น Low คีย์บอร์ดจะส่งคำสั่งยับยั้ง

เมื่อคีย์บอร์ดส่งหรือรับข้อมูลจากระบบ จะสร้างสัญญาณนาฬิกาไปยังข้อมูลเวลา ระบบสามารถป้องกันคีย์บอร์ดจากการส่งข้อมูลโดยให้ Clock line เป็น Low Data line อาจจะเป็น Low หรือ High ในระหว่างนี้การทำBAT คีย์บอร์ดจะยอมให้ Clock และ Data line เป็น High

Timing diagram



รูปที่ 2.4 ไทม์มิ่งไดอะแกรม



รูปที่ 2.4 ไทม์มิ่งไดอะแกรม(ต่อ)

T1 30<50 µs

T2 30<50 µs

T3 5<25 µs

T4 5<T2-5 µs

T5 0<25 µs

คีย์บอร์ดส่งข้อมูล

เมื่อคีย์บอร์ดพร้อมที่จะส่งข้อมูล อันดับแรกจะทำการตรวจสอบสำหรับยับยั้งคีย์บอร์ดหรือระบบร้องขอการส่งสถานะบน Clock และ Data line ถ้า Clock line เป็น Low ข้อมูลจะถูกเก็บในบัฟเฟอร์ของคีย์บอร์ด และคีย์บอร์ดรับข้อมูล

ถ้า Clock และ Data line ทั้งคู่เป็น High คีย์บอร์ดจะส่ง Start bit (0),บิตข้อมูล 8 บิต,พาริตี้บิตและ Stop bit ข้อมูลจะเข้าถึงขอบขาขึ้นและขอบขาลงของ Clock pulse ระหว่างการส่งคีย์บอร์ตจะตรวจสอบ Clock line สำหรับสถานะ Low อย่างน้อยทุกๆ 60 วินาที ถ้าระบบมีสถานะต่ำกว่า Clock line หลังจากคีย์บอร์ดเริ่มส่งข้อมูลสุดท้าย ถ้ามีการโต้แย้งเกิดขึ้นคีย์บอร์ดจะหยุดส่งข้อมูล ถ้าเกิดการโต้แย้งก่อนขอบขาลงของสัญญาณ Clock ลูกที่ 10 (พาริตี้บิต) คีย์บอร์ดบัฟเฟอร์จะกลับ Clock และ Data line เป็นสถานะ High ถ้าไม่เกิดการโต้แย้งจาก Clock ลูกที่ 10 คีย์บอร์ดจะเสร็จสิ้นการส่ง ระบบอาจจะร้องขอหรือไม่ทำการร้องขอคีย์บอร์ดให้ส่งข้อมูลมาใหม่ ภายหลังการส่งระบบสามารถยับยั้งคีย์บอร์ดจนกระทั่งระบบปฏิบัติการป้อนข้อมูลหรือจนกระทั่งร้องขอผลตอบรับที่ส่งไป

คีย์บอร์ดรับข้อมูล

เมื่อระบบพร้อมที่จะส่งข้อมูลไปยังคีย์บอร์ด อันดับแรกจะทำการตรวจสอบ ถ้าคีย์บอร์ตกำลังส่งข้อมูลแต่ไม่มีการเข้าถึงสัญญาณ Clock ลูกที่ 10 ระบบสามารถมองข้ามเอาท์พุตของคีย์บอร์ต โดยให้คีย์บอร์ด Clock line เป็น Low ถ้าคีย์บอร์ดส่งสัญญาณ Clock ลูกที่ 10 ต่อไป ระบบต้องรับการส่ง

ถ้าคีย์บอร์ดไม่มีการส่งหรือระบบทำการยกเลิกเอาท์พุตของคีย์บอร์ด ระบบจะบังคับให้ Clock line คีย์บอร์ดเป็น Low ด้วยเวลามากกว่า 60 ไมโครวินาที ขณะเตรียมการส่งข้อมูล เมื่อระบบพร้อมส่งบิตเริ่มต้น (Start bit) Data line จะเป็น Low ส่งผลให้ Clock line เป็น High

คีย์บอร์ดจะตรวจสอบสถานะของ Clock line ที่เว้นช่วงเวลาไม่เกิน 10 มิลลิวินาที ถ้าระบบร้องขอการส่ง (RTS) จะถูกพบ คีย์บอร์ดจะนับ 11 บิต หลังจากบิตที่ 10 คีย์บอร์ดจะตรวจสอบสำหรับ High บน Data line และถ้า line เป็น High จะถูกทำให้เป็น Low และนับอีกหนึ่งบิต การกระทำสัญญาณนี้ระบบคีย์บอร์ดจะรับข้อมูล ในเวลาที่รับสัญญาณระบบจะกลับสู่สถานะเดิม ในกรณีนี้คีย์บอร์ดจะยอมรับเอาท์พุตหรือไปทำการยับยั้งจนกระทั่งพร้อม

ถ้าคีย์บอร์ดพบว่า Data line มีสถานะเป็น Low หลังจากบิตที่ 10 จะมีความผิดพลาดของเฟรมเกิดขึ้น และคีย์บอร์ดจะนับต่อจนกระทั่ง Data line เป็น High คีย์บอร์ดจะสร้าง Data line เป็น Low แล้วทำการส่งอีกครั้ง

แต่ละระบบคำสั่งหรือการส่งข้อมูลไปยังคีย์บอร์ดจะร้องขอผลตอบรับจากคีย์บอร์ด ก่อนที่ระบบจะสามารถส่งต่อไปยังเอาท์พุต คีย์บอร์ดจะตอบสนองภายใน 20 มิลลิวินาที อย่างน้อยที่สุดระบบจะป้องกันเอาท์พุตของคีย์บอร์ด ถ้าคีย์บอร์ดตอบสนองผิดหรือเกิดพาริตี้ผิดพลาด ระบบจะส่งคำสั่งหรือข้อมูลอีกครั้ง อย่างไรก็ตาม คำสั่ง 2 ไบต์จะร้องขอกระบวนการส่งพิเศษ ถ้าคำสั่งเป็น F3 (จะเซตอัตราเร็วต่อดีเลย์), F0 (เลือก Alternate Scan Codes) หรือ ED (จะทำการเซตหรือรีเซตโหมด) เป็นการส่งหรือตอบรับ (Acknowledged) และประเมินไบต์ว่ามีการส่งหรือตอบสนองผิดหรือมีพาริตี้ผิดพลาด ระบบจะส่งทั้งคำสั่งและประเมินไบต์อีกครั้ง

คีย์บอร์ดบัฟเฟอร์

16 ไบต์แรกและสุดท้าย (First- In-First-Out; FIFO) บัฟเฟอร์ในคีย์บอร์ดจะเก็บค่าสแกนโค้ดจนกระทั่งระบบบัฟเฟอร์พร้อมที่จะรับ

สภาพการล้นของบัฟเฟอร์จะเกิดขึ้นเมื่อมีมากกว่า 16 ไบต์ในคีย์บอร์ดบัฟเฟอร์ โค้ดที่เกินมาจะแทนที่ในไบต์ที่ 17 ถ้าคีย์มากกว่า ก่อนที่ระบบจะเอาท์พุตจะเพิ่มข้อมูลที่สูญเสีย

เมื่อคีย์บอร์ดยินยอมส่งข้อมูล ไบต์ในบัฟเฟอร์จะส่งเป็นปกติและข้อมูลที่เข้ามาจะถูกตรวจจับ (Detect) และส่งโค้ดตอบกลับ จะไม่แสดงตำแหน่งบัฟเฟอร์

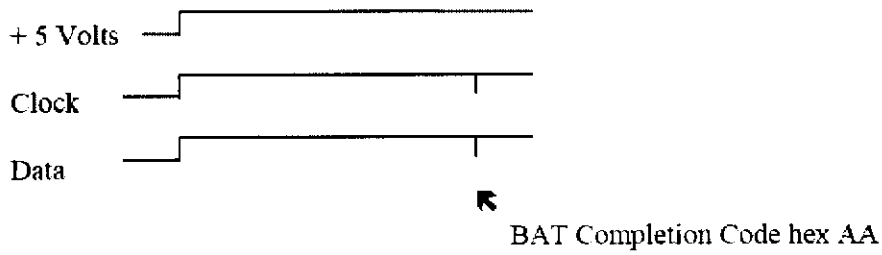
ถ้าคีย์บอร์ดสร้างไบต์หลายระดับ ลำดับทั้งหมดต้องพอดีกับที่ว่างในบัฟเฟอร์ Keystroke จะถูกทิ้งและจะเกิดบัฟเฟอร์ล้น

POWER-ON-ROUTINE

Power-On Reset

คีย์บอร์ดจะสร้างสัญญาณ Power On Reset (POR) เมื่อพลังงานแรกปรากฏที่คีย์บอร์ด POR ที่เกิดขึ้นจะใช้เวลาอย่างน้อยที่สุด 150 มิลลิวินาที และมากที่สุด 2.0 วินาที เวลาของพลังงานแรกที่ปรากฏไปยังคีย์บอร์ด

ชนิดของ Power diagram



x ---- 500 ms -------- 345 ms -----x
POR DELAY BAT +/-20%

รูปที่ 2.5 พาวเวอร์ไดอะแกรม

การทดสอบขั้นพื้นฐาน (Basic Assurance Test)

Basic Assurance Test (BAT) ประกอบจากระบบทดสอบการปฏิบัติการคีย์บอร์ด จะตรวจสอบผลรวมของรอมและแรม ระหว่างการ BAT การกระทำของ Clock และ Data line จะถูกปฏิเสธ LEDs จะติดเมื่อเริ่มต้น และดับในระหว่างการทำ BAT จะใช้เวลาน้อยที่สุด 300 มิลลิวินาที และมากที่สุด 500 มิลลิวินาที การเพิ่มเวลาจะร้องขอโดย POR

โค้ด AA จะถูกส่งไปยังระบบและเริ่มสแกนคีย์บอร์ด ถ้า BAT เกิดความล้มเหลว คีย์บอร์ดจะส่งโค้ดที่ผิดพลาดไปยังระบบ คีย์บอร์ดไม่สามารถส่งคำสั่งเข้าไปได้ในระหว่างนั้น เมื่อโค้ดสมบูรณ์ถูกส่งในช่วงเวลาระหว่าง 450 มิลลิวินาที ถึง 2.5 วินาทีหลังจากการ POR หลังจากคำสั่งรีเซตถูกตอบรับ (acknowledged)

คำสั่งต่างๆ

คำสั่งควบคุมที่ถูกส่งจากพีซีไปยังคีย์บอร์ด(Commands from the System)

ตารางที่ 2.2 แสดงคำสั่งต่างๆที่ถูกส่งด้วยเลขฐาน 16

คำสั่ง	เลขฐาน 16
Set/Reset Status Indicators	ED
Echo	EE
Invalid Command	EF
Select Alternate Scan Codes	F0
Invalid Command	F1
Read ID	F2
Set Typematic Rate/Delay	F3
Enable	F4
Default	F5
Set Default	F6

ตารางที่ 2.2 แสดงคำสั่งต่างๆที่ถูกส่งด้วยเลขฐาน 16 (ต่อ)

คำสั่ง	เลขฐาน 16
Set All Keys - Typematic	F7
- Make/Break	F8
- Make	F9
- Typematic/Make/Break	FA
Set Key Type - Typematic	FB
- Make/Break	FC
- Make	FD
Resend	FE
Reset	FF

คำสั่งต่างๆ จะถูกส่งไปยังคีย์บอร์ดในเวลาต่างกัน คีย์บอร์ดจะตอบรับภายใน 20 มิลลิวินาที ยกเว้นเมื่อกำลังทำ BAT หรือส่งคำสั่งรีเซต

Default (F5)

เป็นคำสั่งรีเซตเงื่อนไขทั้งหมดไปยังพาวเวอร์ของสถานะที่ไม่สามารถทำได้ คีย์บอร์ดจะตอบ ACK, เคลียร์เอาท์พุตบัฟเฟอร์, เซตคีย์ที่ไม่สามารถพิมพ์ได้ (สแกนโค้ดเซตการปฏิบัติการ 3 ครั้งเท่านั้น) และอัตราเร็วในการพิมพ์ต่อดีเลย์ เคลียร์การพิมพ์ครั้งล่าสุด คีย์บอร์ดจะหยุดสแกนและรอคำสั่งถัดไป

Echo (EE)

เมื่อคีย์บอร์ดรับคำสั่งนี้ จะส่ง EE ตอบกลับมาและคีย์บอร์ดจะสแกนต่อเนื่อง

Enable (F4)

ในการตอบรับคำสั่งนี้ คีย์บอร์ดจะตอบ ACK และเคลียร์เอาท์พุตบัฟเฟอร์ เคลียร์การพิมพ์ครั้งล่าสุด

Invalid Command (EE และ F1)

คำสั่ง EE และ F1 จะไม่ถูกรองรับถ้าคำสั่งใดคำสั่งหนึ่งถูกส่ง คีย์บอร์ดจะไม่ตอบ ACK แต่กลับส่งคำสั่งและเริ่มสแกนต่อเนื่อง ระหว่างนี้ไม่มีการกระทำอื่นเกิดขึ้น

Read ID (F2)

คำสั่งนี้จะร้องขอข้อมูลที่ระบุจากคีย์บอร์ด คีย์บอร์ดจะตอบ ACK การสแกนไม่ต่อเนื่องและส่งคีย์บอร์ด ID 2 ไบต์ ไบต์ที่ 2 ต้องตามหลังไบต์แรกที่เสร็จสมบูรณ์ โดยใช้เวลาไม่เกิน 500 ไมโครวินาที หลังจากเอาท์พุตของไบต์ที่ 2 คีย์บอร์ดจะกลับสู่การสแกนอีกครั้ง

Resend (FE)

ระบบจะส่งคำสั่งนี้เมื่อพบความผิดพลาดในการส่งจากคีย์บอร์ด เมื่อได้รับ Resend คีย์บอร์ดจะส่งเอาท์พุตอีกครั้งก่อน

Reset (FF)

ระบบจะออกคำสั่งรีเซตไปยังการเริ่มต้นรีเซตโปรแกรมและคีย์บอร์ดเป็นการทดสอบภายในด้วยตนเอง คีย์บอร์ดจะตอบ ACK และรับรองระบบคอบรับ ACK ก่อนปฏิบัติคำสั่งเสร็จ ระบบจะรับคำสั่ง ACK โดยยกระดับ Clock และ Data line สำหรับเวลาน้อยที่สุด 500 ไมโครวินาที คีย์บอร์ดจะไม่สามารถทำอย่างอื่นได้ในช่วงที่รับคำสั่งรีเซต จนกระทั่ง ACK ถูกรับ หรือจนกระทั่งคำสั่งอื่นถูกส่งมาก่อนคำสั่งนั้น

Select Alternate Scan (F0)

คีย์บอร์ดจะเลือก 1 ใน 3 เซตของสแกนโค้ด คีย์บอร์ดจะตอบรับคำสั่งนี้ด้วย ACK, เคลียทั้งเอาท์พุตบัฟเฟอร์และการกดคีย์ (ถ้า 1 เป็น active) ระบบจะส่งไบต์และตอบรับคีย์บอร์ดอื่นๆเป็น ACK ทางเลือกของไบต์คำสั่งจะมีคำสั่ง 01 จะเลือกสแกนโค้ดให้เป็น 1 คำสั่ง 02 เลือกให้เป็น 2 และคำสั่ง 03 ให้เป็น 3

ถ้าส่งเลขฐาน 16 เป็น 00 คีย์บอร์ดจะตอบ ACK และส่งไบต์ไปบอกระบบว่าสแกนโค้ดพร้อมใช้งาน หลังจากสแกนโค้ดให้มาถูกเซต คีย์บอร์ดจะกลับสู่การสแกน ก่อนการรับคำสั่งโค้ด Select Alternate Scan

Set All Keys (F7,F8,F9,FA)

คำสั่งเหล่านี้จะสั่งคีย์บอร์ดให้เซตทุกคีย์ คีย์บอร์ดจะตอบสนองด้วย ACK, เคลียร์เอาท์พุทบัฟเฟอร์ ให้ทุกคีย์พิมพ์ข้อมูลที่ระบุโดยคำสั่งและสแกนต่อเนื่อง คำสั่งทั้งหมดสามารถถูกส่งโดยใช้สแกนโค้ด

ตารางที่ 2.3 แสดงโค้ดคำสั่งของ Set All Keys

เลขฐาน 16	คำสั่ง
F7	Set All Keys – Typematic
F8	Set All Keys - Make/Break
F9	Set All Keys - Make
FA	Set All Keys - Typematic/Make/Break

Set Default(F6)

คำสั่งนี้จะทำการรีเซตเงื่อนไขทั้งหมดไปยังพลังงาน default state คีย์บอร์ดจะตอบ ACK, เคลียร์เอาท์พุตบัฟเฟอร์ กำหนด Default Key Types (สแกนโค้ดเพียง 3 ครั้งเท่านั้น) และพิมพ์ด้วยอัตราเร็วต่อดีเลย์ เคลียร์การพิมพ์ครั้งล่าสุดและสแกนต่อเนื่อง

Set Key Type (FB, FC, FD)

คำสั่งเหล่านี้จะสั่งการคีย์บอร์ดให้เซต Individual keys ดังรายการข้างล่าง

ตารางที่ 2.4 แสดงโค้ดคำสั่งของ Set Key Type

เลขฐาน 16	คำสั่ง
FB	Set Key Type – Typematic
FC	Set Key Type - Make/Break
FD	Set Key Type - Make

คีย์บอร์ดจะตอบ ACK , เคลียร์เอาท์พุตบัฟเฟอร์และเตรียมรับคีย์ที่ระบุ การระบุคีย์จะทำโดยระบบจะระบุแต่ละคีย์ด้วยค่าสแกนโค้ด ซึ่งกำหนดให้สามารถสแกนโค้ดเซต 3 เฉพาะค่าสแกนโค้ดเซต 3 ที่สามารถใช้กับการระบุค่าคีย์ คำสั่งเหล่านี้สามารถส่งโดยใช้สแกนโค้ดเซตใดก็ได้ แต่ส่งผลเฉพาะสแกนโค้ดเซต 3

Set/Reset Status Indicators (ED)

สถานะทั้ง 3 ของคีย์บอร์ด ได้แก่ Num Lock, Caps Lock และ Scroll Lock คีย์บอร์ดจะทำงานหรือไม่กระทำการใดๆ เมื่อได้รับโค้ดที่ถูกต้องจากระบบ คำสั่งจะเริ่มด้วยไบต์คำสั่ง (ED) คีย์บอร์ดจะตอบรับคำสั่งด้วย ACK การสแกนยังเป็นแบบไม่ต่อเนื่องและรอทางเลือกไบต์จากระบบ การกำหนดบิตสำหรับไบต์ดังตาราง

ตารางที่ 2.5 แสดงการกำหนดบิตในสถานะต่างๆ

บิต	ค่าที่ระบุ
0	Scroll Lock Indicator
1	Num Lock Indicator
2	Caps Lock Indicator
3-7	Reserved (bust be 0s)

ถ้าบิตเป็น 1จะเป็นการเปิด ถ้าเป็น 0 จะเป็นการปิด

คีย์บอร์ดจะตอบสนอง Option byte ด้วย ACK จะเซตค่า Indicator และถ้าก่อนหน้านี้เป็น Enable จะทำการสแกนต่อไป

Set Typematic Rate/Delay

ระบบกำหนดคำสั่ง Set Typematic Rate/Delay เพื่อเปลี่ยนแปลงอัตราการพิมพ์ และดีเลย์ คีย์บอร์ดจะตอบสนองคำสั่งด้วย ACK, หยุดการสแกนและรอระบบในการส่งอัตราเร็วและดีเลย์ในไบต์ด้วย ACK อื่น กำหนดค่าอัตราและดีเลย์ไปยังค่าที่กำหนดและทำการสแกนต่อเนื่อง บิต 6 และ 5 กำหนดเป็นดีเลย์ และบิต 4,3,2,1 และ 0 บิต 7 เป็นบิตซ้ายสุดเป็น 0 เสมอ ดีเลย์เป็น 1 บวกด้วยค่าไบนารี(Binary) ของบิต 6 และ 5 คูณด้วย 250 มิลลิวินาที ±20%

อัตราการพิมพ์ (เมคโค้ดต่อวินาที) เป็น 1 สำหรับคาบและรายการในตาราง

ตารางที่ 2.6 แสดงอัตราเร็วการพิมพ์กับจำนวนบิต

บิต	อัตราเร็วการพิมพ์ ±20%	บิต	อัตราเร็วการพิมพ์ ±20%
0	30	10000	7.5
1	26.7	10001	6.7
10	24	10010	6
11	21.8	10011	5.5
100	20	10100	5
101	18.5	101	4.6
110	17.1	10110	4.3
111	16	10111	4
1000	15	11000	3.7
1001	13.3	11001	3.3
1010	12	11010	3
1011	10.9	11011	2.7
1100	10	11100	2.5
1101	9.2	11101	2.3
1110	8	11110	2.1
1111	8	11111	2

อัตราการพิมพ์ = 10.9 ตัวอักษรต่อวินาที ± 20%

ดีเลย์ = 500 มิลลิวินาที ± 20%

การปฏิบัติการคำสั่งนี้จะหยุดโดยปราศจากการเปลี่ยนแปลงอัตราการกระตุ้น ถ้าคำสั่งอื่นถูกรับแทนของอัตราการเปลี่ยนแปลงต่อดีเลย์ไบต์

คำสั่งควบคุมที่ถูกส่งจากคีย์บอร์ดไปยังพีซี (Commands to the System)

ตารางที่ 2.7 แสดงคำสั่งที่คีย์บอร์ดจะส่งไปยังระบบด้วยเลขฐานหก

คำสั่ง	เลขฐาน 16
Key Detection Error/Overrun	00(Code Sets 2 and 3)
Keyboard ID	83AB
BAT Completion Code	AA
BAT Failure Code	FC
Echo	EE
Acknowledge (ACK)	FA
Resent	FE
Key Detection Error/Overrun	FF(Code Set 1)

คำสั่งคีย์บอร์ดจะส่งไปยังระบบตามคำอธิบายข้างล่างเรียงลำดับตามตัวอักษร

Acknowledge (FA)

คีย์บอร์ดจะส่ง ACK ไปยังค่าอินพุตที่ถูกต้อง ถ้าคีย์บอร์ดถูกแทรกด้วยคำสั่ง Echo และ Resend ระหว่างสั่ง ACK จะละทิ้ง ACK และตอบรับคำสั่งใหม่

BAT Completion Code (AA)

ตามความสมบูรณ์ของ BAT คีย์บอร์ดจะส่ง AA และโค้ดอื่นจะถูกล้มเลิก

BAT Failure (FC)

ถ้า BAT มีการล้มเหลวเกิดขึ้น คีย์บอร์ดจะส่งโค้ดนี้ การสแกนจะไม่ต่อเนื่องและรอสำหรับระบบตอบรับหรือทำการรีเซต

Echo (EE)

คีย์บอร์ดส่งโค้ดนี้ในการตอบรับคำสั่ง Echo

Keyboard ID (83AB)

คีย์บอร์ด ID ประกอบด้วย 2 ไบต์ 83AB คีย์บอร์ดจะตอบการอ่าน ID ด้วย ACK และสแกนไม่ต่อเนื่อง และส่ง ID 2 ไบต์ Low ไบต์จะถูกส่งก่อนตามด้วย High ไบต์ หลังจากเอาท์พุทของคีย์บอร์ต ID คีย์บอร์ดจะเริ่มทำการสแกน

Key Detection Error (00 หรือ FF)

คีย์บอร์ดจะส่งคีย์ป้องกันความผิดพลาดของตัวอักษรถ้าคีย์บอร์ดทำการยุติสวิตช์ที่ไม่สามารถระบุได้ ถ้าคีย์บอร์ดใช้สแกนโค้ดเซต 1 โค้ดคือ FF สำหรับเซต 2 และ 3 โค้ดคือ 00

Overrun (00 หรือFF)

การ Overrun ตัวอักษรถูกกำหนดตำแหน่งในคีย์บอร์ดบัฟเฟอร์และแทนที่โค้ดตัวสุดท้ายเมื่อความจุของบัฟเฟอร์ที่เกิน โค้ดจะถูกส่งไปยังระบบเมื่อไปถึงส่วนบนสุดของบัฟเฟอร์ ถ้าคีย์บอร์ดกำลังใช้สแกนโค้ดเซตเป็น 1 โค้ดเป็น FF สำหรับเซต 2 และ 3 โค้ดเป็น 00

Resent (FE)

คีย์บอร์ดจะส่งคำสั่ง Resent ตามด้วยการรับค่าอินพุตที่ผิดหรือพาริตี้อินพุตผิดพลาดถ้าระบบไม่ส่งอะไรเลยไปยังคีย์บอร์ด จะไม่มีผลตอบสนองถูกร้องขอ

การเริ่มทำงานของคีย์บอร์ดเมื่อเริ่มเปิดเครื่องคอมพิวเตอร์

Keyboard:	AA Self-test passed	;Keyboard controller init
Host:	ED Set/Reset Status Indicators	
Keyboard:	FA Acknowledge	
Host:	00 Turn off all LEDs	
Keyboard:	FA Acknowledge	
Host:	F2 Read ID	
Keyboard:	FA Acknowledge	
Keyboard:	AB First byte of ID	
Host:	ED Set/Reset Status Indicators	;BIOS init
Keyboard:	FA Acknowledge	
Host:	02 Turn on Num Lock LED	
Keyboard:	FA Acknowledge	
Host:	F3 Set Typematic Rate/Delay	;Windows init
Keyboard:	FA Acknowledge	
Host:	20 500 ms / 30.0 reports/sec	
Keyboard:	FA Acknowledge	
Host:	F4 Enable	
Keyboard:	FA Acknowledge	
Host:	F3 Set Typematic Rate/delay	
Keyboard:	FA Acknowledge	
Host:	00 250 ms / 30.0 reports/sec	
Keyboard:	FA Acknowledge	

The i8042 คีย์บอร์ด Controller

การเขียนโปรแกรมเราไม่ต้องลงไปติดต่อโดยตรงกับคีย์บอร์ด เราก็จะติดต่อกับตัว Controller ของคีย์บอร์ดแทน ตัว controller นี้จะดูแลทุกระดับสัญญาณ และดูแลรายละเอียดของโปรโตคอลดีพอๆ กับการหาวิธีการแปลง การแปล และการจัดการกับสแกนโค้ดและคำสั่ง (Command)

Intel 8042/compatible microcontroller ถูกใช้เป็นตัว controller ในคีย์บอร์ด ในคอมพิวเตอร์รุ่นใหม่ ๆ ตัว microcontroller ที่ซ่อนอยู่ใน Motherboard's Chip Set ซึ่งเป็นการรวมหลายๆ controllers ใน single package กระนั้นอุปกรณ์นี้ก็ยังคงอยู่ และ controller ของคีย์บอร์ด ยังคงหมายถึง "8042"

ขึ้นอยู่กับ Motherboard คีย์บอร์ด Controller อาจทำการใน 1 หรือ 2 mode "AT-compatible mode" หรือ "PS/2 compatible mode" PS/2 compatible mode ถูกใช้ถ้า motherboard นั้นรองรับ PS/2 เมาส์ ถ้ากรณีนี้ 8042 จะเป็นตัวคีย์บอร์ด Controller และเมาส์ Controller , คีย์บอร์ด Controller จะถูกตรวจพบโดยอัตโนมัติ ซึ่ง mode มันจะใช้ตามสายที่ต่อกับคีย์บอร์ด port

8042 จะประกอบด้วย Register ดังนี้

- A one-ไบต์ input buffer – ประกอบด้วย ไบต์ ที่อ่านจากคีย์บอร์ด; read only
- A one-ไบต์ output buffer – ประกอบด้วย ไบต์ ที่อ่านจากคีย์บอร์ด; write only
- A one-ไบต์ status registers – 8 status flags; read only
- A one-ไบต์ control registers – 7 status flags; read/write

3 register แรก (Input, output, status) เป็นการเข้าถึงข้อมูลโดยตรงโดยทาง port 0x60 และ 0x64 รีจีสเตอร์สุดท้าย (control) เป็นการอ่านโดยใช้คำสั่ง "Read command byte" และเขียนโดยใช้คำสั่ง "Write command byte"

การเขียนไปยัง port 0x64 ไม่ต้องระบุ Register บัสส่งคำสั่งไปให้ 8042 แปลได้เลย ถ้าคำสั่งอนุญาตตัวแปร ตัวแปรนี้จะส่งไปที่ port 0x60 เช่นเดียวกัน ผลลัพธ์อื่น จะกลับมาจากการใช้คำสั่งอ่านจาก Port 0x60 ก็ได้

สถานะของรีจิสเตอร์ (Status Register)

สถานะแฟลก ของ 8042 จะอ่านมาจาก port 0x64 มันจะประกอบด้วย error information, status information และ indicate data ไม่ถูกแสดงใน input และ output buffer, flags ถูกกำหนดตามตารางที่ 2.8

ตารางที่ 2.8 สถานะแฟลกของ Status Register

	MSB							**LSB**
AT-compatible mode :	PERR	RxTo	TxTo	INH	A2	SYS	IBF	OBF
PS/2-compatible mode :	PERR	To	MOBF	INH	A2	SYS	IBF	OBF

OBF (Output Buffer Full) ใช้แสดงเมื่อมันตกลงที่จะเขียน output buffer

0 : Output buffer ว่าง - ตกลงเขียนไปยัง Port 0x60

1 : Output buffer เต็ม - ไม่เขียนไปยัง Port 0x60

IBF (Input Buffer Full) ใช้แสดงเมื่ออินพุต สามารถใช้ได้ใน input buffer

0 : Input buffer ว่าง - ไม่มี อินพุต ที่ต้องอ่านจาก Port 0x60

1 : Input buffer เต็ม - อินพุตใหม่ สามารถอ่านจาก Port 0x60

SYS (System flag) ถูกอ่านเมื่อ power on Reset, หรือ software Reset

0 : Power Up value - ระบบอยู่ในกระบวนการ Power on Reset

1 : ได้รับ BAT code - ระบบพร้อมถูก Initialize

A2 (Address line A2) ใช้ภายใน

0 : A2 = 0 - Port 0x60 ถูกเขียนไปล่าสุด

1 : A2 = 1 - Port 0x64 ถูกเขียนไปล่าสุด

INH (Inhibit flag) ใช้แสดงว่าคีย์บอร์ด ไม่ถูกยับยั้งการสื่อสาร

0 : คีย์บอร์ดคลีอก = 0 - คีย์บอร์ดถูกยับยั้ง

0 : คีย์บอร์ดคลีอก = 1 - คีย์บอร์ดไม่ถูกยับยั้ง

2.3 ทฤษฎีของ PS/2 เมาส์

การอินเตอร์เฟสทางไฟฟ้า/โปรโตคอล

PS/2 เมาส์ ใช้โปรโตคอลเดียวกับ PS/2(AT) คีย์บอร์ด เป็นมาตรฐานของ IBM technical reference manual PS/2 เมาส์ สามารถรองรับอินพุตตามนี้

การเคลื่อนที่แนวนอน (X Movement)

การเคลื่อนที่แนวตั้ง (Y Movement)

ปุ่มซ้าย (Left Button)

ปุ่มกลาง (Middle Button)

ปุ่มขวา (Right Button)

เมาส์จะอ่านค่านี้ตามจังหวะความถี่ อัพเดทเคาน์เตอร์ (Counter) และแฟลกหลายๆ ค่าเพื่อส่งกลับค่าของการเคลื่อนที่ และสถานะของปุ่ม ในการทดลองนี้จะครอบคลุมไมโครซอฟ Intellimouse ซึ่งประกอบด้วยมาตรฐานการรองรับอินพุตทั่วไป ทั้งแบบที่มีล้อเลื่อน (Scrolling wheel) และ ปุ่ม 2 ปุ่ม

มาตรฐาน เมาส์ มี 2 ตัวนับเก็บการเคลื่อนที่ คือ ตัวนับแนวนอน และ ตัวนับแนวตั้ง มีค่า 9 บิต 2's complement และแต่ละอันจะเกี่ยวข้องกับโอเวอร์โฟลว์แฟลกตามด้วยสถานะของปุ่มกดทั้ง 3 ปุ่ม จะส่งไปยัง โฮสต์ (Computer) ในรูปแบบของชุดข้อมูล (Data packet) ขนาด 3 ไบต์, ตัวนับ ทั้ง 2 จะแทนจำนวนของการเคลื่อนที่ จะปรากฏตั้งแต่การการเคลื่อนที่ครั้งล่าสุด จะถูกส่งไปยังโฮสต์

เมื่อเมาส์ อ่านค่า อินพุตของมัน มันจะบันทึกสถานะล่าสุดของปุ่ม แล้วตรวจสอบการเคลื่อนที่ ถ้ามีการเคลื่อนไหวเกิดขึ้น มันจะเพิ่ม (+x หรือ +y) หรือลด (-x หรือ –y) ค่าการเคลื่อนที่ของมัน ถ้าเกิดโอเวอร์โฟลว์มันจะ เซต (Set) ที่แฟลกโอเวอร์โฟลว์

ตัวแปรที่คำนวณจำนวนของการ(+x หรือ +y) และ (-x หรือ –y) นั่นคือตัวแปรที่มีชื่อว่า "Resolution" โดยค่าเริ่มต้น ของ Resolution คือ 4 ตัว/ มม. และโฮสต์อาจจะทำการเปลี่ยนค่านี้โดยใช้คำสั่ง "Set Resolution" (0xE8)

มีตัวแปรที่ไม่มีผลกับการเคลื่อนที่ตัวนับทั้ง 2 แต่จะมีผลกับการรายงานค่าของเคาน์เตอร์ ตัวแปรนี้คือ "Scaling" ค่า Default ของ Scaling ของเมาส์ จะใช้เป็น 1:1 Scaling ซึ่งไม่มีผลกับการรายงาน เมาส์

การเคลื่อนที่ อย่างไรก็ตาม โฮสต์อาจจะเลือก 1:2 Scaling โดยใช้คำสั่ง “Set Scaling” 2:1 (0xE7) ถ้า 2:1 Scaling ทำงาน เมาส์จะ Apply ตามตารางที่ 2.9 ไปยังตัวนับก่อนจะส่ง ตัวนับของมันไปยังโฮสต์

ตารางที่ 2.9 การรายงานค่าของเคาน์เตอร์

การเคลื่อนที่ ตัวนับ	Reported การเคลื่อนที่
0	0
1	1
2	1
3	3
4	6
5	9
N > 5	2 x N

การส่ง Data ไปจากอุปกรณ์ต่อร่วม ประกอบด้วย 11 บิต การไหลของข้อมูลถูกส่งแบบอนุกรมบนดาต้าไลน์ (Data line) ตารางข้างล่างนี้ แสดงฟังก์ชันของแต่ละบิต, บิตพาริตี้ (Parity bit) จะเป็น “1” หรือ “0” และบิตข้อมูลมี 8 บิต บวกกับบิตพาริตี้ ทุกครั้งต้องมีเลขคี่ของ “1”

ตารางที่ 2.10 รายละเอียดแต่ละบิตของเมาส์

Bit	Function
11	Stop Bit(always 1)
10	Parity bit(odd parity)
9	Data bit7 (MSB)
8	Data bit6
7	Data bit5
6	Data bit4
5	Data bit3
4	Data bit2
3	Data bit1
2	Data bit0
1	Start Bit(always 0)

การเคลื่อนที่ Data Packet

PS/2 เมาส์ จะส่งการเคลื่อนที่แนวตั้งแนวนอน และสถานะของปุ่มตามชุดข้อมูล ด้านล่างนี้

การเคลื่อนที่ตัวนับจะเป็น 9 บิต 2's Complement integers ซึ่ง MSB(Most Significant Bit) จะปรากฏตาม Sign Bitในไบต์ 1 ของการเคลื่อนที่ data packet ตัว ตัวนับ นี้ จะถูกอัพเดท(update) เมื่อ เมาส์อ่านค่า อินพุตของมัน และค้นหาการเคลื่อนที่ที่เกิดขึ้น ค่าเหล่านี้จะเป็นจำนวนของการเคลื่อนที่ที่ปรากฏตั้งแต่การเคลื่อนที่ data packet ล่าสุด ถูกส่งไปยังโฮสต์ (หลังจากแพคเก็ตหนึ่งถูกส่งไปยังโฮสต์ การเคลื่อนที่ตัวนับ ก็จะทำการ Reset) ช่วงพิสัยของค่านี้สามารถถูกแสดงโดยการเคลื่อนที่ ตัวนับ คือ 0–255 ถึง +255 ถ้าเกินกว่านี้ ก็ถือว่าเกิด Overflow โดย Overflow Bit จะถูก Set และตัวนับจะไม่ถูกเพิ่มขึ้นหรือลดลง จนกระทั่งมันจะ Reset ใหม่อีกครั้ง

ตามที่กล่าวไปแล้วว่า การเคลื่อนที่ตัวนับ จะ Reset เมื่อใดก็ตามที่การเคลื่อนที่ data packet ถูกส่งไปยังโฮสต์เสร็จสิ้นแล้ว จะ Reset หลังจากเมาส์ ได้รับคำสั่งจากโฮสต์ (คำสั่ง "Reset" (0xFE))

Modes of Operation

ในการรายงานข้อมูล จะมี 4 โหมดมาตรฐานของการปฏิบัติการ (Operation)

Reset: เมื่อเปิดเครื่อง (Power up) เริ่มต้นเมาส์จะทำการเตรียมตัวให้พร้อมกับการทำงานของคอมพิวเตอร์(Initialization) และทำการตรวจสอบตัวเอง (Self-diagnostics) หลังจากนั้นจะทำการ Reset mode คือ เมื่อได้รับรับคำสั่ง "Reset" (0xFF) จากโฮสต์

Stream: เป็น default mode (หลังจากทำ Reset Mode แล้ว) ซึ่งเมาส์จำการบอกการเคลื่อนที่ของดาต้าแพคเก็ต (movement data packet) เมื่อมีการเคลื่อนที่เกิดขึ้นหรือมีสถานะการกดปุ่มที่เปลี่ยนไป

Remote: เป็น mode ที่มีประโยชน์ในบางสถานการณ์ คือโฮสต์จะทำการร้องขอสำหรับ movement data packets และอาจจะสั่งโดยใช้คำสั่ง "Set Remote Mode" (0xF0) ไปยังเมาส์

Wrap : โหมดนี้ไม่ได้ใช้ประโยชน์ทางใดทางหนึ่งเฉพาะ ยกเว้นใช้ในการทดสอบ การเชื่อมต่อระหว่างเมาส์กับโฮสต์ , Wrap mode อาจทำโดยการส่งคำสั่ง "Set Wrap Mode" (0xFE) ไปยัง เมาส์ การออกจาก Wrap mode , โฮสต์ ต้องส่งคำสั่ง "Reset" (0xFF) หรือ "Reset Wrap Mode" (0xEC) ถ้าส่งคำสั่ง "Reset" (0xFF) ไปแล้ว เมาส์ ได้รับคำสั่งแล้ว เมาส์ จะทำการเข้าสู่ Reset mode ถ้าใช้คำสั่ง "Reset Wrap Mode" (0xEC) เมื่อเมาส์รับคำสั่งแล้ว เมาส์จะเข้าสู่ Wrap Mode ก่อน

Reset Mode: เมาส์ทำการ Reset mode เมื่อเปิดเครื่อง (Power up) หรือในการส่งสัญญาณ response กับคำสั่ง "Reset" (0xFF) หลังจากเข้าสู่ mode นี้แล้ว เมาส์จะทำการตรวจสอบด้วยตัวมันเองตาม BAT (Basic Assurance Test) และ Set ตามค่า default ด้านล่างนี้

Sample rate	- 100 samples/sec
Resolution	- 4 ตัว/มม.
Scaling	- 1:1
Data Reporting	- Disabled

จากนั้นเมาส์ ก็จะส่ง BAT completion code อย่างใดอย่างหนึ่งไป 0xAA หมายถึง BAT successful, 0xFC หมายถึง Error ถ้า โฮสต์ ได้รับ response อื่นนอกจาก 0xAA มันอาจจะวนรอบการ check ใหม่ และให้เมาส์ Reset กระบวนการ BAT ใหม่ ตาม BAT completion code (0xAA หรือ 0x FC) เมาส์ ส่ง device ID ของมัน คือ 0x00 นี้คือสิ่งที่แยกมันออกจาก คีย์บอร์ด หรือ เมาส์ ใช้งานใน Extended mode อย่างไรก็ตามจะพบว่าไบออส (BIOS) จะทำการส่งคำสั่ง "Reset" (0xFF) ทันทีที่ได้รับค่า 0xAA หลังจากที่ทำการเปิดเครื่อง

หลังจากเมาส์ส่ง Device ID ของมันเสร็จ มันจะเข้าสู่ stream mode ด้านบนดู ค่า default จะมี "Data Reporting Disabled" หมายถึง เมาส์จะไม่ส่ง การเคลื่อนที่ data packet ใดๆไปยัง โฮสต์ จนกระทั่งจะได้รับคำสั่ง "Enable Data Reporting" (0xF4)

Stream Mode: ใน mode นี้ เมาส์ส่งการเคลื่อนที่ data เมื่อมันตรวจจับการเคลื่อนที่ หรือการเปลี่ยนแปลงสถานะของปุ่มกด 1 หรือมากกว่า 1 ปุ่ม ค่า maximum rate นี้ data reporting อาจจะปรากฏตามที่เรารู้จักคือ "Sample rate" ตัวแปรนี้มีช่วงจาก 10 samples/sec ถึง 200 samples/sec ค่าเริ่มต้น ของมันคือ 100 samples/sec และโฮสต์อาจจะเปลี่ยนค่าโดยใช้คำสั่ง "Set Sample Rate" (0xF3) โหมดนี้เป็นโหมดเริ่มต้นของ เมาส์

หมายเหตุ : ถ้าการ report ไม่สามารถทำได้เมื่อเริ่มต้น เมาส์จะไม่มีการส่งค่า movement data packets ออกมาจนกว่าจะได้รับคำสั่ง " Enable Data Reporting" (0xF4)

Remote Mode: ในโหมดนี้ เมาส์จะอ่านค่า อินพุตของมัน และ update ค่าของตัวนับ/flag ณ sampling rate ล่าสุด แต่มันจะแจ้งเฉพาะ การเคลื่อนที่ (และการเปลี่ยนสถานะของปุ่ม) เมื่อข้อมูลถูกร้องขอโดยโฮสต์ โฮสต์จะส่งคำสั่ง "Read data" (0xEB) หลังจากรับคำสั่งนี้แล้ว เมาส์จะส่ง การเคลื่อนที่ data packet และ Reset การเคลื่อนที่ตัวนับของมัน

เมาส์จะเข้าสู่ Remote mode เมื่อได้รับคำสั่ง "Set Remote Mode" (0xF0) โดยใน mode นี้จะไม่ค่อยได้ใช้ประโยชน์มากนัก

Wrap Mode: โหมด นี้เป็นการส่งกลับ (Echoing) โหมด คือ ทุกๆ ไบต์ที่เมาส์ได้รับ จะส่งกลับไปยังโฮสต์ เหตุการณ์นี้ ถ้าไบต์แทนด้วยคำสั่งที่ผิด เมาส์จะไม่ตอบรับคำสั่งนั้น (Response to that command) นั่นคือ มันจะส่งเฉพาะไบต์นั้นกลับไปยังโฮสต์ มีข้อยกเว้น 2 ประการ ในกรณีนี้ คำสั่ง "Reset" (0xFF) และ "Reset Wrap Mode" (0xEC) เมาส์ จะตอบรับกับ 2 คำสั่งนี้ แต่จะไม่ส่ง echo กลับไปยัง โฮสต์

ตารางที่ 2.11 ไบต์แสดงสถานะต่าง ๆ ของเมาส์

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ไบต์ 1	Y overfLow	X overfLow	Y sign Bit	X sign Bit	Always 1	Middle Btn	Right Btn	Left Btn
ไบต์ 2	X การเคลื่อนที่							
ไบต์ 3	Y การเคลื่อนที่							
ไบต์ 4	Z การเคลื่อนที่							

ตาราง 2.12 แสดงการรูปแบบ 3 ไบต์ที่ใช้ในการเซต Scaling , Resolution และ Sample rate

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Always0	Mode	Enable	Scaling	Always0	Left Btn	Middle Btn	Right Btn
Byte 2	Resolution							
Byte 3	Sample Rate							

ตาราง 2.13 แสดงค่าที่ใช้ในการเซต Resolution

Byte Read from Host	Resolution
00	1 count/mm
01	2 count/mm
02	4 count/mm
03	8 count/mm

แสดงขั้นตอนการติดต่อระหว่างคอมพิวเตอร์กับเมาส์แบบ Microsoft Intellimouse

(Power-on Reset)

Mouse : AA Self-test passed

Mouse : 00 Mouse ID

Host : FF Reset command

Mouse : FA Acknowledge

Mouse	:	AA	Self-test passed	
Mouse	:	00	Mouse ID	
Host	:	FF	Reset command	
Mouse	:	FA	Acknowledge	
Mouse	:	AA	Self-test passed	
Mouse	:	00	Mouse ID	
Host	:	FF	Reset command	
Mouse	:	FA	Acknowledge	
Mouse	:	AA	Self-test passed	
Mouse	:	00	Mouse ID	
Host	:	F3	Set Sample Rate	(Attempt to Enter Microsoft
Mouse	:	FA	Acknowledge	Scrolling Mouse mode)
Host	:	C8	decimal 200	
Mouse	:	FA	Acknowledge	
Host	:	F3	Set Sample Rate	
Mouse	:	FA	Acknowledge	
Host	:	64	decimal 100	
Mouse	:	FA	Acknowledge	
Host	:	F3	Set Sample Rate	
Mouse	:	FA	Acknowledge	
Host	:	50	decimal 80	
Mouse	:	FA	Acknowledge	
Host	:	F2	Read Device Type	
Mouse	:	FA	Acknowledge	
Mouse	:	03	Mouse ID	(Response 03 if Microsoft scrolling mouse)
Host	:	E8	Set Resolution	
Mouse	:	FA	Acknowledge	
Host	:	03	8 count/mm	
Mouse	:	FA	Acknowledge	
Host	:	E6	set scaling 1:1	
Mouse	:	FA	Acknowledge	
Host	:	F3	Set Sample Rate	
Mouse	:	FA	Acknowledge	

Host : 28 decimal 40

Mouse : FA Acknowledge

Host : F4 Enable device

Mouse : FA Acknowledge

ถ้าเมื่อกด left mouse button

Mouse : 09 00001001 bit0 = left button state; bit3 = always 1

Mouse : 00 No X-movement

Mouse : 00 No Y-movement

Mouse : 00 No Z-movement

ถ้าเมื่อปล่อย left mouse button

Mouse : 08 00001000 bit 0 = left button state; bit3 = always 1

Mouse : 00 No X-movement

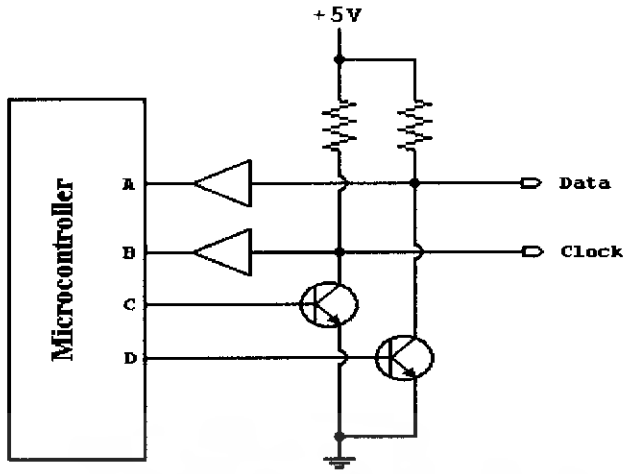
Mouse : 00 No Y-movement

Mouse : 00 No Z-movement

2.4 รายละเอียดของแหล่งจ่ายไฟ

ไฟ Vcc = +5 V กระแสสูงสุด = 100 mA

ดาต้าไลน์ (Data line) และคล็อกไลน์ (Clock line) เป็น Open – Collector ทั้งคู่ และ " Open – Collector " interface จะมีอยู่ 2 สถานะ คือ Low หรือ High impedance ในสถานะ "Low" ทรานซิสเตอร์จะดึงให้ดาต้าไลน์ ไปยังระดับ Ground ในสถานะ High Impedance นั้น จะ Open Circuit และจะไม่ทำให้ดาต้าไลน์ มีสถานะ Low หรือ High ยิ่งไปกว่านั้น ตัวต้านทาน pull up ที่อยู่ระหว่างบัส และ Vcc ค่าของตัวต้านทานไม่สำคัญอาจมีค่าอยู่ประมาณ 1-10 $k\Omega$ โดย ถ้าขนาดของตัวต้านทานมีขนาดใหญ่ก็จะเปลืองไฟเลี้ยง แต่ถ้ามีค่าต่ำการทำงานของขอบขาขึ้นของสัญญาณก็จะเร็วรูปด้านล่างแสดง open-collector interface



รูปที่ 2.6 แสดง Open-Collector Interface

2.5 การติดต่อทั่วไป

เมาส์และคีย์บอร์ดนั้นสนับสนุน Bidirectional synchronous serial protocol โดยเราจะสรุปสถานะของ Data line และคล็อกไลน์ไว้ดังนี้

สถานะของบัส (บัส State)

Data line = High, Clock Line = High: **Idle State**

Data line = High, Clock Line = Low: **Communication Inhibited**

Data line = Low, Clock Line = High: **Host Request – to – Send**

ข้อมูลจะถูกส่งไป 1 ไบต์ จะเท่ากับ 11- 12 บิต ดังนี้:

1 Start bit. : จะมีค่าเป็น "0" เสมอ

8 Data bits : โดยจะส่งบิตที่มีลำดับความสำคัญน้อยที่สุด (LSB: Least Significant Bit) ไปก่อน

1 Odd Parity bit : พาริตี้บิตคี่

1 Stop bit : จะเป็น "1" เสมอ

1 Acknowledge bit : ใช้เฉพาะเวลาสื่อสารในทิศทาง โฮสต์ไปยังอุปกรณ์ เท่านั้น

Parity bit ถูก set (= 1) เมื่อมี "1" ปรากฏเป็นจำนวนคู่ใน 8 Data bits

Parity bit ถูก clear (= 0) เมื่อมี "1" ปรากฏเป็นจำนวนคี่ใน 8 Data bits

Parity bit นี้จะใช้ในกระบวนการ Error Detection

ข้อมูลที่ส่งจากอุปกรณ์ไปยังโฮสต์ จะอ่านที่ขอบขาลงของคล็อก และข้อมูลที่ส่งจากโฮสต์ไปยังอุปกรณ์ (Host – to – device) จะถูกอ่านที่ขอบขาขึ้นของคล็อก โดยที่ความถี่ของคล็อกต้องอยู่ระหว่าง 10 – 16.7 kHz นั่นหมายความว่า คล็อกเป็น "High" เป็นเวลา 30 – 50 µs (microsecond) และเป็น "Low" เป็นเวลา 30 – 50 µs สรุปสุดท้ายว่า ตัวอุปกรณ์(device) จะเป็นตัวสร้างคล็อก แต่โฮสต์จะเป็นตัวควบคุมการสื่อสารทั้งหมด

การติดต่อจากอุปกรณ์ไปยังโฮสต์ (Device – to – Host)

สายข้อมูลและคล็อกไลน์เป็น Open collector ทั้งคู่ ตัวต้านทานที่ต่ออยู่ระหว่างแต่ละสายกับ +5 V ดังนั้น Idle State ของบัสคือ High เมื่อเมาส์หรือคีย์บอร์ดต้องการส่งข้อมูล อย่างแรกมันจะทำการตรวจสอบ คล็อกไลน์ ว่าอยู่ในสถานะ High หรือไม่ ถ้าไม่ใช่ก็แสดงว่า โฮสต์กำลังทำยับยั้งการส่งข้อมูลอยู่ และ device ต้องทำการรับและเก็บข้อมูลหรือคำสั่งที่โฮสต์ส่งมาให้หมด จนกว่าโฮสต์จะทำการปล่อยคล็อกไลน์ คล็อกไลน์ ต้องอยู่ในสถานะ High ต่อเนื่องยาวนานอย่างน้อย 50 ms ก่อนตัวอุปกรณ์ จึงจะสามารถเริ่มต้นการส่งข้อมูลของมันได้

