เมนู อิเล็กทรอนิกส์

ELECTRONIC MENU

โดย

นาย สมโภช ทิพย์อักษร

นาย วีระพล พินเผือก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาตรบัณฑิต

สาขา วิชาอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เมนู อิเล็กทรอนิกส์

ELECTRONIC MENU

โดย

นาย สมโภช ทิพย์อักษร 43015283

นาย วีระพล พินเผือก 43015279

อาจารย์ที่ปรึกษา

ดร. กิติพล ชิตสกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาตรบัณฑิต

สาขา วิชาอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ ปีการศึกษา 2545

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เมนู อิเล็กทรอนิกส์

ผู้จัดทำ

1. นาย สมโภช ทิพย์อักษร 43015283
2. นาย วีระพล พินเผือก 43515279

..... อาจารย์ที่ปรึกษา

(ดร. กิติพล ชิตสกุล)

เมนู อิเล็กทรอนิกส์

ELECTRONIC MENU

นาย สมโภช ทิพย์อักษร 43015283

นาย วีระพล พินเผือก 43515279

โครงงานนี้ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

.....

(ดร.กิติพล ชิตสกุล)

อาจารย์ที่ปรึกษา

เมนูอิเล็กทรอนิกส์

นายสมโภช ทิพย์อักษร 43015283
นายวีระพล พินเผือก 43015279
ดร.กิติพล ชิตสกุล อาจารย์ที่ปรึกษา
ปีการศึกษา 2545

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้กล่าวถึงการนำเอาเครื่องเกมบอย(Gameboy®)มาใช้ติดต่อสื่อสารกับคอมพิวเตอร์ (PC) โดยนำมาประยุกต์ใช้งานเป็นเมนูใช้ในการสั่งอาหาร เกมบอยทำหน้าที่เป็นฐานข้อมูลอาหารและส่งข้อมูลให้กับคอมพิวเตอร์ โดยผู้ใช้งานสามารถที่จะเลือกเมนูที่ต้องการได้ผ่านทางจอแสดงผลแบบผลึกเหลว(LCD)ของเครื่องเกมบอย เมื่อเลือกเมนูแล้วเกมบอยจะส่งข้อมูลให้คอมพิวเตอร์ได้รับรู้ว่ามีการสั่งอาหารเข้ามา ผ่านทางวงจรที่สร้างขึ้นเพื่ออินเตอร์เฟสโดยการรับสัญญาณจากตัวเกมบอยแล้วส่งให้กับคอมพิวเตอร์ตามความเร็วที่กำหนดไว้

ELECTRONIC MENU

Sompoch Tipagsorn 43015283

Veelapon Pinpuck 43015279

Dr. Kitiphol Chitsakul (Advisor)

Academic year 2002

ABSTRACT

This thesis presents an electronic menu used for food ordering in a restaurant. A Gameboy® is used as food data based shown on its LCD monitor. When a user has selected a menu, the Gameboy® send data to a microcomputer (PC) at a setup baud rate via a special circuit designed for interfacing between the Gameboy® and the PC.

กิตติกรรมประกาศ

ในการจัดทำโครงงานครั้งนี้จะไม่ประสบความสำเร็จเลย ถ้าไม่มีท่านอาจารย์ที่ปรึกษา ดร.กิติพล ชิตสกุล ที่คอยให้คำแนะนำแนวคิดในการทำงานแต่ละขั้นตอน รวมทั้งการแก้ไขปัญหาต่างๆ และผลักดันให้โครงงานนี้สำเร็จลุล่วงมาได้ด้วยดี ขอกราบขอบพระคุณบิดา มารดา ที่ให้กำลังใจและโอกาสทางการศึกษาและขอบคุณเพื่อนทุกท่าน ที่ได้คอยช่วยเหลือให้คำแนะนำ ให้ยืมเครื่องมืออุปกรณ์ต่างๆ ในการทำโครงงานนี้

ลงชื่อ .....

(นาย สมโภช ทิพย์อักษร)

ลงชื่อ .....