เมาส์และคีย์บอร์ต ใช้ การสื่อสารแบบอนุกรม (Serial protocol) โดยมีรูปแบบเป็น 11 เฟรมบิต ตามนี้คือ:

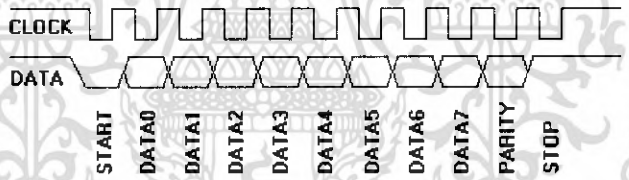
1 Start bit. : จะมีค่าเป็น "0" เสมอ

8 Data bits : โดยจะส่งบิตที่มีลำดับความสำคัญน้อยที่สุด (LSB: Least Significant Bit) ไปก่อน

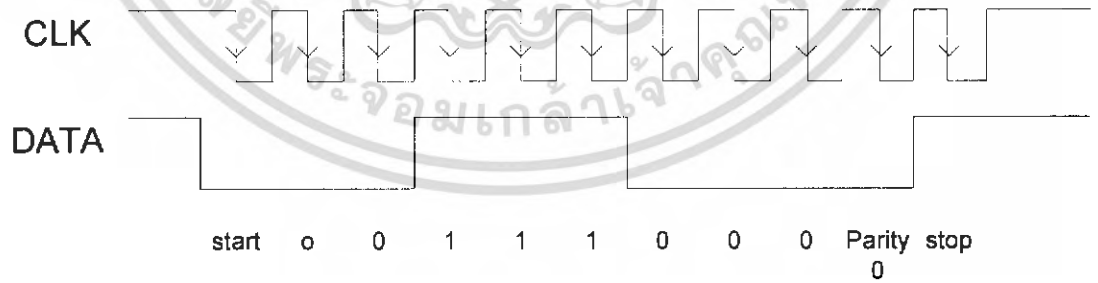
1 Parity bit : พาริตี้บิตคี่

1 Stop bit : จะเป็น "1" เสมอ

เมาส์และคีย์บอร์ดทำการเขียนแต่ละบิตลงใน ดาต้าไลน์เมื่อคล็อกอยู่ในสถานะ High และ มันจะทำการอ่านข้อมูลมาจาก โฮสต์ เมื่อคล็อกมีสถานะเป็น Low ตามรูปด้านล่างนี้



รูปที่ 2.7 แสดงสัญญาณบนดาต้าไลน์ และคล็อกที่เกิดขึ้นเมื่อมีการสื่อสารในทิศทางอุปกรณ์ไปโฮสต์ (Device-to-Host)



รูปที่ 2.8 ตัวอย่างการส่งข้อมูล 'A' จากคีย์บอร์ดไปยังพีซี

ความถี่ของคล็อกประมาณ 10 – 16.7 kHz เวลาจาก ขอบขาขึ้นของคล็อกพัลส์จนถึงการส่งข้อมูลต้องใช้เวลาอย่างน้อย 5 μs และเวลาจากการส่งข้อมูลจนถึงขอบขาลงของคล็อกพัลส์ต้องใช้เวลาระหว่าง 5 – 25 μs

โฮสต์อาจทำการยับยั้งการส่งข้อมูลได้ตลอดเวลา โดยการดึงคลื่อกไลน์ลงมาเป็น Low อย่างน้อย 100 ms ถ้าการส่งถูกยับยั้งก่อนคลื่อกพัลส์ลูกที่ 11 อุปกรณ์ (Device) ต้องยกเลิกการส่งข้อมูลที่กำลังทำอยู่และทำการเตรียมส่งชุดของข้อมูลที่กำลังส่งก่อนถูกทำการยับยั้งจากโฮสต์อีกครั้ง เมื่อโฮสต์ทำการปล่อยคลื่อก ชุดของข้อมูลที่กำลังส่ง ก่อนถูกทำการยับยั้งจากโฮสต์นั้นอาจจะเป็น make code , break code , device ID , mouse movement packet หรืออื่น ๆ ตัวอย่างเช่น ถ้าคีย์บอร์ดถูกทำการขัดจังหวะ (interrupt) ขณะที่กำลังส่งไบต์ ที่ 2 ของ เบรคโค้ด ซึ่งมีทั้งหมด 2 ไบต์ จะไม่ทำการส่งเฉพาะไบต์ที่ 2 แต่จะทำการส่งทั้ง 2 ไบต์ใหม่อีกครั้ง

ถ้าโฮสต์ทำการดึงคลื่อกไลน์ (Clock line) เป็น Low ก่อนเปลี่ยนของคลื่อก(Clock) จาก High ไปยัง Low ของการส่งครั้งแรก หรือ หลังจากขอบขาลงของ คลื่อกพัลส์สุดท้าย ในกรณีนี้อุปกรณ์ไม่ต้องทำการส่งข้อมูลใด ๆ ไปอีกครั้ง อย่างไรก็ตาม ถ้าข้อมูลใหม่ถูกสร้างและรอถูกส่ง มันจะต้องทำการกัน (buffer) ข้อมูลไว้ก่อนจนกระทั่งโฮสต์ จะทำการปล่อยคลื่อกไลน์ เพื่อรองรับในกรณีนี้คีย์บอร์ดจึงมี 16 ไบต์ บัฟเฟอร์ และถ้าข้อมูลที่อยู่ในบัฟเฟอร์ (buffer) เต็มแล้วแต่เรายังต้องการส่งข้อมูลอีก ตัวอักษรที่ถูกกดหลังบัฟเฟอร์เต็มจะถูกละทิ้งจนกว่าบัฟเฟอร์จะว่าง

การติดต่อจากโฮสต์ไปยังตัวอุปกรณ์ (Host – to – Device)

ก่อนการส่งทุกครั้งตัวอุปกรณ์จะสร้างคลื่อก ถ้าโฮสต์ต้องการส่งข้อมูล อย่างแรก มันต้องทำการดึงคลื่อกไลน์และดาต้าไลน์ให้อยู่ในสถานะ "Request – to- Send" ตามข้างล่างนี้

- ยับยั้งการส่งข้อมูลด้วยการดึงคลื่อกไลน์ เป็น Low เป็นเวลาอย่างน้อย 100 μs
- จากนั้นทำการดึง ดาต้าไลน์เป็น Low ก็จะอยู่ในสถานะ "Request – to- Send"แล้วจึงปล่อยคลื่อก

อุปกรณ์ (Device) ควรทำการตรวจสอบ สำหรับสถานะนี้ไม่ควรใช้เวลาเกิน 10 ms เมื่ออุปกรณ์ตรวจพบสถานะนี้ มันจะเริ่มสร้างคลื่อกและคลื่อกในข้อมูล 8 บิต และ 1 – stop bit โฮสต์ จะทำการส่งบิตข้อมูลลงไปยังดาต้าไลน์เมื่อคลื่อกไลน์เป็น Low เท่านั้นและข้อมูลจะถูกอ่านโดย device เมื่อคลื่อกเป็น High กระบวนการนี้จะเกิดตรงข้ามกันเมื่อเป็นการสื่อสารในทิศทางอุปกรณ์ไปโฮสต์

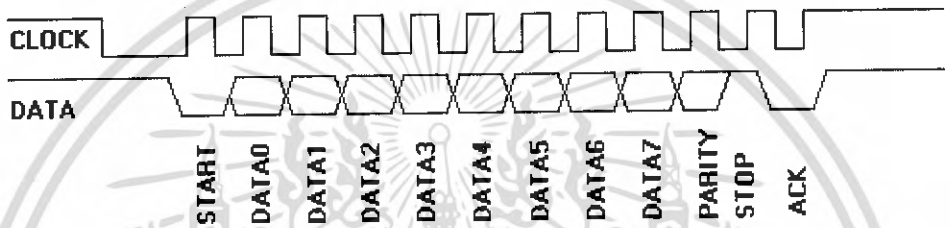
หลังจากอุปกรณ์ได้รับ Stop bit แล้ว อุปกรณ์จะทำการส่งสัญญาณ "acknowledge" ว่ารับข้อมูลเรียบร้อยแล้วโดยการทำให้ดาต้าไลน์เป็น Low และทำการสร้างคลื่อกพัลส์สุดท้ายไป 1 ลูก ถ้า โฮสต์ ไม่ทำการปล่อยดาต้าไลน์หลังจากคลื่อกพัลส์ลูกที่ 11 อุปกรณ์จะทำการสร้างคลื่อกพัลส์ จนกระทั่ง ดาต้าไลน์ถูกปล่อย

โฮสต์ อาจทำการยกเลิกการส่งก่อนคลื่อกพัลส์ลูกที่ 11 โดยการดึงคลื่อกเป็น Low อย่างน้อย 100 μs

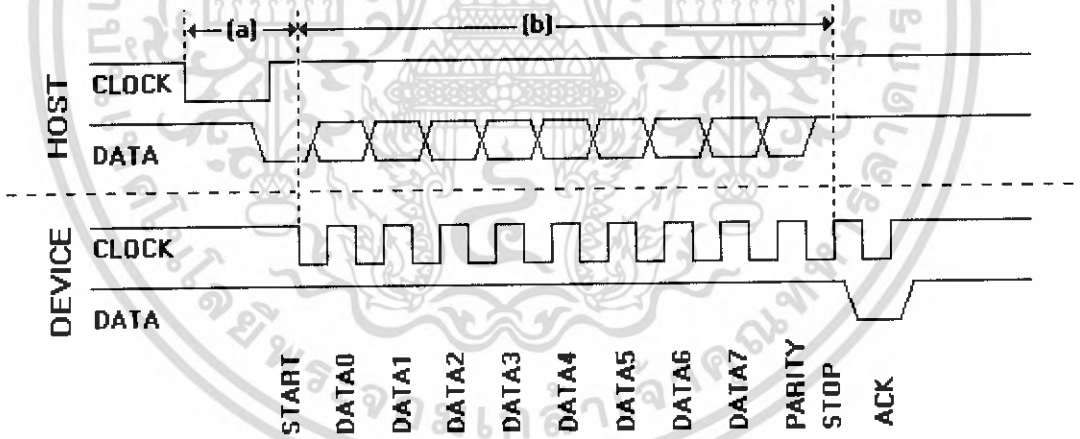
ด้านล่างเป็นขั้นตอนการทำงานเมื่อส่งข้อมูลไปยัง Device

1. ดึงคลื่อกให้เป็น Low อย่างน้อย 100 us
2. ดึงดาต้าไลน์ให้เป็น Low
3. ปล่อยคลื่อกไลน์

4. คอยการ acknowledge ของ device โดยรอรับคล็อกไลน์ เป็น Low
5. Set / Reset คาต้าไลน์เพื่อส่งข้อมูลบิตที่ 1
6. คอยให้ device ส่งคล็อกเป็น High
7. คอยให้ device ส่งคล็อกเป็น Low
8. ทำซ้ำข้อ 5-7 สำหรับอีกคาต้าบิต (Data bit) ที่เหลืออีก 7 บิต และ Parity bit
9. ปล่อย Data line
10. คอยให้ device ส่ง Data Low มา
11. คอยให้ device ส่งคล็อกเป็น Low มา
12. คอยให้ device ปล่อยคาต้า และ คล็อก



รูปที่ 2.9 แสดงสัญญาณบนคาต้าไลน์และคล็อก ที่เกิดขึ้นเมื่อมีการสื่อสารในทิศทางโฮสต์ไปยังอุปกรณ์



รูปที่ 2.10 แสดงช่วงเวลาที่เกิดการสื่อสารระหว่างโฮสต์ไปอุปกรณ์

ตามรูปที่ 2.10 มี 2 จำนวนที่โฮสต์จะคำนึงถึง (a) คือเวลาที่ device เริ่มgenerate คล็อกพัลส์ หลังจาก โฮสต์ทำการ initial โดยทำให้คล็อก เป็น Low อย่างน้อย 15 ms (b) เป็นเวลาการส่งชุดข้อมูล อย่างน้อยต้องใช้เวลา 2 ms ถ้าทั้ง (a)และ(b) ไม่เป็นตามนี้ โฮสต์ จะ สร้างสัญญาณผิดพลาด (generate error)จากนั้น หลังจาก "acknowledge" ได้รับเรียบร้อย ถ้าคำสั่งจาก โฮสต์ ต้องการ "response" , response ต้องไม่ได้รับช้ากว่า 20 ms หลังจาก โฮสต์ ปล่อยคล็อกไลน์ถ้าไม่มี response โฮสต์จะสร้าง สัญญาณผิดพลาด

2.6 ระบบการรับข้อมูล (System Receiving Data)

ค่าเวลา (Timing) ของการรับข้อมูลจากอุปกรณ์ต่อพ่วง (เมาส์/คีย์บอร์ด) ต่อไปจะอธิบายลำดับการทำงานที่เกิดขึ้นเมื่อระบบกำลังทำการรับ data จากอุปกรณ์ต่อพ่วง

1. คีย์บอร์ด/เมาส์ จะตรวจสอบคล็อก ถ้าคล็อกไลน์ (Clock line) เป็น "0" output ที่มาจากคีย์บอร์ด/เมาส์ จะไม่ได้รับการอนุญาต

2. คีย์บอร์ด/เมาส์ ตรวจสอบดาต้าไลน์ (Data line) ถ้าดาต้าไลน์เป็น "0" ตัวควบคุม (Controller) ทำการรับข้อมูลจากคอมพิวเตอร์

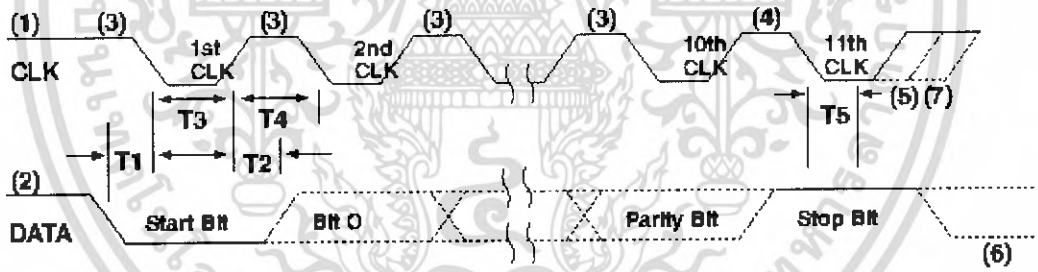
3. คีย์บอร์ด/เมาส์ ตรวจสอบ คล็อกไลน์ (Clock line) ในระหว่างทำการส่งถ้าใช้เวลาไม่เกิน 100 µs ถ้า คีย์บอร์ด ค้นหา ระบบทำการครอบครองคล็อกไลน์ (Clock line) ไว้ การส่งจะสิ้นสุดลง ระบบสามารถจบการส่งได้ทุกเวลา ตลอดเวลาในช่วงคล็อก10 ลูกแรก

4. สุดท้ายจะตรวจสอบการสิ้นสุดการส่ง โดยกระทำการหลังจากผ่าน Clock 10 ลูกแรกไป อย่างน้อย 5 µs

5. คอมพิวเตอร์สามารถให้คล็อกไลน์ (Clock line) inactive (เป็น "0") ไว้เพื่อยับยั้งการส่งครั้งต่อไปไว้ก่อน

6. คอมพิวเตอร์สามารถ Set ดาต้าไลน์ (Data line) ให้ inactive ถ้าไบต์ ถูกส่งไปยังอุปกรณ์ดาต้าไลน์ (Data line) จะ Set เป็น inactive เมื่อ Start Bit (0 เสมอ) ปรากฏบนดาต้าไลน์

7. ระบบปล่อยคล็อกไลน์ (Clock line) อนุญาตให้ส่งครั้งต่อไป



รูปที่ 2.11 เวลาต่างๆที่คอมพิวเตอร์ใช้ในการรับข้อมูล

ตารางที่ 2.14 รายละเอียดของเวลาในการรับข้อมูล

Period	TIMING PARAMETER	Min	Max
T1	เวลาการส่งข้อมูล ที่ขอบขาลงของสัญญาณ CLK	5 µs	25 µs
T2	เวลาขอบขาขึ้นของสัญญาณ CLK ขณะที่มีการส่งข้อมูล	5 µs	T4-5 µs
T3	ช่วงเวลาที่สัญญาณ CLK เป็น "0"	30 µs	50 µs
T4	ช่วงเวลาที่สัญญาณ CLK เป็น "1"	30 µs	50 µs
T5	เวลาที่ใช้ในการยับยั้งอุปกรณ์ไม่ให้ส่งสัญญาณหลังจากผ่านสัญญาณ CLK ลูกที่ 11 แล้ว	> 0 µs	50 µs

2.7 ระบบการส่งข้อมูล (System Sending Data)

1. คอมพิวเตอร์ตรวจสอบว่า คีย์บอร์ด/เมาส์ กำลังดำเนินกระบวนการส่งข้อมูลมาหรือไม่ ถ้าการส่งอยู่ในขั้นดำเนินการถัดจาก Clock 10 ลูกแรก ระบบต้องทำการรับข้อมูล

2. คีย์บอร์ด/เมาส์ จะตรวจสอบคล็อกไลน์ (Clock line) ถ้าคล็อกเป็น "0" กระบวนการ I/O จะถูกยับยั้ง

3. คีย์บอร์ด/เมาส์ ตรวจสอบดาต้าไลน์ (Data line) ถ้าดาต้าไลน์เป็น "0" คอมพิวเตอร์ต้องทำการส่งข้อมูล ดาต้าไลน์ (Data line) จะเป็น "0" เมื่อบิตสตาร์ท (Start Bit) ("0" เสมอ) ปรากฏบน ดาต้าไลน์

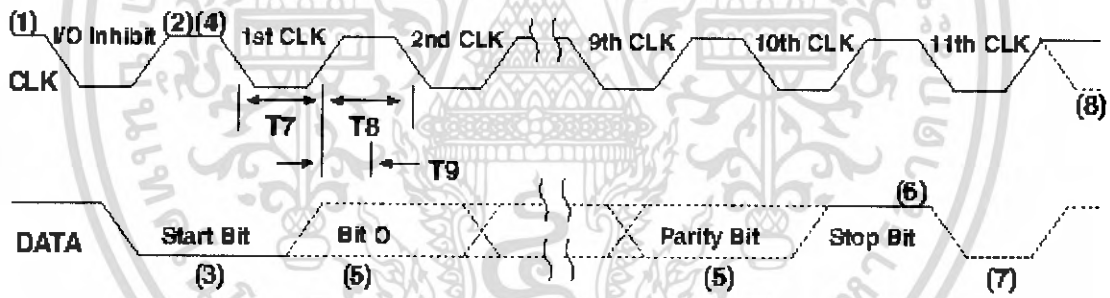
4. คีย์บอร์ด/เมาส์ จะ Set คล็อกไลน์เป็น "0" ระบบจะส่งบิตแรกลงบนดาต้าไลน์ แต่ละครั้งที่คีย์บอร์ด/เมาส์ Set คล็อกไลน์เป็น "0" ระบบจะส่งบิตต่อ ๆไปลงบนดาต้าไลน์ (Data line) จนกระทั่งทุกบิต ถูกส่งไปจนครบ

5. คีย์บอร์ด/เมาส์

6. คีย์บอร์ด/เมาส์ตรวจสอบ positive-level ของ Stop-Bit หลังจากคล็อค 10 ลูกแรก ถ้าเป็น "0" คีย์บอร์ด/เมาส์ จะเกิดที่ครั้งต่อไปเพื่อส่งหรือส่งซ้ำไปยังคอมพิวเตอร์

7. คีย์บอร์ด/เมาส์ ดึงดาต้าไลน์ให้เป็น "0"

8. คอมพิวเตอร์สามารถดึงคล็อก (Clock line) ให้เป็น "0" เพื่อยับยั้งการส่งข้อมูลมาจากคีย์บอร์ด



รูปที่ 2.12 เวลาต่าง ๆ ที่คอมพิวเตอร์ใช้ในการส่งข้อมูล

ตารางที่ 2.15 รายละเอียดของเวลาในการส่งข้อมูล

Period	TIMING PARAMETER	Min	Max
T7	ช่วงเวลาที่สัญญาณ CLK เป็น "0"	30 µs	50 µs
T8	ช่วงเวลาที่สัญญาณ CLK เป็น "1"	30 µs	50 µs
T9	เวลาจากสัญญาณ CLK เปลี่ยนจาก "0" เป็น "1"	30 µs	50 µs
T4	ช่วงเวลาที่สัญญาณ CLK เป็น "0"	30 µs	50 µs
T5	เวลาที่ใช้ในการยับยั้งอุปกรณ์ไม่ให้ส่งสัญญาณหลังจากผ่านสัญญาณ CLK ลูกที่ 11 แล้ว	> 0 µs	50 µs

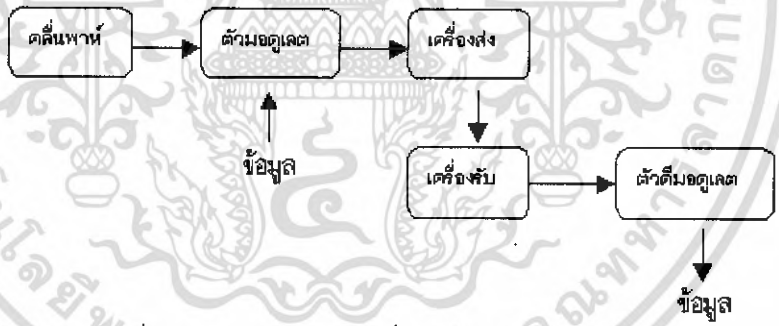
ส่วนของเครื่องรับ-ส่งวิทยุ

สัญญาณอิเล็กทรอนิกส์ที่ใช้ในการสื่อสารสามารถแบ่งได้เป็น 2 ประเภทคือ สัญญาณอนาล็อก และสัญญาณดิจิตอล สัญญาณอนาล็อก ได้แก่ สัญญาณเสียง และสัญญาณในธรรมชาติทั้งหมด ปัญหาที่สำคัญของสัญญาณ อนาล็อกก็คือเรื่องสัญญาณรบกวน ซึ่งในบางครั้งอาจทำให้ระบบไม่สามารถใช้งานได้เลย ดังนั้นจึงมีการนำสัญญาณดิจิตอลเข้ามาแทนที่

1. สัญญาณแบบอนาล็อก (Analog Signal) จะเป็นสัญญาณแบบต่อเนื่องที่ทุกๆ ค่า ที่เปลี่ยนแปลงไปของระดับสัญญาณจะมีความหมาย การส่งสัญญาณแบบนี้จะถูกรบกวนให้มีการแปลความหมายผิดพลาดได้ง่าย เนื่องจากค่าทุกค่าถูกนำมาใช้งาน ซึ่งสัญญาณแบบอนาล็อกนี้จะเป็นสัญญาณที่สื่อกลางในการสื่อสารส่วนมากใช้อยู่เช่น สัญญาณเสียงในสายโทรศัพท์ เป็นต้น

2. สัญญาณแบบดิจิตอล (Digital Signal) จะประกอบขึ้นจากระดับสัญญาณเพียง 2 ค่าคือ สัญญาณระดับสูงสุด และสัญญาณระดับต่ำสุด ดังนั้นจะมีประสิทธิภาพ และความน่าเชื่อถือสูงกว่าแบบอนาล็อกเนื่องจากมีการใช้งานค่าสองค่า เพื่อนำมาตีความหมายเป็น on/off หรือ0/1 เท่านั้น ซึ่งเป็นสัญญาณที่คอมพิวเตอร์ใช้ในการติดต่อสื่อสารกัน

การมอดูเลต (Modulation) เป็นการผสมสัญญาณของข้อมูลเข้าไปกับสัญญาณอีกสัญญาณหนึ่งเรียกว่า คลื่นพาห์ (carrier) ซึ่งสัญญาณนี้มีความถี่ที่เหมาะกับช่องสัญญาณนั้นๆ เพื่อให้ข้อมูลที่ส่งเข้าไปในช่องสัญญาณเดินทางได้ไกลมากขึ้น การเลือกวิธีมอดูเลตขึ้นอยู่กับปัจจัยหลายประการ เช่น ชนิดของสัญญาณ แบนด์วิดท์ ประสิทธิภาพของระบบที่ต้องการ และความด้านทานต่อสัญญาณรบกวน เป็นต้น



รูปที่ 2.13 รูปแบบของการสื่อสารในการรับส่งสัญญาณ

จากรูป แสดงรูปแบบของการสื่อสารในการรับส่งสัญญาณอย่างง่าย โดยคลื่นพาห์ผสมสัญญาณข้อมูลที่ตัวมอดูเลต (Modulator) แล้วส่งไปที่เครื่องส่ง จากเครื่องส่งไปยังเครื่องรับจะเป็นช่องสัญญาณสำหรับลำเลียงสัญญาณผสมนี้ สัญญาณผสมจากเครื่องรับจะไปเข้าตัวดีมอดูเลต (Demodulate) เพื่อแยกสัญญาณข้อมูลออกมา เพื่อให้เราสามารถสื่อสารไปได้ไกลยิ่งขึ้น จำเป็นต้องมอดูเลตสัญญาณข่าวสารลงพาหะ วิธีมอดูเลตแบ่งออกเป็น 2 แบบที่สำคัญ คือ มอดูเลตทางแอมปลิจูด และมอดูเลตเชิงมุม ซึ่งการมอดูเลต ทั้งสองแบบนี้เป็นการมอดูเลตเชิงอนาล็อก (Analog)

การสื่อสารอนาล็อกเป็นระบบที่ออกแบบให้ส่งข้อมูลสัญญาณอนาล็อกเช่น สัญญาณเสียง แต่ได้มีการพัฒนาจนประยุกต์ให้สามารถส่งข่าวสารได้ด้วย ในปัจจุบันปัญหาสำคัญสำหรับการสื่อสาร

อนาล็อกก็คือเรื่องสัญญาณรบกวน แต่เนื่องจากสัญญาณในธรรมชาติทั้งหมดเป็นสัญญาณอนาล็อกจึงยังคงเห็นการพัฒนาของการสื่อสารแบบอนาล็อกในปัจจุบัน เช่น การมอดูเลตแอมปลิจูด (Amplitude Modulation หรือ AM) การมอดูเลตความถี่ (Frequency Modulation หรือ FM)

การมอดูเลตที่ใช้อยู่ในปัจจุบันมี 3 วิธี ได้แก่

1. การมอดูเลตแอมปลิจูด (Amplitude Modulation หรือ AM) วิธีนี้แอมปลิจูดของคลื่นพาห์จะเปลี่ยนแปลงตามสัญญาณของข้อมูลที่เข้ามา การมอดูเลตแบบ AM เป็นวิธีที่ง่ายที่สุดในการมอดูเลต แต่คุณภาพของสัญญาณไม่ดี มีความต้านทานสัญญาณรบกวนต่ำ เหมาะกับข้อมูลที่ไม่ต้องการคุณภาพมากนัก เช่น สัญญาณเสียง เป็นต้น

2. การมอดูเลตความถี่ (Frequency Modulation หรือ FM) วิธีการนี้เป็นการเปลี่ยนแปลงความถี่ของคลื่นพาห์ตามสัญญาณของข้อมูลที่เข้ามา การมอดูเลตแบบความถี่ ให้คุณภาพที่ดีกว่าการมอดูเลตแบบแอมปลิจูด แต่ระบบจะซับซ้อนกว่า

3. การมอดูเลตเฟส (Phase Modulation หรือ PM) เป็นการมอดูเลตที่ใช้ในการเปลี่ยนแปลงเฟสของคลื่นพาห์ตามสัญญาณข้อมูลที่เข้ามา ทั้งคุณภาพของสัญญาณและความซับซ้อนไม่ค่อยแตกต่างจากการมอดูเลตแบบความถี่เท่าใดนัก ข้อแตกต่างระหว่างการมอดูเลตแบบความถี่กับการมอดูเลตแบบเฟส คือการมอดูเลตแบบเฟสใช้คลื่นพาห์เพียงความถี่เดียว การมอดูเลตและดีมอดูเลตสามารถทำได้ประหยัดกว่า แต่ไม่ได้หมายความว่าซับซ้อนน้อยกว่า

ในที่นี้เราจะกล่าวถึงการมอดูเลตแอมปลิจูดซึ่งใช้ในการรับส่งสัญญาณในการทดลองครั้งนี้

2.8 การมอดูเลตแอมปลิจูดแบบแถบข้างคู่ขจัดคลื่นพาห์

ในกระบวนการมอดูเลตแอมปลิจูดนั้นขนาดของสัญญาณข่าวสารจะถูกใช้ไปบังคับค่าแอมปลิจูด A_c ของคลื่นพาห์ $A_c\cos(\omega_c + \theta_c)$ ให้เปลี่ยนตามในขณะที่ความถี่ ω_c และเฟส θ_c จะยังมีค่าคงเดิม สมการของคลื่นที่มอดูเลตแล้ว $\phi_{DSB-SC}(t)$ จะเป็นดังนี้คือ

$$\phi_{DSB-SC}(t) = km(t)\cos(\omega_c t + \theta_c) \tag{2-1}$$

โดย k คือค่าคงที่ ที่กระบวนการมอดูเลตจัดการควบคุมสัดส่วนของ A_c ให้แปรผันตามสัญญาณข่าวสาร $m(t)$

เพื่อความสะดวกในการวิเคราะห์สัญญาณ เราสามารถสมมุติให้ $k = 1$ และ $\theta_c = 0$ ได้ โดยไม่เสียความหมายของการวิเคราะห์สัญญาณโดยทั่วไป แต่อย่างไรเพียงแต่จะทำให้รูปของสมการที่กระชับขึ้น คือ จะได้สมการเป็น

$$\phi_{DSB-SC}(t) = m(t)\cos(\omega_c t) \tag{2-2}$$

เพื่อที่จะทำความเข้าใจถึงการเปลี่ยนแปลงที่เกิดขึ้นในโดเมนความถี่ จะสมมุติให้ $M(\omega)$ คือค่าฟังก์ชันสเปกตรัมของ $m(t)$ กล่าวคือ

$$m(t) \leftrightarrow M(\omega) \tag{2-3}$$

โดยอาศัยคุณสมบัติของการแปลงฟูเรียร์ จะได้

$$m(t)\cos(\omega_c t) \leftrightarrow \frac{1}{2}\left[M(\omega+\omega_c)+M(\omega-\omega_c)\right] \tag{2-4}$$

นั่นคือสัญญาณเอเอ็มในโดเมนความถี่ $\Phi_{DSB-SC}(\omega)$ จะมีค่าเป็น

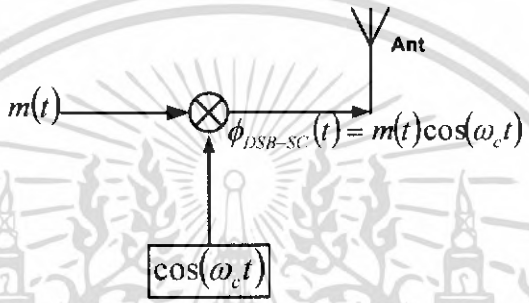
$$\Phi_{DSB-SC}(\omega) = \frac{1}{2}\left[M(\omega+\omega_c)+M(\omega-\omega_c)\right] \tag{2-5}$$

ถ้าแบนด์วิดท์ของ $M(\omega)$ ตามสมการ (2-3) มีค่าเป็น B เฮิรตซ์หรือ W เรเดียนต่อวินาที $(W = 2\pi B)$ เราจะรู้จากสมการ (2-5) ได้ว่า $\Phi_{DSB-SC}(\omega)$ จะมีแบนด์วิดท์เป็น $2W$ เรเดียนต่อวินาที ดังรูปที่ 2.14 และเป็นที่น่าสังเกตว่า ค่าความถี่กึ่งกลางแบนด์ของสัญญาณเอเอ็มนั้นอยู่ที่ความถี่ของคลื่นพาห์คือ ω_c ซึ่งแกนตั้งที่ ω_c ในโดเมนของความถี่นี้จะแบ่งสเปกตรัมที่เกิดขึ้นออกเป็นสองส่วนที่สมมาตรกัน ส่วนของสัญญาณที่มีสเปกตรัมอยู่ที่ความถี่สูงกว่าและต่ำกว่าความถี่ ω_c นั้นมีชื่อว่าแถบข้าง ส่วนบนหรือไซด์แบนด์สูง (Upper sideband) นิยมเขียนย่อแทนด้วย ยูเอสบี (USB) และแถบข้างส่วนล่างหรือไซด์แบนด์ต่ำ (Lower side band) นิยมเขียนย่อด้วย แอลเอสบี (LSB) ตามลำดับในทำนองเดียวกันบนแกนความถี่ด้านลบค่าความถี่ $-\omega_c$ ก็จะทำให้เกิดไซด์แบนด์สูงและไซด์แบนด์ต่ำเช่นกันแต่ในกรณีของความถี่ด้านลบไซด์แบนด์สูงจะหมายถึงส่วนสเปกตรัมที่มีความถี่เป็นลบมากกว่า $-\omega_c$ และไซด์แบนด์ต่ำจะหมายถึงส่วนสเปกตรัมที่มีความถี่เป็นลบน้อยกว่า $-\omega_c$

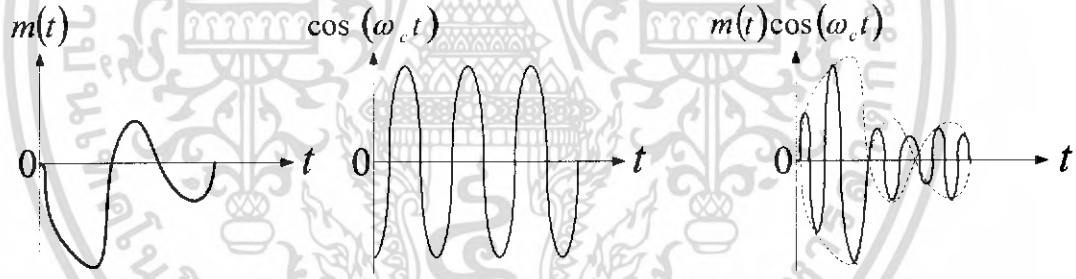
ควรสังเกตอีกด้วยว่า ถ้าสัญญาณ $m(t)$ นั้นไม่มีส่วนประกอบเชิงความถี่ที่เป็นไฟดรงไซด์แบนด์ทั้งสองที่เกิดจากการมอดูเลตก็จะอยู่ห่างกันอย่างเด่นชัด และสเปกตรัมของสัญญาณเอเอ็มก็จะไม่เกิดมีองค์ประกอบความถี่ที่ $\omega = \pm\omega_c$ ปรากฏ กล่าวคือ ส่วนประกอบสัญญาณที่มีความถี่ตรงกับความถี่ของคลื่นพาห์จะถูกขจัดออกไปในกรณีดังกล่าว ด้วยเหตุนี้เองทำกระบวนการมอดูเลตสัญญาณแบบนี้ได้ชื่อว่า **การมอดูเลตแอมปลิจูดแบบไซด์แบนด์คู่ขจัดคลื่นพาห์** (AM double-sideband-suppressed carrier) **หรือการมอดูเลตแอมปลิจูดแบบแถบข้างคู่ขจัดคลื่นพาห์**ซึ่งเรียกย่อว่าดีเอสบี เอสซี (DSB-SC) ตามสมการ (2-5) จะบอกให้รู้ว่าการมอดูเลตแบบนี้จะทำให้เกิดการย้ายสเปกตรัมของสัญญาณข่าวสาร $M(\omega)$ ออกไปจากเดิม $\pm\omega_c$ (หมายถึงย้ายไปจากเดิม $+\omega_c$ และ $-\omega_c$ ตามลำดับ) ในการที่จะแยกสัญญาณข่าวสาร $m(t)$ ออกมาจาก $\phi_{DSB-SC}(t)$ นั้นลักษณะการที่ทำหน้าที่นี้เรียกว่า การกู้สัญญาณ หรือ การดี

มอดูเลต (Demodulate) สัญญาณหรือการตรวจจับอีกครั้งหนึ่ง ซึ่งเมื่อทำเช่นว่านี้แล้ว จะได้ผลลัพธ์ $v_d(t)$ เป็น

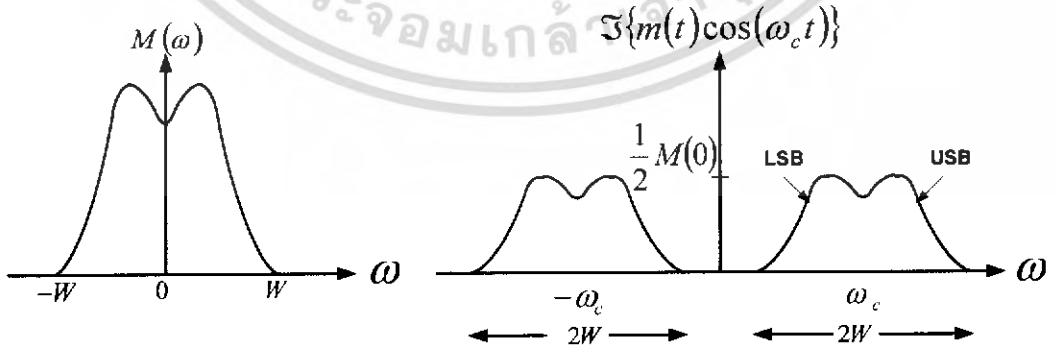
$$\begin{aligned} v_d(t) &= \phi_{DSB-SC}(t)\cos(\omega_c t) \\ &= \{m(t)\cos(\omega_c t)\}\cos(\omega_c t) \\ &= m(t)\cos^2(\omega_c t) \\ &= \frac{1}{2}m(t) + \frac{1}{2}m(t)\cos(2\omega_c t) \end{aligned} \tag{2-6}$$



(ก) แผนภาพแสดงกระบวนการมอดูเลตสัญญาณดีเอสบี เอสซี



(ข) สัญญาณข่าวสาร $m(t)$ (ค) คลื่นพาห์ $\cos(\omega_c t)$ (ง) สัญญาณดีเอสบี เอสซี



(จ) ความหนาแน่นของสเปกตรัมของ $m(t)$ (ฉ) ความหนาแน่นของสัญญาณดีเอสบี เอสซี

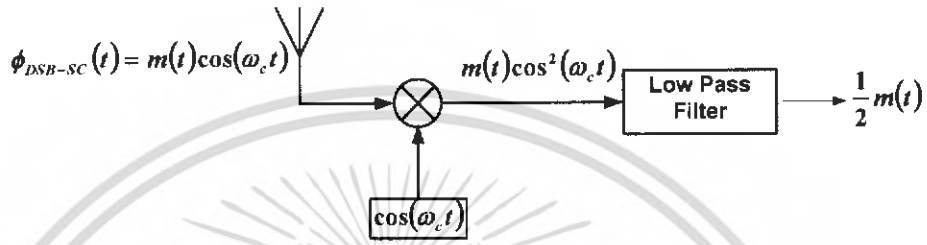
ในที่นี้ $W = 2\pi B$

รูปที่ 2.14 กระบวนการมอดูเลตสัญญาณดีเอสบีเอสซี

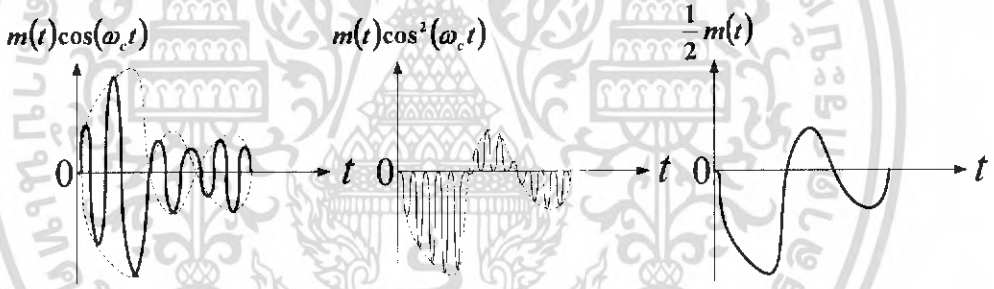
และโดยคุณสมบัติของการแปลงฟูเรียร์จะได้

$$\phi_{DSB-SC}(t)\cos(\omega_c t) \leftrightarrow \frac{1}{2}M(\omega)+\frac{1}{4}\left[M(\omega+2\omega_c)+M(\omega-2\omega_c)\right] \qquad (2\text{-}7)$$

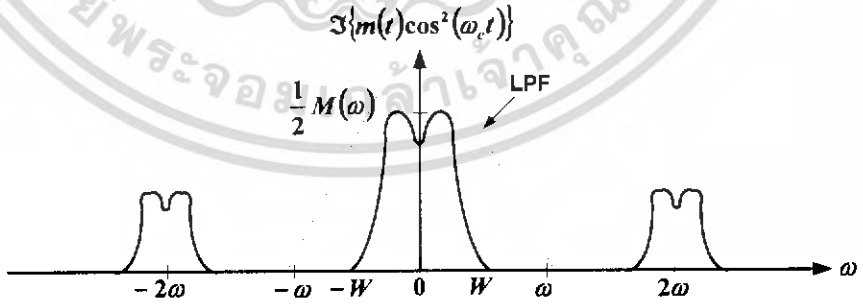
เมื่อนำสัญญาณ $\phi_{DSB-SC}(t)\cos(\omega_c t)$ นี้ไปผ่านวงจรกรองความถี่ต่ำผ่านเพื่อขจัดสัญญาณที่มีความถี่สูงคือพจน์ที่อยู่ในวงเล็บใหญ่ของ (2-4) ออก ก็จะเหลือเพียงแต่ส่วนของสเปกตรัม $\frac{1}{2}M(\omega)$ ออกมาซึ่งก็คือสเปกตรัมของสัญญาณเดิม $\frac{1}{2}m(t)$ นั่นเอง



(ก) แผนภาพแสดงกระบวนการดีมอดูเลตสัญญาณดีเอสบี เอสซี



(ข) สัญญาณดีเอสบี เอสซี (ค) สัญญาณ $v_o(t)$ (ง) สัญญาณที่ดีมอดูเลตออกมาได้



(จ) สเปกตรัมของสัญญาณ $v_o(t)$

รูปที่ 2.15 การดีมอดูเลตสัญญาณดีเอสบี เอสซี และสัญญาณต่าง ๆ พร้อมทั้งค่าความหนาแน่นสเปกตรัมของสัญญาณที่อินพุตของวงจรกรองความถี่ต่ำผ่าน

อย่างไรก็ดีในการตรวจจับสัญญาณด้วยวิธีการดังกล่าวนี้ มักจะพบปัญหาอันเนื่องมาจากคลื่นพาห์ที่เครื่องรับสร้างขึ้นมานั้น มักจะมีความถี่และเฟสไม่เท่ากับความถี่และเฟสของคลื่นพาห์จากเครื่องส่ง การที่จะเข้าใจถึงผลของความคลาดเคลื่อนของคลื่นพาห์นั้น ทำได้โดยสมมุติว่าเมื่อมีสัญญาณจากเครื่องส่งคือ $\phi_{DSB-SC}(t) = m(t)\cos(\omega_c t)$ นั้นคลื่นพาห์จากออสซิลเลเตอร์ทางเครื่องรับมีความคลาดเคลื่อนไปเล็กน้อยเป็น $\cos\{(\omega_c + \Delta\omega)t + \delta\}$ โดย $\Delta\omega$ และ δ คือค่าความคลาดเคลื่อนทางความถี่และเฟสตามลำดับ

ตามกระบวนการดีมอดูเลต เครื่องรับจะทำการดีมอดูเลตสัญญาณโดยใช้คลื่นพาห์ $\cos\{(\omega_c + \Delta\omega)t + \delta\}$ คูณกับ $\phi_{DSB-SC}(t)$ ทำให้ได้สัญญาณสัพท์เป็น

$$\begin{aligned} v_d(t) &= \phi_{DSB-SC}(t)\cos\{(\omega_c + \Delta\omega)t + \delta\} \\ &= m(t)\cos(\omega_c t)\cos\{(\omega_c + \Delta\omega)t + \delta\} \\ &= \frac{1}{2}m(t)\cos(\Delta\omega t + \delta) + \frac{1}{2}m(t)\cos\{(2\omega_c + \Delta\omega)t + \delta\} \end{aligned} \quad (2\text{-}8)$$

วงจรกรองความถี่ต่ำผ่านในเครื่องรับจะขจัดส่วนความถี่สูง คือ พจน์หลังสุดของสมการ (2-8) ออก คงเหลือสัญญาณเอาท์พุต $v_o(t)$ ของวงจรตรวจจับสัญญาณ คือ

$$v_o(t) = \frac{1}{2}m(t)\cos(\Delta\omega t + \delta) \quad (2\text{-}9)$$

ถ้าเครื่องผลิตคลื่นพาห์ขึ้นมาโดยไม่มีความคลาดเคลื่อนกล่าวคือ $\Delta\omega = 0$ และ $\delta = 0$ จากสมการ(2-9) จะพบว่า เพราะ $\cos(0) = 1$ ดังนั้น $v_o(t)$ จึงมีค่าเท่ากับ $\frac{1}{2}m(t)$ ซึ่งเป็นสัญญาณข่าวสาร $m(t)$ ที่ถูกต้อง แต่ถ้าเครื่องรับสร้างคลื่นพาห์ได้มีความถี่ถูกต้อง (คือ $\Delta\omega = 0$) แต่ทว่ายังมีความคลาดเคลื่อนทางเฟสอยู่บ้าง (คือ $\delta \neq 0$) ในกรณีนี้เอาท์พุต ตามสมการ (2-8) จะกลายเป็น

$$v_o(t) = \frac{1}{2}m(t)\cos(\delta) \quad (2\text{-}10)$$

เนื่องจาก $|\cos(\delta)| \leq 1$ ดังนั้นผลที่เกิดขึ้นก็คือ สัญญาณเอาท์พุตจะมีระดับลดลง และ ถ้า $\delta = \pm\frac{\pi}{2}$ ระดับของสัญญาณเอาท์พุตก็จะเป็นศูนย์ (เพราะ $\cos\left(+\frac{\pi}{2}\right) = 0$)

ถ้าสมมุติว่าเครื่องรับสร้างคลื่นพาห์ที่มีเฟสเริ่มต้นถูกต้อง $(\delta = 0)$ แต่มีความถี่คลาดเคลื่อน $(\Delta\omega \neq 0)$ จะพบว่าเอาท์พุตตามสมการ (2-9) จะได้เป็น

$$v_o(t) = \frac{1}{2}m(t)\cos(\Delta\omega t) \quad (2\text{-}11)$$

ซึ่งแสดงให้เห็นว่า $v_o(t)$ มีลักษณะผิดเพี้ยนไปจาก $m(t)$ ด้วยอิทธิพลของการบังคับขนาด (การคูณ) ด้วยคลื่นรูปไซน์ที่มีความถี่ $\Delta\omega$ ซึ่งเป็นความถี่ต่ำ ผลที่เกิดขึ้นนี้เรียกว่า มีการบีต (Beating) ด้วยความถี่ต่ำนั้น ทำให้เอาท์พุตไม่ใช่สัญญาณข่าวสาร $m(t)$ เดิม

เราได้รู้ว่าการตรวจจับสัญญาณดีเอสบี เอสซี นั้นทำด้วยการมอดูเลตคลื่นพาห์ที่สร้างขึ้นทางเครื่องรับซ้ำเข้ากับสัญญาณ $(\phi_{DSB-SC}(t))$ นั้นอีกครั้งหนึ่ง แล้วใช้วงจรกรองความถี่ต่ำผ่านมาขจัดส่วนประกอบที่เป็นความถี่สูงออก ก็จะได้สัญญาณข่าวสารเดิมกลับคืนมา ประเด็นสำคัญในกระบวนการนี้คือ คลื่นพาห์ทางเครื่องรับจะต้องมีความถี่และเฟสตรงกับคลื่นพาห์จากเครื่องส่งพอดี การตรวจจับสัญญาณจึงจะมีประสิทธิภาพเต็มที่ จึงทำให้กระบวนการตรวจจับสัญญาณแบบนี้มีชื่อว่า การดีมอดูเลตแบบสัมพัทธ์ (Synchronous demodulation) หรือการดีมอดูเลตแบบร่วมนัยน์ (Coherent demodulation)

การสร้างวงจรมอดูเลตสัญญาณแบบแถบข้างคู่ขจัดคลื่นพาห์ อาจทำได้โดยใช้วงจรมอดูเลตแบบตัด (Chopper modulator) แบบไม่เชิงเส้น ฯลฯ และที่เคยได้รับความนิยมมากคือ ริงมอดูเลเตอร์ (Ring modulator) สำหรับวงจรรับนั้น วงจรเครื่องรับที่น่าสนใจคือเครื่องรับแบบคอสแทส ซึ่งเป็นเครื่องรับที่นิยมใช้กันอยู่ สามารถที่จะปรับค่าความถี่ของสัญญาณจากออสซิลเลเตอร์ภายในเครื่องรับให้ตรงกับความถี่ของคลื่นพาห์จากเครื่องส่งได้เลยอัตโนมัติ

2.9 การมอดูเลตแอมปลิจูดสำหรับวิทยุกระจายเสียง

จากที่ได้รู้ว่าการดีมอดูเลตสัญญาณดีเอสบี เอสซี และสัญญาณเอสเอสบี สามารถทำได้โดยวิธีการดีมอดูเลตแบบสัมพัทธ์ ซึ่งเป็นวิธีการที่จะต้องใช้การผลิตคลื่นพาห์ทางด้านเครื่องรับให้มีความถี่และเฟสให้เท่ากับค่าของคลื่นพาห์จากเครื่องส่ง และการที่จะควบคุมให้เครื่องรับสามารถสร้างคลื่นพาห์ที่ถูกต้องได้ จะต้องอาศัยวงจรที่มีความซับซ้อนพอควร ซึ่งย่อมเป็นการแน่นอนว่า เมื่อเป็นเช่นนี้ต้นทุนในการสร้างเครื่องรับดังกล่าวก็ย่อมจะต้องมีราคาสูงด้วย เพื่อลดปัญหาในเรื่องนี้ จึงได้เกิดการคิดหาวิธีการที่จะดีมอดูเลตสัญญาณ ดีเอสบี เอสซี ชนิดที่ไม่จำเป็นต้องมีการสร้างคลื่นพาห์ทางด้านเครื่องรับขึ้น วิธีการนี้ทำได้โดยการใช้คลื่นพาห์ $\cos(\omega_c t)$ ซึ่งมีขนาดที่เหมาะสมส่งรวมไปกลับสัญญาณดีเอสบี เอสซี จากเครื่องส่ง การกระทำเช่นนี้ทำให้คลื่นสัญญาณที่ส่งมีรูปฟังก์ชันเป็น

$$\phi_{AM}(t) = A_c \cos(\omega_c t) + m(t)\cos(\omega_c t) \tag{2-12}$$

โดยในที่นี้ A_c คือ ค่าแอมปลิจูดของคลื่นพาห์ที่ใช้เพิ่มเข้ามาและเมื่อจัดรูป (2-12) ใหม่จะได้

$$\phi_{AM}(t) = A_c\left(1 + \frac{m(t)}{A_c}\right)\cos(\omega_c t) \tag{2-13}$$

ปกติค่าความถี่ของคลื่นพาห์ ω_c จะมีค่าสูงกว่าค่าความถี่ที่เป็นองค์ประกอบสูงสุดของสัญญาณ $m(t)$ มาก ดังนั้นถ้าขนาดของคลื่นพาห์ A_c มีค่ามากพอที่จะทำให้พจน์ในวงเล็บแรกของ (2-13) มีค่า

ไม่เป็นลบตลอดเวลาทุกค่า t ใด ๆ แล้ว สัญญาณ $\phi_{AM}(t)$ ตามสมการ (2-13) ก็จะมียอดคลื่นที่ค่อย ๆ เปลี่ยนไปตามเอ็นเวโลป (Envelope) หรือกรอบที่มีลักษณะเหมือนกับสัญญาณ $m(t)$ ทุกประการ เพราะฉะนั้น ถ้านำสัญญาณ $\phi_{AM}(t)$ นี้ไปป้อนให้กับวงจรตรวจจับกรอบสัญญาณ (Envelope detector) แล้วก็จะทำให้เกิดการตรวจจับแยกสัญญาณ $m(t)$ กลับออกมาได้

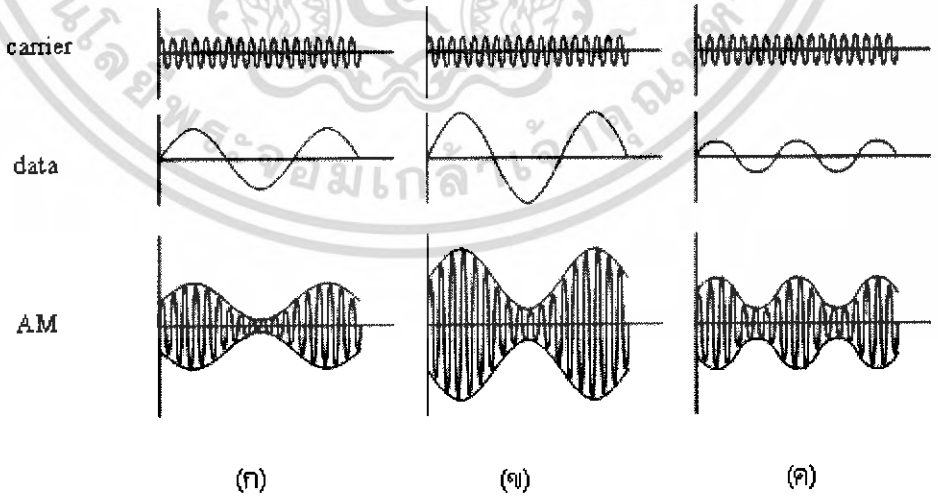
อย่างไรก็ดี สมการของสัญญาณเอเอ็ม (Standard AM) โดยทั่วไปนั้นนิยมเขียนดัดแปลงไปจาก (2-13) เล็กน้อยคือ

$$\phi_{AM}(t) = A_c(1 + k_a m(t))\cos(\omega_c t) \qquad (2\text{-}14)$$

โดยที่ k_a นั้นเทียบได้กับ $\frac{1}{A_c}$ และมีชื่อเรียกว่าความไวของการมอดูเลตแอมปลิจูด (AM modulation sensitivity) แต่อย่างไรก็ดี k_a นี้ไม่จำเป็นต้องเท่ากับ $\frac{1}{A_c}$ แต่จะต้องมีค่าที่ไม่ทำให้ $1 + k_a m(t)$ มีค่าเป็นลบ

รูปคลื่นของสัญญาณเอเอ็ม

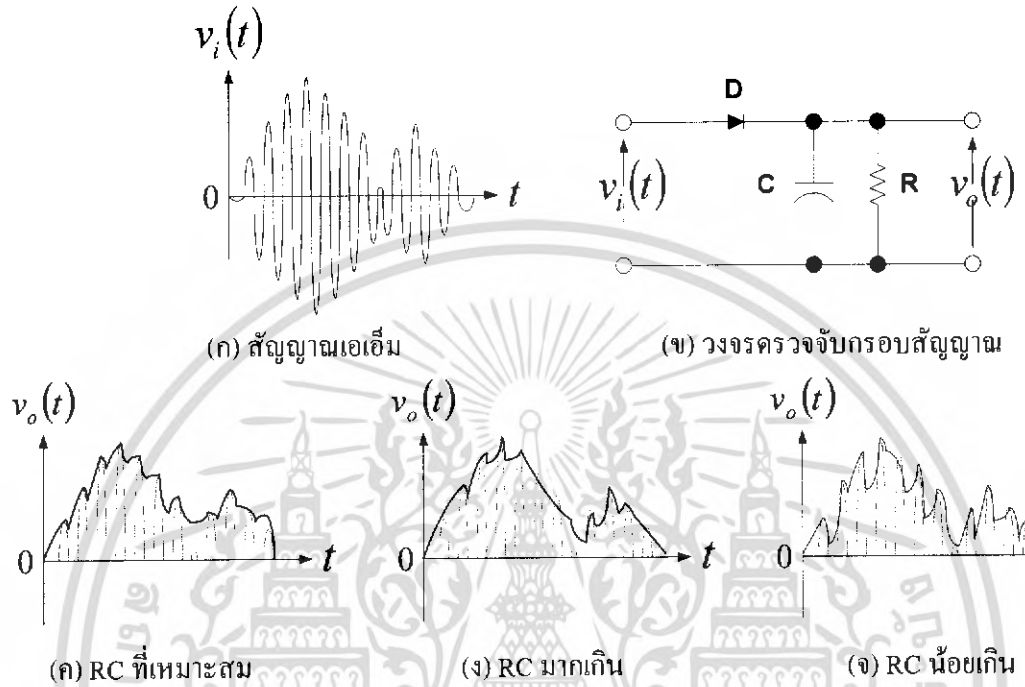
จากรูปที่ 2.16 (ก) จะเห็นว่ารูปของสัญญาณเอเอ็ม นั้นจะมีความถี่ตามคลื่นพาห์แต่ความสูงจะเปลี่ยนตามสัญญาณข้อมูล ซึ่งเมื่อสัญญาณข้อมูลมีความสูงมากที่สุดทางด้านบวก รูปสัญญาณเอเอ็มจะมีความถี่มากที่สุดและคลื่นสัญญาณเอเอ็มมีความสูงต่ำสุด เมื่อสัญญาณข้อมูลมีค่าลบสูงสุดจากรูปที่2.16 (ข) สัญญาณข้อมูลมีความสูงเพิ่มขึ้นจะทำให้รูปเอเอ็มมีค่าสูงสุดมากกว่าเดิมและค่าต่ำสุดต่ำกว่าเดิม ส่วนในรูปที่2.16 (ค) สัญญาณข้อมูลมีขนาดเล็กลงแต่ความถี่มากขึ้น จะทำให้รูปคลื่นเอเอ็ม มีค่าสูงสุดและต่ำสุดลดลง และอัตราการแกว่งขึ้นลงเพิ่มขึ้นเท่ากับความถี่ของสัญญาณข้อมูล



รูปที่ 2.16 รูปสัญญาณเอเอ็ม ที่เปลี่ยนแปลงตามสภาวะของสัญญาณข้อมูล

2.10 วงจรตรวจจับกรอบสัญญาณ

วงจรตรวจจับกรอบสัญญาณ คือวงจรที่ให้เอาท์พุตมีลักษณะเป็นไปตามกรอบหรือแนวทางเดินของยอดคลื่นของสัญญาณอินพุต วงจรตรวจจับกรอบสัญญาณแบบง่ายจะเป็นวงจรชนิดไม่เชิงเส้นดังแสดงในรูปที่ 2.17



(ก) สัญญาณเอเอ็ม (ข) วงจรตรวจจับกรอบสัญญาณ

(ค) RC ที่เหมาะสม (ง) RC มากเกิน (จ) RC น้อยเกิน

รูปที่ 2.17 ประกอบการอธิบายการทำงานของวงจรตรวจจับกรอบสัญญาณ

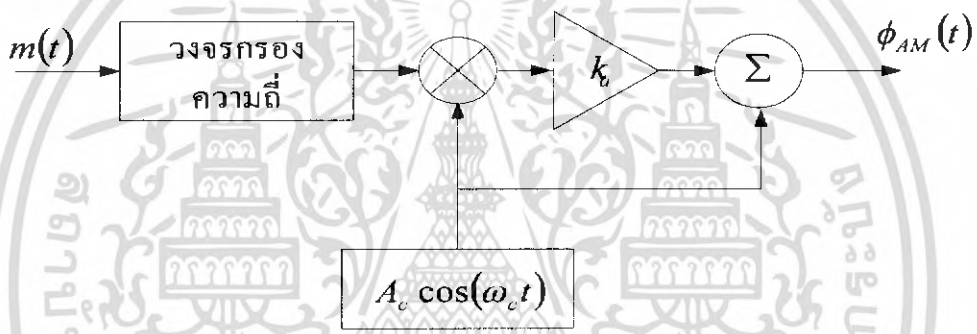
การทำงานของวงจรนี้ เมื่อมีสัญญาณ $\phi_{AM}(t)$ เข้ามาสู่วงจรในตอนแรกนั้น ขณะที่สัญญาณมีค่าเพิ่มขึ้นในทางบวก ไดโอด D จะนำกระแส ทำให้เกิดการประจุตัวเก็บประจุ C ให้มีแรงดันเอาท์พุตเกิดขึ้นคร่อม C ตามค่าการเพิ่มของสัญญาณอินพุต จนกระทั่งถึงค่าสูงสุด (Peak) หรือยอดของสัญญาณ แต่ต่อจากนี้เมื่ออินพุต $\phi_{AM}(t)$ ลดลงก็จะทำให้ค่าแรงดันไฟฟ้าทางอาโนดของ D ต่ำกว่าคาโทด ในสถานการณ์เช่นนี้ไดโอด D จะหยุดนำกระแสและกั้นไม่ให้ C คายประจุผ่านตัวมันได้ ดังนั้นการคายประจุของ C จึงมีทางเดียวคือ คายประจุผ่าน R ไปอย่างช้า ๆ จนกระทั่งถึงเวลาที่ $\phi_{AM}(t)$ จะมีค่ากลับเป็นบวก และมีค่าเพิ่มขึ้นมากกว่าแรงดันไฟฟ้าคร่อม C ไดโอด D จึงนำกระแสอีก และการประจุของ C ทำให้เอาท์พุตเป็นไปตามค่าของสัญญาณ $\phi_{AM}(t)$ อีกไปจนถึงค่ายอดของคลื่นพาห์ถัดมา แล้วพฤติกรรมการทำงานจะเกิดซ้ำในลักษณะเดิมอีกกล่าวคือการคลายประจุของ C ผ่าน R อีกครั้งหนึ่ง พฤติกรรมเช่นนี้จะเกิดซ้ำแล้วซ้ำอีกเรื่อยไป ดังนั้นถ้าเราเลือกค่าคงตัวเชิงเวลา (Time constant) RC ที่ใช้ให้เหมาะสม ก็จะได้เอาท์พุตของวงจรเป็นไปตามเส้นทึบดังรูปที่ 2.17 (ค) แต่ถ้าค่าคงตัวเชิงเวลาของวงจรตรวจจับกรอบสัญญาณมีค่ามากเมื่อเทียบกับคาบเวลาที่สั้นที่สุดของสัญญาณที่ประกอบอยู่ใน $m(t)$ แล้วก็จะทำให้เอาท์พุตของวงจรเกิดลดตามยอดคลื่นของ $\phi_{AM}(t)$ ไม่ทันจะทำให้เกิดการผิดเพี้ยนของสัญญาณเอาท์พุตดังรูปที่ 2.17 (ง) และในกรณีที่เกิดการใช้ RC มีค่าน้อยเมื่อเทียบกับคาบเวลาของ

คลื่นพาห์ก็จะทำให้เกิดการกระเพื่อม (Ripple) บนสัญญาณเอาท์พุตมากดังรูปที่ 2.17(จ) อย่างไรก็ดี ค่าความถี่ของคลื่นพาห์ ω_c ที่ใช้ในทางปฏิบัตินั้นจะมีค่าสูงกว่าความถี่ของสัญญาณที่เป็นส่วนประกอบของ $m(t)$ มาก หรือกล่าวอีกนัยหนึ่งได้ว่า $m(t)$ จะเปลี่ยนแปลงช้ามากเมื่อเทียบกับคลื่นพาห์ที่ใช้จึงทำให้การเลือกค่าเวลาคงตัวของวงจร RC ทำได้อย่างเหมาะสม ง่าย กล่าวคือจะทำให้การกระเพื่อมบนเอาท์พุตของวงจรมีค่าน้อยมากได้ จนไม่ต้องคำนึงถึงในทางปฏิบัติ เงื่อนไขสำหรับการเลือก ค่า RC ที่เหมาะสมสามารถสรุปได้ในรูป ของอสมการดังนี้

$$\frac{1}{f_c} << RC << \frac{1}{f_m} \tag{2-15}$$

โดยในที่นี้ f_c และ f_m คือค่าความถี่ของคลื่นพาห์และค่าความถี่สูงสุดที่ประกอบอยู่ในสัญญาณข่าวสาร $m(t)$ ตามลำดับ

โดยอาศัยสมการที่ (2-13) เราสามารถสร้างสัญญาณเอเอ็มได้โดยระบบที่ง่ายดังรูปที่ 2.18



รูปที่ 2.18 แผนผังระบบการสร้างสัญญาณเอเอ็มแบบง่าย

อย่างไรก็ดีเราสามารถที่จะสร้างสัญญาณเอเอ็ม โดยวิธีอื่น เช่น ใช้วงจรสวิตช์ หรือวงจรไม่เชิงเส้นได้เช่นเดียวกับการสร้างสัญญาณดีเอสบี เอสซี

2.11 การแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก (D/A)

อุปกรณ์อิเล็กทรอนิกส์ที่ทำหน้าที่แปลงสัญญาณข้อมูลดิจิตอลให้เป็นสัญญาณอนาล็อกย่านความถี่เสียง เราเรียกว่า โมเด็ม (MODEM หรือ *Mo*dulator *Dem*odulator)

เทคนิคการแปลงสัญญาณข้อมูลดิจิตอลให้เป็นสัญญาณอนาล็อกนั้นมีอยู่ด้วยกัน 3 วิธีคือ

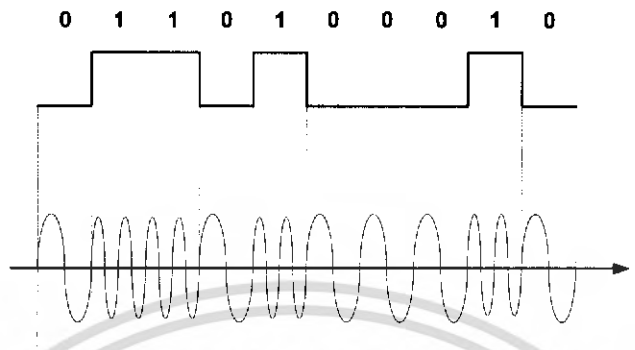
1.การมอดูเลตเชิงเลขทางแอมปลิจูด (Amplitude-Shift keying หรือ ASK)

2.การมอดูเลตเชิงเลขทางความถี่ (Frequency-Shift Keying หรือ FSK)

3.การมอดูเลตเชิงเลขทางเฟส (Phase-Shift Keying หรือ PSK)

การมอดูเลตเชิงเลขทางความถี่ (FSK)

ในการมอดูเลตแบบ FSK ขนาดของคลื่นพาห์จะไม่เปลี่ยนแปลง ที่เปลี่ยนแปลงคือความถี่ของคลื่นพาห์ นั่นคือเมื่อบิตมีค่าเป็น "1" ความถี่ของคลื่นพาห์จะสูงกว่าปกติ และเมื่อบิตมีค่าเป็น "0" ความถี่ของคลื่นพาห์ก็จะต่ำกว่าปกติ

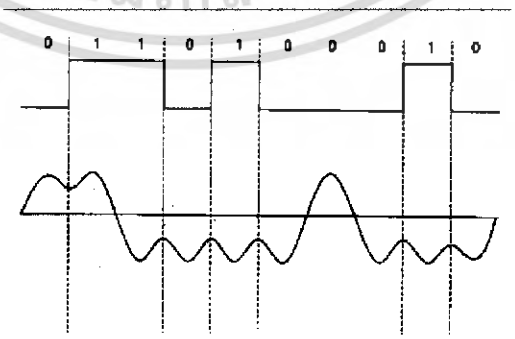


รูปที่ 2.19 การแปลงสัญญาณ D/A แบบ FSK

เทคนิคการมอดูเลตแบบจีเอฟเอสเค (GFSK: Gaussian Frequency Shift Keying) ซึ่งพัฒนามาจากเทคนิคการมอดูเลตแบบเอฟเอสเค (FSK) เพื่อเพิ่มประสิทธิภาพในการใช้แบนด์วิดท์ เนื่องจากการนำสัญญาณดิจิตอลมามอดูเลตนั้นสเปกตรัมของสัญญาณเอฟเอสเคที่ได้จะมีแบนด์วิดท์ที่กว้างซึ่งเป็นการไม่ประหยัด ทำให้ได้ช่วงสัญญาณในการส่งที่น้อย ดังนั้นจึงได้ใช้เกาส์เซียนฟิลเตอร์กรองสัญญาณข่าวสารก่อนที่จะนำไปมอดูเลต สเปกตรัมของสัญญาณที่ได้หลังการมอดูเลตจะมีแบนด์วิดท์ที่แคบกว่า เรียกวิธีการนี้ว่า การมอดูเลตแบบจีเอฟเอสเค (GFSK)

การมอดูเลตเชิงเลขทางเฟส(PSK)

หลักการของ PSK คือค่าของขนาดและความถี่ของคลื่นพาห์จะไม่มีการเปลี่ยนแปลง แต่ที่จะเปลี่ยนคือเฟสของสัญญาณ กล่าวคือเมื่อมีการเปลี่ยนแปลงสภาวะของบิตจาก "1" ไปเป็น "0" หรือเปลี่ยนจาก "0"ไปเป็น "1" เฟสของคลื่นจะเปลี่ยน (Shift) ไป180 องศาด้วย วิธีการแบบ PSK จะมีสัญญาณรบกวนเกิดขึ้นน้อยที่สุด ได้สัญญาณที่มีคุณภาพดีที่สุด แต่วงจรการทำงานจะยุ่งยากกว่าและราคาสูงกว่า



รูปที่ 2.20 การแปลงสัญญาณ D/A แบบ PSK

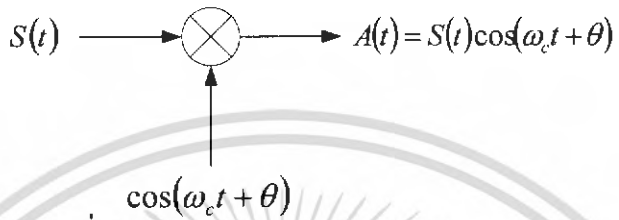
การมอดูเลตเชิงเลขทางแอมปลิจูด (ASK)

คลื่นที่ถูกมอดูเลตทางแอมปลิจูด ถูกทำให้แอมปลิจูดของคลื่นพาห์เปลี่ยนแปลงเป็นสัดส่วนกับสัญญาณข้อมูล ซึ่งทำได้โดยการคูณคลื่นพาห์ไซน์เข้ากับสัญญาณข้อมูล คลื่นที่ถูกมอดูเลตทางแอมปลิจูด A(t) สามารถกำหนดโดย

$$A(t) = S(t) \times \cos(\omega_c t + \theta)$$

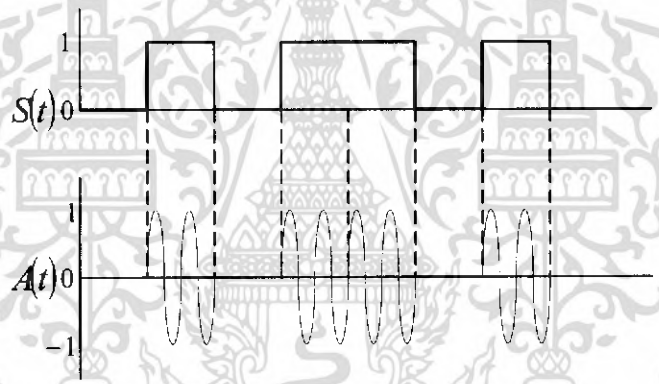
$S(t)$ = สัญญาณข้อมูล

$\cos(\omega_c t + \theta)$ = คลื่นพาห์



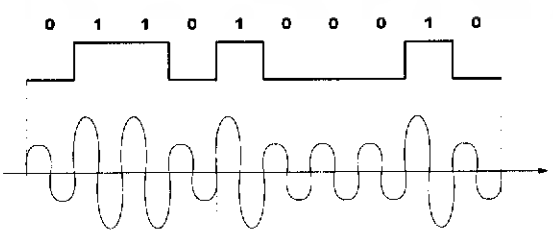
รูปที่ 2.21 รูปแบบของการมอดูเลตทางแอมปลิจูด

พิจารณาสัญญาณข้อมูล ซึ่งเป็นยูนิโพล่าพัลส์ เมื่อพัลส์เป็น 1 จะมีคลื่นพาห์ออกมา และ เมื่อพัลส์เป็น 0 จะไม่มีคลื่นพาห์ออกมา การมอดูเลตทางแอมปลิจูดแบบนี้เรียกว่า ON-OFF ASK หรือ ON-OFF Keying (OOK)



รูปที่ 2.22 ON-OFF ASK

ความถี่ของคลื่นพาห์ (Carrier Wave) ซึ่งทำหน้าที่นำสัญญาณอนาล็อกผ่านตัวกลางสื่อสารนั้นจะคงที่ ลักษณะของสัญญาณมอดูเลตเมื่อค่าของบิตของสัญญาณข้อมูลดิจิตอลมีค่าเป็น "1" ขนาดของคลื่นพาห์จะสูงขึ้นกว่าปกติ และเมื่อบิตมีค่าเป็น "0" ขนาดของคลื่นพาห์จะตกลงกว่าปกติ การมอดูเลต ASK มักไม่ค่อยได้รับความนิยม เพราะจะถูกรบกวนจากสัญญาณอื่นได้ง่าย

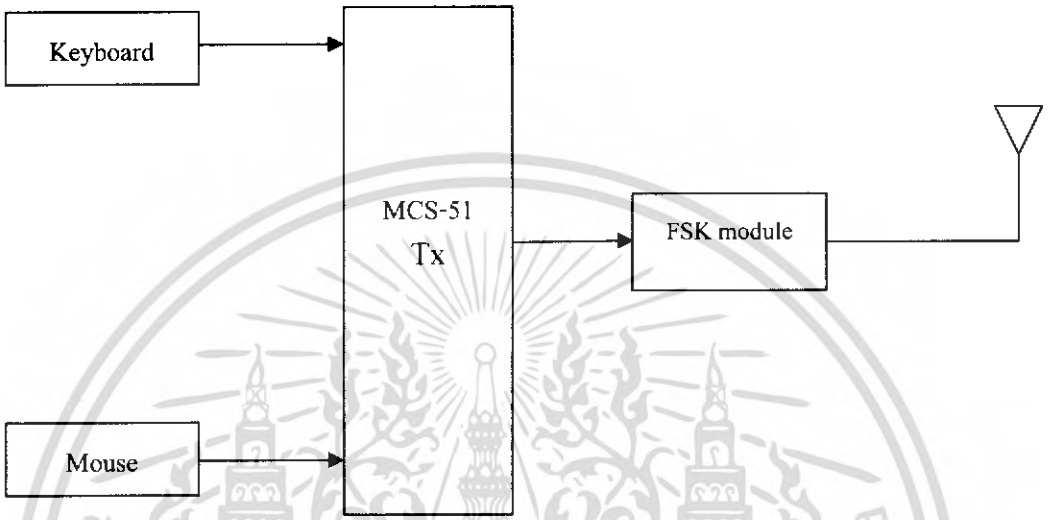


รูปที่ 2.23 การแปลงสัญญาณ D/A แบบ ASK

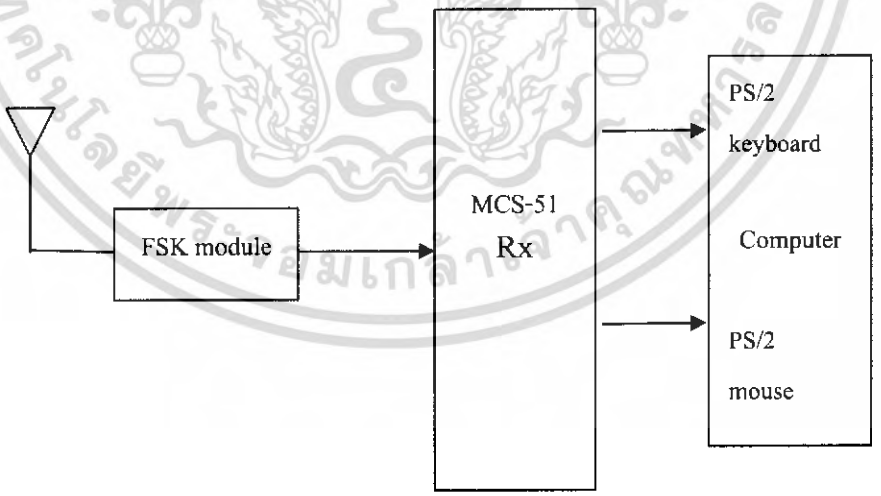
บทที่ 3
การคำนวณและการสร้างภาพรวมของระบบ

3.1 หลักการทำงาน

ประกอบด้วยบล็อกไดอะแกรม 2 ด้านดังนี้



รูปที่ 3.1 บล็อกไดอะแกรมด้านคีย์บอร์ด และเมาส์



รูปที่ 3.2 บล็อกไดอะแกรมด้านคอมพิวเตอร์

โดยระบบจะแบ่งออกเป็น 2 ด้าน คือ

1. ด้านเมาส์และคีย์บอร์ดเชื่อมต่อกับไมโครคอนโทรลเลอร์
2. ด้านคอมพิวเตอร์เชื่อมต่อกับไมโครคอนโทรลเลอร์

3.1.1 ไมโครคอนโทรลเลอร์ด้านคีย์บอร์ดและเมาส์ (ด้านส่ง)จะมีการทำงานดังนี้

เมาส์และคีย์บอร์ดจะส่งข้อมูลมาเก็บและประมวลผลที่ไมโครคอนโทรลเลอร์ แล้วจัดการกับข้อมูลในรูปของโปรโตคอลเพื่อส่งข้อมูลต่อไปยังเอฟเอสเคโมดูล (FSK module) เพื่อส่งสัญญาณไปทางฝั่งคอมพิวเตอร์โดยเมื่อต้องการจำลองให้ไมโครคอนโทรลเลอรเสมือนเป็นคอมพิวเตอร์เพื่อส่งข้อมูลสั่งให้เมาส์และคีย์บอร์ดเริ่มทำงานได้ โดยวิธีการนี้เรียกว่าการทำแฮนด์เช็คคิ่ง (Hand-checking)

3.1.2 ไมโครคอนโทรลเลอร์ด้านคอมพิวเตอร์(ด้านรับ)จะมีการทำงานดังนี้

เมื่อภาครับหรือ FSK module จะทำการดีเท็คสัญญาณ FSK ที่ส่งมาแล้วแปลงเป็นชุดข้อมูลเพื่อส่งเข้าไปที่ไมโครคอนโทรลเลอร์เพื่อทำการประมวลผลข้อมูล และแปลงข้อมูลให้เหมาะสมสำหรับการสื่อสารระหว่างคีย์บอร์ดและเมาส์กับคอมพิวเตอร์ จึงทำการส่งข้อมูลไปยังคอมพิวเตอร์ โดยในด้านนี้จะทำการแฮนด์เช็คคิ่งเพื่อใช้ติดต่อกับคอมพิวเตอร์เสมือนว่ามีการต่อคีย์บอร์ดและเมาส์อยู่

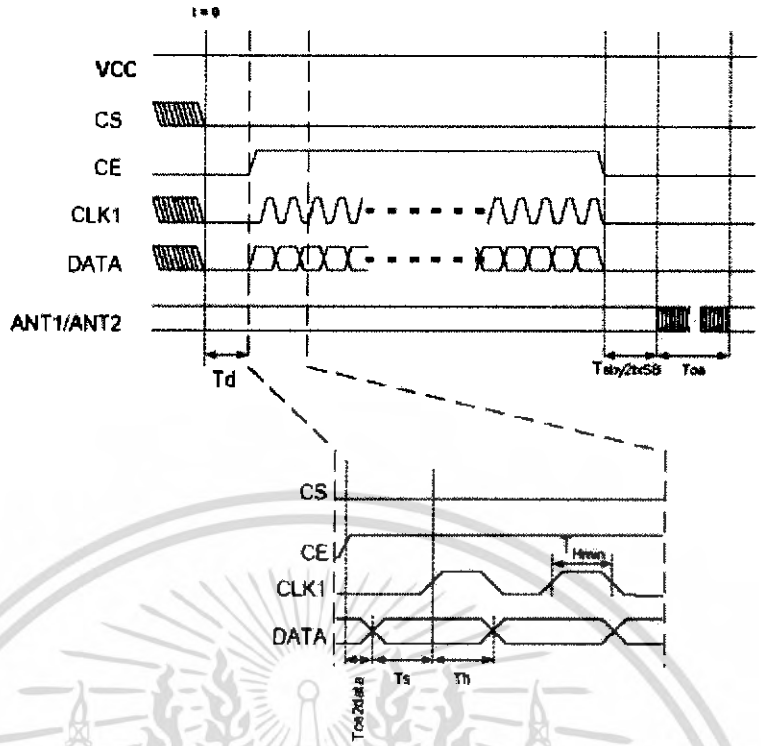
3.2 การทำงานของ FSK module

ในที่นี้เราเลือกใช้ TRW-2.4 GHz เป็น FSK module ซึ่งมีข้อดีดังนี้คือ สามารถกำหนดให้เป็นตัวรับหรือส่งได้โดยใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุมการทำงาน และยังสามารถทำการเข้ารหัสและตรวจสอบความผิดพลาดได้ภายในตัวของมันเอง เมื่อทำงานภายใต้การทำงานแบบ ช็อคเบิร์ส (Shock Burst mode)

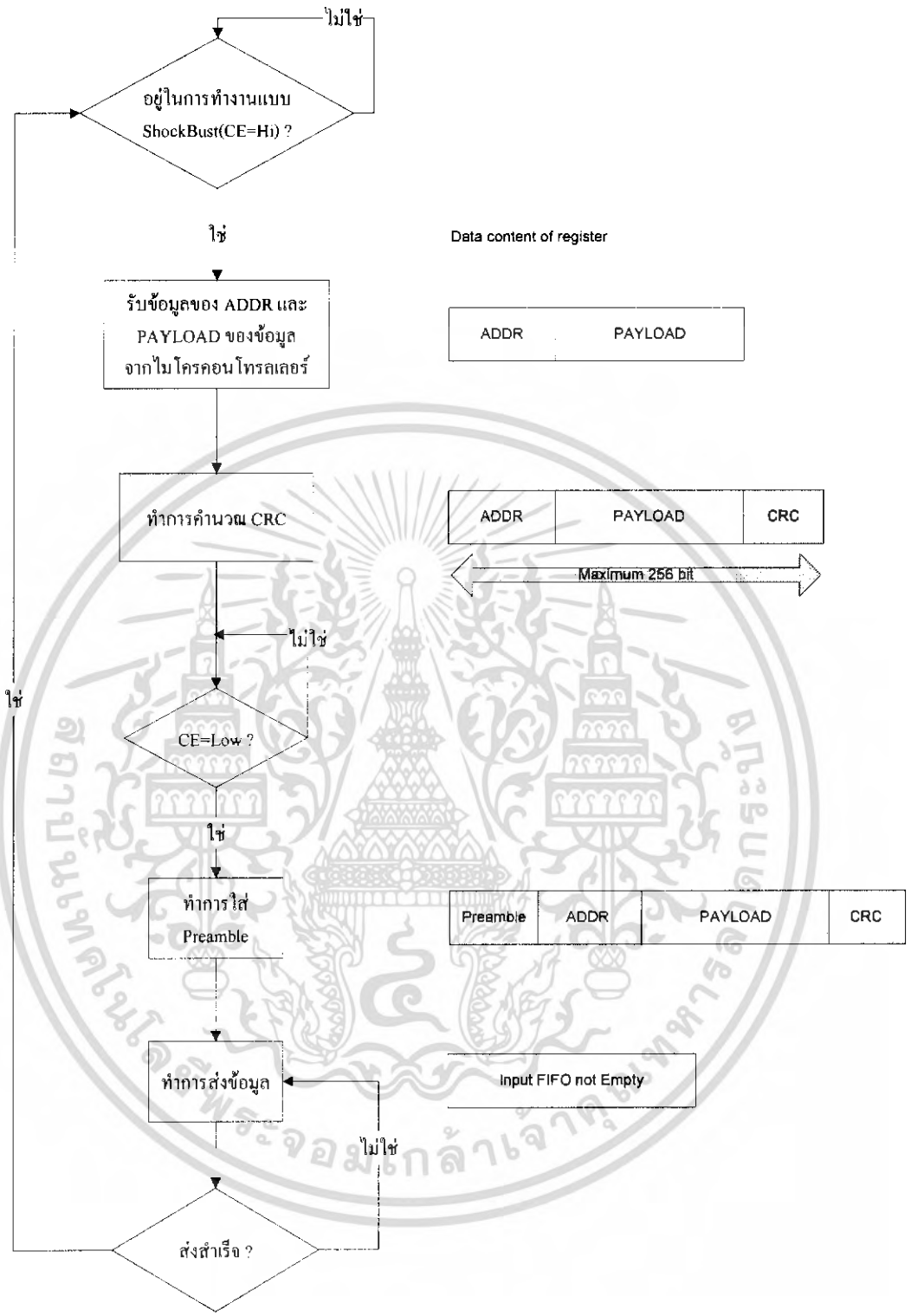
3.2.1 การทำงานเมื่อเป็นตัวส่ง

ขาที่ต่อกับไมโครคอนโทรลเลอร์ : CE,CLK1,DATA โดยมีขั้นตอนการทำงานดังนี้

- เมื่อไมโครคอนโทรลเลอร์ มีข้อมูลที่จะส่ง จะกำหนดให้ CE เป็น High เป็นการกระตุ้นให้ TRW-2.4 GHz ทำการประมวลผลข้อมูล
- แอดเดรสของตัวรับ (RX address) และข้อมูลในเพย์โหลด (payload) จะจับเวลาเข้าสู่ช่วงเวลาระบบย่อยของ TRW-2.4 GHz
- ไมโครคอนโทรลเลอร์กำหนดให้ CE เป็น Low จะเป็นการกระตุ้นให้ทำการส่งแบบ Shock Bust
- TRW-2.4 GHz Shock Bust:
  - RF front end มีกำลังงานสูงขึ้น
  - RF package ถูกทำให้สมบูรณ์ (เพิ่ม preamble, คำนวณ CRC)
  - ข้อมูลถูกส่งด้วยความเร็วสูง (250 kbps หรือ 1 Mbps แล้วแต่การกำหนด)
  - TRW-2.4 GHz กลับสู่สถานะเตรียมพร้อม เมื่อสิ้นสุดการทำงาน



รูปที่ 3.3 แผนผังเวลาของ Shock Burst แบบส่ง

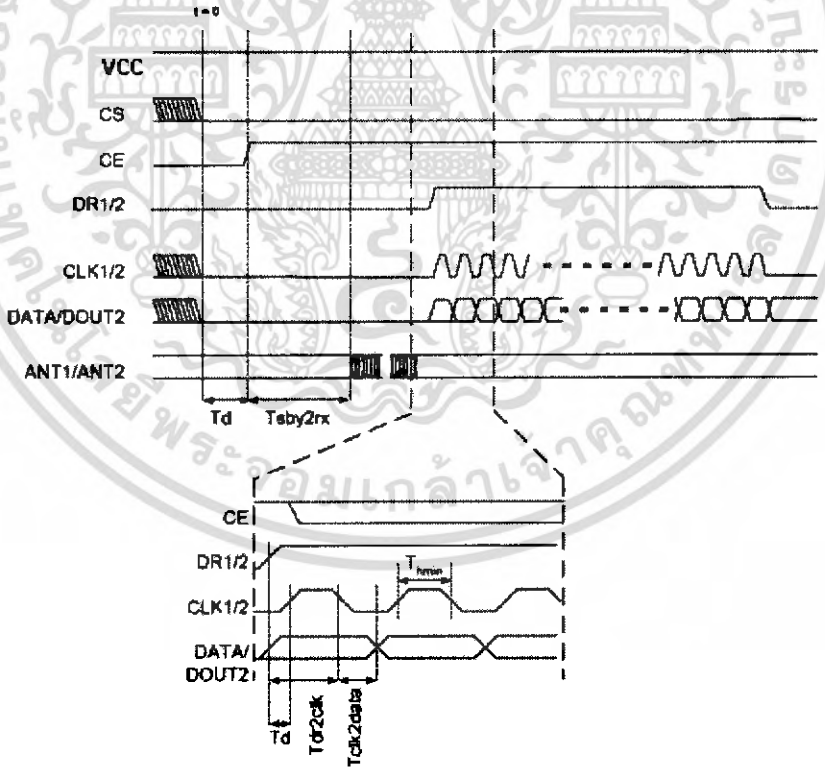


รูปที่ 3.4 แสดงแผนภูมิการทำงานของ TRW-2.4 GHz เมื่อทำงานใน ShockBurst แบบส่ง

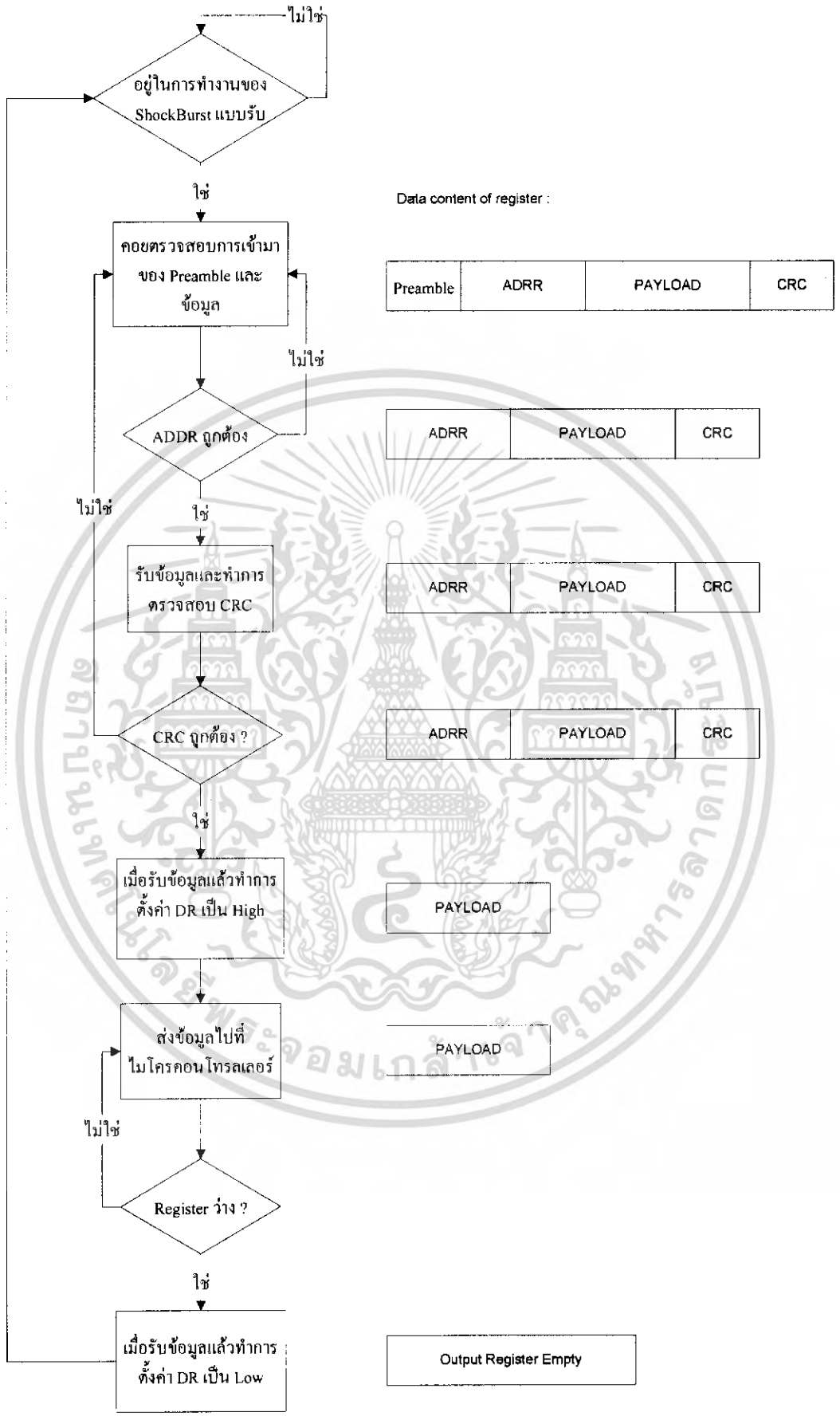
3.2.2 การทำงานเมื่อเป็นตัวรับ

ขาที่ต่อกับไมโครคอนโทรลเลอร์ : CE,DR1,CLK1,DATA โดยมีขั้นตอนการทำงานดังนี้

- แอดเดรสที่ถูกต้องและขนาดของเพย์โหลดของสัญญาณ RF packages ที่เข้ามาจะถูกกำหนดให้กับ TRW-2.4 GHz
- กระตุ้นให้เป็นการทำงานของตัวรับโดยกำหนดให้ CE เป็น Low
- หลังจาก 200 ไมโครวินาที TRW-2.4 GHz จะคอยตรวจสอบการสื่อสารที่เข้ามา
- เมื่อรับ package ที่ถูกต้องแล้ว TRW-2.4 GHz จะทำการลบ preamble,address,CRC ออกไป
- จากนั้น TRW-2.4 GHz จะแจ้งให้ไมโครคอนโทรลเลอร์รู้โดยการกำหนดให้ DR1 มีสถานะเป็น High
- ไมโครคอนโทรลเลอร์จะ กำหนดให้ CE เป็น Low เพื่อป้องกัน RF front end (โหมดที่กระแสต่ำ)
- ไมโครคอนโทรลเลอร์จะส่งสัญญาณนาฬิกาที่เหมาะสมกับข้อมูลในเพย์โหลด
- เมื่อข้อมูลทั้งหมดถูกรับเสร็จสิ้นแล้ว TRW-2.4 GHz จะกำหนดให้ค่า DR1 เป็น Low อีกครั้งและพร้อมที่จะรับข้อมูลชุดต่อไปที่กำลังจะเข้ามา โดยที่ CE จะรักษาสถานะเป็น High อยู่ระหว่างทำการรับข้อมูล ถ้า CE ถูกกำหนดให้มีสถานะเป็น Low จะเป็นการเริ่มรับข้อมูลใหม่



รูปที่ 3.5 แผนผังเวลาของ Shock Burst แบบรับ



รูปที่ 3.6 แสดงแผนภูมิการทำงานของ TRW-2.4 GHz เมื่อทำงานใน Shock Burst แบบรับ

3.3 การทำงานของไมโครคอนโทรลเลอร์ในการสื่อสารระหว่างอุปกรณ์กับคอมพิวเตอร์

3.3.1 การทำงานของทางด้านส่ง

จะมีโปรแกรมการทำงานในส่วนย่อยๆดังนี้

<u>การรับข้อมูลจากอุปกรณ์มายังไมโครคอนโทรลเลอร์</u>

รูปที่ 3.7 แผนภูมิการทำงานของการรับข้อมูลจากอุปกรณ์มายังไมโครคอนโทรลเลอร์

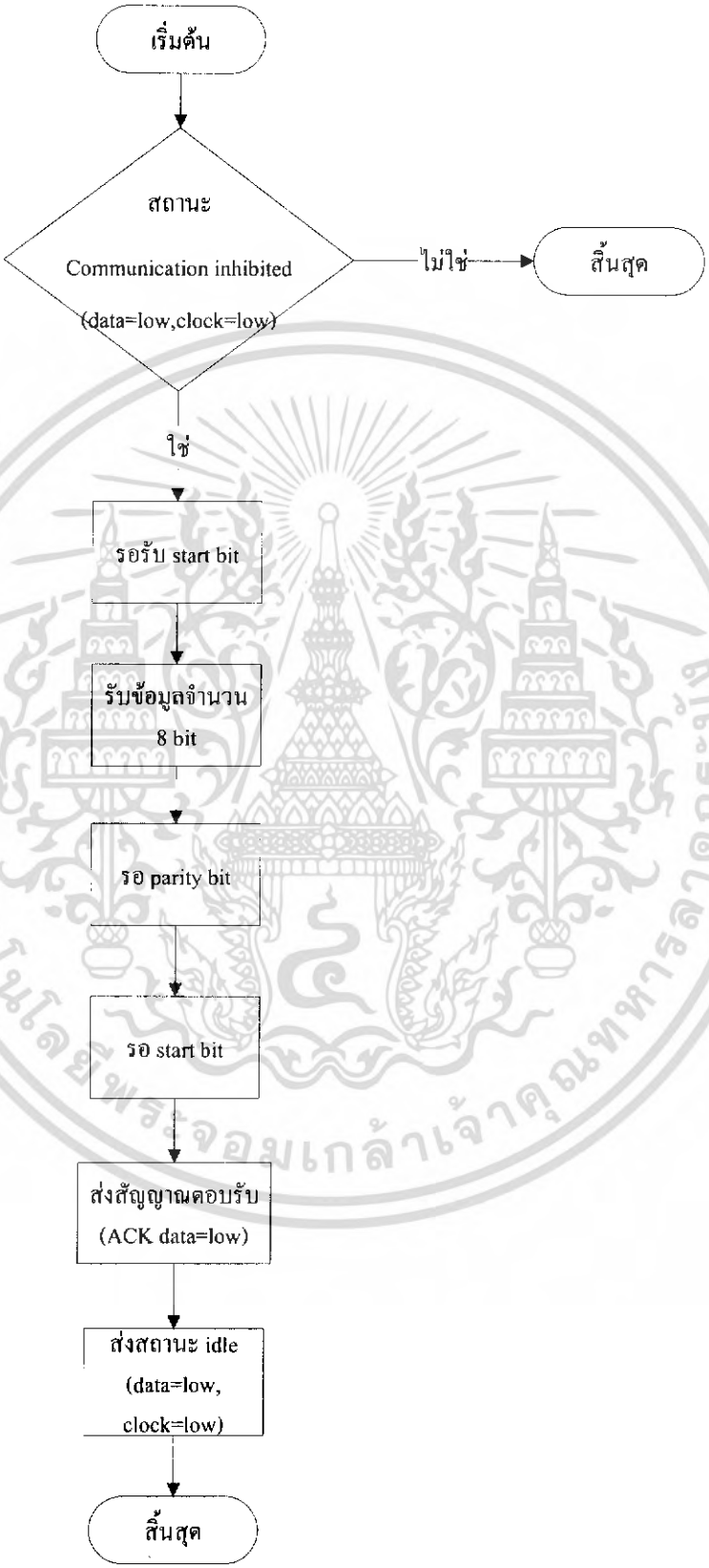
การส่งข้อมูลจากไมโครคอนโทรลเลอร์ไปยังอุปกรณ์

รูปที่ 3.8 แผนภูมิการทำงานของการส่งข้อมูลจากไมโครคอนโทรลเลอร์ไปยังอุปกรณ์

3.3.2 การทำงานของทางด้านรับ

จะมีโปรแกรมการทำงานในส่วนย่อยๆดังนี้

<u>การรับข้อมูลจากคอมพิวเตอร์มายังไมโครคอนโทรลเลอร์</u>



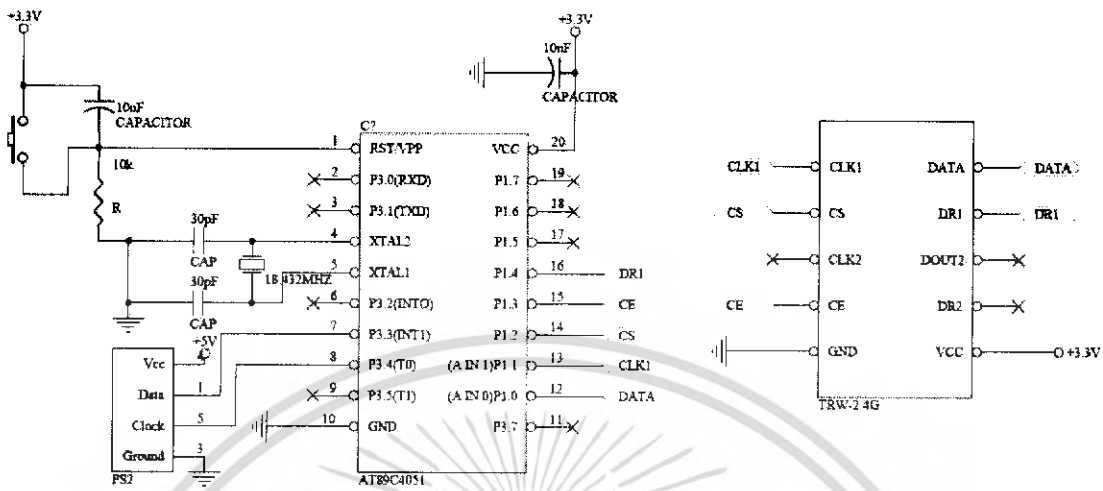
รูปที่ 3.9 แผนภูมิการทำงานของการรับข้อมูลจากคอมพิวเตอร์มายังไมโครคอนโทรลเลอร์

การส่งข้อมูลจากไมโครคอนโทรลเลอร์ไปยังคอมพิวเตอร์

รูปที่ 3.10 แผนภูมิการทำงานของการส่งข้อมูลจากไมโครคอนโทรลเลอร์ไปยังคอมพิวเตอร์

3.4 การส่งข้อมูลจากคีย์บอร์ดไปยังคอมพิวเตอร์

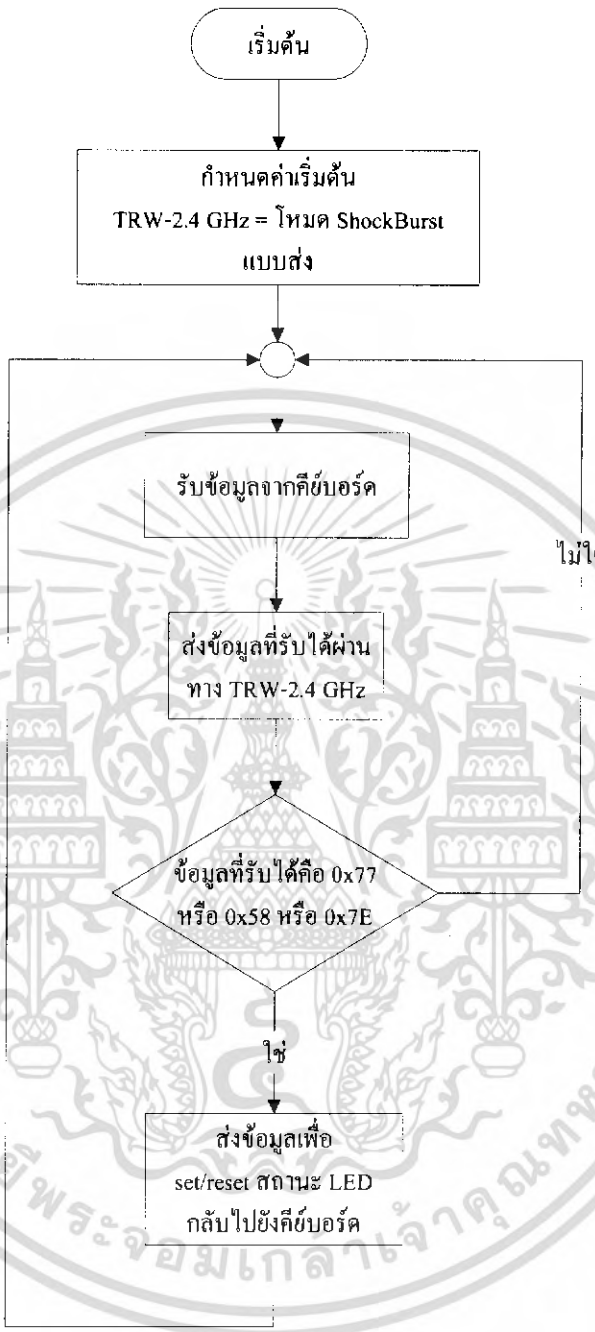
ในภาคการศึกษานี้ได้ทำการส่งข้อมูลจากคีย์บอร์ดไปยังคอมพิวเตอร์โดยมีวงจรการทำงานทั้งทางด้านส่งและรับเหมือนกันดังรูปที่ 3.11



รูปที่ 3.11 วงจรที่ใช้ในการส่งข้อมูลจากคีย์บอร์ดไปยังคอมพิวเตอร์

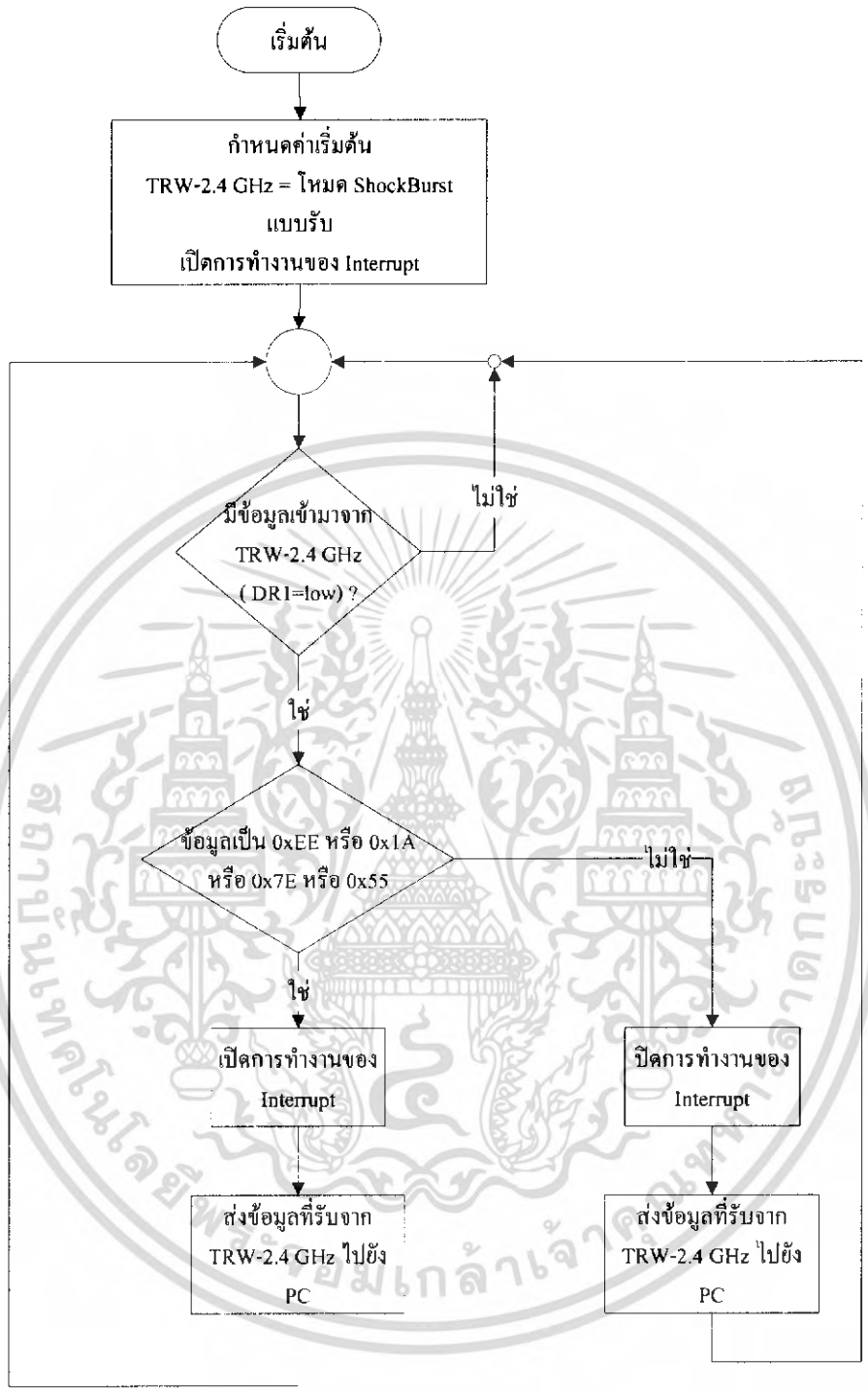
โดยในการทำงานนั้นข้อมูลที่ส่งมาจากคีย์บอร์ดจะมีจำนวน 11 บิต โดยจะมีข้อมูลที่เป็นสแกนโค้ดอยู่จำนวน 8 บิต ไมโครคอนโทรลเลอร์ทางด้านส่งจะทำการส่งเฉพาะข้อมูลในส่วนนี้ ออกไปยัง FSK module จากนั้นเมื่อ FSK module ทางด้านรับรับข้อมูลได้ จะทำการส่งต่อให้กับไมโครคอนโทรลเลอร์ทางด้านรับ เพื่อทำการสร้างสัญญาณที่เหมือนกับที่คีย์บอร์ดส่งออกมา เพื่อส่งเข้าคอมพิวเตอร์

3.4.1 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านส่ง

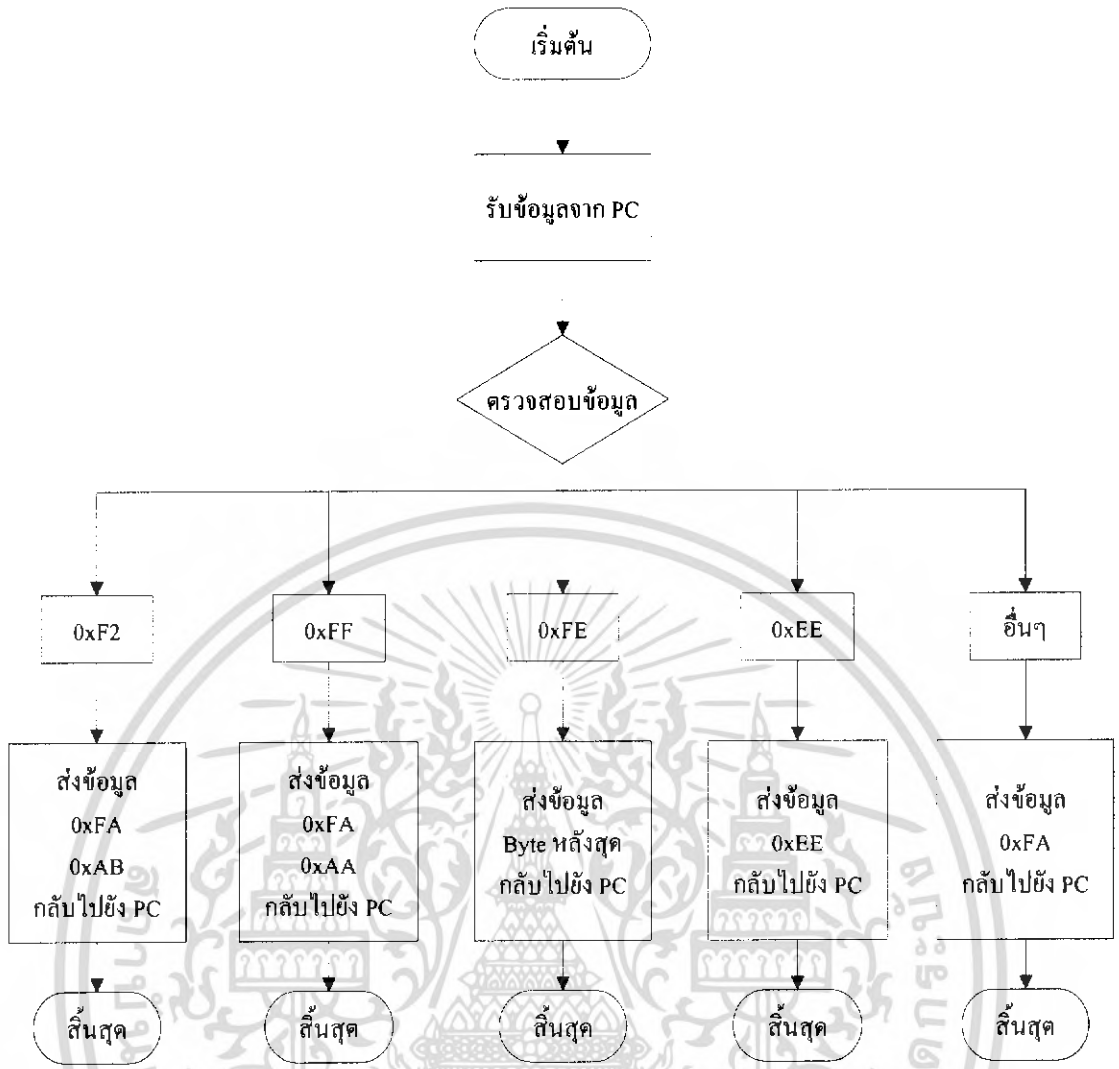


รูปที่ 3.12 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านส่ง

3.4.2 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านรับ



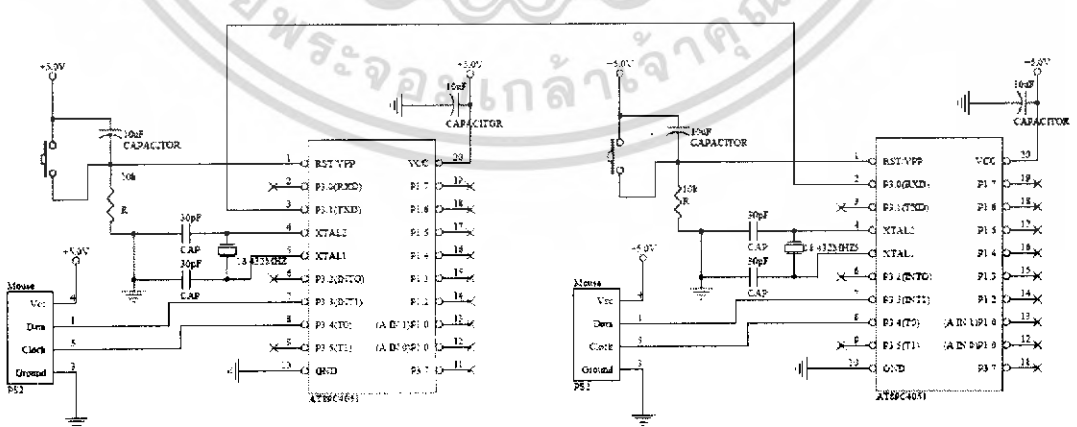
รูปที่ 3.13 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านรับ



รูปที่ 3.14 แผนภูมิการอินเตอร์รัพต์ของไมโครคอนโทรลเลอร์ทางด้านรับ

3.5 การส่งข้อมูลจากอุปกรณ์เมาส์ไปยังคอมพิวเตอร์ผ่านทางพอร์ตสื่อสารอนุกรม

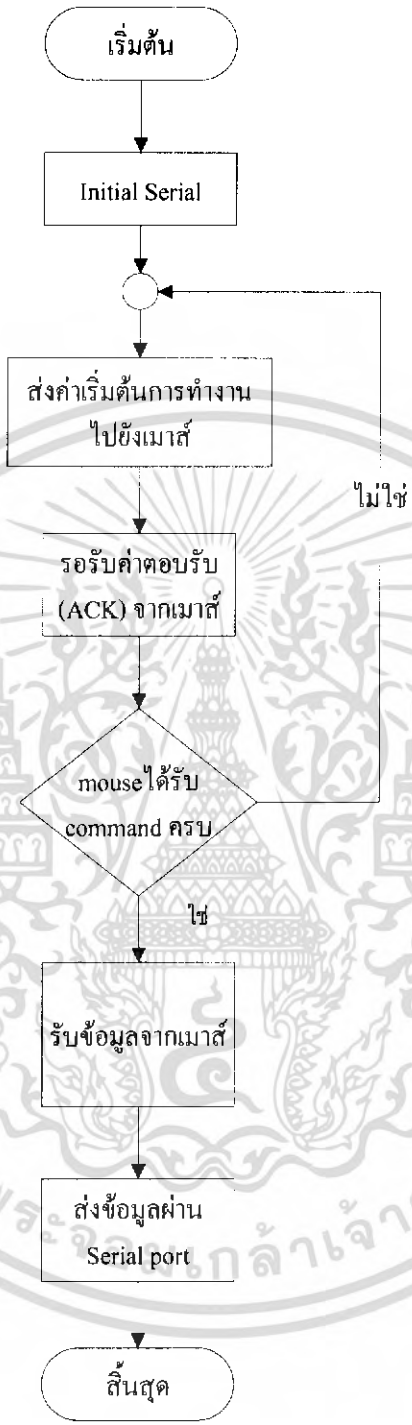
ในที่นี้ได้ทำการส่งข้อมูลจากเมาส์ไปยังคอมพิวเตอร์โดยมีวงจรการทำงานดังนี้



รูปที่ 3.15 วงจรที่ใช้ในการส่งและรับข้อมูลจากเมาส์ไปยังคอมพิวเตอร์ผ่านทางพอร์ตสื่อสารอนุกรม

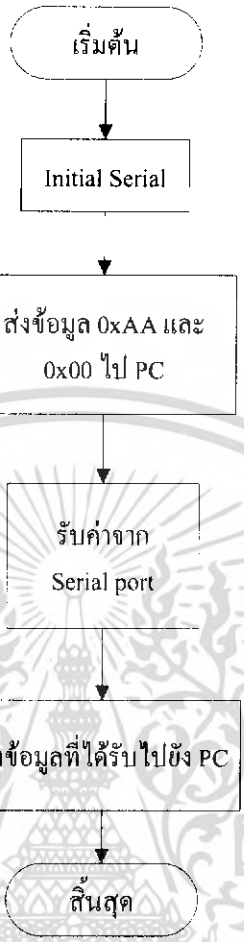
ข้อมูลที่ส่งมาจากเมาส์จะมีข้อมูลจำนวนถึง 4 ไบต์ (สำหรับ Microsoft Intellimouse) แต่ละไบต์จะมีข้อมูลจำนวน 8 บิต (แต่ในการทำงานจะมี 1 start bit , ข้อมูล 8 bits, 1 Parity bit และ 1 stop bit) โดยไบต์แรกจะเป็นข้อมูลของการกดซ้ายหรือขวา ,ไบต์ที่สองเป็นข้อมูลของการเคลื่อนที่ซ้ายหรือขวา (แนวแกน X) , ไบต์ที่สามเป็นการเคลื่อนที่ขึ้นหรือลง(ในแนวแกน Y) และไบต์สุดท้ายเป็นการเคลื่อนที่ของ scrolling wheel (แนวแกน Z) โดยในการทำงานนั้นจะต้องเซตค่าการทำงานเริ่มต้นให้กับอุปกรณ์เมาส์ก่อน โดยทำการเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ทางด้านส่งให้พร้อมจะรับสัญญาณที่เมาส์ส่งออกมาเพื่อบอกให้คอมพิวเตอร์รู้สถานะของตัวเมาส์เองว่าพร้อมจะทำงานแล้ว ไมโครคอนโทรลเลอร์ทางด้านส่ง จะทำหน้าที่เป็นเสมือนคอมพิวเตอร์ที่คอยส่งค่าคำสั่งต่างๆที่จะตั้งค่าการทำงานของเมาส์ไปยังอุปกรณ์เมาส์แล้วจะทำการรอค่าคำสั่งตอบรับของเมาส์เพื่อทำการส่งคำสั่งต่อไป และรอการตอบสนองของเมาส์จนกระทั่งครบทุกคำสั่ง เมาส์จึงพร้อมทำงานได้ และเมื่อเมาส์ส่งข้อมูลจำนวน 4 ไบต์ออกมาจะทำการส่งผ่านทางพอร์ตสื่อสารอนุกรมเพื่อส่งไปทางไมโครคอนโทรลเลอร์ทางด้านรับ ซึ่งการทำงานเริ่มต้นของไมโครคอนโทรลเลอร์ทางด้านรับจะทำงานเสมือนเป็นอุปกรณ์เมาส์ที่ทำการส่งค่าคำสั่งให้คอมพิวเตอร์รู้ว่ามีการอุปกรณ์ต่ออยู่และคอยรับคำสั่งจากคอมพิวเตอร์เพื่อส่งค่าตอบสนองกลับให้แก่คอมพิวเตอร์ จากนั้นจะทำการรับค่าข้อมูลที่ส่งมาทางพอร์ตสื่อสารอนุกรมแล้วนำข้อมูลส่งผ่านไปยังคอมพิวเตอร์

3.5.1 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านส่งเมื่อทำการส่งผ่านพอร์ตสื่อสารอนุกรม

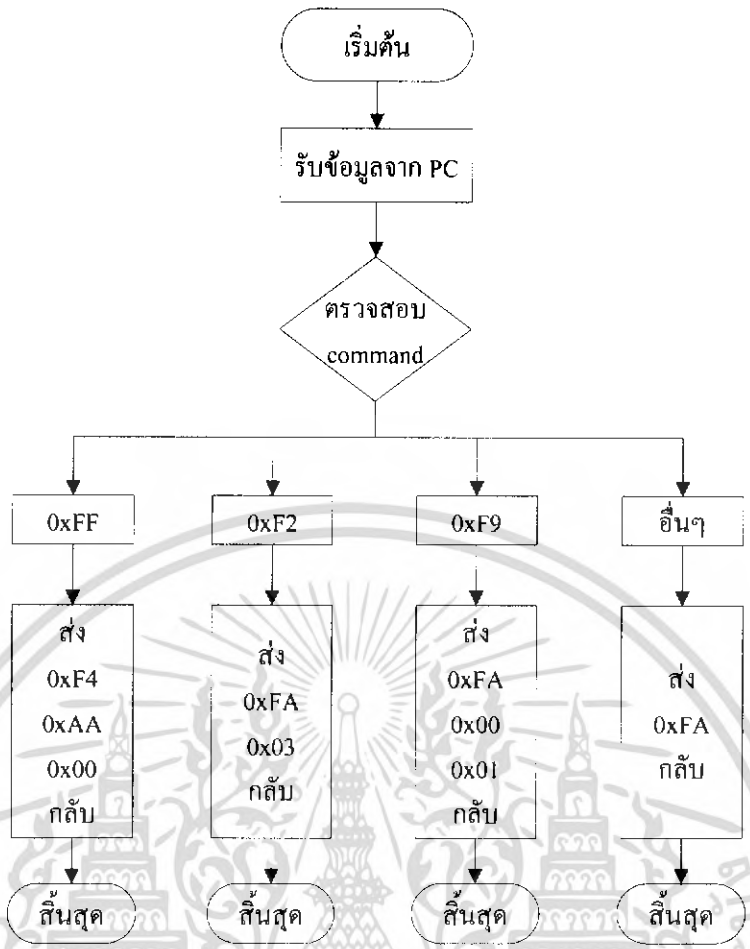


รูปที่ 3.16 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านส่งเมื่อทำการส่งผ่านพอร์ตสื่อสารอนุกรม

3.5.2 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านรับเมื่อรับค่าที่ได้จากพอร์ตสื่อสารอนุกรม

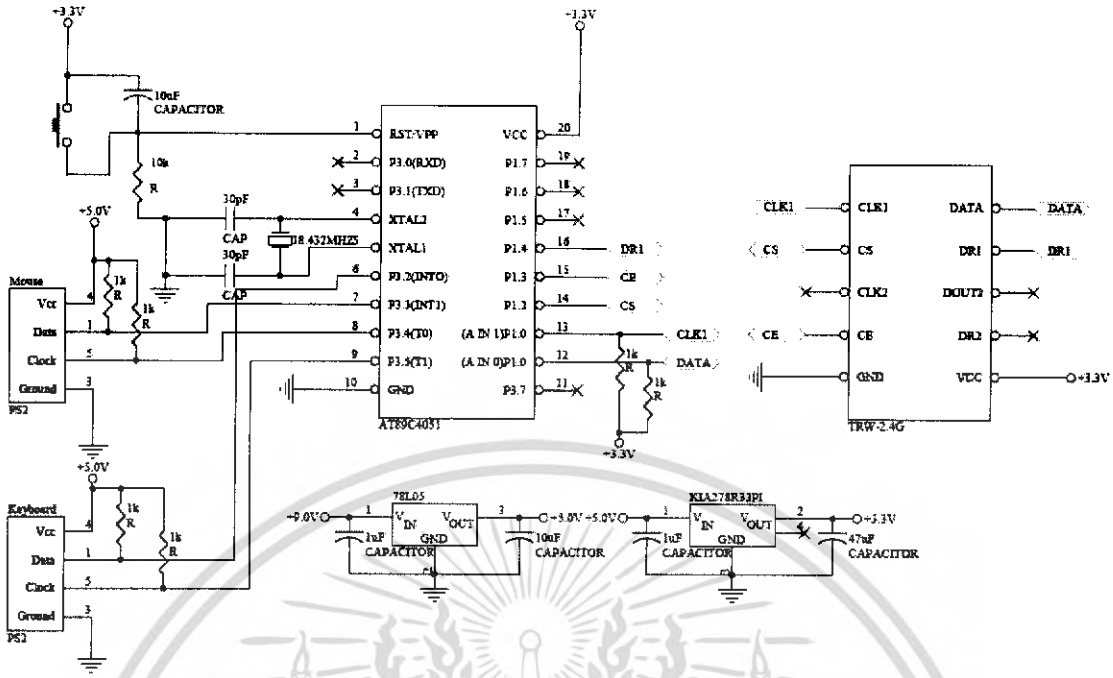


รูปที่ 3.17 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านรับเมื่อทำการรับค่าที่ได้ ผ่านพอร์ตสื่อสารอนุกรม

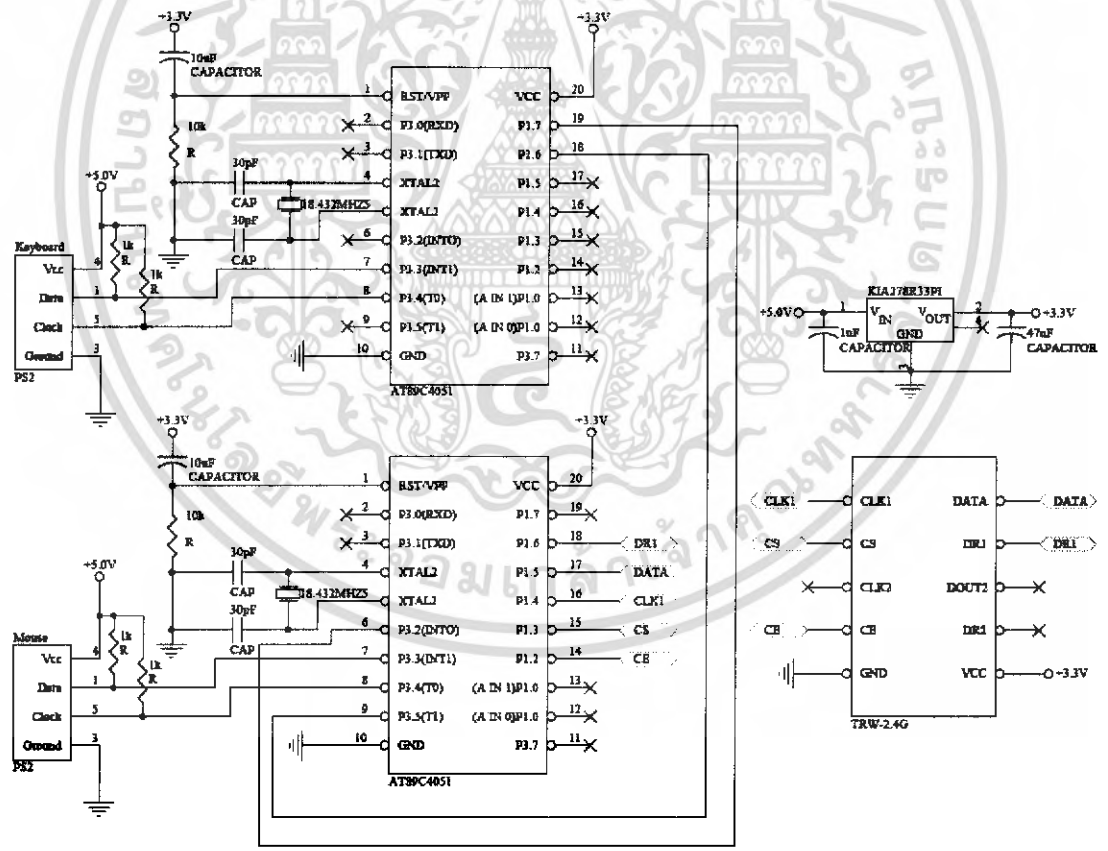


รูปที่ 3.18 แผนภูมิการอินเตอร์รัพต์ของไมโครคอนโทรลเลอร์ทางด้านรับ

3.6 การทำงานของเมาส์และคีย์บอร์ดไร้สาย



รูปที่ 3.19 วงจรของเมาส์และคีย์บอร์ดไร้สายทางด้านส่ง

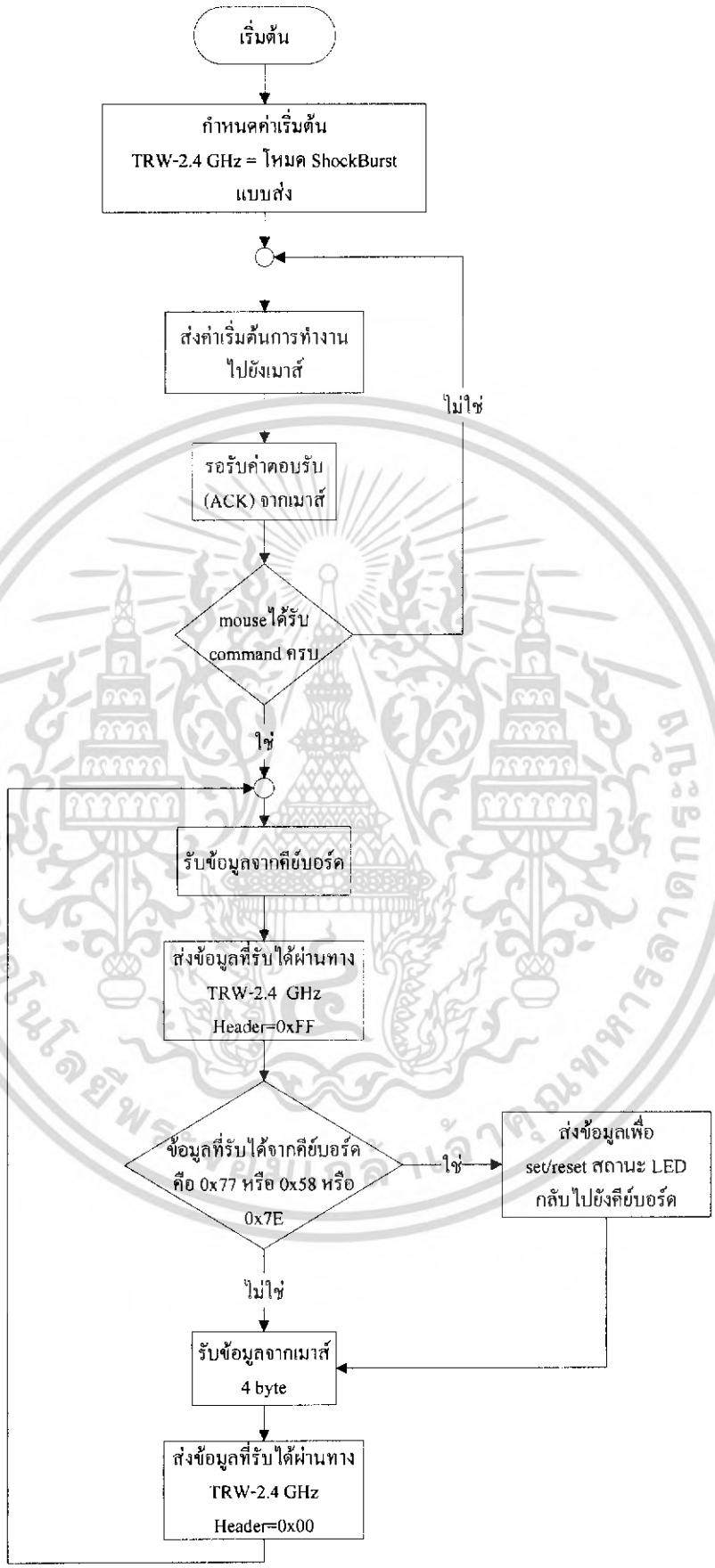


รูปที่ 3.20 วงจรของเมาส์และคีย์บอร์ดไร้สายทางด้านรับ

การทำงานของเมาส์และคีย์บอร์ดไร้สายทางด้านส่งนั้นจะคล้ายกับการทำงานของเมาส์และคีย์บอร์ดเดี่ยวๆ แต่จะต่างกันที่การส่งข้อมูลผ่าน TRW-2.4 GHz นั้นจะทำการใส่ส่วนหัวของข้อมูล (header) เพื่อทำการแยกข้อมูลของเมาส์ และคีย์บอร์ดออกจากกันเพื่อทำให้ด้านรับนั้นสามารถทราบได้ว่าข้อมูลที่ส่งมานั้นเป็นของเมาส์หรือคีย์บอร์ด โดยถ้าเป็นข้อมูลของคีย์บอร์ดจะใช้ header เป็น 0xFF ส่วนข้อมูลของเมาส์จะเป็น 0x00

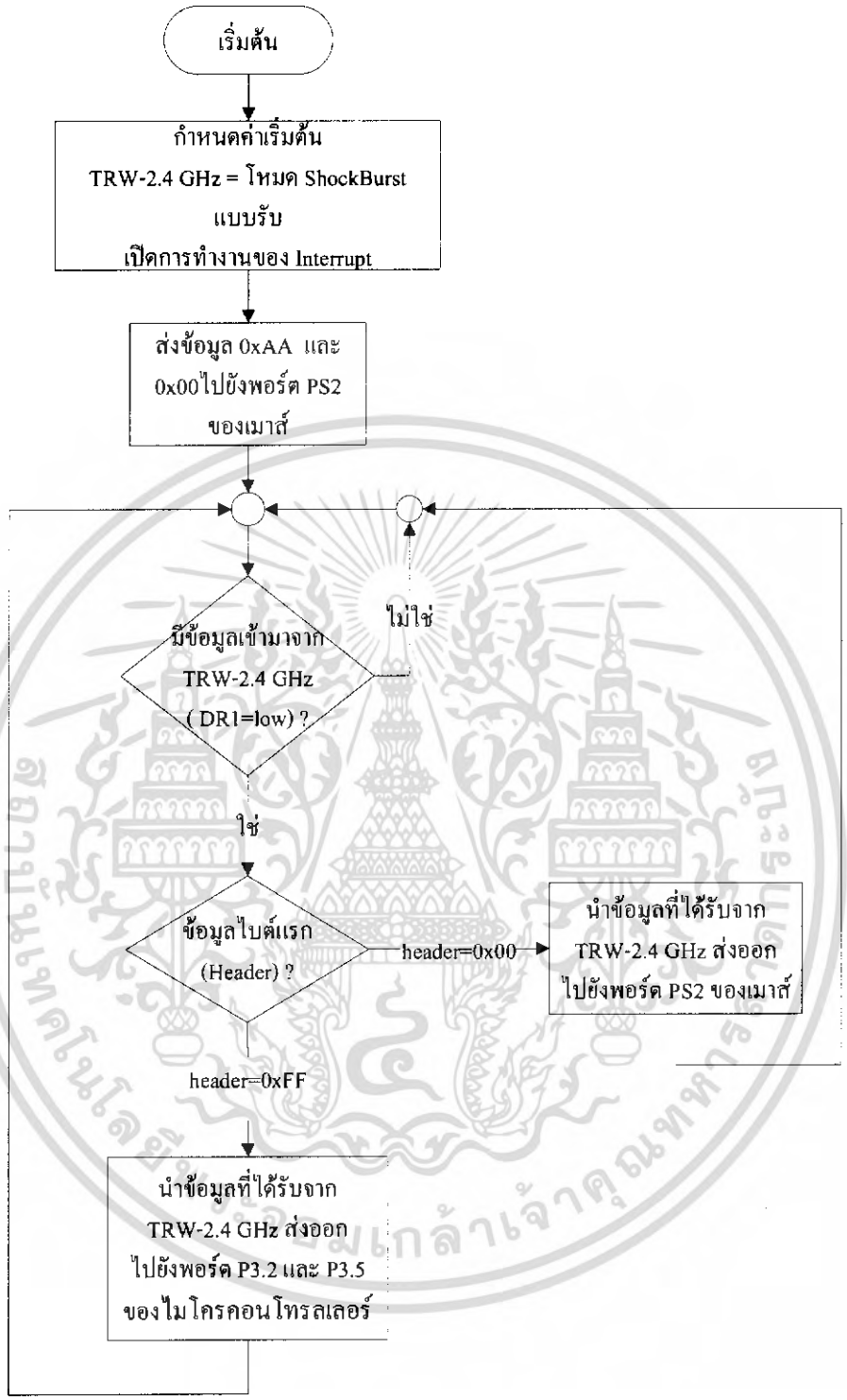
การทำงานของเมาส์และคีย์บอร์ดไร้สายทางด้านรับ จะใช้ไมโครคอนโทรลเลอร์จำนวน 2 ตัว เพื่อทำการเริ่มต้นการทำงานให้กับ เมาส์และคีย์บอร์ด โดยการทำงานของไมโครคอนโทรลเลอร์ที่ต่ออยู่กับคอมพิวเตอร์ที่พอร์ต PS2 ของเมาส์นั้นจะทำการรับข้อมูลจาก TRW-2.4G มาแล้วพิจารณาว่าเป็นข้อมูลของเมาส์หรือคีย์บอร์ด ถ้าเป็นข้อมูลของเมาส์จะทำการส่งข้อมูลของทางพอร์ต P3.4 และ P3.4 เพื่อทำการสื่อสารกับคอมพิวเตอร์ ถ้าเป็นข้อมูลของคีย์บอร์ดจะทำการส่งข้อมูลของทางพอร์ต P3.2 และ P3.5 เพื่อส่งข้อมูลไปให้ไมโครคอนโทรลเลอร์ที่ต่ออยู่กับคอมพิวเตอร์ที่พอร์ต PS2 ของคีย์บอร์ดเพื่อทำการสื่อสารกับคอมพิวเตอร์

3.6.1 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านส่ง

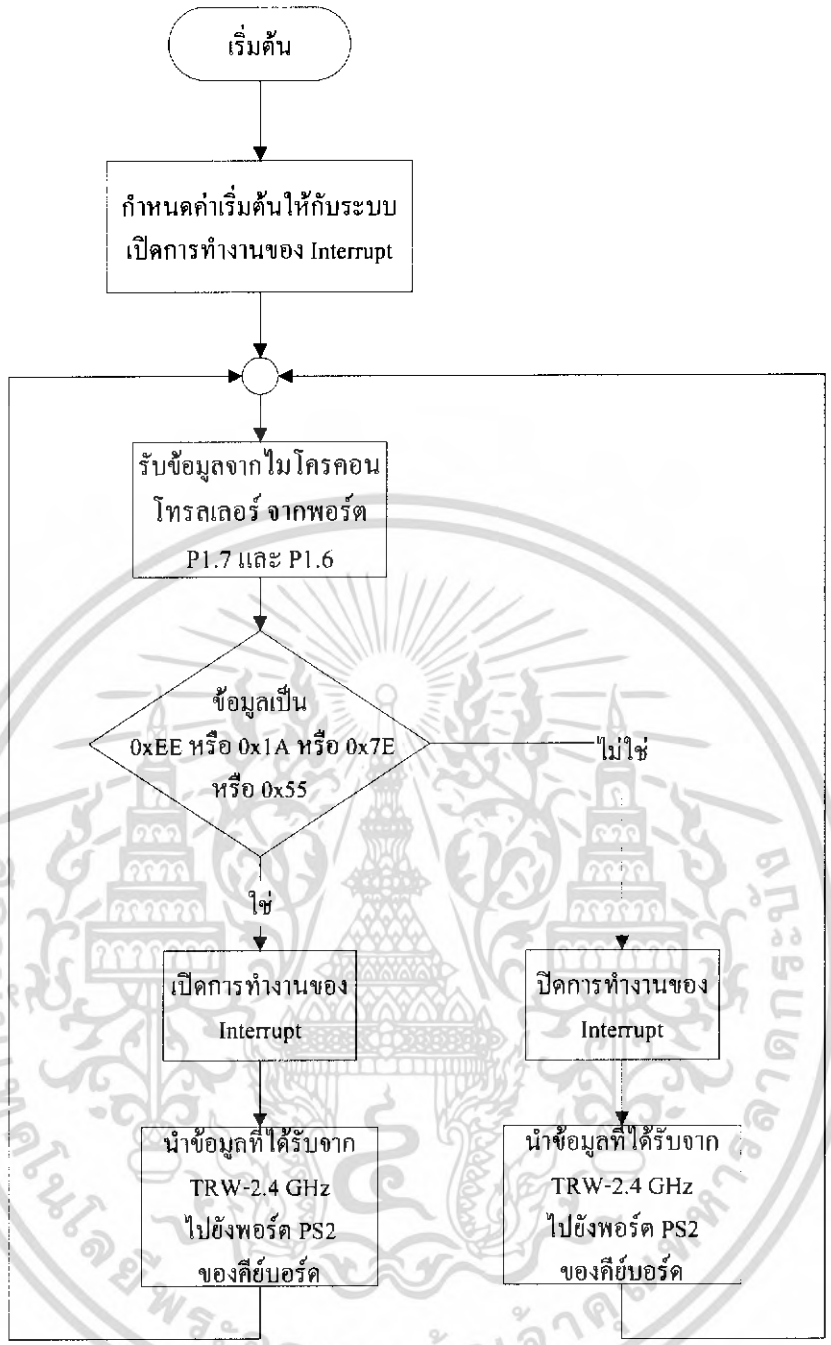


รูปที่ 3.21 แผนภูมิการทำงานของเมาส์และคีย์บอร์ดไร้สายทางด้านส่ง

3.6.2 แผนภูมิการทำงานของไมโครคอนโทรลเลอร์ทางด้านรับ



รูปที่ 3.22 แผนภูมิการทำงานของเมาส์และคีย์บอร์ดไร้สายทางด้านที่ต่ออยู่กับเมาส์



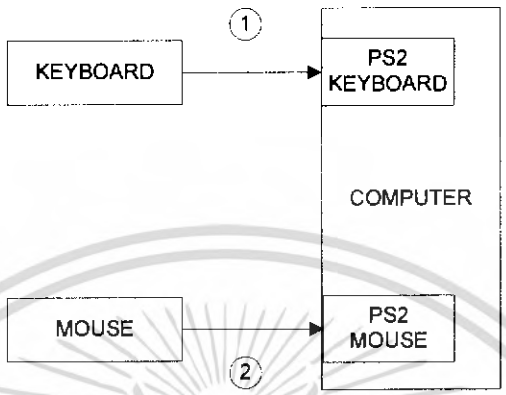
รูปที่ 3.23 แผนภูมิการทำงานของเมาส์และคีย์บอร์ดไร้สายทางด้านที่ต่ออยู่กับคีย์บอร์ด

บทที่ 4

การทดลองและผลการทดลอง

การทดลองที่1

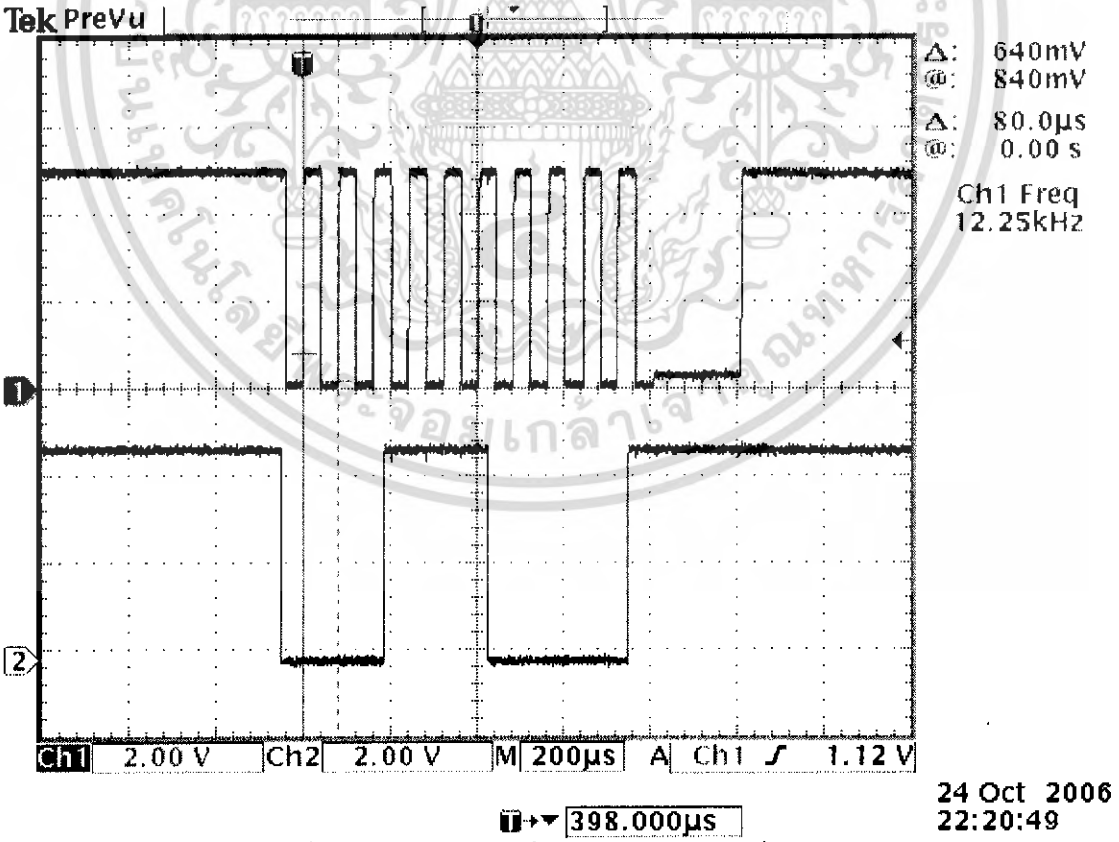
เป็นการศึกษารูปแบบการสื่อสารระหว่างคีย์บอร์ดและเมาส์กับคอมพิวเตอร์ โดยทำการวัดสัญญาณที่ตำแหน่งที่ 1 และ 2 ดังรูปที่ 4.1



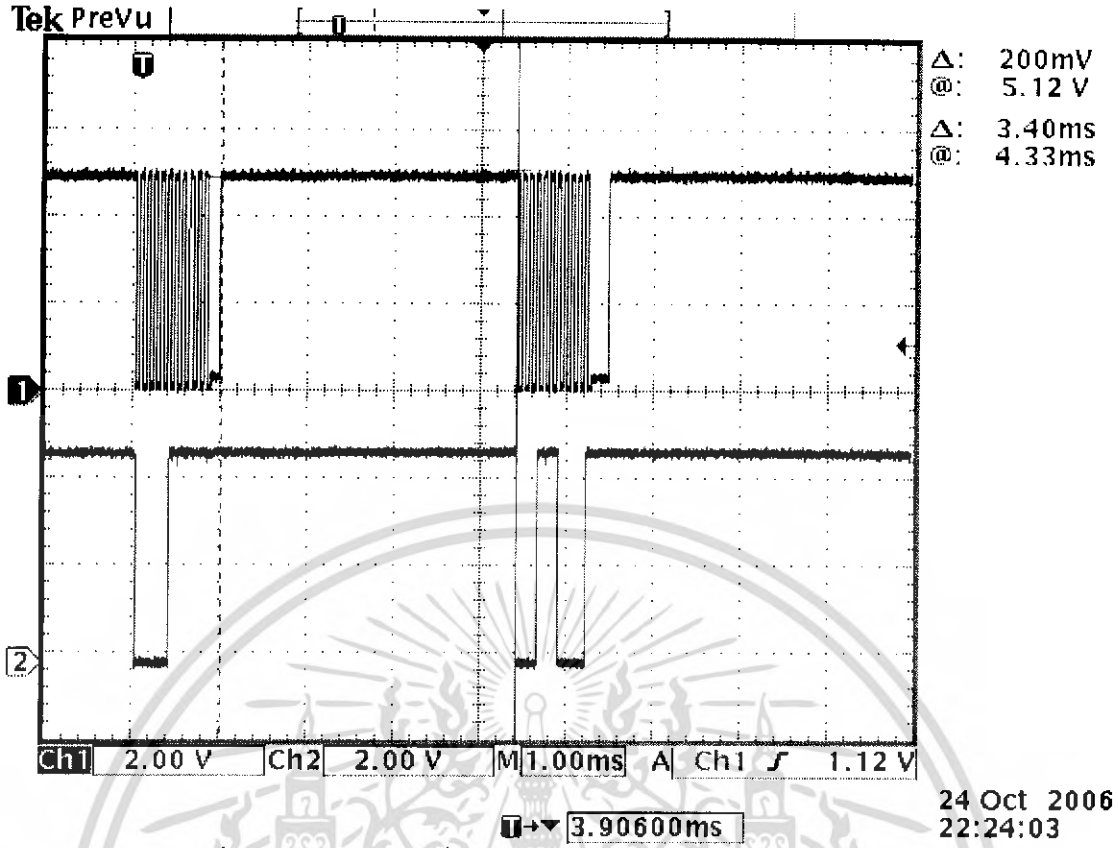
รูปที่ 4.1 บล็อกไดอะแกรมการสื่อสารระหว่างคีย์บอร์ดและเมาส์กับคอมพิวเตอร์

ผลการทดลอง

1. สัญญาณการสื่อสารระหว่างคีย์บอร์ดกับคอมพิวเตอร์ที่จุดที่ 1 จากรูปที่ 4.1 โดยใช้ CH1 เป็นสัญญาณนาฬิกา CH2 เป็นสัญญาณข้อมูล



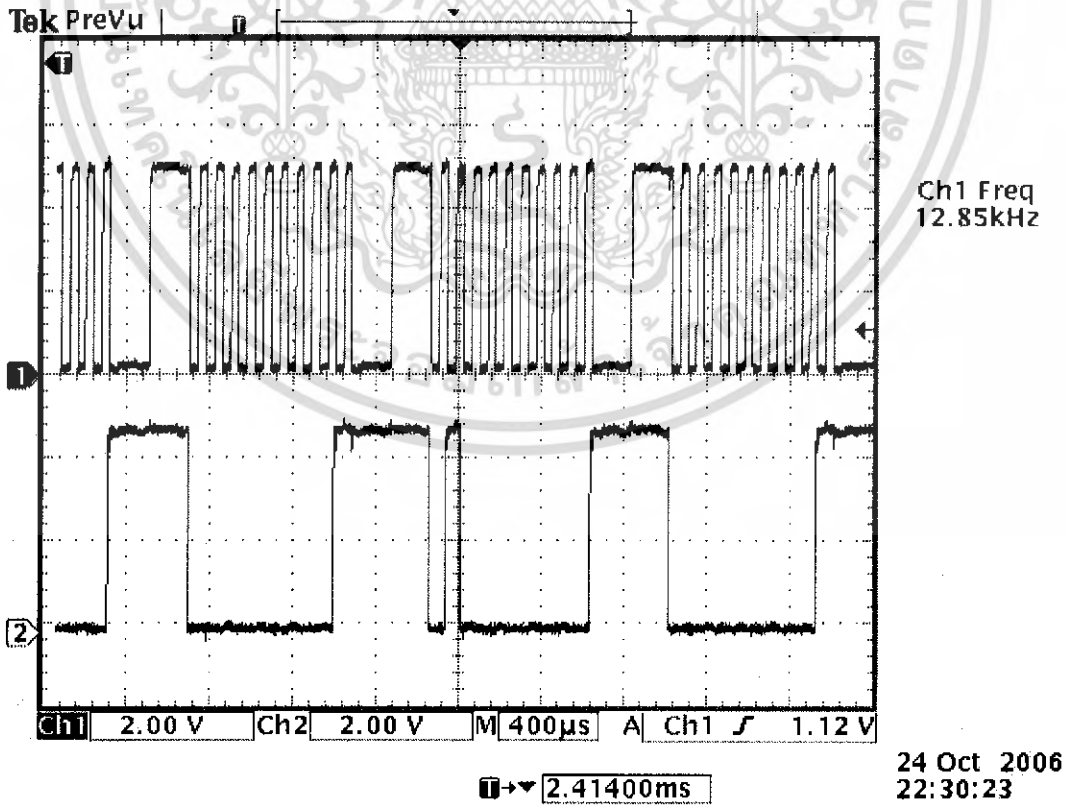
รูปที่ 4.2 แสดงสัญญาณเมื่อทำการกดคีย์บอร์ดที่ปุ่ม “A”



รูปที่ 4.3 แสดงสัญญาณเมื่อทำการปล่อยคีย์บอร์ดหลังจากการกดปุ่ม "A"

2. สัญญาณการสื่อสารระหว่างเมาส์กับคอมพิวเตอร์ที่จุดที่ 2 จากรูปที่ 4.1

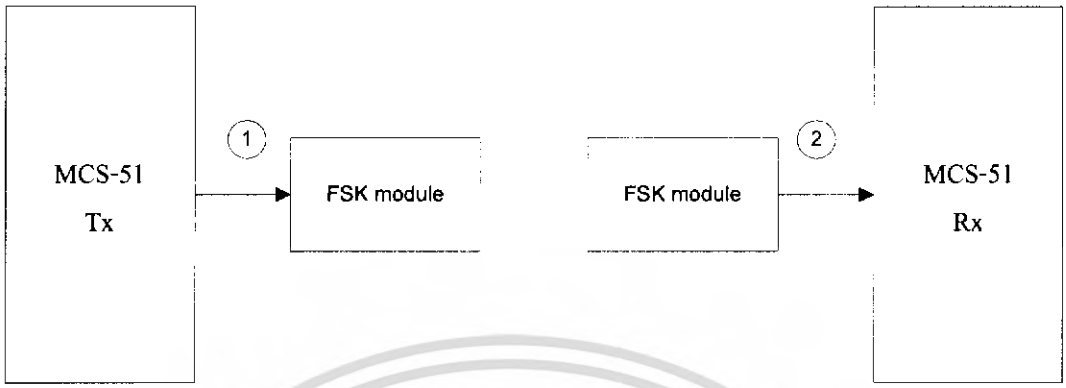
โดยใช้ CH1 เป็นสัญญาณนาฬิกา CH2 เป็นสัญญาณข้อมูล



รูปที่ 4.4 แสดงสัญญาณเมื่อทำการเลื่อนเมาส์

การทดลองที่ 2

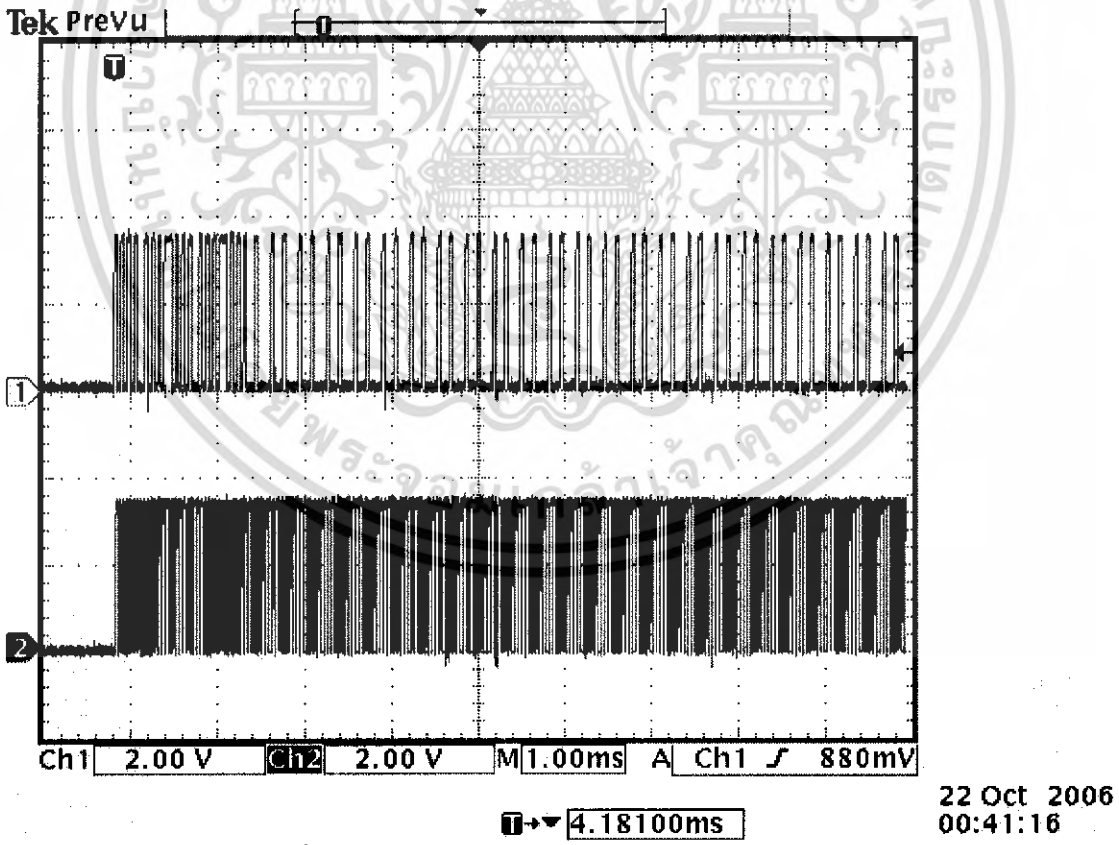
การควบคุม FSK module TRW-2.4 GHz โดยทำการวัดสัญญาณ ณ ตำแหน่งที่ 1 และ 2 ดังรูปที่ 4.5 โดยในที่นี้ ได้ทำการส่งข้อมูลเป็น 42H หรือ 0100 0010B เป็นจำนวน 25 ไบต์



รูปที่ 4.5 บล็อกไดอะแกรมการสื่อสารระหว่าง FSK module

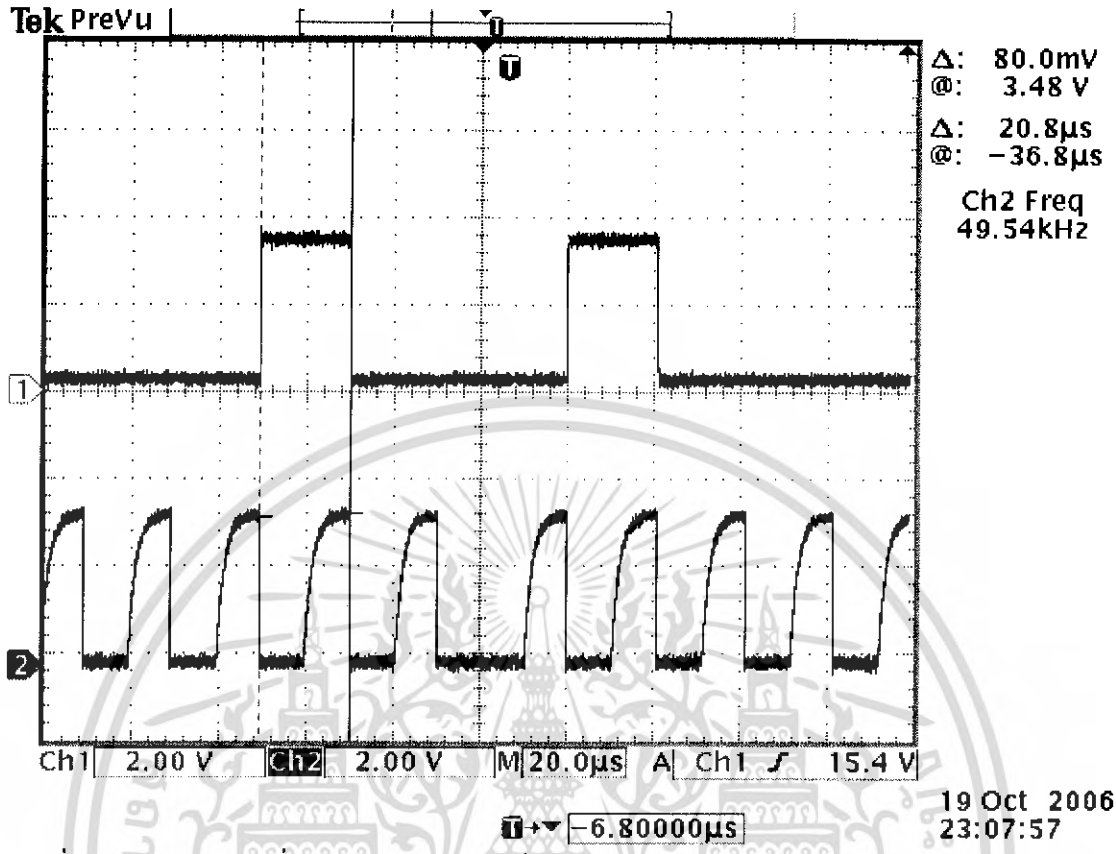
ผลการทดลอง

1. วัดสัญญาณ ณ ตำแหน่งที่ 1 จากรูปที่ 4.5 ที่ทำการส่งจากทางไมโครคอนโทรลเลอร์ด้านส่ง (Tx) ที่เชื่อมต่อกับคีย์บอร์ดไปยัง FSK module TRW-2.4 GHz ด้านส่งเพื่อดูค่าสัญญาณที่เราส่งออกไป ดังรูปที่ 4.6 โดยให้ CH1 แสดงสัญญาณข้อมูล ส่วน CH2 แสดงสัญญาณนาฬิกา



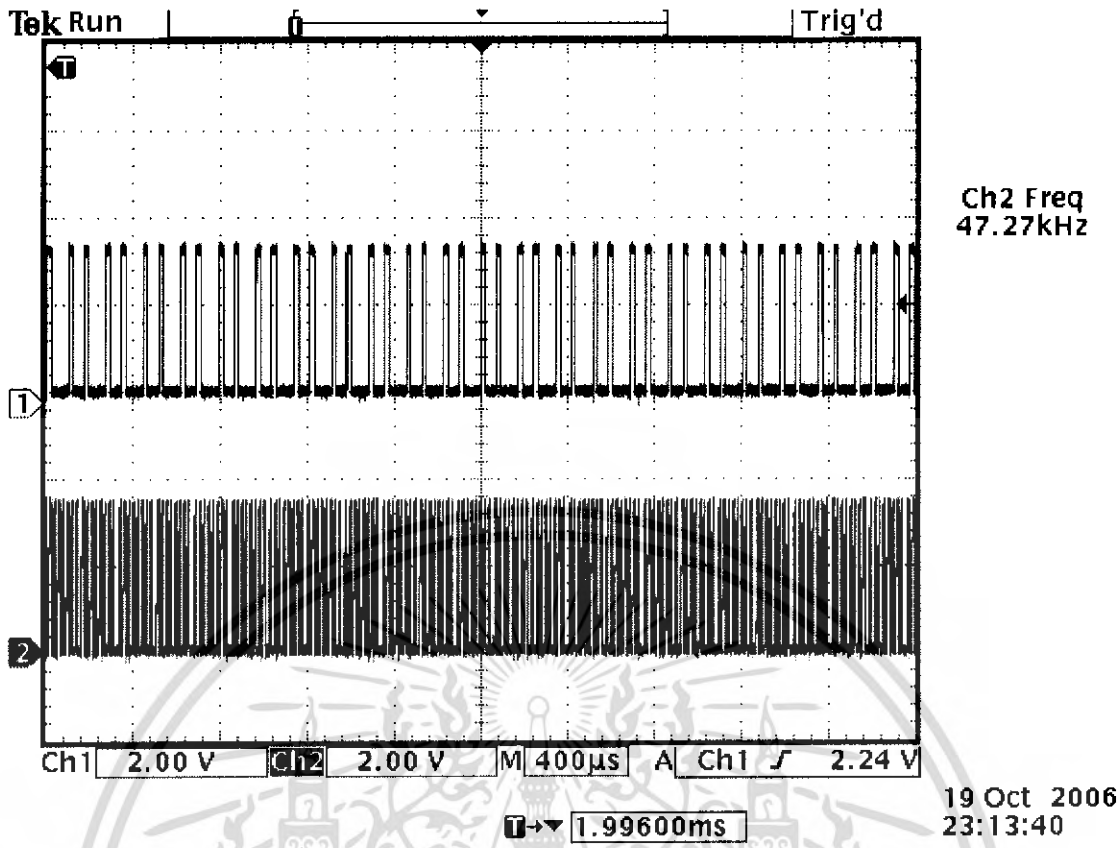
รูปที่ 4.6 แสดงสัญญาณที่ไมโครคอนโทรลเลอร์ด้านส่ง ส่งให้ FSK module TRW-2.4 GHz ด้านส่ง

จากรูปที่ 4.6 จะสังเกตได้ว่า สัญญาณข้อมูลใน CH1 ช่วงแรกจะเป็นแอดเดรสที่ส่งจากนั้นจึงเป็นสัญญาณข้อมูลที่เราต้องการจะส่งจริงๆ

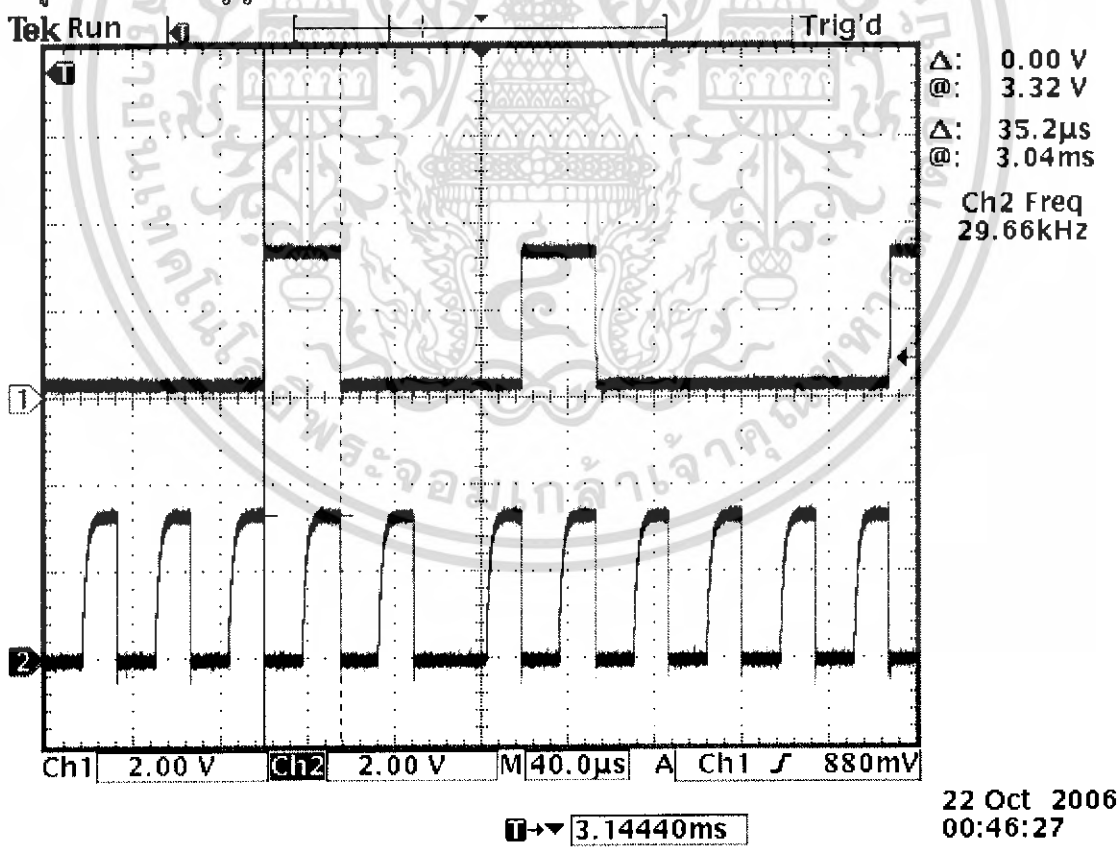


รูปที่ 4.7 แสดงสัญญาณที่ไมโครคอนโทรลเลอร์ด้านส่ง ส่งให้ FSK module TRW-2.4 GHz ด้านส่งโดยแสดงให้เห็นถึงข้อมูลที่ต้องการจะส่งออกไปคือ 42H

2. วัดสัญญาณ ณ ตำแหน่งที่ 2 จากรูปที่ 4.5 โดย FSK module TRW-2.4 GHz ที่ด้านรับจะส่งค่าสัญญาณข้อมูลให้กับไมโครคอนโทรลเลอร์ที่ด้านรับ โดยผลการทดลองจะได้ดังรูป 4.8 ให้ CH1 แสดงสัญญาณข้อมูล ส่วน CH2 แสดงสัญญาณนาฬิกา

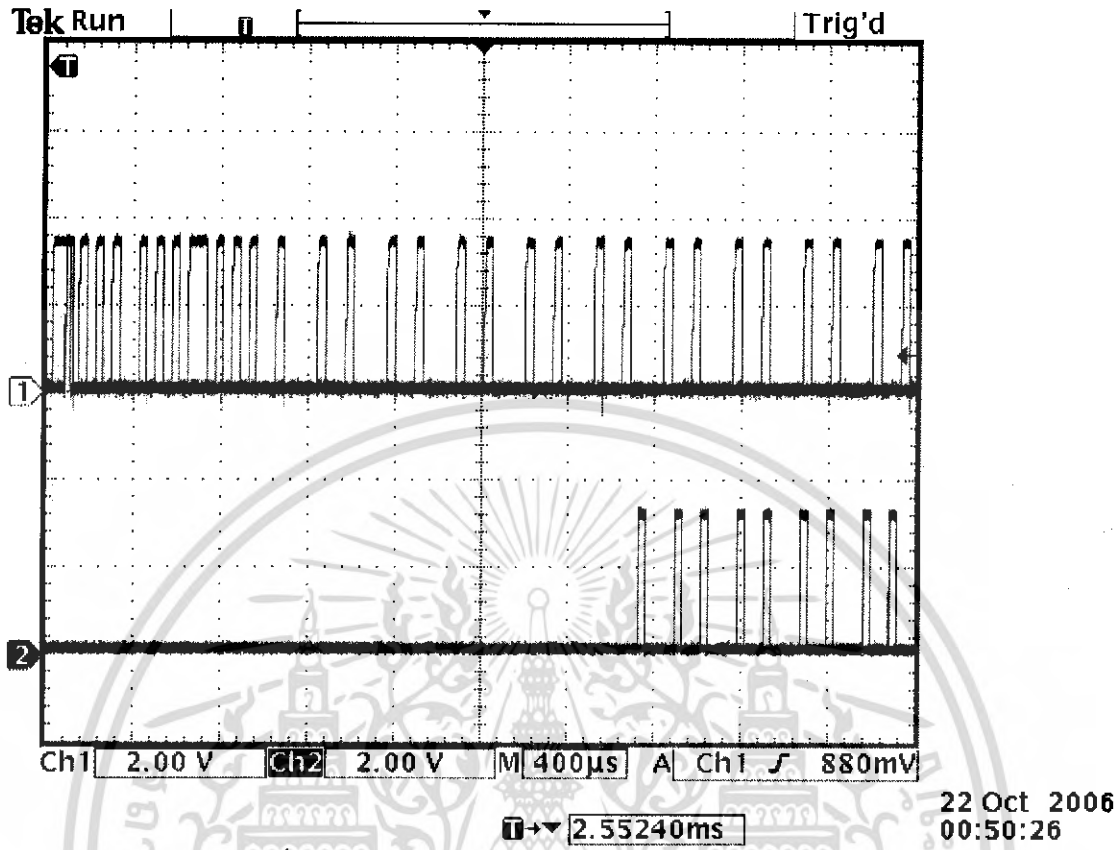


รูปที่ 4.8 แสดงสัญญาณที่ FSK module TRW-2.4 GHz รับได้และส่งต่อให้กับไมโครคอนโทรลเลอร์



รูปที่ 4.9 แสดงสัญญาณที่ FSK module TRW-2.4 GHz รับได้และส่งต่อให้กับไมโครคอนโทรลเลอร์ โดยแสดงให้เห็นถึงข้อมูลที่รับได้คือ 42H

3. วัดสัญญาณข้อมูล ณ ตำแหน่งที่ 1 เทียบกับ ตำแหน่งที่ 2 จากรูปที่ 4.5 โดย CH1 เป็นข้อมูล ณ ตำแหน่งที่ 1 CH2 เป็นข้อมูล ณ ตำแหน่งที่ 2

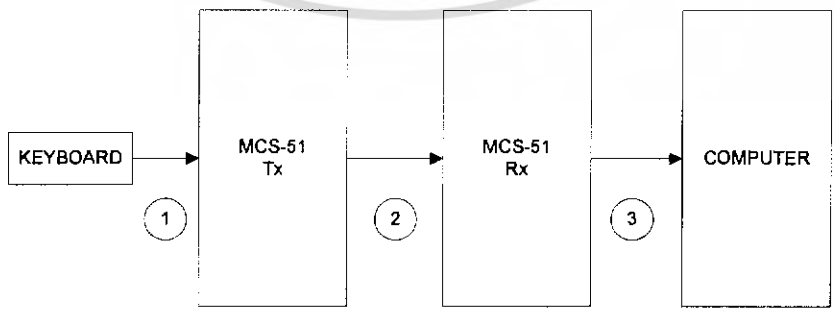


รูปที่ 4.10 สัญญาณเปรียบเทียบข้อมูลของตำแหน่งที่ 1 และ 2

จากรูปที่ 4.10 จะเห็นได้ว่าทางด้านส่งจะต้องทำการส่งแอดเดรสรวมมากับข้อมูล แล้วทางด้านรับจะทำการประมวลผลแล้วส่งเฉพาะข้อมูลออกมา

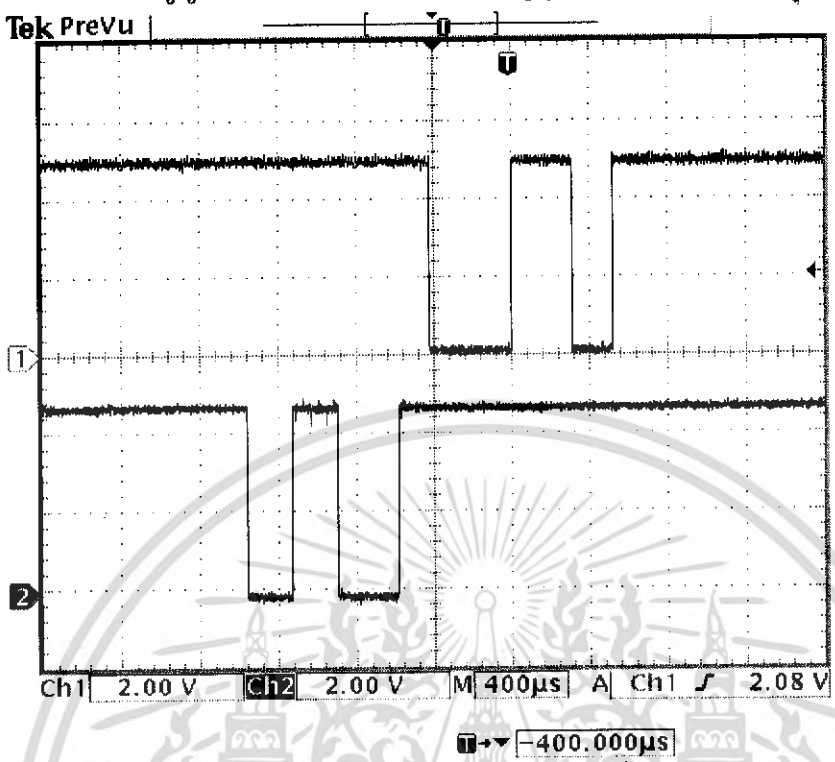
การทดลองที่ 3

ทำการส่งข้อมูลจากคีย์บอร์ดโดยผ่านพอร์ตสื่อสารอนุกรมของไมโครคอนโทรลเลอร์ โดยต่อวงจรตามบล็อกไดอะแกรมดังรูปที่ 4.11 แล้วทำการวัดสัญญาณตามจุดต่างๆ โดยในที่นี้เป็นสัญญาณเมื่อทำการกดปุ่ม "A" บนคีย์บอร์ด



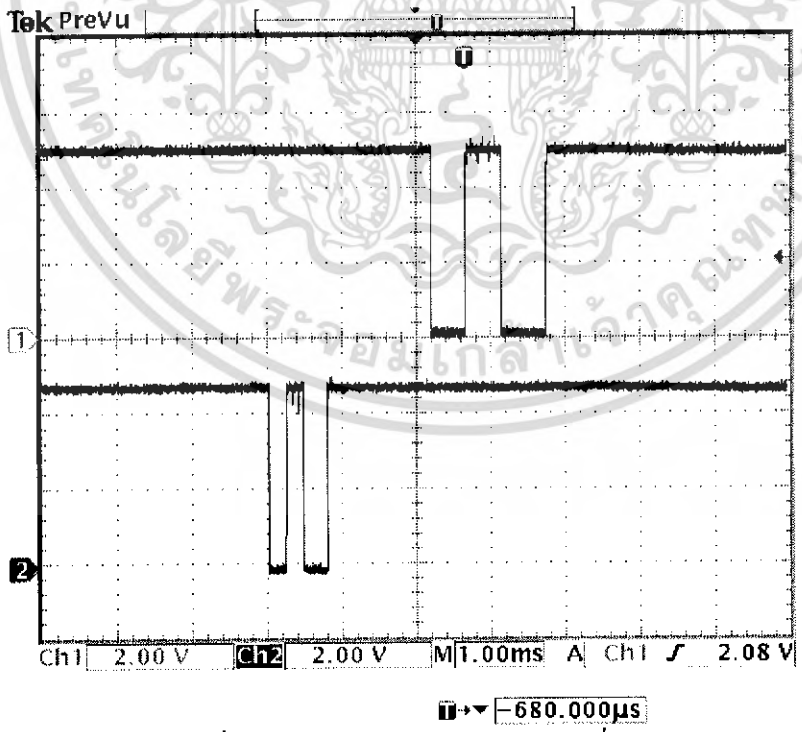
รูปที่ 4.11 บล็อกไดอะแกรมการส่งข้อมูลของคีย์บอร์ดผ่านทางพอร์ตสื่อสารอนุกรมของไมโครคอนโทรลเลอร์

1. วัดสัญญาณข้อมูลที่ส่งจากคีย์บอร์ด เทียบกับ สัญญาณที่ไมโครคอนโทรลเลอร์ด้านส่งส่งออกไป โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่1 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 2



รูปที่ 4.12 สัญญาณข้อมูลที่ส่งจากคีย์บอร์ด เทียบกับ สัญญาณที่ไมโครคอนโทรลเลอร์ด้านส่งส่งออกไป

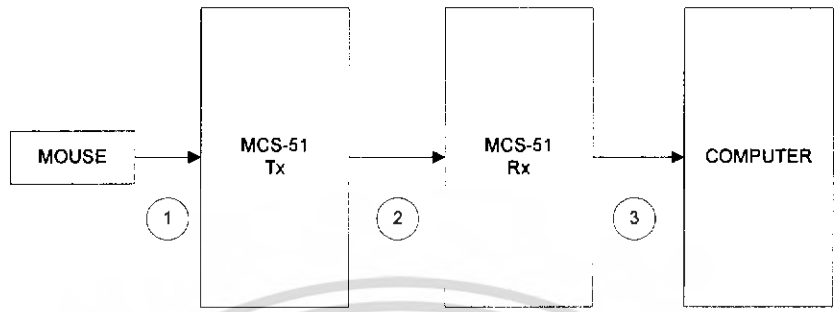
2. วัดสัญญาณข้อมูลที่ส่งจากคีย์บอร์ด เทียบกับ สัญญาณที่ไมโครคอนโทรลเลอร์ด้านรับส่งออกไปให้กับคอมพิวเตอร์ โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่1 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 3



รูปที่ 4.13 สัญญาณข้อมูลที่ส่งจากคีย์บอร์ด เทียบกับ สัญญาณที่ไมโครคอนโทรลเลอร์ด้านรับส่งออกไปให้กับคอมพิวเตอร์

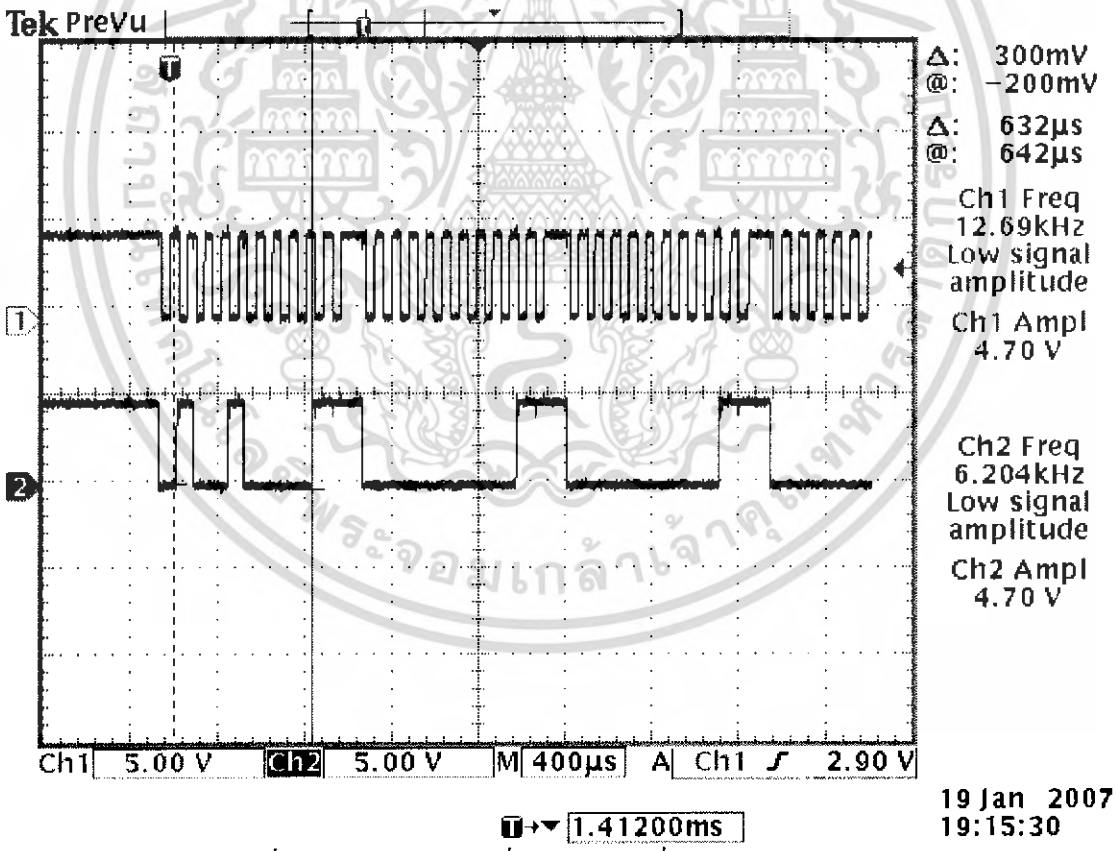
การทดลองที่ 4

ทำการส่งข้อมูลจากเมาส์ที่ทำการเซตค่าเริ่มต้นแล้วส่งผ่านพอร์ตสื่อสารอนุกรมของไมโครคอนโทรลเลอร์ โดยต่อวงจรตามบล็อกไดอะแกรมดังรูปที่ 4.14 แล้วทำการวัดสัญญาณตามจุดต่างๆ โดยในข้อที่ 1 - 10 กำหนดให้ CH1 เป็นสัญญาณนาฬิกา CH2 เป็นสัญญาณข้อมูล



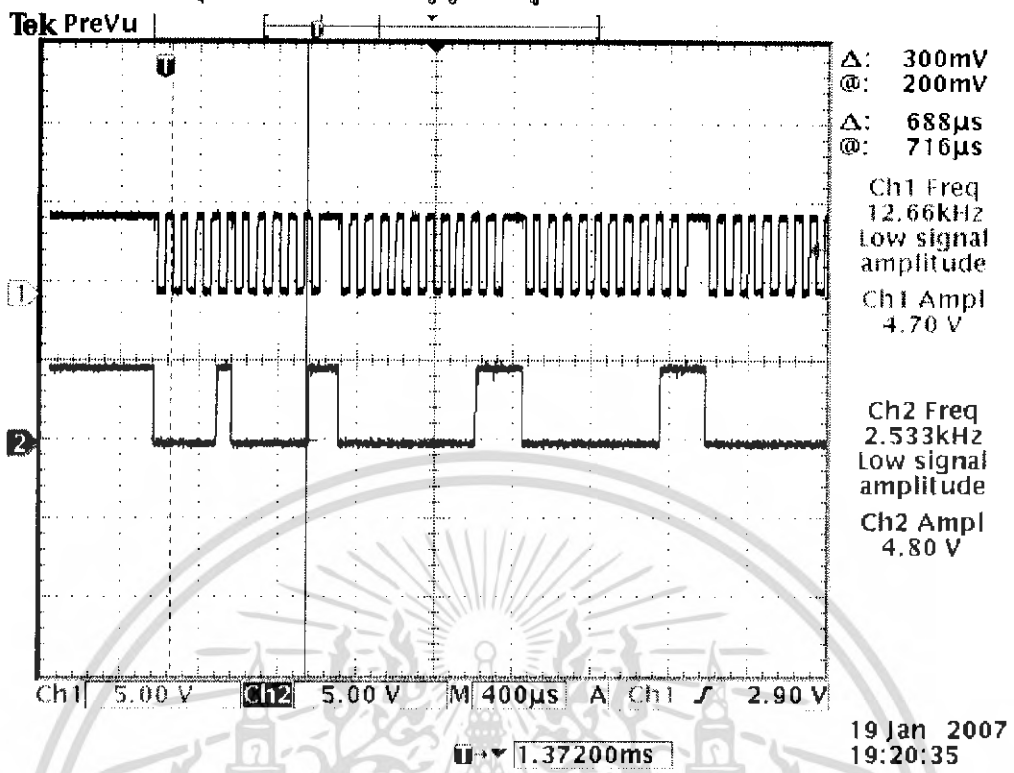
รูปที่ 4.14 บล็อกไดอะแกรมการส่งข้อมูลของเมาส์ผ่านทางพอร์ตสื่อสารอนุกรมของไมโครคอนโทรลเลอร์

1. เมื่อกดปุ่มซ้ายของเมาส์วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ในตำแหน่งที่ 1



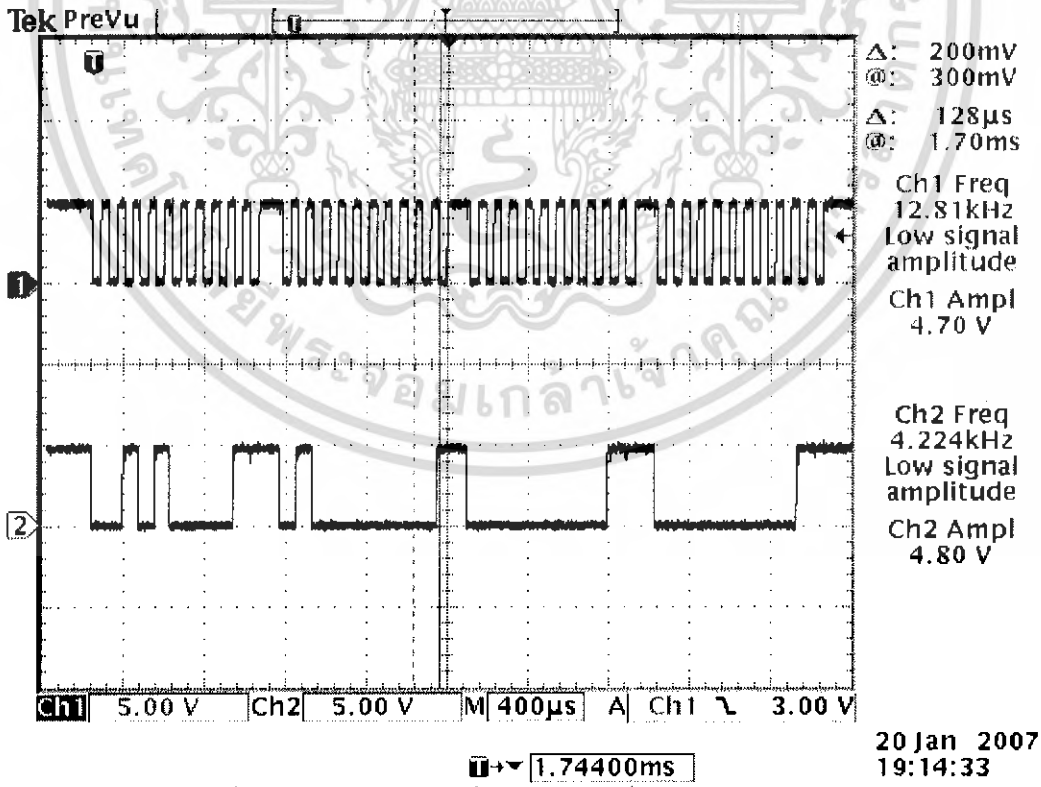
รูปที่ 4.15 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อกดปุ่มซ้ายของเมาส์

2. เมื่อปล่อยปุ่มซ้ายของเมาส์ วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ในตำแหน่งที่ 1



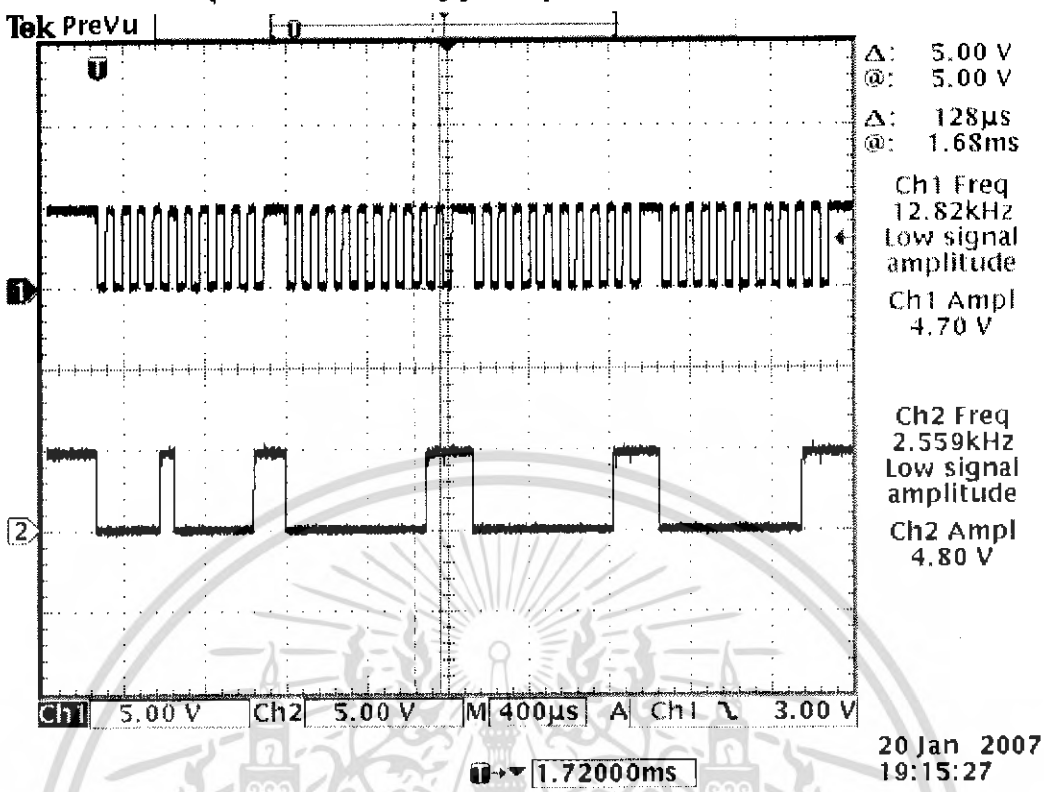
รูปที่ 4.16 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อปล่อยปุ่มซ้ายของเมาส์

3. เมื่อกดปุ่มขวาของเมาส์ วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ในตำแหน่งที่ 1



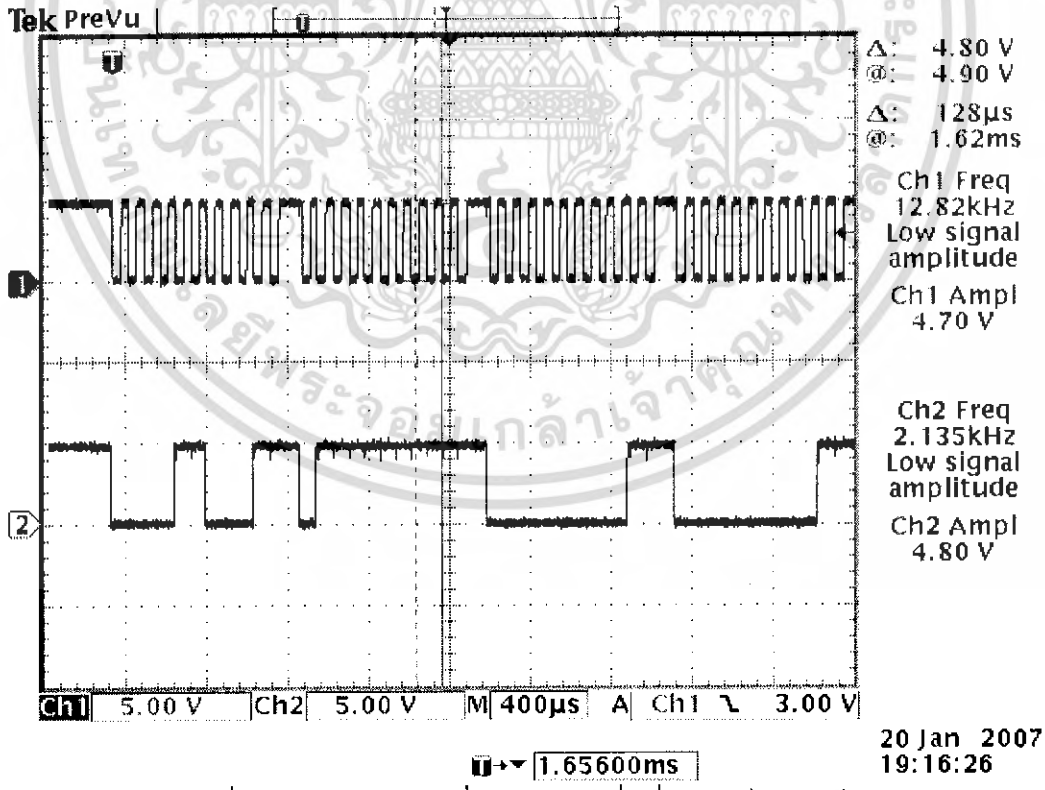
รูปที่ 4. 17 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อกดปุ่มขวาของเมาส์

4. เมื่อปล่อยปุ่มขวาของเมาส์ วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ในตำแหน่งที่ 1



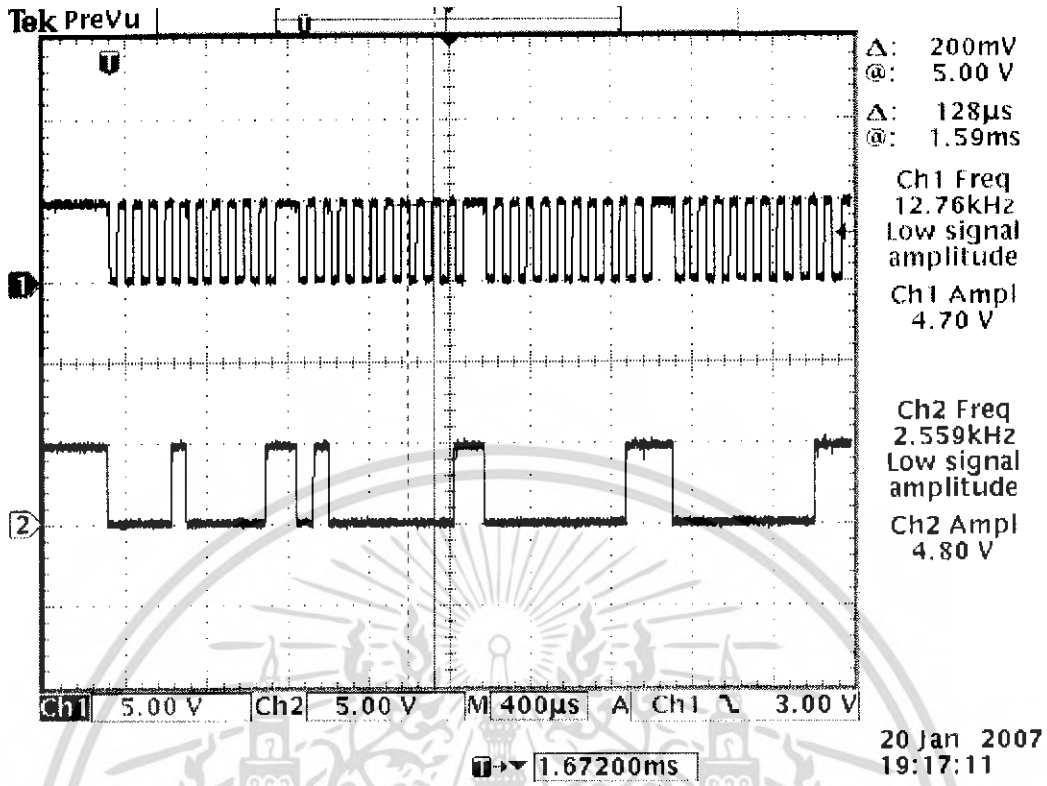
รูปที่ 4.18 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อปล่อยปุ่มขวาของเมาส์

5. เมื่อเลื่อนเมาส์ไปทางซ้าย วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ในตำแหน่งที่ 1



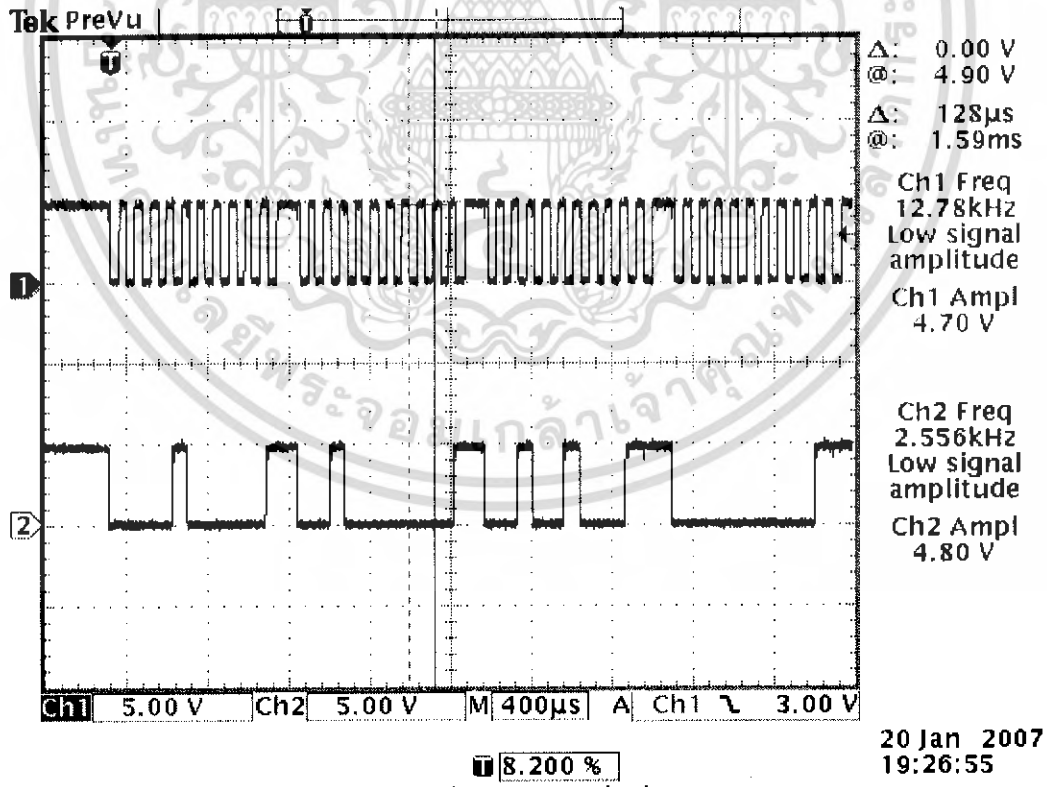
รูปที่ 4.19 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อเลื่อนเมาส์ไปทางซ้าย

6. เมื่อเลื่อนเมาส์ไปทางขวา วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ในตำแหน่งที่ 1



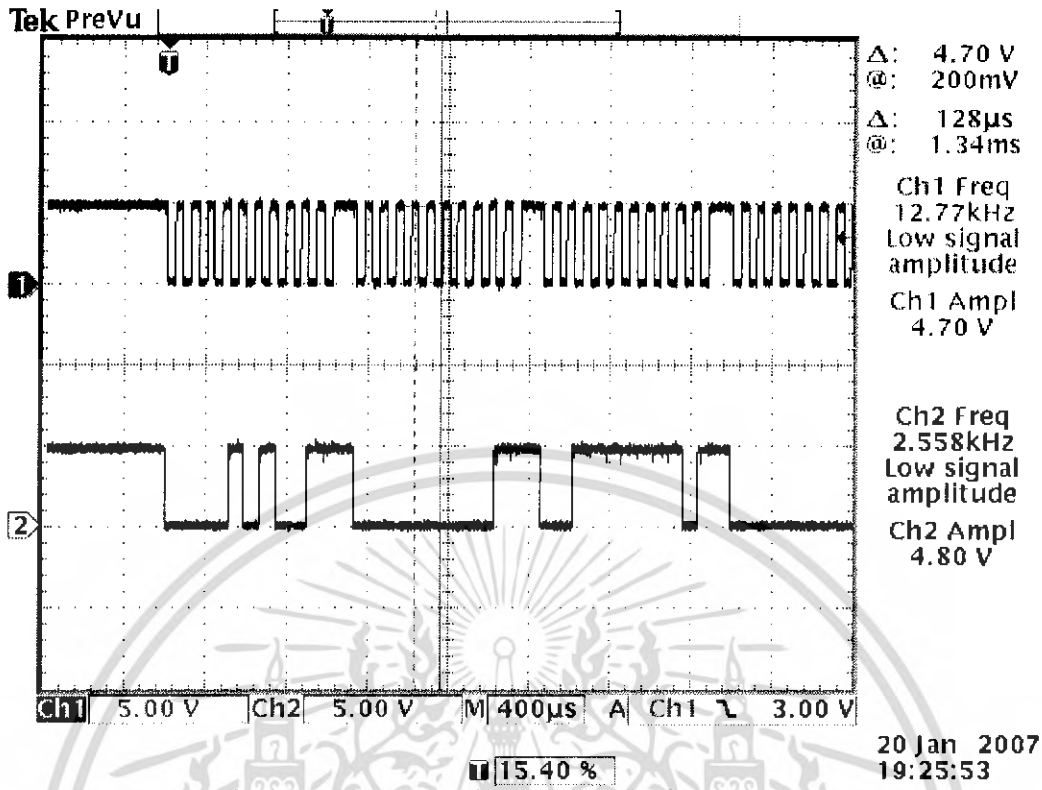
รูปที่ 4.20 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อเลื่อนเมาส์ไปทางขวา

7. เมื่อเลื่อนเมาส์ไปข้างหน้า วัดสัญญาณข้อมูลที่ส่งจากเมาส์ในตำแหน่งที่ 1



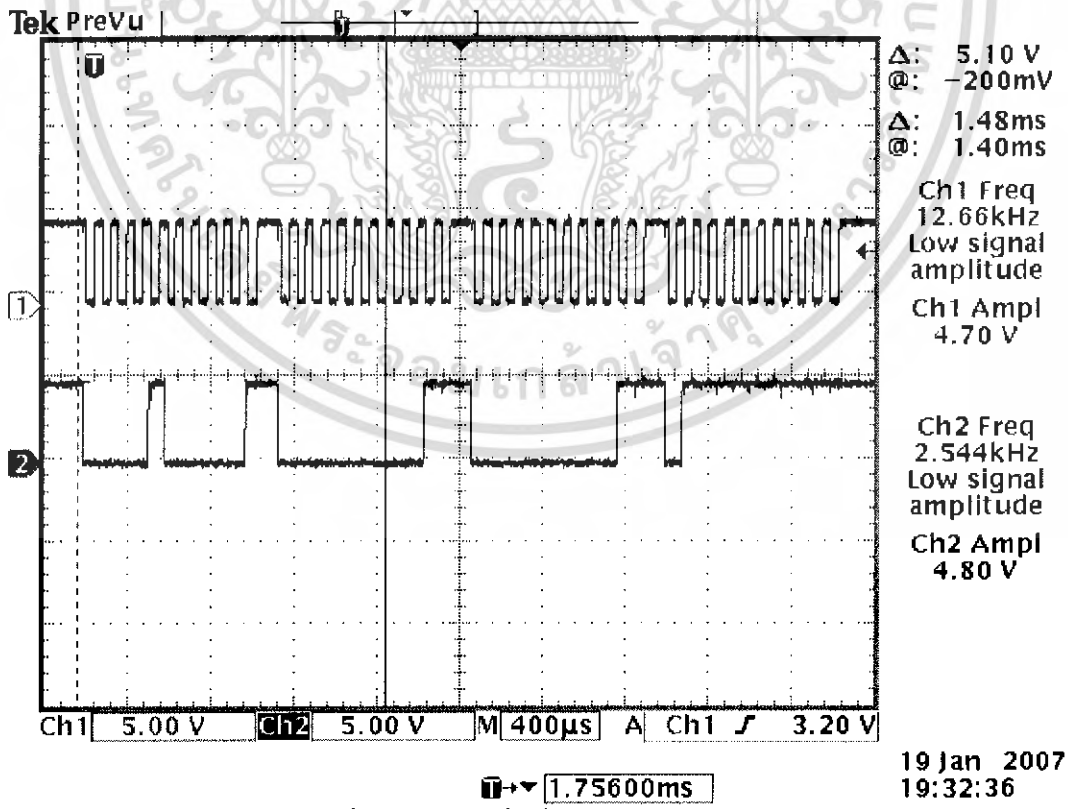
รูปที่ 4.21 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อเลื่อนเมาส์ในไปข้างหน้า

8. เมื่อเลื่อนเมาส์เข้าหาตัว วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ในตำแหน่งที่ 1



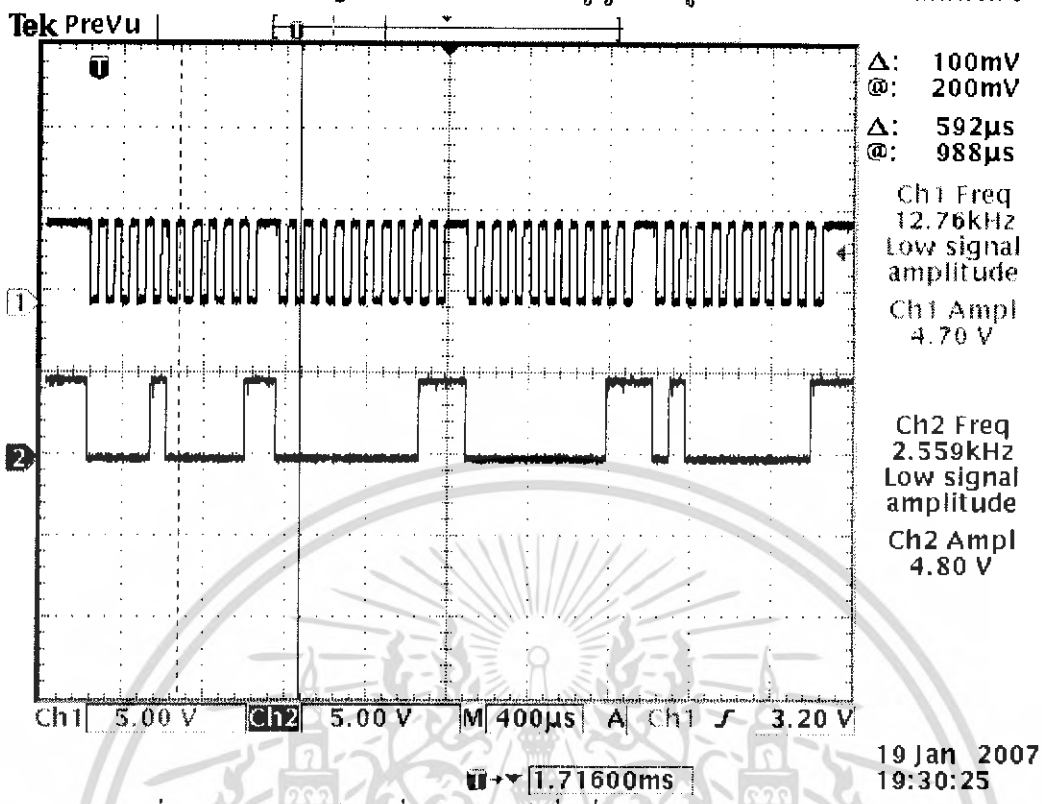
รูปที่ 4. 22 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อเลื่อนเมาส์เข้าหาตัว

9. เมื่อเลื่อน scrolling wheel ของเมาส์ในไปข้างหน้า วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ใน ตำแหน่งที่ 1



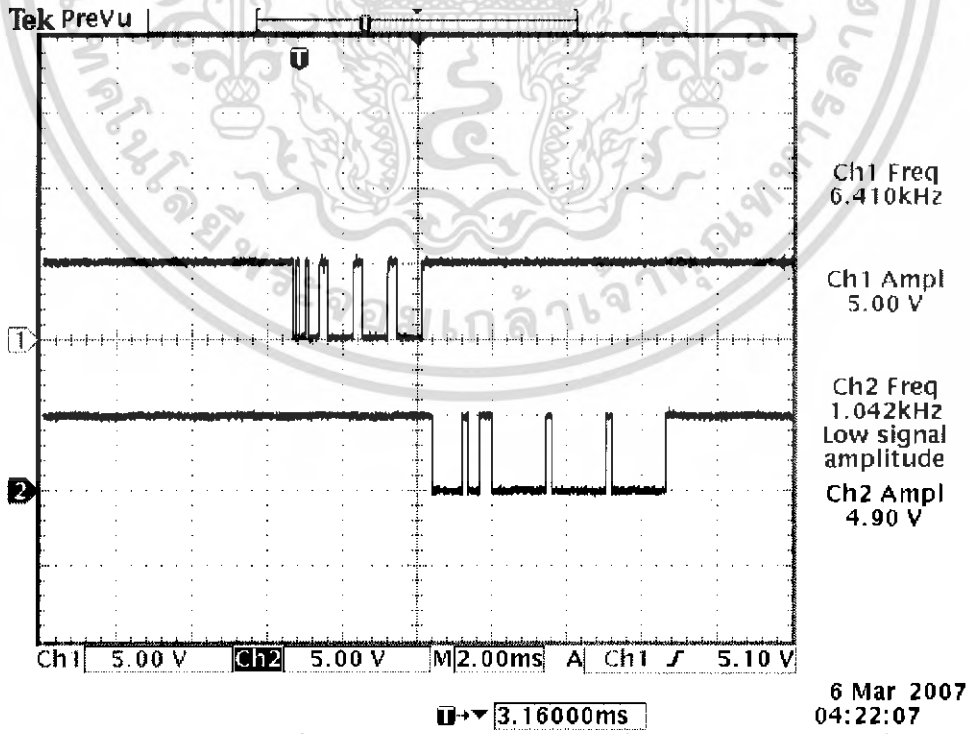
รูปที่ 4. 23 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อเลื่อน scrolling wheel ของเมาส์ในไปข้างหน้า

10. เมื่อเลื่อน scrolling wheel เข้าหาตัว วัดสัญญาณข้อมูลที่ส่งจากเมาส์ ในตำแหน่งที่ 1



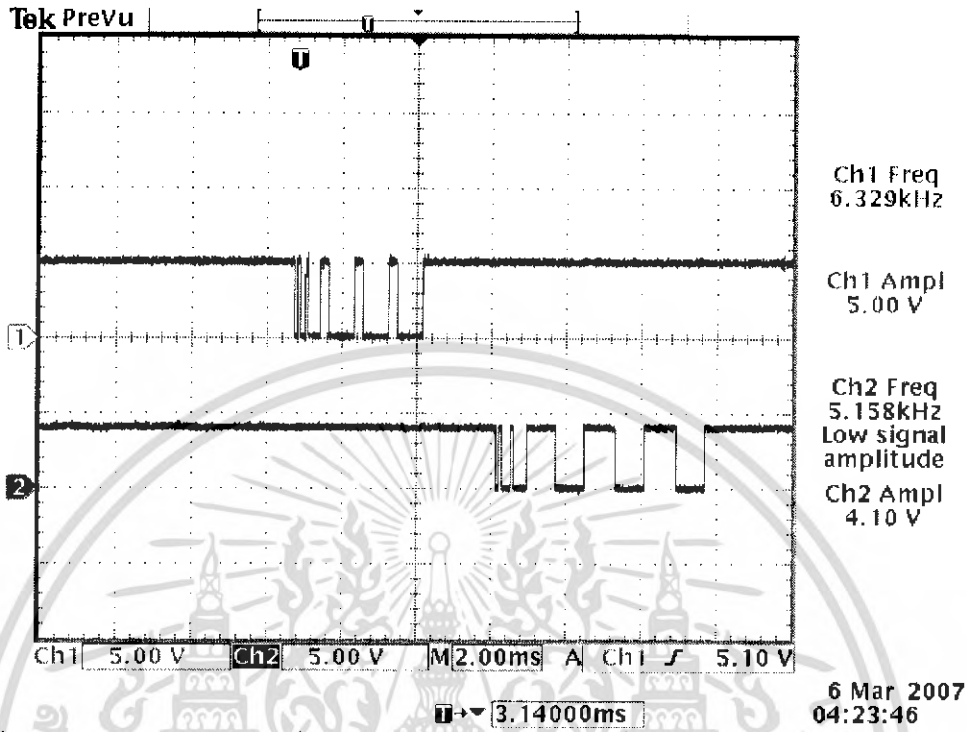
รูปที่ 4.24 สัญญาณข้อมูลที่ส่งจากเมาส์ เมื่อเลื่อน scrolling wheel ของเมาส์เข้าหาตัว

11.เมื่อทำการกดปุ่มซ้ายแล้ววัดเทียบสัญญาณระหว่างสัญญาณที่ส่งออกจากเมาส์ในตำแหน่งที่ 1 กับสัญญาณที่ส่งออกทางพอร์ตอนุกรมในตำแหน่งที่ 2 โดย CH1 แทนตำแหน่งที่ 1 และ CH2 แทนตำแหน่งที่ 2



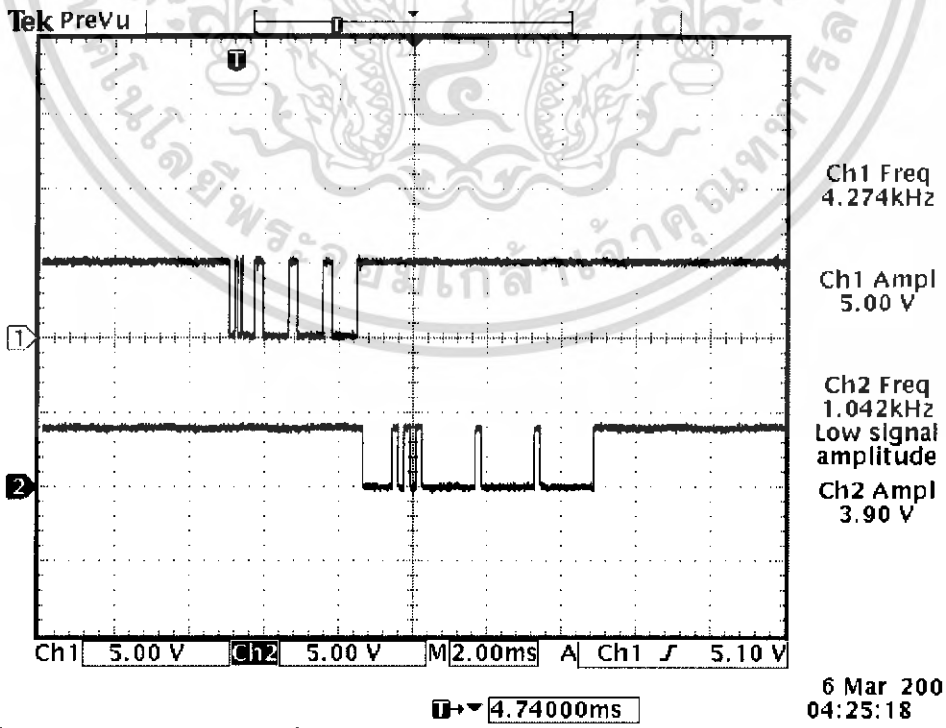
รูปที่ 4.25 แสดงสัญญาณข้อมูลที่ออกมาจากเมาส์และสัญญาณที่ออกจากพอร์ตอนุกรมเมื่อกดปุ่มซ้าย

12.เมื่อทำการกดปุ่มซ้ายแล้ววัดเทียบสัญญาณระหว่างสัญญาณที่ส่งออกจากเมาส์ใน ตำแหน่งที่ 1 กับสัญญาณที่ส่งออกจากไมโครคอนโทรลเลอร์ทางด้านรับเข้าสู่คอมพิวเตอร์ในตำแหน่งที่ 3 โดย CH1 แทนตำแหน่งที่ 1 และ CH2 แทนตำแหน่งที่ 3



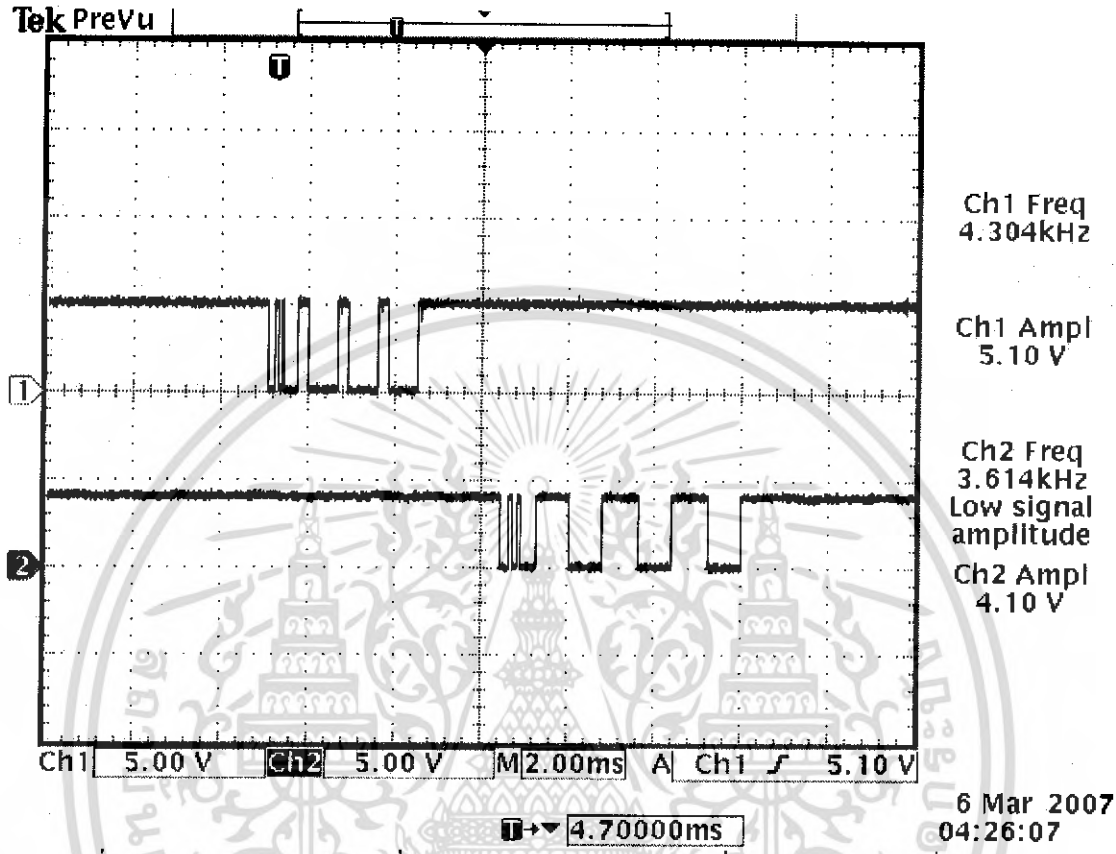
รูปที่ 4.26 แสดงสัญญาณข้อมูลที่ออกมาจากเมาส์และสัญญาณที่เข้าสู่คอมพิวเตอร์เมื่อกดปุ่มซ้าย

13.เมื่อทำการกดปุ่มขวาแล้ววัดเทียบสัญญาณระหว่างสัญญาณที่ส่งออกจากเมาส์ในตำแหน่งที่ 1 กับสัญญาณที่ส่งออกทางพอร์ตอนุกรมในตำแหน่งที่ 2 โดย CH1 แทนตำแหน่งที่ 1 และ CH2 แทนตำแหน่งที่ 2



รูปที่ 4.27 แสดงสัญญาณข้อมูลที่ออกมาจากเมาส์และสัญญาณที่ออกจากพอร์ตอนุกรมเมื่อกดปุ่มขวา

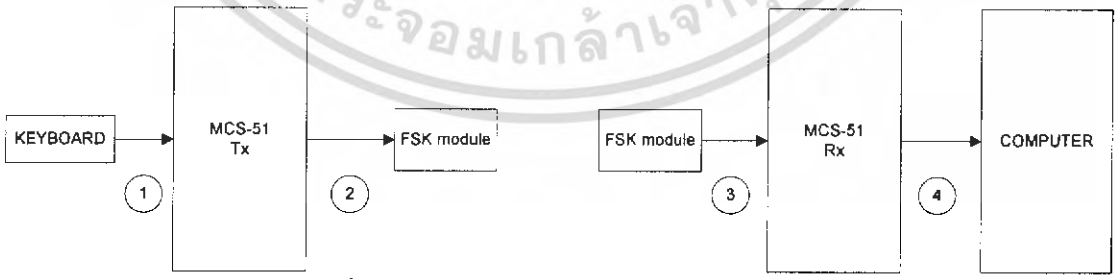
14.เมื่อทำการกดปุ่มขวาแล้ววัดเทียบสัญญาณระหว่างสัญญาณที่ส่งออกจากเมาส์ในตำแหน่งที่ 1 กับสัญญาณที่ส่งออกจากไมโครคอนโทรลเลอร์ทางด้านรับเข้าสู่คอมพิวเตอร์ในตำแหน่งที่ 3 โดย CH1 แทนตำแหน่งที่ 1 และ CH2 แทนตำแหน่งที่ 3



รูปที่ 4.28 แสดงสัญญาณข้อมูลที่ออกมาจากเมาส์และสัญญาณที่เข้าสู่คอมพิวเตอร์เมื่อกดปุ่มขวา

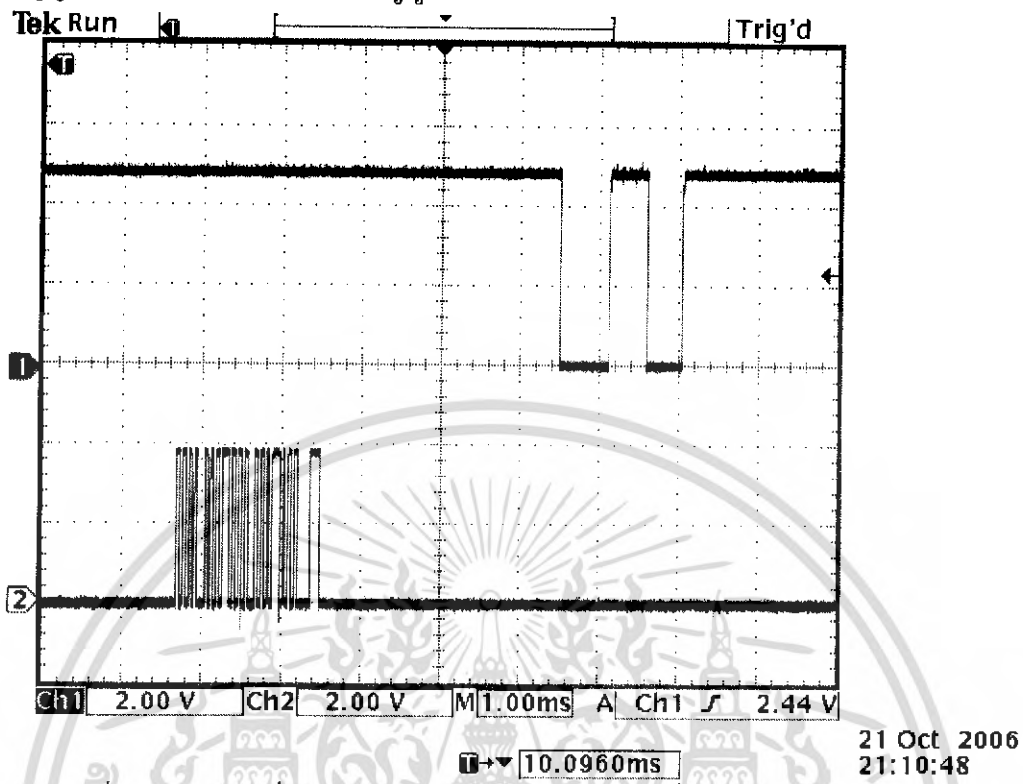
การทดลองที่ 5

ทำการส่งข้อมูลจากคีย์บอร์ดไร้สายโดยต่อวงจรตามบล็อกไดอะแกรมดังรูปที่ 4.29 แล้วทำการวัดสัญญาณตามจุดต่างๆ โดยในที่นี้เป็นสัญญาณเมื่อทำการกดปุ่ม "A" บนคีย์บอร์ด



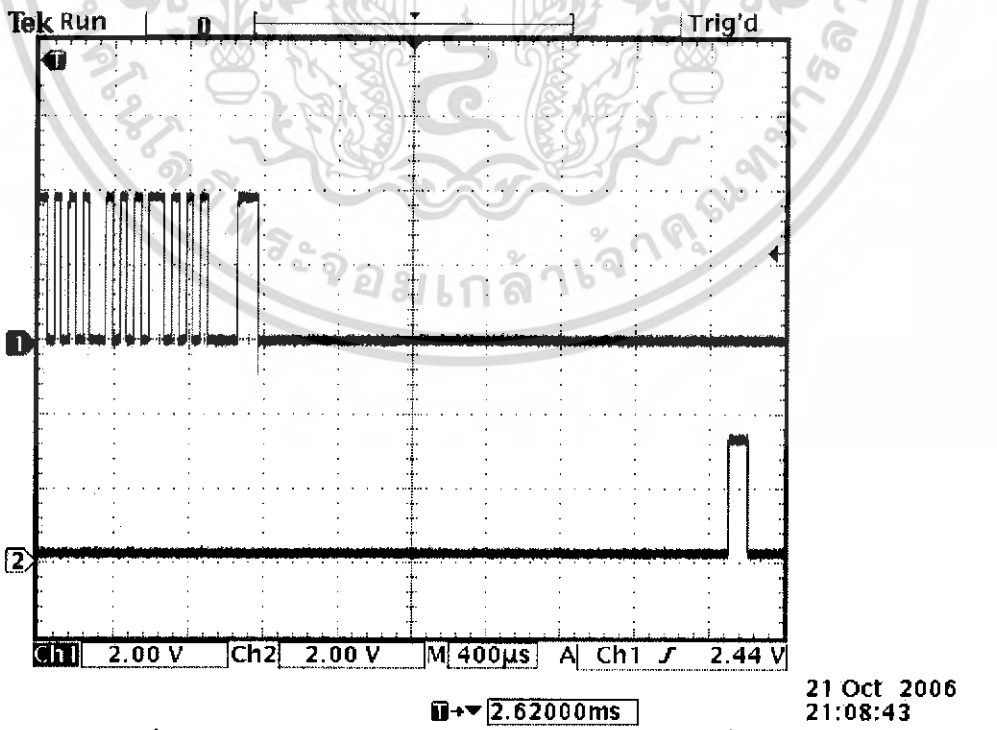
รูปที่ 4.29 บล็อกไดอะแกรมของคีย์บอร์ดไร้สาย

1. วัดสัญญาณข้อมูลที่ส่งจากคีย์บอร์ด เทียบกับ สัญญาณที่ส่งออกไปยัง FSK module โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่1 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 2



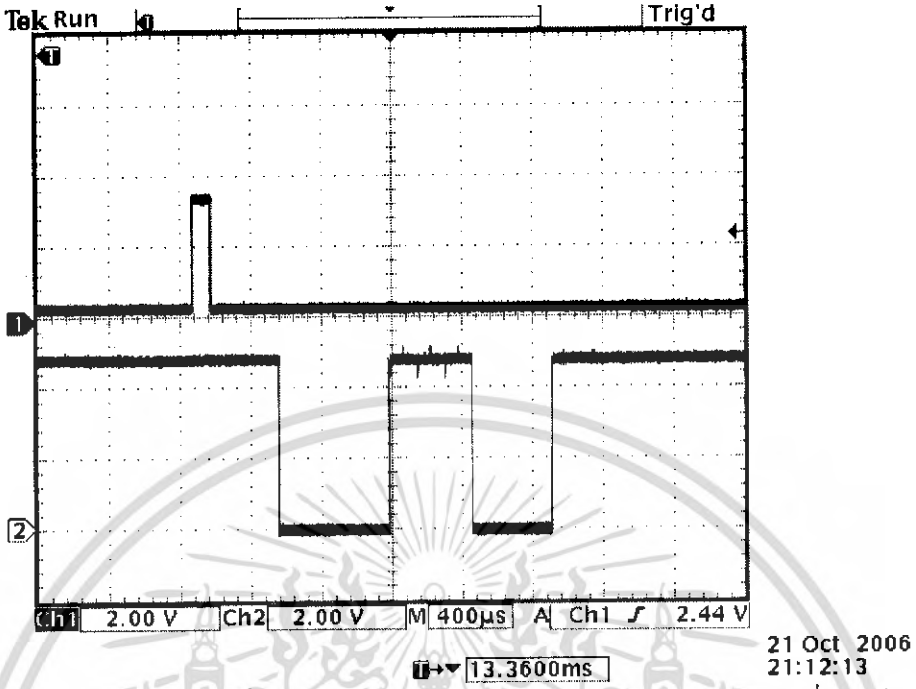
รูปที่ 4.30 สัญญาณที่คีย์บอร์ดส่งออกมาเทียบกับสัญญาณที่ส่งออกไปยัง FSK module

2. วัดสัญญาณที่ส่งออกไปยัง FSK module ด้านส่ง เทียบกับ สัญญาณที่ FSK module ด้านรับรับได้และส่งต่อออกไปยังไมโครคอนโทรลเลอร์ด้านรับ โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่2 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 3



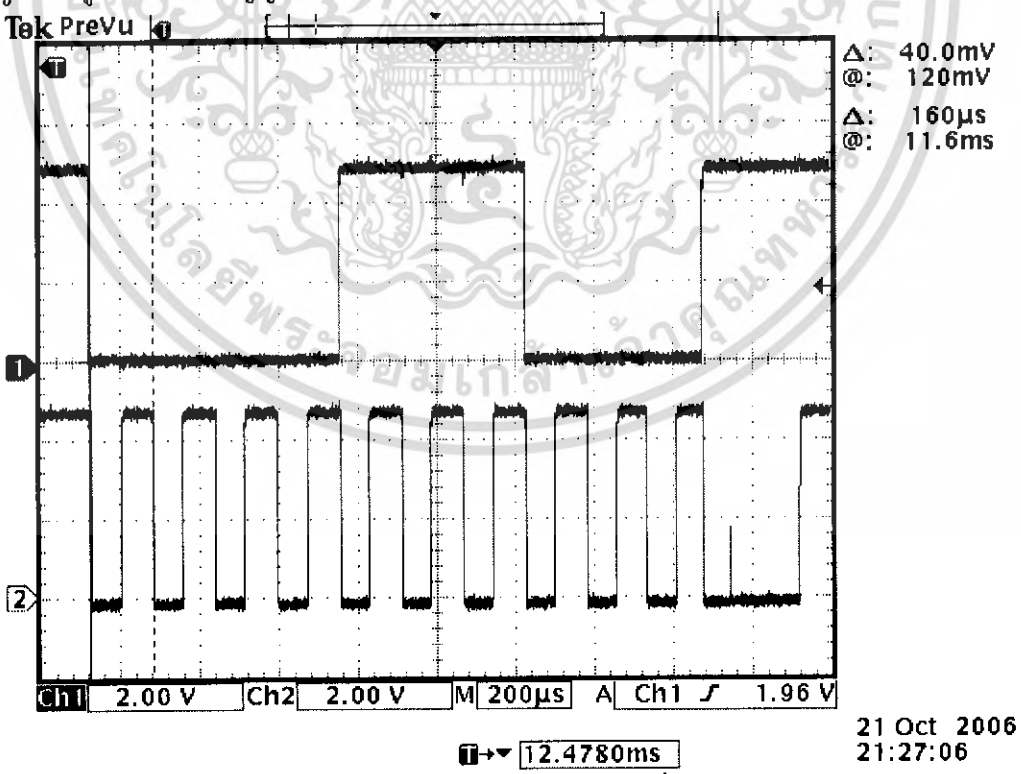
รูปที่ 4.31 สัญญาณที่ส่งออกไปยัง FSK module ด้านส่ง เทียบกับ สัญญาณที่ FSK module ด้านรับรับได้

3. วัดสัญญาณที่ FSK module ด้านรับรับได้ เทียบกับ สัญญาณข้อมูลที่ส่งต่อไปให้กับคอมพิวเตอร์ โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่ 3 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 4



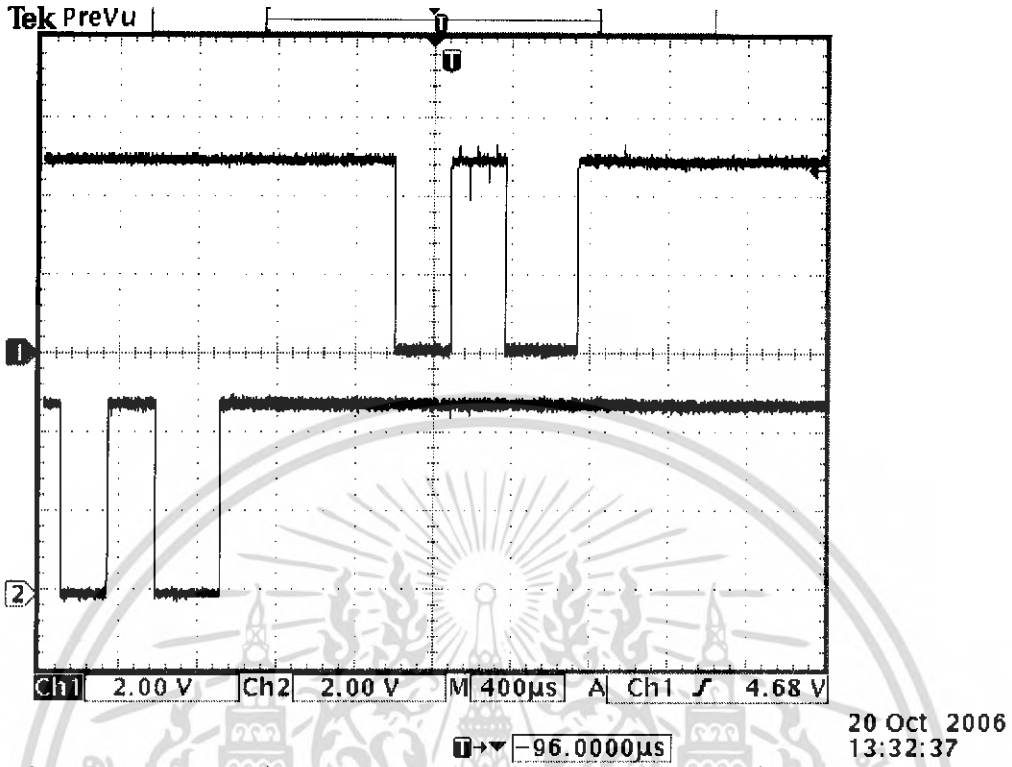
รูปที่ 4.32 สัญญาณสัญญาณที่ FSK module ด้านรับรับได้ เทียบกับ สัญญาณข้อมูลที่ส่งต่อไปให้กับคอมพิวเตอร์

4. วัดสัญญาณข้อมูลและสัญญาณนาฬิกาที่เข้าสู่คอมพิวเตอร์ ณ ตำแหน่งที่ 4 โดย CH1 เป็นสัญญาณข้อมูล CH2 เป็นสัญญาณนาฬิกา



รูปที่ 4.33 สัญญาณข้อมูลและสัญญาณนาฬิกาที่เข้าสู่คอมพิวเตอร์

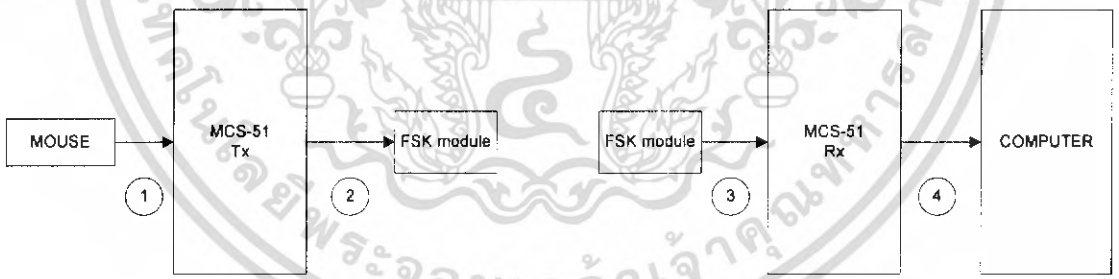
5. วัดสัญญาณข้อมูลที่ส่งออกจากคีย์บอร์ด เทียบกับ สัญญาณข้อมูลที่ส่งไปให้กับคอมพิวเตอร์ โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่ 1 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 4



รูปที่ 4.34 สัญญาณข้อมูลที่ส่งออกจากคีย์บอร์ด เทียบกับ สัญญาณข้อมูลที่ส่งไปให้กับคอมพิวเตอร์

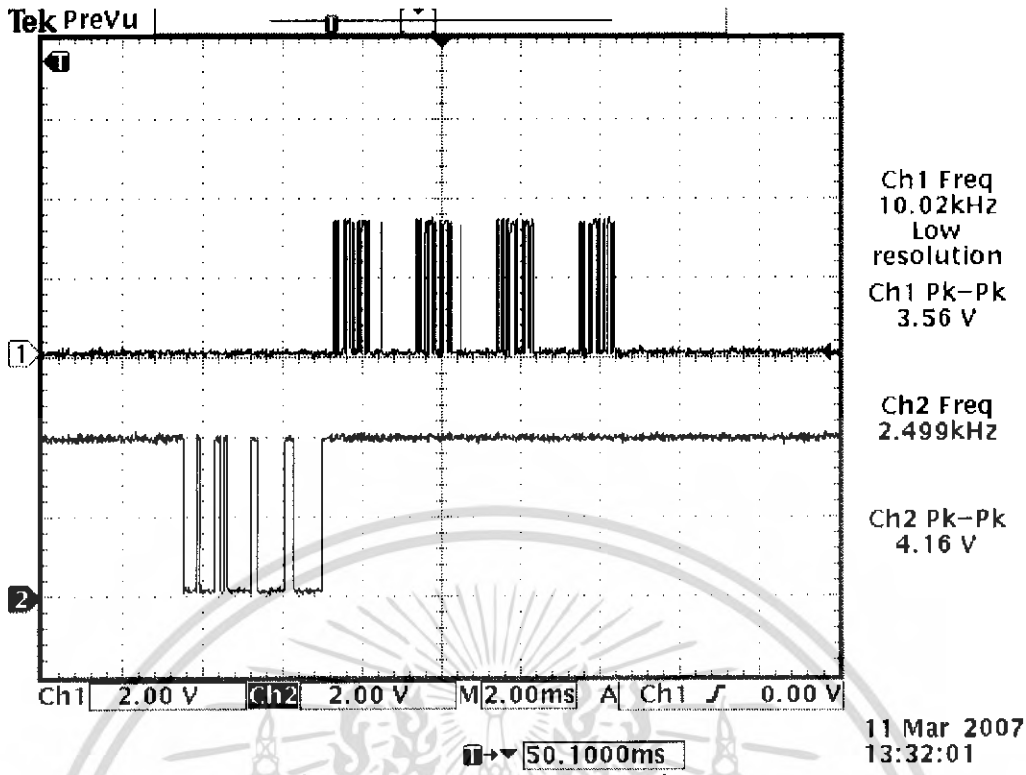
การทดลองที่ 6

ทำการส่งข้อมูลจากเมาส์ไร้สายโดยต่อวงจรตามบล็อกไดอะแกรมดังรูปที่ 4.35 แล้วทำการวัดสัญญาณตามจุดต่างๆ โดยในที่นี้เป็นสัญญาณเมื่อทำการปล่อยปุ่มขวาของเมาส์



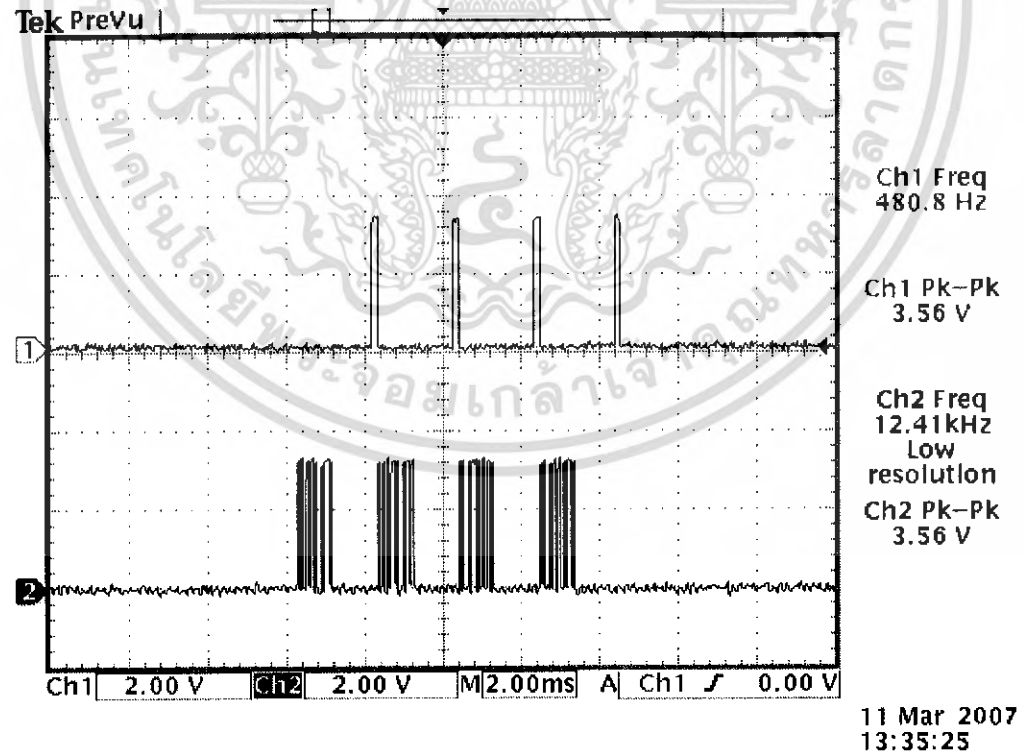
รูปที่ 4.35 บล็อกไดอะแกรมของเมาส์ไร้สาย

1. วัดสัญญาณข้อมูลที่ส่งจากเมาส์เมื่อปล่อยปุ่มขวา เทียบกับ สัญญาณที่ส่งออกไปยัง FSK module โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่ 2 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 1



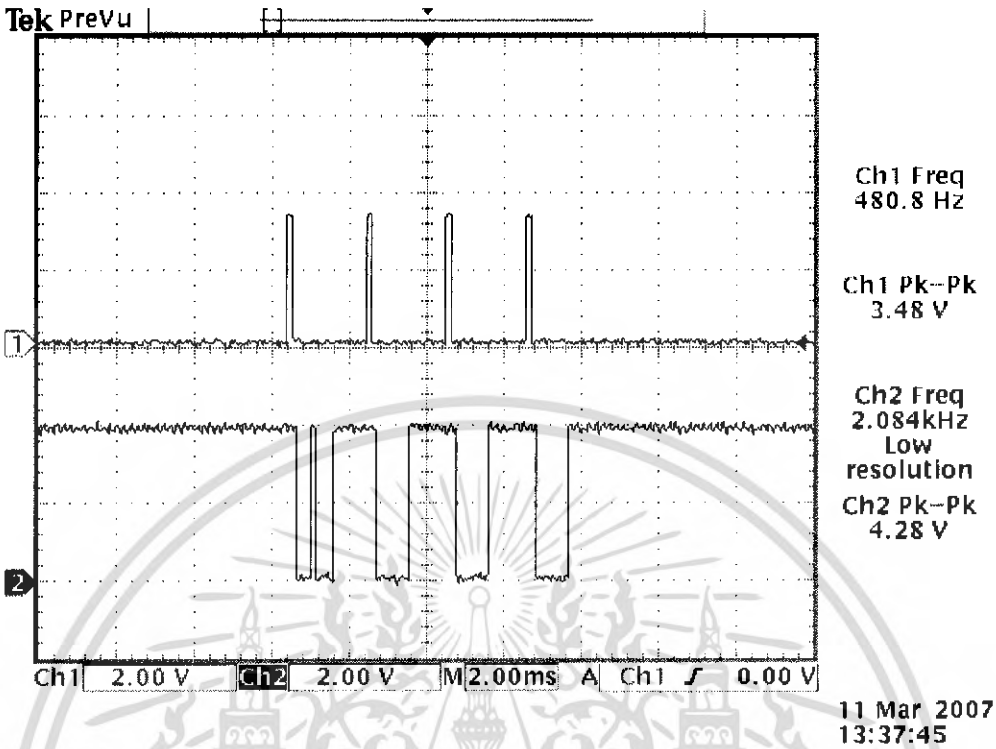
รูปที่ 4.36 สัญญาณที่เมาส์ส่งออกมาเทียบกับสัญญาณที่ส่งออกไปยัง FSK module

2. วัดสัญญาณเมาส์ที่ส่งออกไปยัง FSK module ด้านส่ง เทียบกับ สัญญาณที่ FSK module ด้านรับรับได้และส่งต่อออกไปยังไมโครคอนโทรลเลอร์ด้านรับ โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่3 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 2



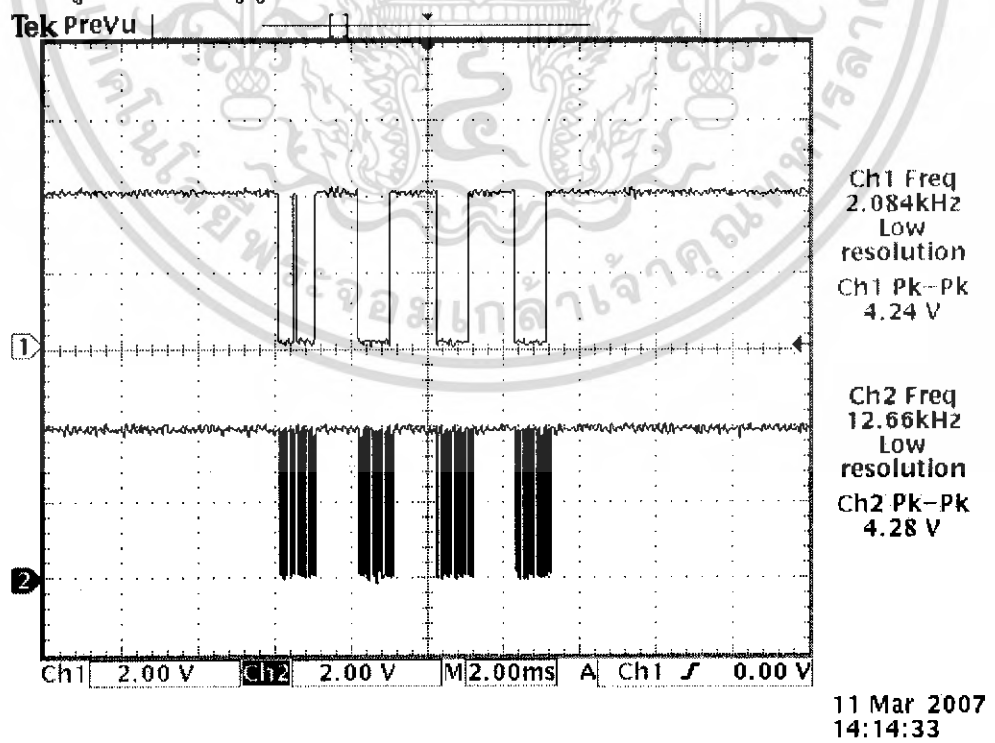
รูปที่ 4.37 สัญญาณที่เมาส์ส่งออกไปยัง FSK module ด้านส่ง เทียบกับ สัญญาณที่ FSK module ด้านรับรับได้

3. วัดสัญญาณที่ FSK module ด้านรับรับได้ เทียบกับ สัญญาณข้อมูลที่ส่งต่อไปให้กับคอมพิวเตอร์ โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่ 3 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 4



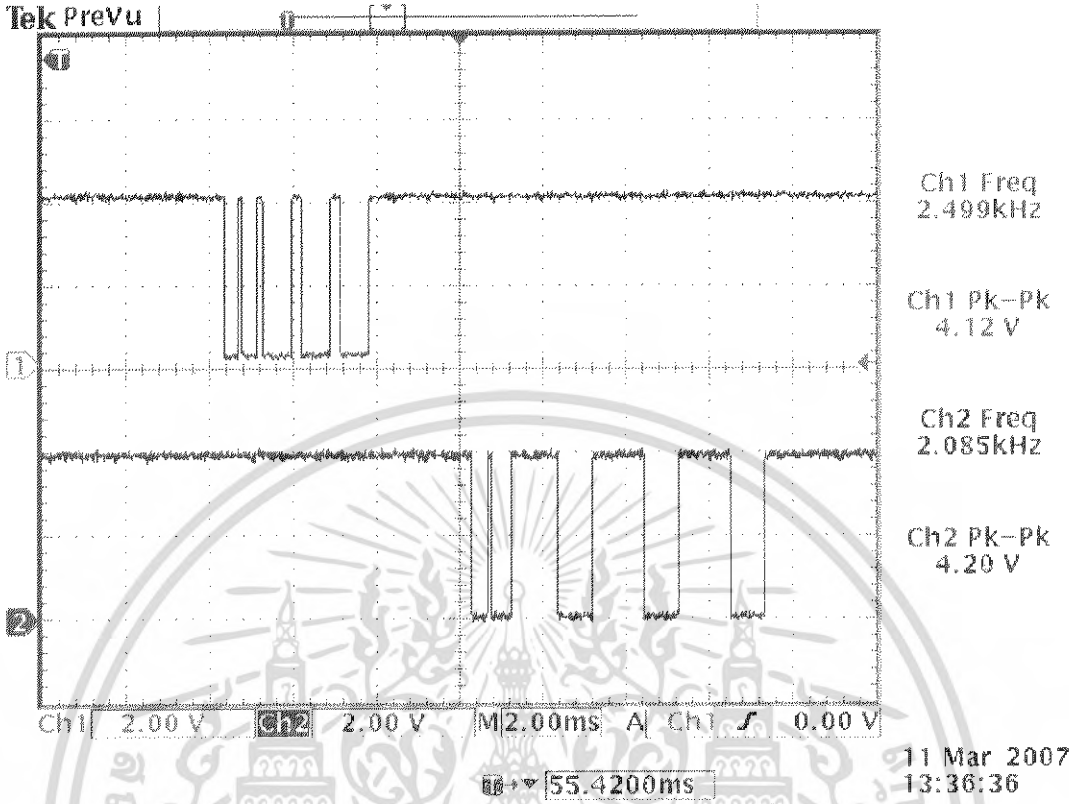
รูปที่ 4.38 สัญญาณสัญญาณที่ FSK module ด้านรับรับได้ เทียบกับ สัญญาณข้อมูลที่ส่งต่อไปให้กับคอมพิวเตอร์

4. วัดสัญญาณข้อมูลและสัญญาณนาฬิกาที่เข้าสู่คอมพิวเตอร์ ณ ตำแหน่งที่ 4 โดย CH1 เป็นสัญญาณข้อมูล CH2 เป็นสัญญาณนาฬิกา



รูปที่ 4.39 สัญญาณข้อมูลและสัญญาณนาฬิกาที่เข้าสู่คอมพิวเตอร์

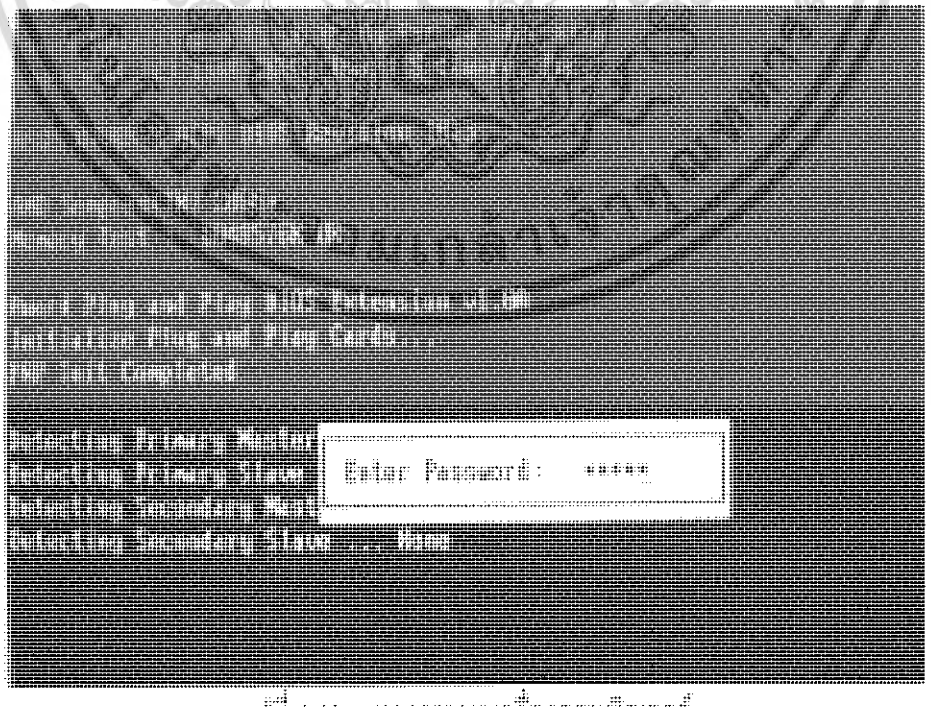
5. วัดสัญญาณข้อมูลที่ส่งออกจากเมาส์ เทียบกับ สัญญาณข้อมูลที่ส่งไปให้กับคอมพิวเตอร์ โดย CH1 เป็นสัญญาณ ณ ตำแหน่งที่ 1 CH2 เป็นสัญญาณ ณ ตำแหน่งที่ 4



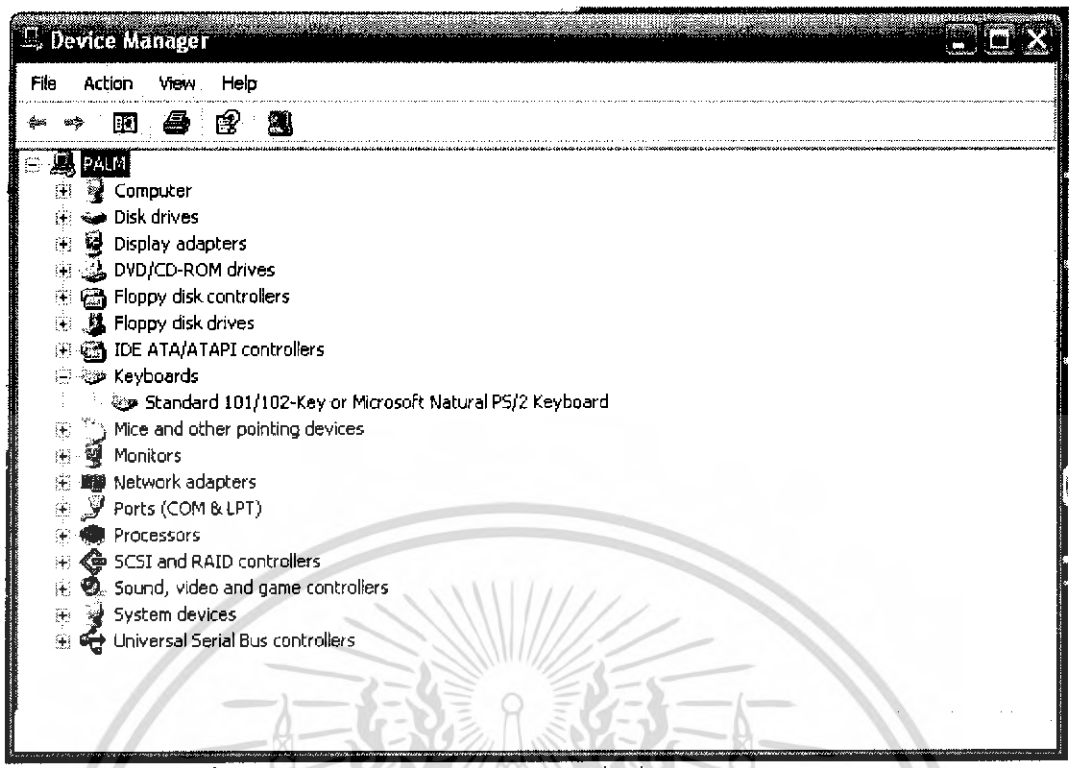
รูปที่ 4.40 สัญญาณข้อมูลที่ส่งออกจากเมาส์ เทียบกับ สัญญาณข้อมูลที่ส่งไปให้กับคอมพิวเตอร์

การทดลองที่ 7

1. ทำการติดตั้งใช้งานคีย์บอร์ดไร้สายตั้งแต่ก่อนเปิดเครื่องคอมพิวเตอร์ แล้วทำการเปิดเครื่องคอมพิวเตอร์เพื่อเริ่มใช้งาน

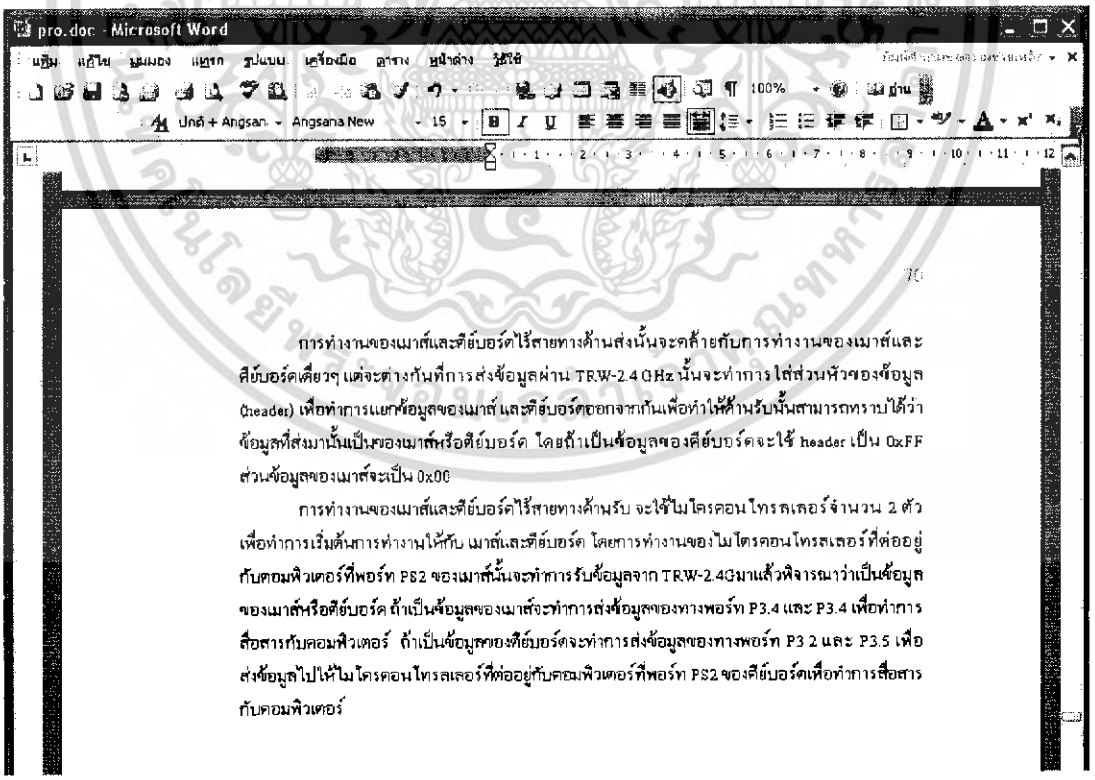


รูปที่ 4.41 แสดงการบูตเครื่องคอมพิวเตอร์



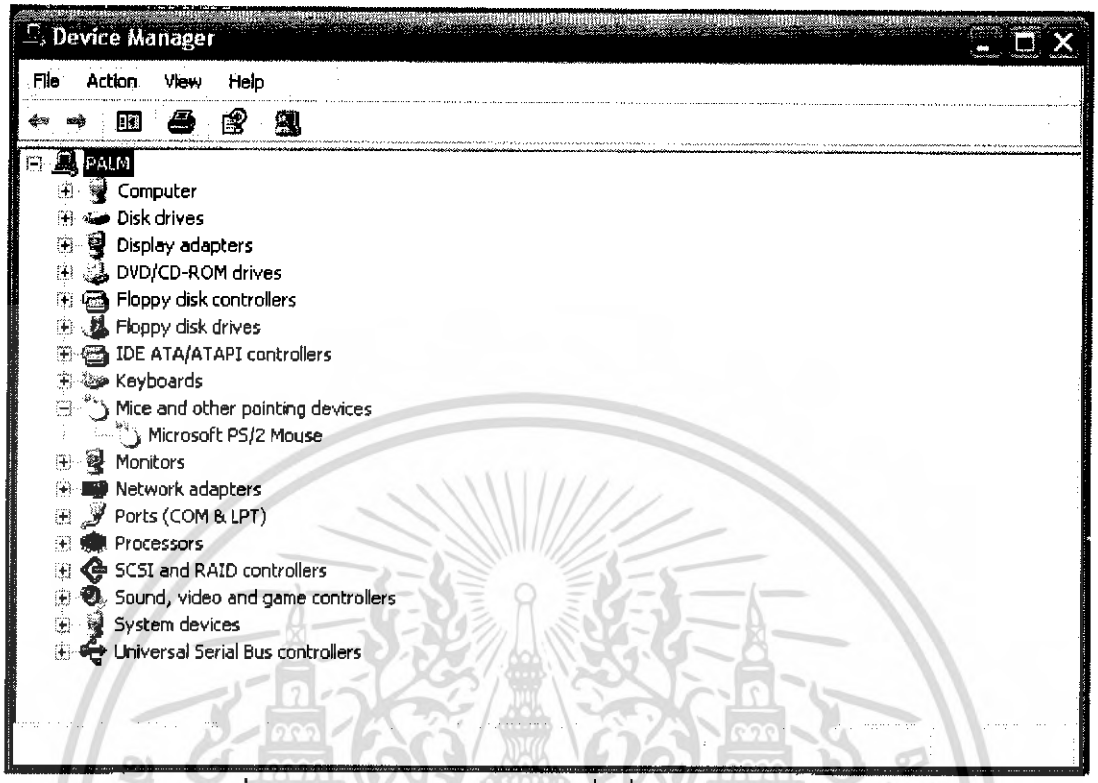
รูปที่ 4.42 แสดงไดร์ฟเวอร์ของคีย์บอร์ดที่เครื่องคอมพิวเตอร์ตรวจพบ

2. ทำการพิมพ์โดยใช้คีย์บอร์ดไร้สายลงในโปรแกรม ไมโครซอฟท์ เวิร์ด (Microsoft Word) แล้วทำการบันทึกผลทดลอง



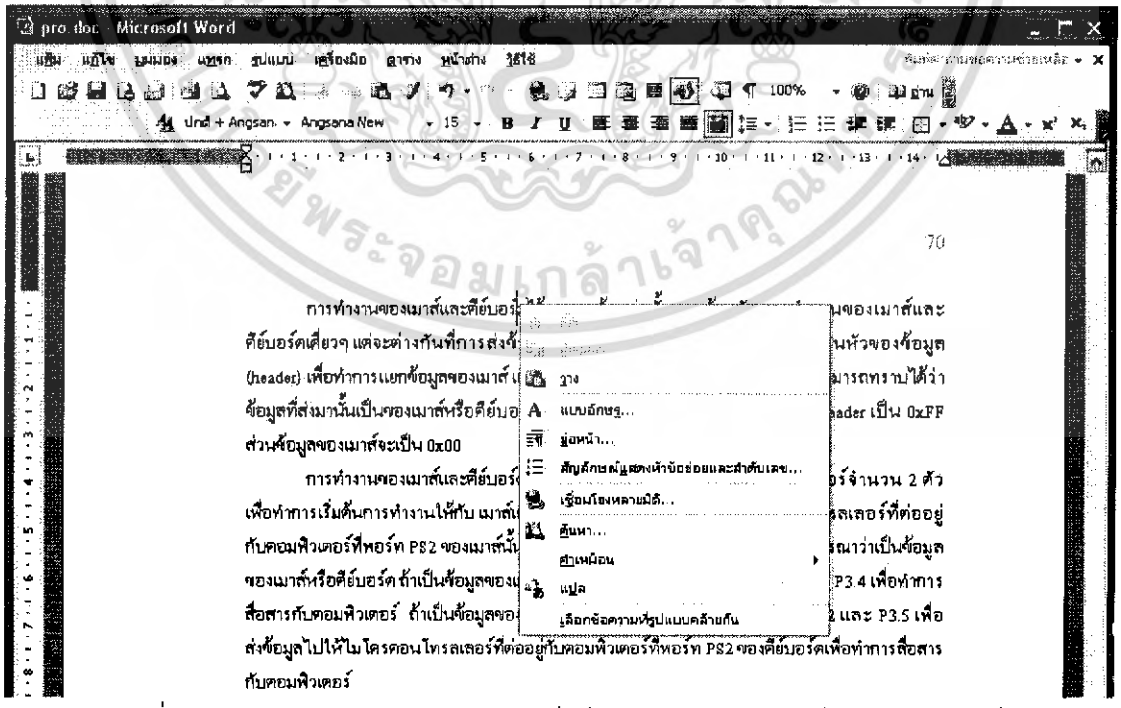
รูปที่ 4.43 ผลจากการพิมพ์ใช้งานจริง แล้วทำการบันทึกจากหน้าจอคอมพิวเตอร์

3. ทำการติดตั้งใช้งานเมาส์ตั่งแต่ก่อนเปิดเครื่องคอมพิวเตอร์ แล้วทำการเปิดเครื่องคอมพิวเตอร์เพื่อเริ่มใช้งาน



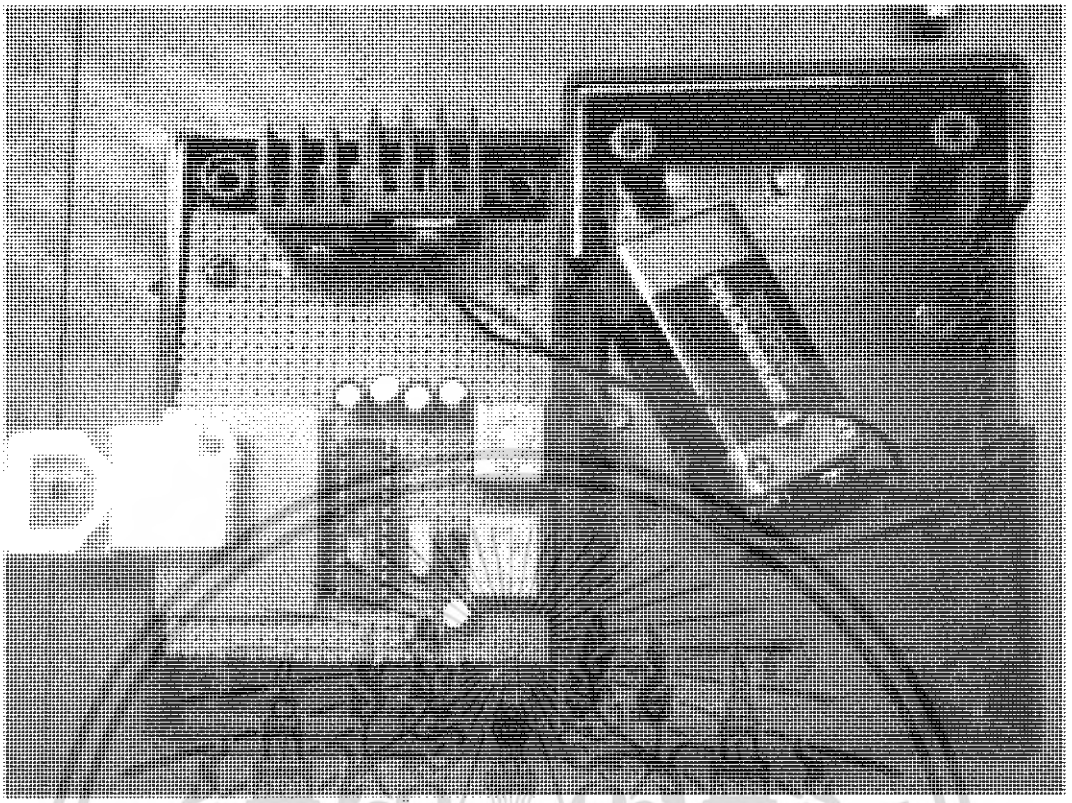
รูปที่ 4.44 แสดงไดร์ฟเวอร์ของเมาส์ที่เครื่องคอมพิวเตอร์ตรวจพบ

4. ทำการกดปุ่มขวาโดยใช้เมาส์ไร้สายลงในโปรแกรม ไมโครซอฟท์ เวิร์ด (Microsoft Word) แล้วทำการบันทึกผลทดลอง



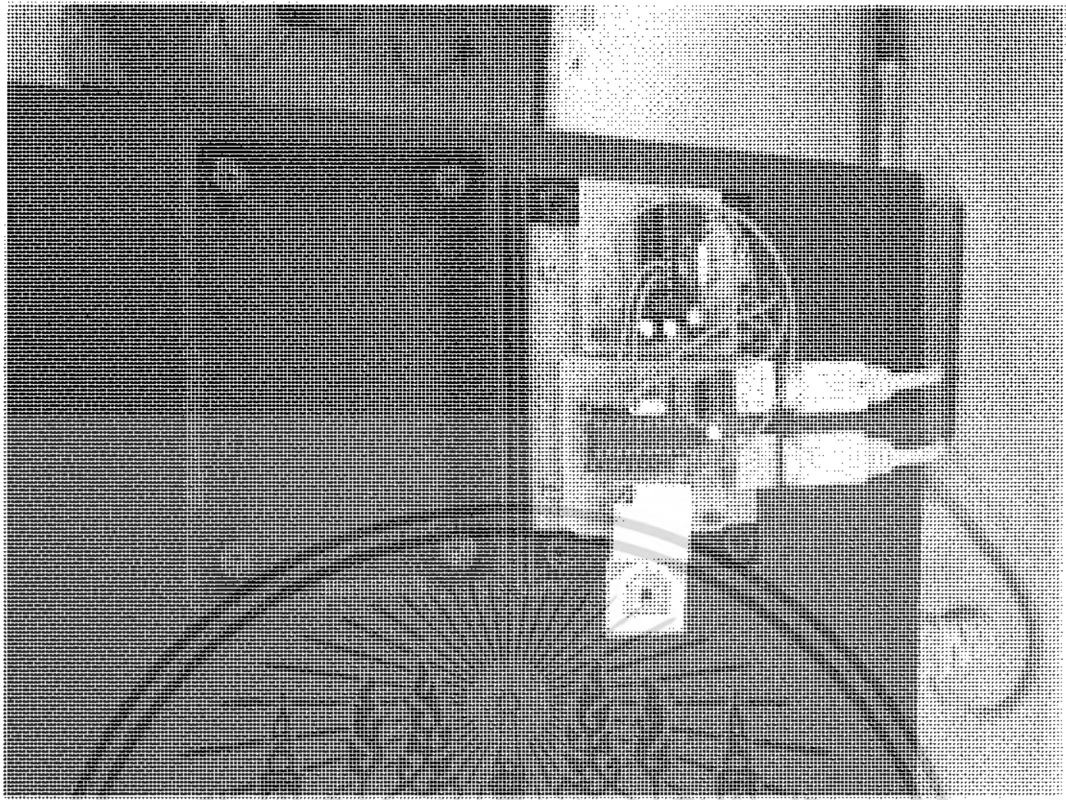
รูปที่ 4.45 ผลจากการกดปุ่มขวาของเมาส์แล้วทำการบันทึกจากหน้าจอคอมพิวเตอร์

5. อุปกรณ์เมาส์และคีย์บอร์ดไร้สายที่สมบูรณ์พร้อมใช้งานจริง

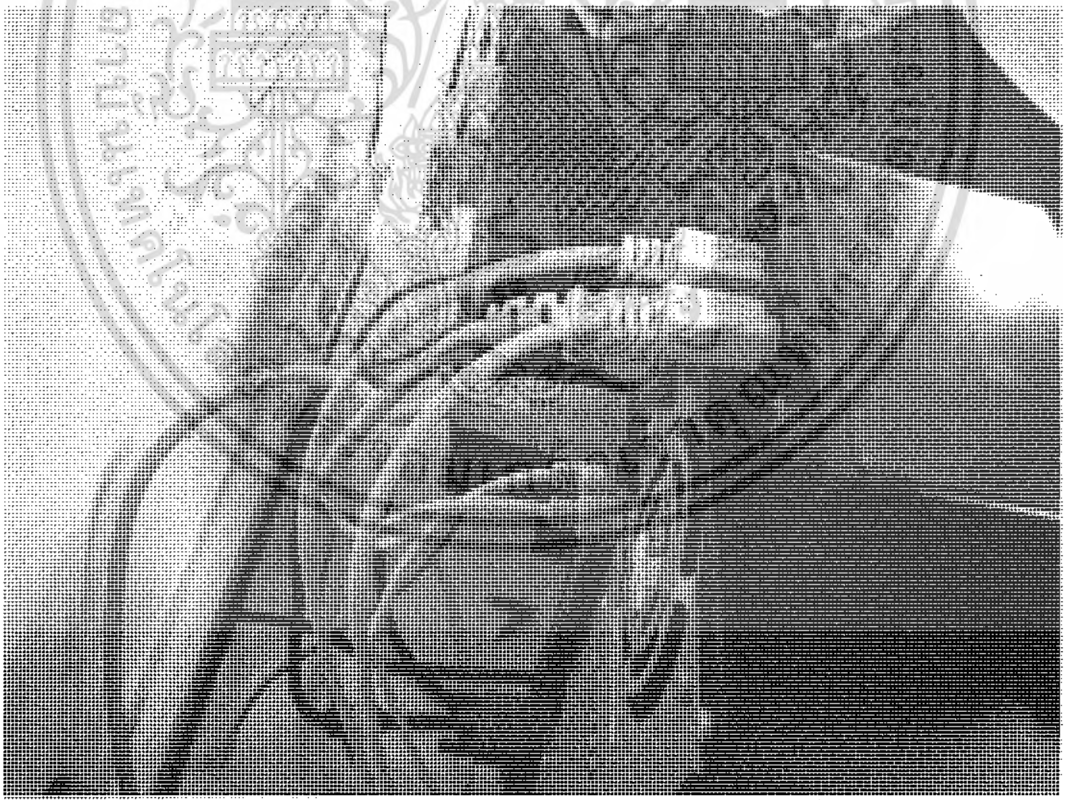


รูปที่ 4.46 ภาพในอุปกรณ์ด้านล่าง

รูปที่ 4.47 เมื่อทำการต่อเมาส์และคีย์บอร์ดสำหรับการใช้งานที่ด้านล่าง



รูปที่ 4.48 ภายในอุปกรณ์ด้านรับ



รูปที่ 4.49 เมื่อทำการต่อเข้ากับคอมพิวเตอร์สำหรับการใช้งานที่ด้านรับ

บทที่ 5
บทวิจารณ์และสรุปผล

5.1 บทวิจารณ์

จากการศึกษาการทำงานในส่วนต่างๆของการทำงานของคีย์บอร์ดและเมาส์รวมทั้งการทดลองทำการติดต่อระหว่างคีย์บอร์ดกับคอมพิวเตอร์และเมาส์กับคอมพิวเตอร์ได้พบปัญหาและมีข้อสังเกต ดังนี้

5.1.1 ในส่วนของกระบวนการติดต่อสื่อสารไร้สายโดยใช้ FSK module TRW-2.4 GHz

จากการทดลองติดต่อสื่อสารด้วย FSK module TRW-2.4 GHz สามารถติดต่อสื่อสารแบบไร้สาย (wireless) ได้ในระยะทางประมาณ 8 เมตร แต่ปัญหาคือยังไม่สามารถส่งระยะไกลได้เมื่อทำการทดลองส่งผ่านห้องที่มีกำแพงหลายชั้น แต่ภายหลังเมื่อนำมาทดลองในที่แจ้งสามารถส่งได้ไกลประมาณ 50 เมตร

5.1.2 ในส่วนของกระบวนการติดต่อสื่อสารระหว่างคีย์บอร์ดกับคอมพิวเตอร์แบบใช้สาย

จากการทดลองการติดต่อสื่อสารคีย์บอร์ดกับคอมพิวเตอร์แบบมีสาย สามารถติดต่อสื่อสารได้จริง โดยจากผลการทดลองนั้นสามารถส่งค่าสแกนโค้ดจากด้านคีย์บอร์ดไปสู่ด้านรับทางคอมพิวเตอร์ได้ โดยใช้ไมโครคอนโทรลเลอร์ควบคุม ซึ่งสามารถใช้ออสซิลโลสโคปจับค่าสัญญาณข้อมูลได้เป็นค่าสแกนโค้ดที่เหมือนกันทั้งทางด้านส่งและด้านรับ และสามารถแสดงผลออกทางหน้าจอคอมพิวเตอร์ได้จริง

5.1.3 ในส่วนของกระบวนการติดต่อสื่อสารระหว่างคีย์บอร์ดกับคอมพิวเตอร์แบบไร้สาย

จากการทดลองติดต่อสื่อสารระหว่างคีย์บอร์ดกับคอมพิวเตอร์แบบไร้สาย (Wireless) นั้นสามารถติดต่อสื่อสารได้อย่างสมบูรณ์ คือเมื่อทำการกดคีย์บอร์ดเพื่อส่งค่าตัวอักษรสามารถทำให้หน้าจอคอมพิวเตอร์แสดงค่าตัวอักษรได้ รวมทั้งการใช้งานตั้งแต่ตอนเริ่มเปิดเครื่องคอมพิวเตอร์

5.1.4 ในส่วนของกระบวนการติดต่อสื่อสารระหว่างเมาส์กับคอมพิวเตอร์แบบใช้สาย

จากผลการทดลองเขียนโปรแกรมการติดต่อสื่อสารระหว่างเมาส์กับคอมพิวเตอร์ที่มีการสื่อสารใช้สายโดยผ่านพอร์ตสื่อสารอนุกรม จากการทดลองสามารถทำให้อุปกรณ์เมาส์นั้นเซตค่าเริ่มต้นเพื่อให้พร้อมใช้งานได้โดยมีไมโครคอนโทรลเลอร์ทางด้านส่งเป็นตัวควบคุมการเซตค่า และทำให้คอมพิวเตอร์สามารถตรวจสอบการต่อของเมาส์ได้จากการควบคุมด้วยไมโครคอนโทรลเลอร์ทางด้านรับ และสามารถใช้ในการส่งข้อมูลและรับข้อมูลเมื่อทำการใช้เมาส์ได้จริง

5.1.5 ในส่วนของกระบวนการติดต่อสื่อสารระหว่างเมาส์กับคอมพิวเตอร์แบบไร้สาย

จากการทดลองติดต่อสื่อสารระหว่างเมาส์กับคอมพิวเตอร์แบบไร้สาย (Wireless) นั้นสามารถติดต่อสื่อสารได้อย่างสมบูรณ์ คือเมื่อทำการกดปุ่มซ้ายและขวาของเมาส์ เลื่อนเมาส์ไปตามทิศทางต่างๆ และทำการเลื่อน scrolling wheel สามารถทำให้หน้าจอคอมพิวเตอร์แสดงการทำงานของลูกศรได้จริง รวมทั้งสามารถใช้งานตั้งแต่ตอนเริ่มเปิดเครื่องคอมพิวเตอร์

5.2 บทสรุป

- จากการทดลองทำคีย์บอร์ดไร้สายสามารถติดต่อได้จริง โดยไม่มีความผิดพลาดเมื่ออยู่ในระยะที่กำหนด (50 เมตร)

- จากการทดลองทำเมาส์ไร้สายสามารถติดต่อได้จริง โดยไม่มีความผิดพลาดเมื่ออยู่ในระยะที่กำหนด (50 เมตร)

- เมื่อทำการทดลองใช้วงจรที่ใช้ทั้งเมาส์และคีย์บอร์ดรวมกันสามารถใช้งานได้จริงในระยะประมาณ 50 เมตร โดยมีปัญหาในการใช้คือไม่สามารถส่งข้อมูลทั้งเมาส์และคีย์บอร์ดในเวลาเดียวกันได้

บรรณานุกรม

[1] ผศ.ธีรวัฒน์ ประกอบผล , "การพัฒนาไมโครคอนโทรลเลอร์ด้วยภาษาซี" ส.ส.ท. ,2545

[2] วิวัฒน์ กิรานนท์ , " วิศวกรรมการสื่อสาร " คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2544

[3] รศ.สมยศ จุณณะปิยะ , "การประยุกต์ใช้งานไมโครคอนโทรลเลอร์" สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2546

[4] สุชาติ กังวารจิตต์ , "เครื่องรับส่งวิทยุและระบบวิทยุสื่อสาร" แผนกวิศวกรรม กองบังคับการตำรวจสื่อสาร กรมตำรวจ , 2538

[5] อุดม รานอก, "ภาษา C สำหรับงานควบคุมไมโครคอนโทรลเลอร์ MCS-51" นนทบุรี : ไอดีซี อินโฟ ดิสทริบิวเตอร์ เซ็นเตอร์ จำกัด, 2548

[6] http://www.beyondlogic.org/keyboard/keybrd.htm

[7] http://www.hut.fi/~then/mytexts/mouse.html

[8] http://www.simandl.cz/stranky/elektro/keyboard/keyboard_a.htm

[9] http://www.networktechinc.com/ps2-prots.html

[10] http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/PS2/ps2.htm

[11] http://www.epanorama.net/documents/pc/mouse.html

[12] http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/mouse/mouse.html

[13] http://www.uni.net.th/~08_2543/chap04.html

[14] http://www.fareastern.ac.th/acad/bc/pichate/network/net_c2/chapter2.htm

[15] http://www.computer-engineering.org/index.php?title=PS/2_Keyboard_Interface

[16]http://www.computer-engineering.org/index.php?title=PS/2_Mouse/Keyboard_Protocol

ภาคผนวก

Source Code Program

โปรแกรมของเมาส์และคีย์บอร์ดไร้สายทางด้านส่ง

```
#pragma code
#include <AT892051.h>
#include <string.h>
#include <intrins.h>
#define CE  P1_4
#define CS  P1_3
#define DAT P1_1
#define CLK P1_2
#define DR1 P1_0
#define MODE_TX 0
#define MODE_RX 1

sbit DAT_key = P3^2;
sbit CLK_key = P3^5;

sbit SDA_mou = P3^3;
sbit SCL_mou = P3^4;

unsigned char
o,l,dat,led,datakey,y,w,datmou,datamou,arr_point,rec_dat[4];

bit
inbit,parity_bit,scroll_bit,num_bit,cap_bit,inbitmou,parity_bitmou;

/***********TRW-2.4G************/
void Wait(unsigned char q)
{
      int i;
      while(q)
      {
            i = 0x00;
            while(i--);
            q--;
      }
}

void Init_TRW24G(void)
{
      Wait(100);
      CE  = 0;
      CS  = 0;
      CLK = 0;
      DAT = 0;
      DR1 = 0;
      Wait(100);
}

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

void Write_TRW24(unsigned char Data)
{
```

```
	unsigned char j;
	bit Out;
	for (j=0;j<8;j++)
	{
		Out = Data & 0x80;
		DAT = Out;

		Data = Data << 1;
		CLK_TRW24();
	}
}
void SetMode_TRW24( unsigned char Mode)
{
	Wait(200);
	CE = 0;
	CS = 1;
	Write_TRW24(0x8E); /* Reserved for testing */
	Write_TRW24(0x08); /* Reserved for testing */
	Write_TRW24(0x1C); /* Reserved for testing */

	Write_TRW24(0x10); /* Length of Bit Ch 2 */
	Write_TRW24(0x10); /* Length of Bit Ch 1 */

	Write_TRW24(0xC0); /* Address 5 Byte Ch 2 */
	Write_TRW24(0xAA);
	Write_TRW24(0x55);
	Write_TRW24(0xAA);
	Write_TRW24(0x55);

	Write_TRW24(0xCC); /* Address 5 Byte Ch 1 */
	Write_TRW24(0x33);
	Write_TRW24(0xCC);
	Write_TRW24(0x33);
	Write_TRW24(0xCC);

	Write_TRW24(0xA3); /* Number of Address bit + CRC */
	Write_TRW24(0x4F); /* RF Programming */
	Write_TRW24(0x88+Mode);

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

	CS = 0; Wait(200);
}
void Send_TRW24key(unsigned char Data)
{
	Wait(1);
	CS = 0;
	CE = 1;
	Write_TRW24(0xCC);	/*Address*/
	Write_TRW24(0x33);	/*Address*/
	Write_TRW24(0xCC);	/*Address*/
	Write_TRW24(0x33);	/*Address*/
	Write_TRW24(0xCC);	/*Address*/
	Write_TRW24(0xFF);	/*header keyboard*/
	Write_TRW24(Data);	/*keyboard data*/
	CLK = 0;
	CE = 0;
	Wait(1);
```

```
}
void Send_TRW24mou(unsigned char Data)
{
      Wait(1);
      CS = 0;
      CE = 1;
      Write_TRW24(0xCC);       /*address*/
      Write_TRW24(0x33);       /*address*/
      Write_TRW24(0xCC);       /*address*/
      Write_TRW24(0x33);       /*address*/
      Write_TRW24(0xCC);       /*address*/
      Write_TRW24(0x00);       /*header mouse*/
      Write_TRW24(Data);       /*mouse data*/
      CLK = 0;
      CE = 0;
      Wait(1);
}
/***********TRW-2.4G end*************/
void delay50()
      {
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();
      }
void delay25()
      {
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();
      }
void one_bit()                         /*wait for one bit*/
      {
            while (CLK_key == 1);
            while (CLK_key == 0);
      }
void one_bitmou()
      {
            while (SCL_mou == 1);
            while (SCL_mou == 0);
      }
void gen_led()                         /*gen led keyboard*/
      {
            led = 0x00;
            led = led<<1;
            led = led | scroll_bit;
            led = led<<1;
            led = led | num_bit;
```

```
                led = led<<1;
                led = led | cap_bit;
                for (o=0;o<5;o++) led = led<<1;
        }
unsigned char Read_key(void)  /*read data from keyboard*/
{
        one_bit();                      /*start bit*/
        for (l=0;l<8;l++)
        {
                while (CLK_key == 1);
                inbit = DAT_key;
                dat = dat<<1;
                dat = dat | inbit;
                while (CLK_key == 0);
        }
        one_bit();                      /*parity bit*/
        one_bit();                      /*stop bit*/
        return(dat);
}
unsigned char Read_mou(void)  /*read data from mouse*/
        {

        one_bitmou();                   /*start bit*/
        for (y=0;y<8;y++)
        {
                while (SCL_mou == 1);
                inbitmou = SDA_mou;
                datmou = datmou<<1;
                datmou = datmou | inbitmou;
                while (SCL_mou == 0);
        }
                one_bitmou();           /*parity bit*/
                one_bitmou();           /*stop bit*/
                return(datmou);
         }

void gen_parity(unsigned char temp_dat)   /*gen_parity keyboard*/
        {
                bit outbit;
                int m;
                parity_bit = 1;
                for (m=0;m<8;m++)
                        {
                                outbit = temp_dat & 0x80;
                                parity_bit ^= outbit;
                                temp_dat = temp_dat<<1;
                        }
        }
void gen_paritymou(unsigned char temp_datmou)  /*gen_parity mouse*/
        {
                bit outbitmou;
                int x;
                parity_bitmou = 1;
                for (x=0;x<8;x++)
                        {
                                outbitmou = temp_datmou & 0x80;
                                parity_bitmou ^= outbitmou;
                                temp_datmou = temp_datmou<<1;
                        }
        }
void send_to_key(unsigned char pc_dat) /*send data to keyboard*/
```

```
        {
              bit outbit;
              int n;
              gen_parity(pc_dat);
              CLK_key = 0;       /*inhibit communication*/
              delay50();
              delay50();
              delay25();         /*delay 125 us*/
              DAT_key = 0;       /*Request-to-Send + Send Start bit*/
              CLK_key = 1;
              while (CLK_key == 1);
              for (n=0;n<8;n++)
              {

                    outbit = pc_dat & 0x80;
                    DAT_key = outbit;
                    pc_dat = pc_dat<<1;
                    while (CLK_key == 0);
                    while (CLK_key == 1);

              }
              DAT_key = parity_bit;
          while (CLK_key == 0);
              while (CLK_key == 1);
              DAT_key = 1;
              while (CLK_key == 0);
              while (CLK_key == 1);          /*wait for ACK*/
              while (CLK_key == 0);
              while (DAT_key == 0);
        }
    void send_to_mou(unsigned char old_datmou) /*send data to mouse*/
        {
              bit outbitmou;
              int z;
              gen_paritymou(old_datmou);
              SCL_mou = 0;       /*inhibit communication*/
              delay50();
              delay50();
              delay25();         /*delay 125 us*/
              SDA_mou = 0;       /*Request-to-Send + Send Start bit*/
              SCL_mou = 1;
              while (SCL_mou == 1);
              for (z=0;z<8;z++)
              {
                    outbitmou = old_datmou & 0x80;
                    old_datmou = old_datmou<<1;
                    SDA_mou = outbitmou;
                    while (SCL_mou == 0);
                    while (SCL_mou == 1);
              }
              SDA_mou = parity_bitmou;
          while (SCL_mou == 0);
              while (SCL_mou == 1);
              SDA_mou = 1;
              while (SCL_mou == 0);
              while (SCL_mou == 1);    /*wait for ACK*/
              while (SCL_mou == 0);
              while (SDA_mou == 0);
        }
    void main(void)
    {           /*initial trw*/
```

```
Init_TRW24G();
SetMode_TRW24(MODE_TX);
/*initial keyboard*/
scroll_bit = 0;num_bit = 1;cap_bit = 0;
/*initial mouse*/
datamou = Read_mou();
while(datamou!=0x55);
datamou = Read_mou();
while(datamou!=0x00);
send_to_mou(0xFF);
datamou = Read_mou();
while(datamou!=0x5F);
SCL_mou=1;
SDA_mou=1;
datamou = Read_mou();
while(datamou!=0x55);
datamou = Read_mou();
while(datamou!=0x00);
send_to_mou(0xFF);
datamou = Read_mou();
while(datamou!=0x5F);
SCL_mou=1;
SDA_mou=1;
datamou = Read_mou();
while(datamou!=0x55);
datamou = Read_mou();
while(datamou!=0x00);
send_to_mou(0xFF);
datamou = Read_mou();
while(datamou!=0x5F);
SCL_mou=1;
SDA_mou=1;
datamou = Read_mou();
while(datamou!=0x55);
datamou = Read_mou();
while(datamou!=0x00);
send_to_mou(0xCF);
datamou = Read_mou();
while(datamou!=0x5F);
send_to_mou(0x13);
datamou = Read_mou();
while(datamou!=0x5F);
send_to_mou(0xCF);
datamou = Read_mou();
while(datamou!=0x5F);
send_to_mou(0x26);
datamou = Read_mou();
while(datamou!=0x5F);
send_to_mou(0xCF);
datamou = Read_mou();
while(datamou!=0x5F);
send_to_mou(0x0A);
datamou = Read_mou();
while(datamou!=0x5F);
send_to_mou(0x4F);
datamou = Read_mou();
while(datamou!=0x5F);
SCL_mou=1;
SDA_mou=1;
datamou = Read_mou();
while(datamou!=0xC0);
```

```
                send_to_mou(0x17);
                datamou = Read_mou();
                while(datamou!=0x5F);
                send_to_mou(0x40);
                datamou = Read_mou();
                while(datamou!=0x5F);
                send_to_mou(0x67);
                datamou = Read_mou();
                while(datamou!=0x5F);
                send_to_mou(0xCF);
                datamou = Read_mou();
                while(datamou!=0x5F);
                send_to_mou(0x14);
                datamou = Read_mou();
                while(datamou!=0x5F);
                send_to_mou(0x2F);
                datamou = Read_mou();
                while(datamou!=0x5F);
                arr_point = 0;
                /***************main******************/
while(1)
        {
                        DAT_key = 1;
                        CLK_key = 1;
                        SDA_mou = 1;
                        SCL_mou = 1;
                if(DAT_key == 0)
                {
        /************************keyboard*****************************/
                        datakey=Read_key();
                        Send_TRW24key(datakey);
                        if (datakey == 0xEE || dat == 0x1A || dat == 0x7E)
                            {
                                datakey=Read_key();
                                while (datakey != 0x0F);
                                Send_TRW24key(datakey);
                                datakey=Read_key();
                                Send_TRW24key(datakey);
                                switch (datakey)
                                        {
        case 0xEE : num_bit    = ~num_bit;        break;
        case 0x1A : cap_bit    = ~cap_bit;        break;
        case 0x7E : scroll_bit = ~scroll_bit;     break;
                                        }
                                gen_led();
                                send_to_key(0xB7);
                                datakey=Read_key();
                                while (datakey != 0x5F);
                                send_to_key(led);
                                datakey=Read_key();
                                while (datakey != 0x5F);
                            }
        /***********************keyboard end**************************/
                }

                if(SDA_mou == 0)
                {

        /************************mouse*********************************/
```

```
                datamou = Read_mou();

                        if(datamou!=0x55)
                        {
                              rec_dat[arr_point] = datamou;
                              arr_point++;
                        }
                        if (arr_point == 4)
                        {
                              for(w=0;w<4;w++)
                              {
                                    Send_TRW24mou(rec_dat[w]);
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    delay50();
                                    rec_dat[w] = 0;
                              }

                              arr_point = 0;
                        }
      /*************************mouse end*****************************/
            }

      }
}
```

โปรแกรมของเมาส์และคีย์บอร์ดไร้สายทางด้านรับ;ไมโครคอนโทรลเลอร์ ที่ต่อกับเมาส์

```
#pragma code
#include <AT892051.h>
#include <string.h>
#include <intrins.h>
#define CE  P1_2
#define CS  P1_3
#define DAT P1_5
#define CLK P1_4
#define DR1 P1_6
#define MODE_TX0
#define MODE_RX 1

sbit DAT_key = P3^2;
sbit CLK_key = P3^5;

sbit DAT_mou = P3^3;
sbit CLK_mou = P3^4;

bit inbit,databit,databitmou,inbitmou,parity_bitmou;
unsigned char m,n,header,re_dat,input,x,y,datmou,re_datmou;

/***********TRW-2.4G*************/
void Wait(unsigned char q)
{
      int i;
      while(q)
      {
            i = 0x00;
            while(i--);
            q--;
      }
}

void Init_TRW24G(void)
{
      Wait(100);
      CE  = 0;
      CS  = 0;
      CLK = 0;
      DAT = 0;
      DR1 = 0;
      Wait(100);
}

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

void Write_TRW24(unsigned char Data)
{
      unsigned char j;
      bit Out;
      for (j=0;j<8;j++)
      {
            Out = Data & 0x80;
            DAT = Out;
            Data = Data << 1;
```

```
            CLK_TRW24();
        }
}
unsigned char Read_TRW24(void)
{
        unsigned char k,Temp;
        bit red;
        DAT = 1;
        for (k=0;k<8;k++)
        {
            Temp = Temp << 1;
            CLK = 1;
            red = DAT;
            if (red) { Temp = Temp + 0x01;}
            CLK = 0;
        }
        return(Temp);
}
void SetMode_TRW24( unsigned char Mode)
{
        Wait(200);
        CE = 0;
        CS = 1;
        Write_TRW24(0x8E); /* Reserved for testing */
        Write_TRW24(0x08); /* Reserved for testing */
        Write_TRW24(0x1C); /* Reserved for testing */

        Write_TRW24(0x10); /* Length of Bit Ch 2 */
        Write_TRW24(0x10); /* Length of Bit Ch 1 */

        Write_TRW24(0xC0); /* Address 5 Byte Ch 2 */
        Write_TRW24(0xAA);
        Write_TRW24(0x55);
        Write_TRW24(0xAA);
        Write_TRW24(0x55);

        Write_TRW24(0xCC); /* Address 5 Byte Ch 1 */
        Write_TRW24(0x33);
        Write_TRW24(0xCC);
        Write_TRW24(0x33);
        Write_TRW24(0xCC);

        Write_TRW24(0xA3); /* Number of Address bit + CRC */
        Write_TRW24(0x4F); /* RF Programming */
        Write_TRW24(0x88+Mode);

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

        CS = 0; Wait(200);
}

/***********TRW-2.4G end*************/
void delay50()
        {
        _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
        _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
```

```
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();
      }
void delay25()
      {
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      }
/*clock keyboard*/
void Clock_high()
      {
            CLK_key = 1;
            delay25();
       }
void Clock_low()
      {
            CLK_key = 0;
            delay25();
       }
/*clock mouse*/
void Clock_highmou()
      {
            CLK_mou = 1;
            delay50();
       }

void Clock_lowmou()
      {
            CLK_mou = 0;
            delay50();
       }
void gen_paritymou(unsigned char temp_datmou)  /*gen_parity mouse*/
      {
            bit outbitmou;
            int z;
            parity_bitmou = 1;
            for (z=0;z<8;z++)
                  {
                        outbitmou = temp_datmou & 0x80;
                        parity_bitmou ^= outbitmou;
                        temp_datmou = temp_datmou<<1;
                  }
      }
void send_data_key(unsigned char old_dat)/*send data key to mcs key*/
      {

            while(CLK_key == 0);
            DAT_key = 0;
            Clock_low();
            Clock_high();
            for (m=0;m<8;m++)
            {
                  databit = old_dat & 0x80;
                  DAT_key = databit;
```

```
                old_dat = old_dat<<1;
                Clock_low();
                Clock_high();

            }
            DAT_key = 1;
            CLK_key = 1;

      }
void send_data_mou(unsigned char old_datmou)/*send data mouse to pc*/
      {
            gen_paritymou(old_datmou);
            while(CLK_mou == 0);
            DAT_mou = 0;      /*check clock high at least 50us*/
            CLK_mou = 0;
            delay50();
            CLK_mou = 1;
            delay50();
            for (x=0;x<8;x++)
            {
                databitmou = old_datmou & 0x80;
                DAT_mou = databitmou;
                old_datmou = old_datmou<<1;
                CLK_mou = 0;
                delay50();
                CLK_mou = 1;
                delay50();
            }
            DAT_mou = parity_bitmou;
            CLK_mou = 0;
            delay50();
            CLK_mou = 1;
            delay50();
            DAT_mou = 1;
            CLK_mou = 0;
            delay50();
            CLK_mou = 1;
      }

void receive_from_mou() interrupt 2 /*interrupt from INT1;pc boot
mouse*/
      {
            if (DAT_mou == 0 && CLK_mou == 0)
            {
                while (CLK_mou == 0);
                Clock_highmou();  /*start bit*/
                Clock_lowmou();
                for (y=0;y<8;y++)
                {
                    inbitmou = DAT_mou;
                    re_datmou = re_datmou<<1;
                    re_datmou = re_datmou | inbitmou;
                    Clock_highmou();
                    Clock_lowmou();
                }
                Clock_highmou();  /*parity bit*/
                Clock_lowmou();
                Clock_highmou();  /*stop bit*/
                Clock_lowmou();
                DAT_mou = 0;      /*ACK*/
                Clock_highmou();
```

```
                    Clock_lowmou();
                    DAT_mou = 1;
                    CLK_mou = 1;
                    switch(re_datmou)
                          {
case 0xFF:
      send_data_mou(0x5F);send_data_mou(0x55);send_data_mou(0x00);
      break;
case 0x4F:  send_data_mou(0x5F);    send_data_mou(0xC0);    break;
case 0x7F:  send_data_mou(input);   break;
case 0x6F:  send_data_mou(0x5F);    send_data_mou(0x26);
      send_data_mou(0x40);    send_data_mou(0x67); break;
default   :  send_data_mou(0x5F);    break;
                          }
            }
      }
void main(void)
{
            /*initial interrupt*/
            IT1 = 0x01;
            IT0 = 0x01;
            IE = 0x85;

            /*initial trw*/
            Init_TRW24G();
            SetMode_TRW24(MODE_RX);

            /*main*/
      while(1)
      {
            DAT_key = 1;
            CLK_key = 1;
            DAT_mou = 1;
            CLK_mou = 1;
            while(!DR1);
            header=Read_TRW24();
            if(header==0xFF)
            {
            /*******************keyboard*****************/
            input=Read_TRW24();
            send_data_key(input);
            /*******************keyboard end**************/
            }
            if(header==0x00)
            {
            /*******************mouse********************/
            input = Read_TRW24();
            if(input!=0x55)
            {send_data_mou(input);}

            /*******************mouse end****************/
            }

      }
}
```

โปรแกรมของเมาส์และคีย์บอร์ดไร้สายทางด้านรับ;ไมโครคอนโทรลเลอร์ ที่ต่อกับคีย์บอร์ด

```
#pragma code
#include <AT892051.h>
#include <string.h>
#include <intrins.h>

sbit DAT_key = P3^3;
sbit CLK_key = P3^4;

sbit DAT_mcs = P1^7;
sbit CLK_mcs = P1^6;

bit inbit,parity_bit,databit;
unsigned char m,n,o,dat,l,re_dat,input;


void delay50()
      {
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_()
;_nop_();
      _nop_();_nop_();_nop_();
      }
void delay100()
      {
      delay50();delay50(); delay50(); delay50(); delay50();
      }
void delay3600()
      {
       delay100(); delay100(); delay100(); delay100(); delay100();
delay100(); delay100(); delay100(); delay100();
      }
void delay500ms()
      {
      delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
      delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
      delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
      delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
      delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
      delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
      delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
      delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
```

```
	delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
	delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
	delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
	delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
	delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
	delay3600();delay3600();delay3600();delay3600();delay3600();del
ay3600();delay3600();delay3600();delay3600();delay3600();
	}

/*clock keyboard*/
void Clock_high()
	{
		CLK_key = 1;
		delay50();
	}

void Clock_low()
	{
		CLK_key = 0;
		delay50();
	}

void one_bit()		/*wait for one bit*/
	{
		while (CLK_mcs == 1);
		while (CLK_mcs == 0);
	}

void gen_parity(unsigned char temp_dat)	/*gen_parity keyboard*/
	{
		bit outbit;
		int l;
		parity_bit = 1;
		for (l=0;l<8;l++)
			{
				outbit = temp_dat & 0x80;
				parity_bit ^= outbit;
				temp_dat = temp_dat<<1;
			}
	}

void send_data_key(unsigned char old_dat)/*send data keyboard to pc*/
	{
		gen_parity(old_dat);
		while(CLK_key == 0);
		DAT_key = 0;		/*check clock high at least 50us*/
		Clock_low();
		Clock_high();
		for (m=0;m<8;m++)
		{
			databit = old_dat & 0x80;
			DAT_key = databit;
			old_dat = old_dat<<1;
			Clock_low();
			Clock_high();
		}
```

```
            DAT_key = parity_bit;
            Clock_low();
            Clock_high();
            DAT_key = 1;
            Clock_low();
            Clock_high();
      }
unsigned char Read_key(void) /*read data keyboard from mcs mouse*/
{
      one_bit();                    /* start bit*/
      for (l=0;l<8;l++)
      {
            while (CLK_mcs == 1);
            inbit = DAT_mcs;
            dat = dat<<1;
            dat = dat | inbit;
            while (CLK_mcs == 0);
      }
      return(dat);
}

void receive_from_key()   interrupt 2/*interrupt from INT1;pc boot
keyboard*/
      {
            if (DAT_key == 0 && CLK_key == 0)
            {
                  while (CLK_key == 0);
                  Clock_high();       /*start bit*/
                  Clock_low();
                  for (n=0;n<8;n++)
                  {
                        inbit = DAT_key;
                        re_dat = re_dat<<1;
                        re_dat = re_dat | inbit;
                        Clock_high();
                        Clock_low();
                  }
                  Clock_high();       /*parity bit*/
                  Clock_low();
                  Clock_high();       /*stop bit*/
                  Clock_low();
                  DAT_key = 0;        /*ACK*/

                  Clock_high();
                  Clock_low();
                  DAT_key = 1;
                  CLK_key = 1;

                  delay100();
                  switch(re_dat)
                        {
case 0x4F   :     send_data_key(0x5F);     delay100();
      send_data_key(0xD5);    break;
case 0xFF   :   send_data_key(0x5F);       delay500ms();
      send_data_key(0x55);    break;
case 0x7F   :   send_data_key(input);      break;
case 0x77   :     send_data_key(0x77);     break;

default           :     send_data_key(0x5F);     break;

                        }
```

```
            }
        }
void main(void)
{
            /*initial interrupt*/
            IT1 = 0x01;
            IT0 = 0x01;
            IE = 0x85;

            /*main*/
      while(1)
      {
            /*keyboard*/
            DAT_key = 1;
            CLK_key = 1;
            input=Read_key();
if (input == 0xEE || input == 0x1A || input == 0x7E ||input == 0x55)
                  {
                        IE = 0x85;
                        send_data_key(input);
                        for (o=0;o<2;o++)
                        {
                              DAT_key = 1;
                              CLK_key = 1;
                              input = Read_key();
                              send_data_key(input);
                        }

                  }
                  else
                  {
                        IE = 0x00;
                        send_data_key(input);
                  }

      }
}
```

LAIPAC TECH

TRF-2.4G Transceiver Data Sheet

High frequency 2.4G Wireless Transceiver

Antenna, Codec and CRC built in

Data Rate up to 1Mbps

Specification

- Frequency Range: 2.4~2.524 GHz ISM band
- Modulate Mode: GFSK
- Data Rate: 1Mbps; 250Kbps
- Multi channel operation: 125 channels, Channel switching time<200uS, Support frequency hopping
- Emulated full duplex RF link due to the 1Mbits/s on the air data rate
- Simultaneous dual receiver
- Data slicer / clock recovery of data
- Including decoder, encoder and data buffer and CRC computation
- ShockBurst mode for ultra-low power operation and relaxed MCU performance
- Sensitivity: -90dBm
- Built in antenna
- Power supply range: 1.9 to 3.6 V
- Low supply current (TX), typical 10.5mA peak@ -5dBm output power
- Low supply current (RX), typical18mA peak in receive mode
- Supply current in Power Down Mode: 1 uA
- Operating Temperature: -40~+85 Centigrade
- Size: 20.5*36.5*2.4mm
- 100% RF tested
- Competitive price

Applications

- Wireless mouse, keyboard, joystick
- Wireless data communication
- Alarm and security systems
- Home automation
- Wireless Earphone
- Telemetry
- Surveillance
- Automotive

GENERAL DESCRIPTION

Laipac TRF-2.4G Module is an easy to use radio transceiver for the world wide 2.4 - 2.5 GHz ISM band. The transceiver consists of an antenna, a fully integrated frequency synthesizer, a power amplifier, a crystal oscillator and a modulator. Output power and frequency channels are easily programmable by use of the 3-wire serial interface. Current consumption is very low, only 10.5mA at an output power of -5dBm and 18mA in receive mode. Built-in Power Down modes makes power saving easily realizable.

ELECTRICAL SPECIFICATIONS

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

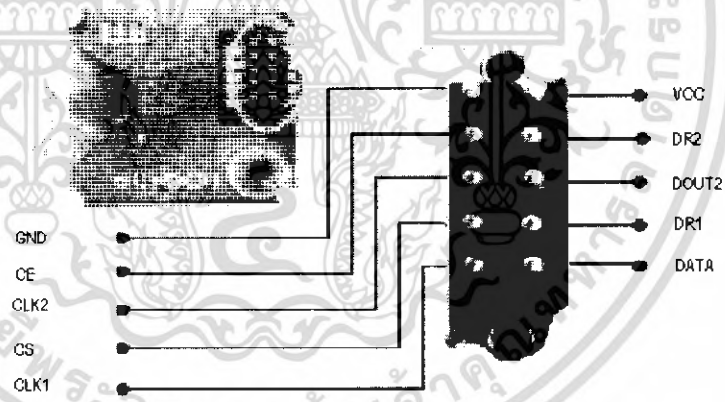
Symbol	Parameter (condition)	Notes	Min.	Ttp.	Max.	Units
	Operating conditions					
VCC	Supply voltage		1.9	3.0	3.6	V
TEMP	Operating Temperature		-40	+27	+85	°C
	Digital input pin					
V_{IH}	HIGH level input voltage		VCC-0.3		VCC	V
V_{IL}	LOW level input voltage		Vss		0.3	V
	Digital output pin					
V_{OH}	HIGH level output voltage (IOH=-0.5mA)		VCC-0.3		VCC	V
V_{OL}	LOW level output voltage (IOL=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		>0		1000	kbps
R_{GFSK}	Data rate Direct Mode	2)	250		1000	kbps
$F_{CHANNEL}$	Channel spacing			1		MHz
	Transmitter operation					
P_{RF}	Maximum Output Power	3)		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_{VCC}	Supply current @ 0dBm output power	4)		13		mA
I_{VCC}	Supply current @ -20dBm output power	4)		8.8		mA
I_{VCC}	Average Supply current @ -5dBm output power, ShockBurst	5)		0.8		mA
I_{VCC}	Average Supply current in stand-by mode	6)		12		µA
I_{VCC}	Average Supply current in power down			1		µA
	Receiver operation					
I_{VCC}	Supply current one channel 250kbps			18		mA
I_{VCC}	Supply current one channel 1000kbps			19		mA
I_{VCC}	Supply current two channels 250kbps			23		mA
I_{VCC}	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

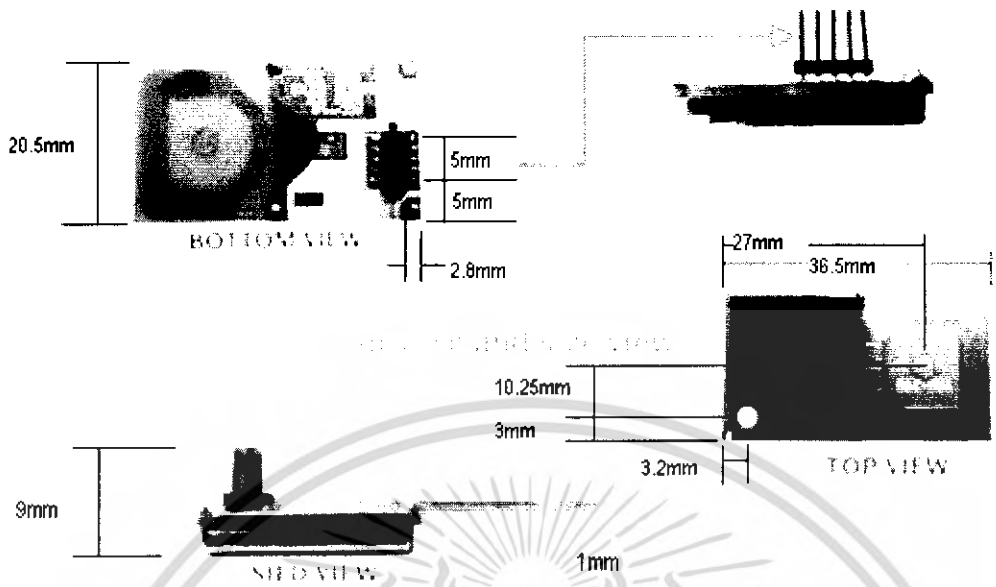
C/ICO	C/I Co-channel			6		dB
C/I1ST	1st Adjacent Channel Selectivity C/I 1MHz			-1		dB
C/I2ND	2nd Adjacent Channel Selectivity C/I 2MHz			-16		dB
C/I3RD	3rd Adjacent Channel Selectivity C/I 3MHz			-26		dB
RXB	Blocking Data Channel 2			-41		dB

1) Usable band is determined by local regulations
2) Data rate must be either 250kbps or 1000kbps.
3) De-embedded Antenna load impedance = 400
4) De-embedded Antenna load impedance = 400 . Effective data rate 250kbps or 1Mbps.
5) De-embedded Antenna load impedance = 400 . Effective data rate 10kbps.
6) Current if 4 MHz crystal is used.

Table 1 TRF-2.4G RF specifications

PIN ASSIGNMENT





Note: The connector pitch size is 1.25mm, mounting hole diameter is 2.8mm

PIN FUNCTIONS

Pin	Name	Pin funtion	Description
1	GND	Power	Gound (0V)
2	CE	Input	Chip Enable activates RX or TX mode
3	CLK2	I/O	Clock outpu/input for RX data channel 2
4	CS	Input	Chip Select activates Configuration mode
5	CLK1	I/O	Clock Input(TX)&I/O(RX) for data channel 1 3-wire interface
6	DATA	I/O	RX data channel 1/TX data input /3-wire interface
7	DR1	Output	RX data ready at data channel 1 (ShockBurst only)
8	DOUT2	Output	RX data channel 2
9	DR2	Output	RX data ready at data channel 2 (ShockBurst only)
10	VCC	Power	Power Supply (+3V DC)

Table 2 TRF-2.4G pin function

MODE OF OPERATION

TRF-2.4G can be set in the following main mode:

Mode	CE	CS
Active (RX /TX)	1	0
Configuration	0	1
Stand by	0	0

Table 3 TRF-2.4G main modes

TRF-2.4G has two active (RX /TX) modes:

- ShockBurst
- Direct Mode

The device functionality in these modes is decided by the content of a configuration word. This configuration word is presented in configuration section.

Absolute Maximum Ratings

Supply voltages

VCC.....- 0.3V to + 3.6V

VSS 0V

Input/Output voltages

V_I .....- 0.3V to VCC + 0.3V

V_O .....- 0.3V to VCC + 0.3V

Total Power Dissipation

P_D (T_A=85°C).....90mW

Temperatures

Operating Temperature.... - 40°C to + 85°C

Storage Temperature..... - 40°C to + 125°C

ShockBurst Mode

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

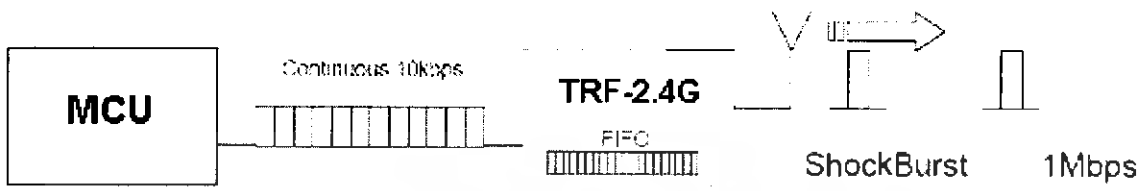


Figure 0 Clocking in data with MCU and sending with ShockBursttechnology

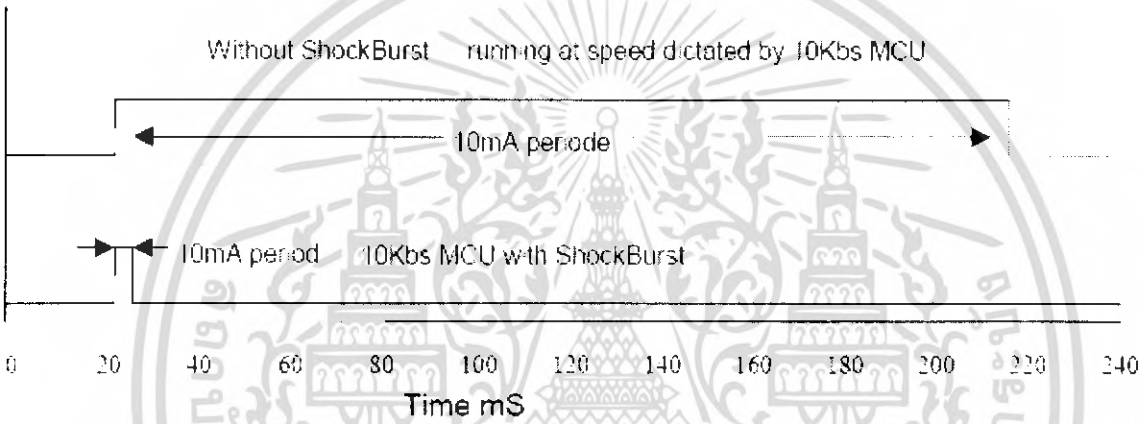


Figure 1 Current consumption with & without ShockBursttechnology

TRF-2.4G ShockBurst Transmit:

MCU interface pins: CE, CLK1, DATA

1. When the application MCU has data to send, set CE high. This activates TRF-2.4G on-board data processing.
2. The address of the receiving node (RX address) and payload data is clocked into the TRF-2.4G. The application protocol or MCU sets the speed <1Mbps (ex: 10kbps).
3. MCU sets CE low, this activates a TRF-2.4G ShockBurst transmission.
4. TRF-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).
   - TRF-2.4G return to stand-by when finished

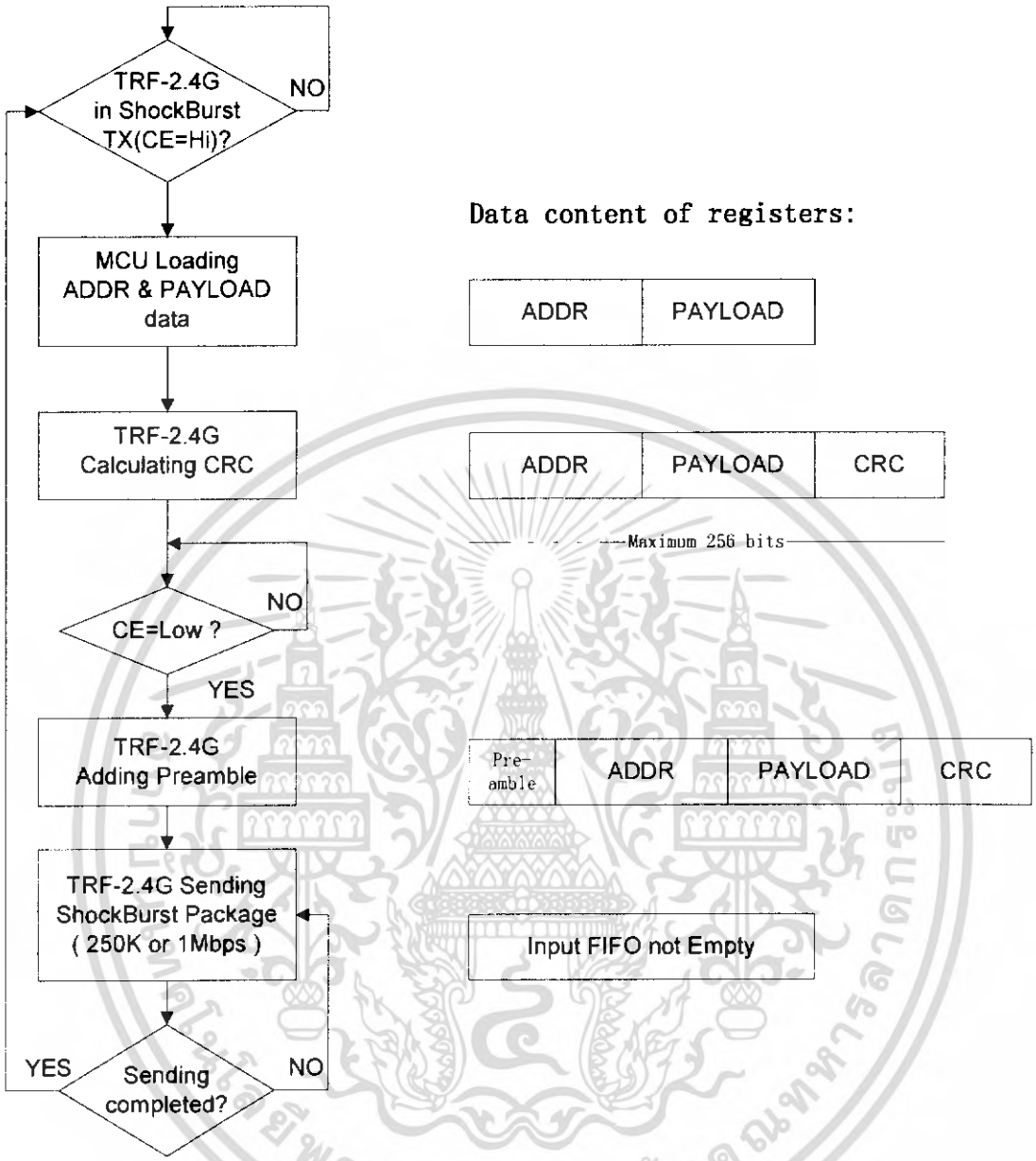


Figure 2 Flow Chart ShockBurst Transmit of TRF-2.4G

TRF-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 TRF-2.4G is configured to ShockBurst RX.
2. To activate RX, set CE high.
3. After 200 s settling, TRF-2.4G is monitoring the air for incoming communication.
4. When a valid package has been received (correct address and CRC found), TRF-2.4G removes the preamble, address and CRC bits.
5. TRF-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 TRF-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 2.

Figure 3 Flow Chart ShockBurst Receive of TRF-2.4G

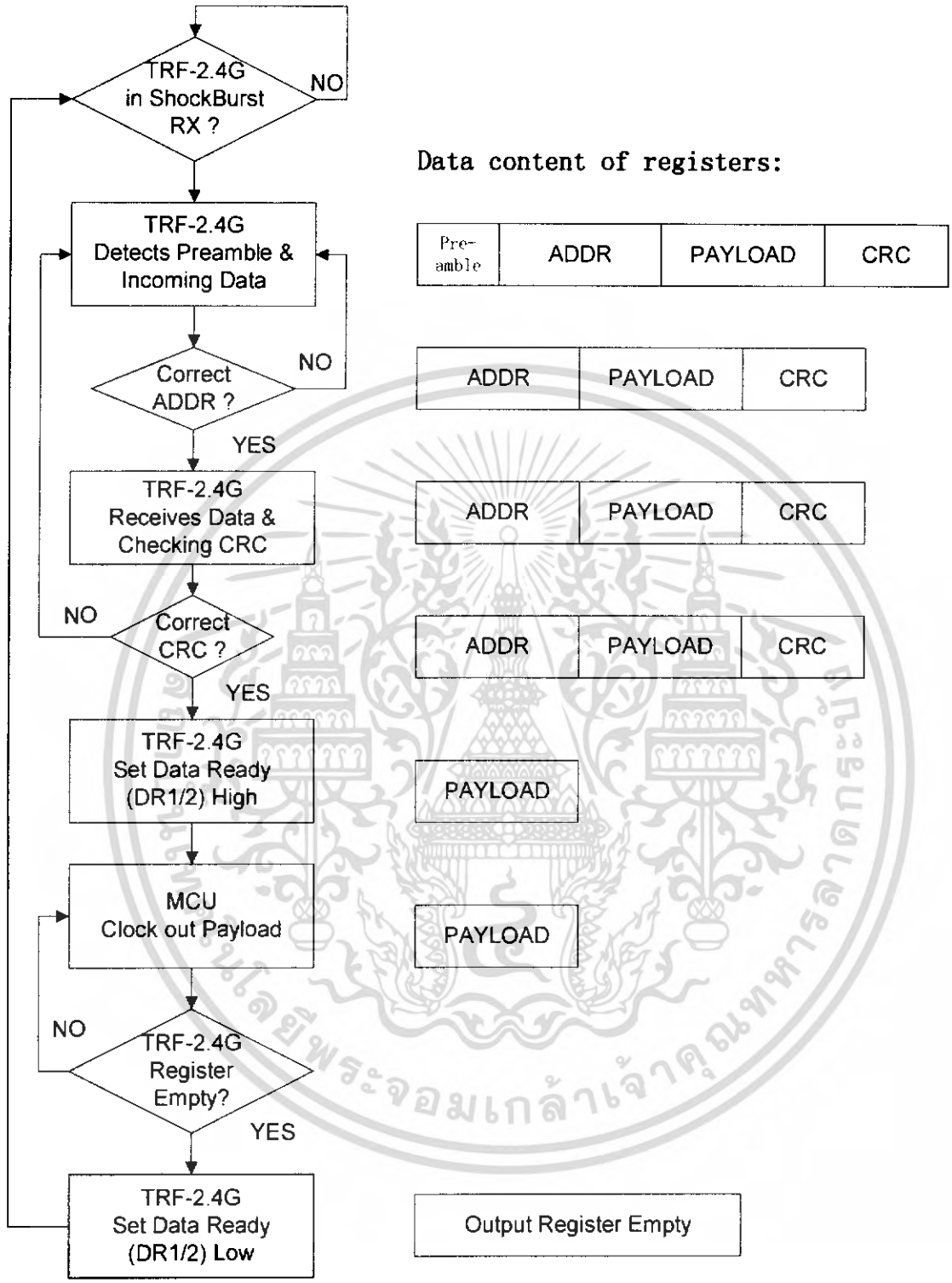


Figure 3 Flow Chart ShockBurst Receive of TRF-2.4G

TRF-2.4G Direct Mode:

In direct mode the TRF-2.4G works like a traditional RF device. Data must be at 1Mbps, or 250kbps at low data rate setting, for the receiver to detect the signals.

Direct Mode Transmit:

MCU interface pins: CE, DATA

1. When application MCU has data to send, set CE high
2. The TRF-2.4G RF front end is now immediately activated, and after 200 seconds settling time, data will modulate the carrier directly.
3. All RF protocol parts must hence be implemented in MCU firmware (preamble, address and CRC).

Direct Mode Receive:

MCU interface pins: CE, CLK1, and DATA

1. Once the TRF-2.4G is configured and powered up (CE high) in direct RX mode, DATA will start to toggle due to noise present on the air.
2. CLK1 will also start to toggle as TRF-2.4G is trying to lock on to the incoming data stream.
3. Once a valid preamble arrives, CLK1 and DATA will lock on to the incoming signal and the RF package will appear at the DATA pin with the same speed as it is transmitted.
4. To enable the demodulator to re-generate the clock, the preamble must be 8 bits toggling hi-low, starting with low if the first data bit low.
5. In this mode no data ready (DR) signals is available. Address and checksum verification must also be done in the receiving MC.

DuoCeiver Simultaneous Two Channel Receive Mode

In both ShockBurst & Direct modes the TRF-2.4G can facilitate simultaneous reception of two parallel independent frequency channels at the maximum data rate.

This means:

- TRF-2.4G can receive data from two 1 Mbps transmitters, 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 DuoCeiver technology provides 2 separate dedicated data channels for RX and replaces the need for two, stand alone receiver systems.

Laipac Technology Inc. www.laipac.com Phone +1-905-7621228 Fax +1-905-7631737

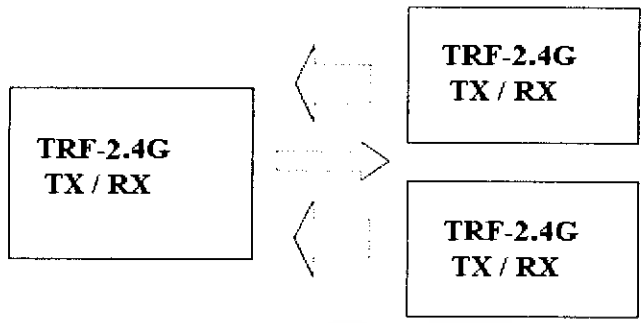
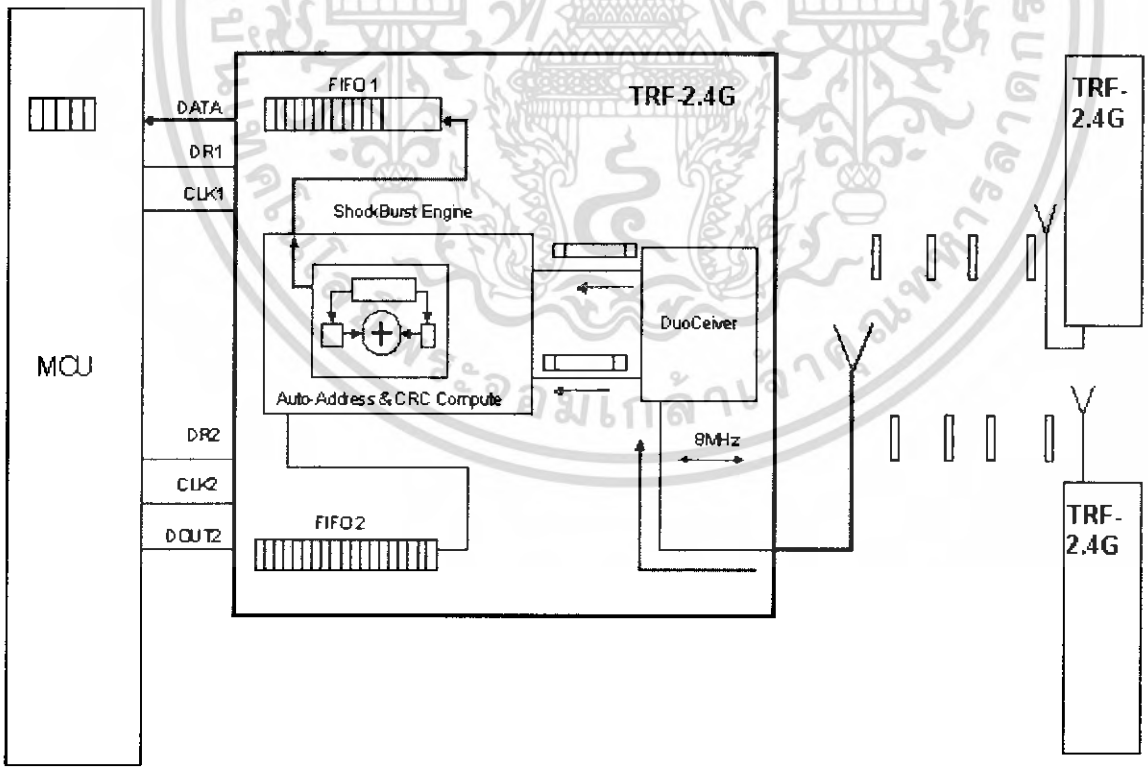


Figure 4 Simultaneous 2 channel receive on TRF-2.4G

There is one absolute requirement for using the second data channel. For the TRF-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 TRF-2.4G must be programmed to receive at the frequency of data channel 1. No time ultiplexing is used in TRF-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.



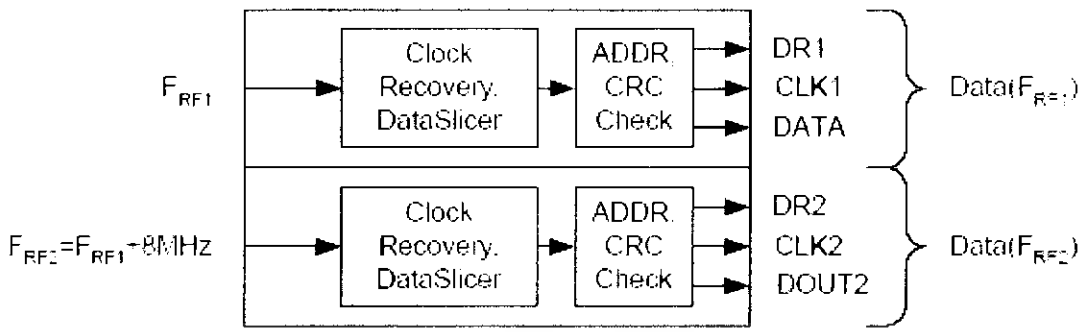


Figure 5 DuoCeiver with two simultaneously independent receive channels.

Configuration Mode

In configuration mode a configuration word of up to 15 bytes is downloaded to TRF-2.4G. This is done through a simple 3-wire interface (CS, CLK1 and DATA). For more information on configuration please refer to the TRF-2.4G Device configuration chapter on next 2nd page.

Stand-By Mode

Stand by mode is used to minimize average current consumption while aintaining short start up times. In this mode, part of the crystal oscillator is active. Current consumption is dependent on crystal frequency (Ex: 12uA @ 4 MHz, 32uA@ 16MHz). The configuration word content is maintained during stand by.

Power Down Mode

In power down the TRF-2.4G is disabled with minimal current consumption, typically less than 1 A. Entering this mode when the device is not active minimizes average current consumption, maximizing battery lifetime. The configuration word content is maintained during power down.

DEVICE CONFIGURATION

All configuration of the TRF-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 use and up to 2 bytes long for direct mode.

Configuration for ShockBurst operation

The configuration word in ShockBurst enables the TRF-2.4G to handle the RF protocol. Once the protocol is completed and loaded into TRF-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 TRF-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 TRF-2.4G to distinguish between address and payload data.
- Address (RX Channel 1 and 2): Destination address for received data.
- CRC: Enables TRF-2.4G on-chip CRC generation and de-coding.

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

PRE-AMBLE	ADDRESS	PAYLOAD	CRC

Figure 6 Data packet set-up

Configuration for Direct Mode operation

For direct mode operation only the two first bytes (bit[15:0]) of the configuring word are relevant.

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 1
	111:104	8	DATA1_W	Length of data payload section RX channel 1
	103:64	40	ADDR2	Up to 5 bytes address for channel 2
	63:24	40	ADDR1	Up to 5 bytes address for channel 1
	23:18	6	ADDR_W	Number of address bits(both RX channels)
	17	1	CRC_L	8 or 16 bits 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 (Factory default 16MHz crystal mounted)
	9:8	2	RF_PWR	RF output power
	7:1	7	RF_CH#	Frequency channel
	0	1	RXEN	RX or TX operation

Table 4 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 TRF-2.4G updates the number of bits actually shifted in during the last configuration.
Ex:
If the TRF-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 VCC 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 TRF-2.4G.
General Device Configuration: bit[15:0]
ShockBurst Configuration: bit[119:0]
Test Configuration: bit[143:120]

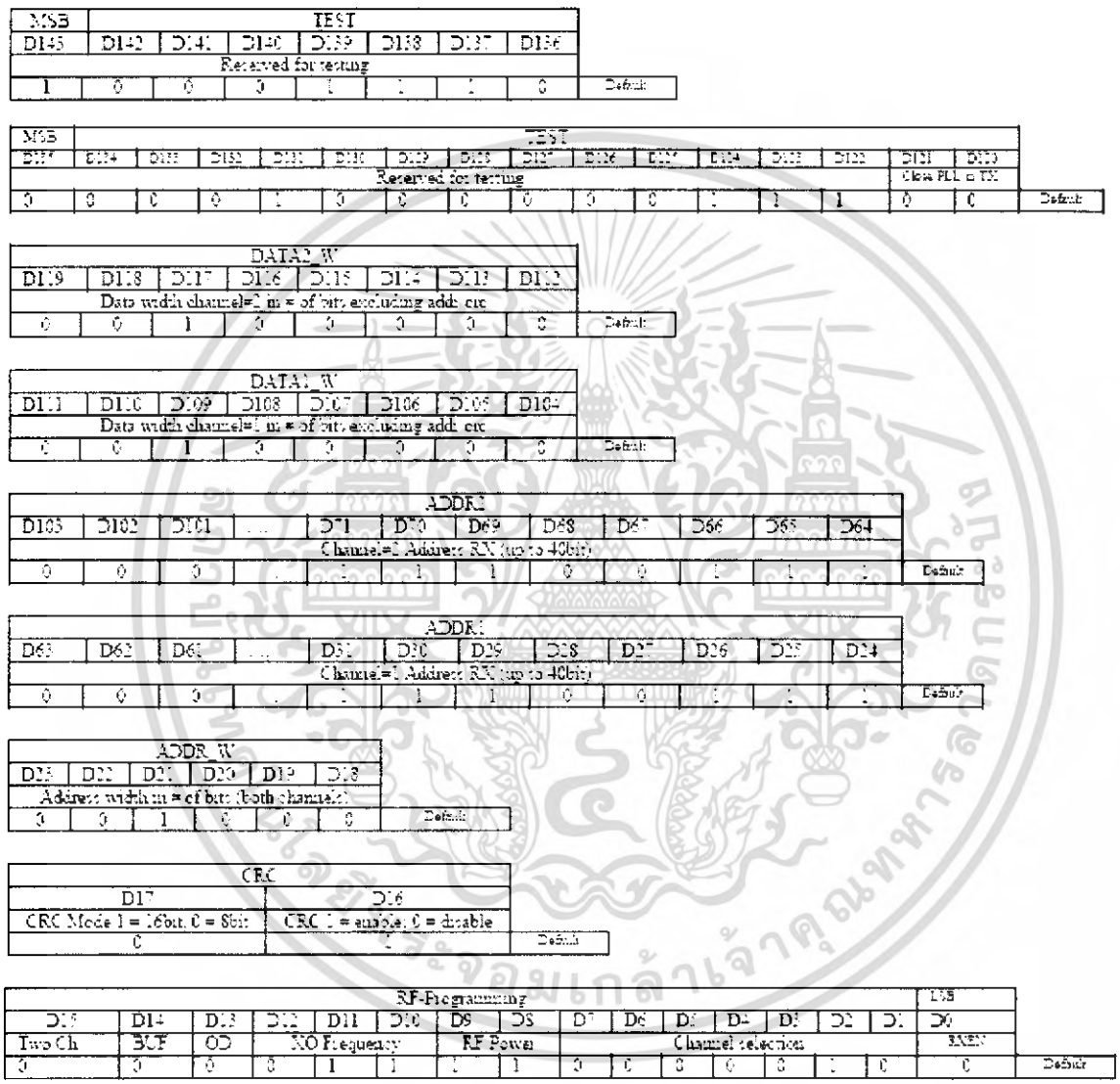


Table 5 Configuration data word

The MSB bit should be loaded first into the configuration register.

ShochBurst configuation:

The section B[119:16] contains the segments of the configuration register dedicated to ShockBurst operational protocol. After VCC is turned on ShockBurst configuration is done once and remains set whilst VCC is present. During

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 8 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 TRF-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 9 RF operational settings.

Bit 15:

RX2_EN:

Logic 0: One channel receive

Logic 1: Two channels receive

NOTE:

In two channels receive, the TRF-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 0: TRF-2.4G operates in direct mode.

Logic 1: TRF-2.4G operates in ShockBurst mode

Bit 13:

RF Data Rate:

Logic 0: 250 kbps

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:

XO_F: Selects the TRF-2.4G crystal frequency to be used:

XO FREQUENCY SELECTION			
D12	D11	D10	Crystal Frequency (MHz)
0	1	1	16
Factory default: 16MHz Crystal is used inside module			

Table 10 Crystal frequency setting.

Bit 9-8:

RF_PWR: Sets TRF-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 11 RF output power setting.

RF channel & direction

RF_CH#							RXEN
7	6	5	4	3	2	1	0

Table 12 Frequency channel + RX / TX setting.

Bit 7 – 1:

RF_CH#: Sets the frequency channel the TRF-2.4G operates on.

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

$Channel_{RF} = 2400MHz + RF\_CH\# * 1.0MHz$

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

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

$Channel_{RF} = 2400MHz + RF\_CH\# * 1.0MHz$ *(Reiceive 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:

$Channel_{RF} = 2400MHz + RF\_CH\# * 1.0MHz + 8MHz$ *(Reiceive 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

operation only the first byte for frequency channel and RX/TX switching need to be changed.

DATAx_W

DATA2_W							
119	118	117	116	115	114	113	112

DATA1_W							
111	110	109	108	107	106	105	104

Table 6 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 7 Address of receiver #2 and receiver #1.

Bit 103 – 64:

ADDR2: Receiver address channel 2, up to 40 bit.

Bit 63 – 24: ADDR1

DATA PACKAGE DESCRIPTION

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 and Direct modes • Preamble is 8 (or 4) bits in length and is dependent of the first data bit in direct mode. PREAMBLE 1st ADDR-BIT 01010101 0 10101010 1 • Preamble is automatically added to the data packet in ShockBurst and thereby gives extra space for payload. • In ShockBurst mode the preamble is stripped from the received output data, in direct mode the preamble is transparent to the output data.
2 ADDRESS	• The address field is required in ShockBurst mode. • 8 to 40 bits length. • Address automatically removed from received packet in ShockBurst mode. In Direct mode MCU must handle address.
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). • In Direct mode the payload size is defined by 1Mbps for 4ms: 4000 bits minus the following: (Preamble: 8 (or 4) bits. + Address: 8 to 40 bits. + CRC: 0, 8 or 16 bits).
4 CRC	• The CRC is optional in ShockBurst mode, and is not used in Direct mode. • 8 or 16 bits length • The CRC is stripped from the received output data.

Table 13 Data package description

IMPORTANT TIMING DATA

The following timing applies for operation of TRF-2.4G.

TRF-2.4G Timing Information

TRF-2.4G timing	Max.	Min.	Name
PWR_DWN => ST_BY mode	3ms		Tpd2sby
PWR_DWN =>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 14 Switching times for TRF-2.4G

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

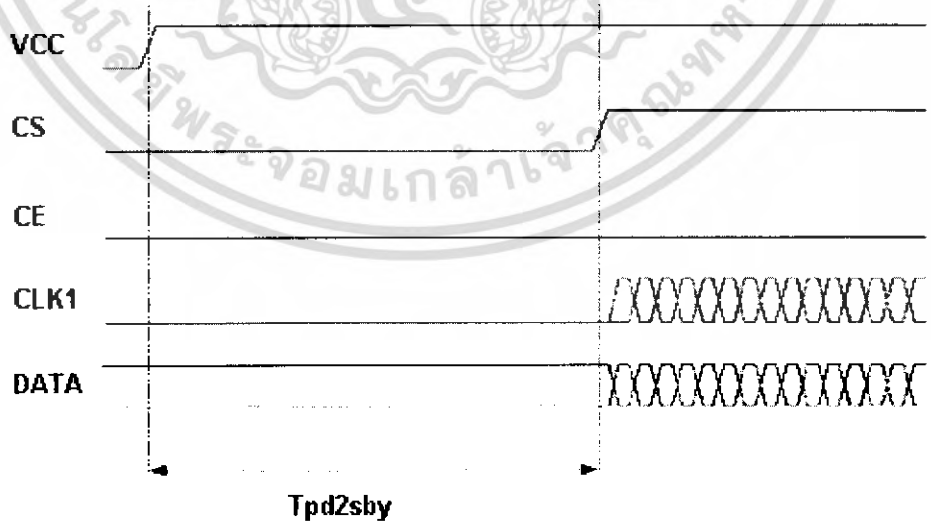


Figure 8 Timing diagram for power down (or VCC off) to stand by mode

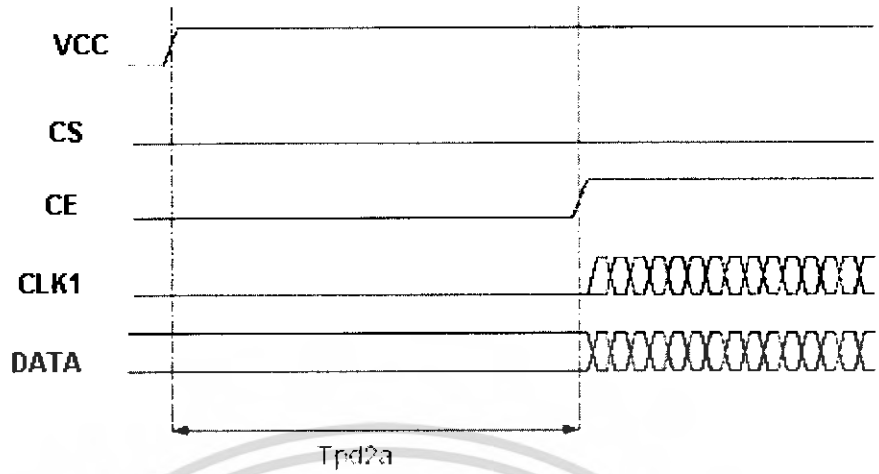


Figure 9 Power down (or VCC off) to active mode

Note that the configuration word will be lost when VCC 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.

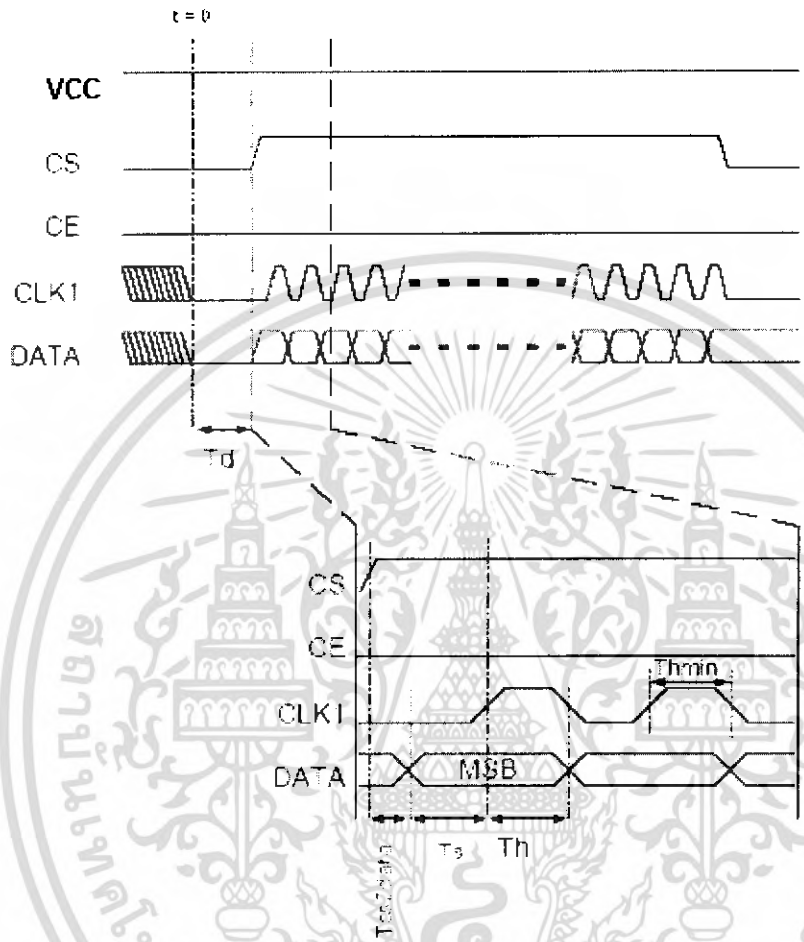


Figure 10 Timing diagram for configuration of TRF-2.4G

If configuration mode is entered from power down, CS can be set high after Tpd2sby as shown in Figure 10.

ShockBurst mode timing

ShockBurst TX:

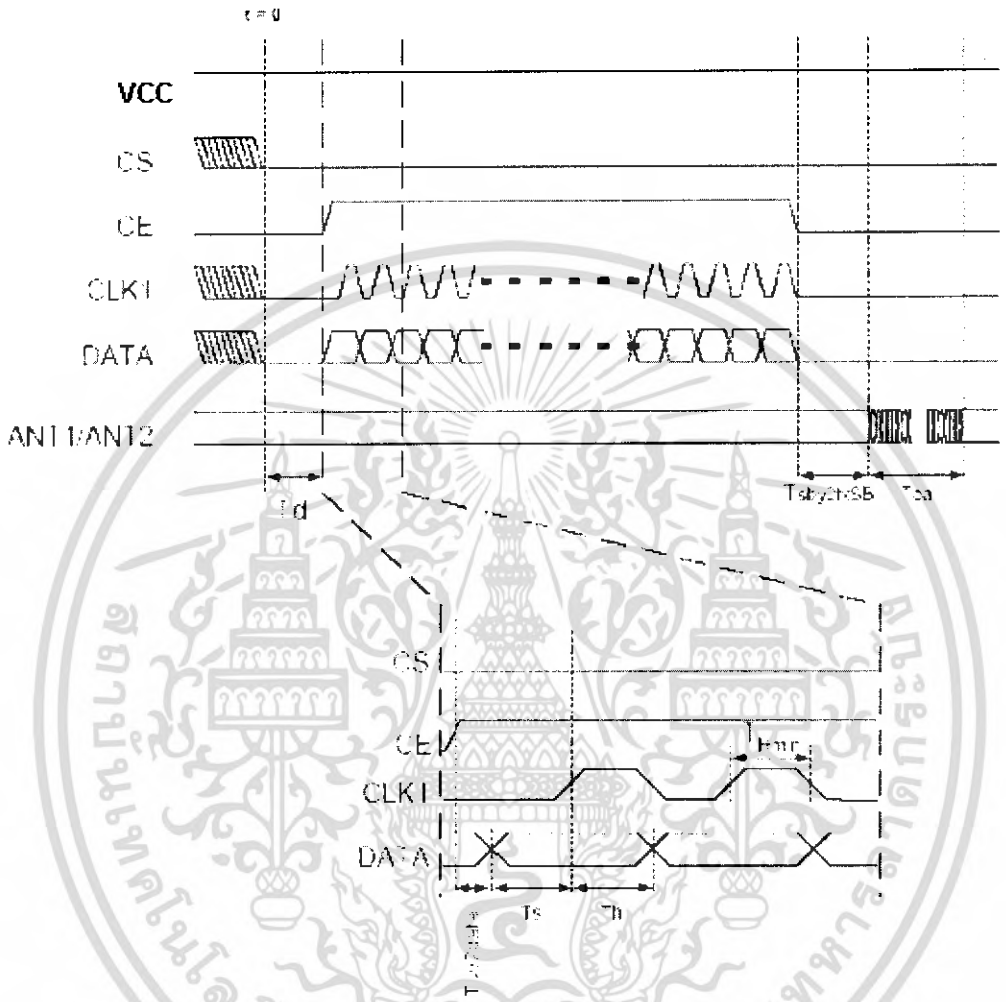


Figure 11 Timing of ShockBurst in TX

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

$$T_{OA} = 1/datarate * (\#databits + 1)$$

ShockBurst RX:

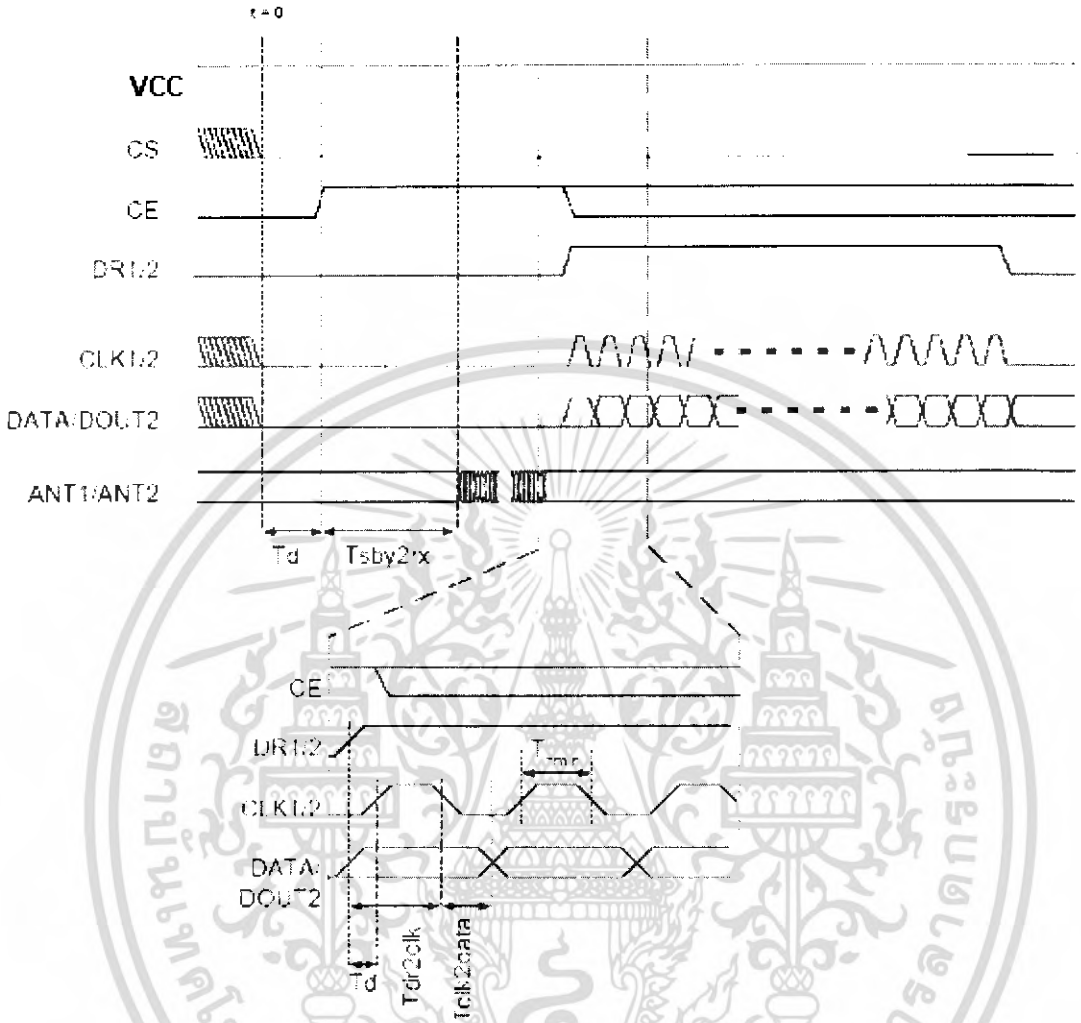


Figure 12 Timing of ShockBurst in RX

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 (200 s) after the DR1 goes low.

Direct Mode

Direct Mode TX

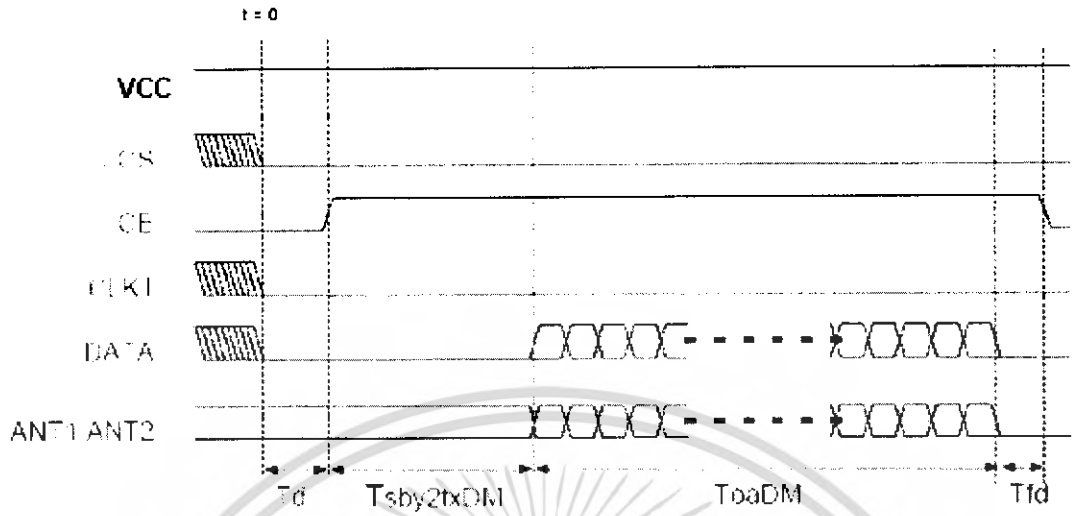


Figure 13 Timing of direct mode TX

In TX direct mode the input data will be sampled by TRF-2.4G and therefore no clock is needed. The clock must be stable at low level during transmission due to noise considerations. The exact delay Tsby2txDM is given by the equation:

$$T_{sby2txDM} = 194uS + 1/F_{XO} * 14 + 2.25uS$$

Direct Mode RX

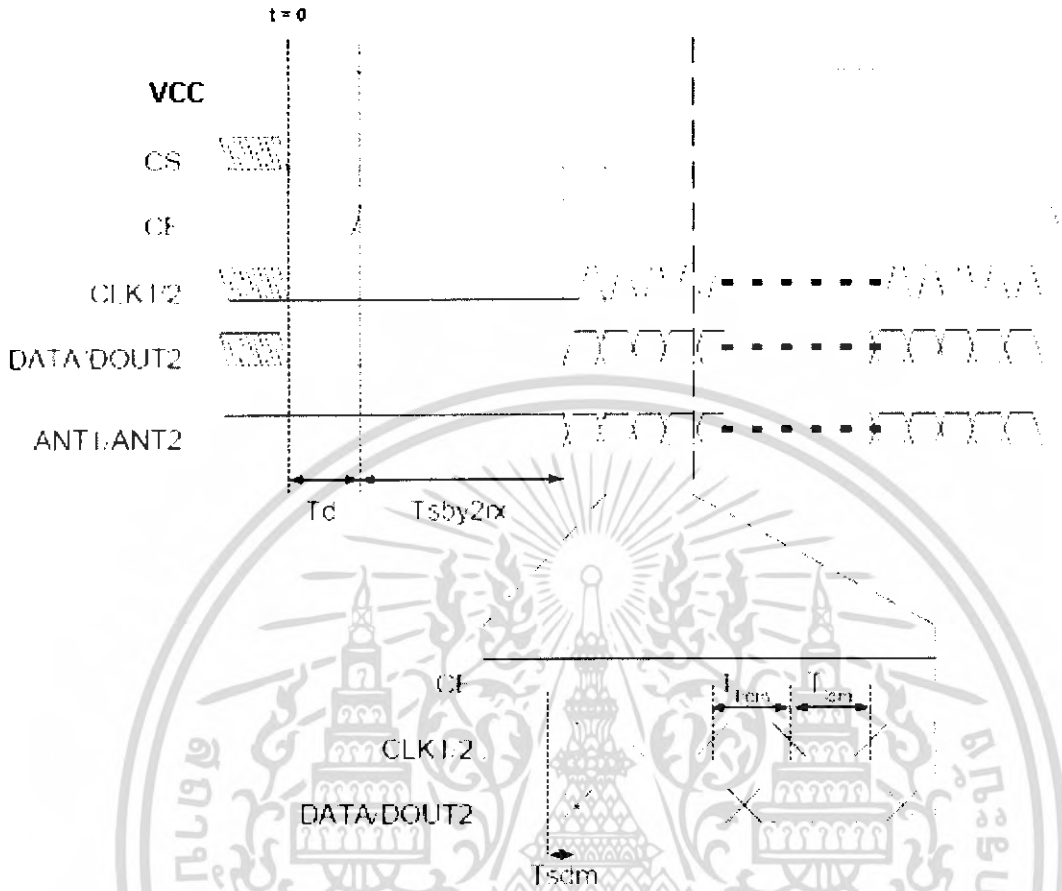


Figure 14 Timing of direct mode RX

Tsby2rx describes the delay from the positive edge of CE to the start detection of (demodulated) incoming data.

PERIPHERAL RFINFORMATION

Antenna output

The ANT1 & ANT2 output pins provide a balanced RF output to the antenna. The pins must have a DC path to VCC, either via a RF choke or via the center point in a dipole antenna. The load impedance seen between the ANT1/ANT2 outputs should be in the range 200-700 . A de-embedded load impedance i.e. impedance seen at drain terminals of the output transistors of 400 . is recommended for maximum output power (0dBm). Lower load impedance (for instance 50) can be obtained by fitting a simple matching network.

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: VCC = 3.0V, VSS = 0V, TA = 27°C, Load impedance = 400 .

Table 15 RF output power setting for the TRF-2.4G.

Configuration Word Example

1 Channel, Freq.: 2410MHz, 1Mbps and Transmit mode:

Bit143	Bit142	Bit41	Bit140	Bit139	Bit138	Bit137	Bit136
1	0	0	0	1	1	1	0
Bit135	Bit134	Bit133	Bit132	Bit131	Bit130	Bit129	Bit128
0	0	0	0	1	0	0	0
Bit127	Bit126	Bit125	Bit124	Bit123	Bit122	Bit121	Bit120
0	0	0	1	1	1	0	0
Bit119	Bit118	Bit117	Bit116	Bit115	Bit114	Bit113	Bit112
1	1	0	0	1	0	0	0
Bit111	Bit110	Bit109	Bit108	Bit107	Bit106	Bit105	Bit104
1	1	0	0	1	0	0	0
Bit103	Bit102	Bit101	Bit100	Bit99	Bit98	Bit97	Bit96
1	1	0	0	0	0	0	0
Bit95	Bit94	Bit93	Bit92	Bit91	Bit90	Bit89	Bit88
1	0	1	0	1	0	1	0
Bit87	Bit86	Bit85	Bit84	Bit83	Bit82	Bit81	Bit80
0	1	0	1	0	1	0	1
Bit79	Bit78	Bit77	Bit76	Bit75	Bit74	Bit73	Bit72
1	0	1	0	1	0	1	0
Bit71	Bit70	Bit69	Bit68	Bit67	Bit66	Bit65	Bit64
0	1	0	1	0	1	0	1
Bit63	Bit62	Bit61	Bit60	Bit59	Bit58	Bit57	Bit56
1	0	1	0	1	0	1	0
Bit55	Bit54	Bit53	Bit52	Bit51	Bit50	Bit49	Bit48
0	1	0	1	0	1	0	1
Bit47	Bit46	Bit45	Bit44	Bit43	Bit42	Bit41	Bit40
1	0	1	0	1	0	1	0
Bit39	Bit38	Bit37	Bit36	Bit35	Bit34	Bit33	Bit32
0	1	0	1	0	1	0	1
Bit31	Bit30	Bit29	Bit28	Bit27	Bit26	Bit25	Bit24
1	0	1	0	1	0	1	0
Bit23	Bit22	Bit21	Bit20	Bit19	Bit18	Bit17	Bit16
1	0	1	0	0	0	1	1
Bit15	Bit4	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
0	**1**	**1**	0	1	1	1	1
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	1	0	1	0	**0**

Table 16 Configuration Example

Laipac Technology Inc.

Headquarter

Laipac Technology, Inc.
55 West Beaver Creek Rd., Unit 1
Richmond Hill, Ontario.
L4B 1K5 - Canada
Tel: 905-762-1228 Fax: 905-763-1737

Europe Office

Laipac Tech Europe SL.
Güell 58 (Edificio CINC)
(17001) Gerona - Spain
Tel:+34 972 940 947 Fax: +34 972 940 948

For European Customers, click here

Africa, Middle East & Other countries, click here.

For Sales : sales@laipac.com
For Customer Service: cs@laipac.com
For General Info : info@laipac.com
For Technical : tech@laipac.com
For Human Resources : diego@laipac.com