(นาย วีระพล พินเผือก)

สารบัญ

สารบัญตาราง

สารบัญรูป

บทที่1
บทนำ

1.1 ความเป็นมาของโครงงาน

โครงงานนี้เป็นการนำเอาเครื่องเกมบอยมาประยุกต์ใช้งานเป็นเมนูอิเล็กทรอนิกส์สำหรับใช้ในการสั่งอาหาร เพื่อเป็นอีกทางลือกหนึ่งในการนำเอาอุปกรณ์อิเล็กทรอนิกส์มาใช้งานแทนมนุษย์ (บริกร) ซึ่งเหตุผลของการนำเอาเครื่องเกมบอยมาใช้งานนั้นเนื่องจาก ตัวเครื่องเกมบอยนั้นมีส่วนของ CPU, LCD, RAM และ keypad และอินพุตและเอาต์พุตพอร์ตอย่างครบถ้วน โดยพบว่าหากจะสร้างระบบขึ้นมาเองจากอุปกรณ์เหล่านี้แล้ว จะมีขนาดที่ค่อนข้างใหญ่ไม่สะดวกในการพกพา อีกทั้งต้นทุนโดยรวมที่ค่อนข้างสูง เมื่อเทียบกับตัวของเครื่องเกมบอย

1.2 ลักษณะของโครงงาน

โครงงานนี้เป็นการนำเอาเครื่องเกมบอยมาประยุกต์ใช้งานเป็นเมนูอิเล็กทรอนิกส์ สามารถส่งข้อมูลให้กับไมโครคอมพิวเตอร์เพื่อประมวลและแสดงหรือพิมพ์รายการอาหาร โดยเริ่มจากการเขียนโปรแกรมสร้างฐานข้อมูลอาหารตัวเกมบอยและควบคุมการแสดงรายการและเลือกเมนูอาหารบนจอ LCD ของเครื่องเกมบอย เมื่อมีการเลือกเมนูเกิดขึ้น โปรแกรมควบคุมจะส่งข้อมูลออกมาทางพอร์ตอนุกรมเพื่อส่งสัญญาณอินฟราเรดไปยังเครื่องคอมพิวเตอร์ โดยจะทำการรับข้อมูลจากเครื่องเกมบอยแล้วทำการส่งข้อมูลไปที่ตัวไมโครคอนโทรเลอร์แล้วผ่านไปยังวงจรส่งสัญญาณอินฟราเรดไปยังบอร์ดรับเพื่อส่งไปยังพอร์ตอนุกรมของเครื่องคอมพิวเตอร์ ตามอัตราเร็วที่กำหนด เมื่อคอมพิวเตอร์ได้รับข้อมูลแล้วก็จะแสดงผลข้อมูลออกทางหน้าจอคอมพิวเตอร์

1.3 โครงสร้างของปริญญานิพนธ์

ปริญญานิพนธ์นี้ได้รวบรวมรายละเอียด ความเป็นมาแนวคิด การสร้างและทดสอบโครงงาน โดยได้มีการแบ่งรายงานออกเป็นบทต่างๆ ทั้งนี้เพื่อความสะดวกต่อการศึกษาและทำความเข้าใจ ในแต่ละบทประกอบด้วยเนื้อหาสำคัญดังนี้

บทที่1 บทนำเกี่ยวกับความเป็นมาของโครงงานและลักษณะของโครงงาน

บทที่2 เป็นรายละเอียดเกี่ยวกับ ทฤษฎีและหลักการพื้นฐานของเครื่องเกมบอย วงจรและการส่งสัญญาณอินฟราเรด และทฤษฎีเบื้องต้นของการส่งข้อมูลแบบอนุกรม

บทที่3 จะเป็นการแสดงวงจรที่ใช้ในโครงงานนี้ซึ่งประกอบด้วย วงจรที่ใช้เป็น cartridge card ของเกมบอย วงจรส่งสัญญาณอินฟราเรด และวงจรที่ใช้เป็นตัวรับข้อมูลแบบอนุกรม

บทที่4 จะเป็นการกล่าวถึงการพัฒนาซอฟต์แวร์ที่ใช้ในโครงงานและการใช้งาน

บทที่5 จะเป็นผลการทดลองวัดสัญญาณเอาท์พุท เช่นเอาท์พุตที่ออกจาก พอร์ต อนุกรม ของเครื่องเกมบอยและ เอาท์พุทที่ออกจากวงจรส่งสัญญาณอินฟราเรด

บทที่6 เป็นบทสรุป

บทที่ 2
ทฤษฎีและหลักการพื้นฐาน

2.1 ข้อมูลทางด้านฮาร์ดแวร์เกี่ยวกับตัวเกมส์บอย

2.1.1 รายละเอียดโดยทั่วไปของตัวเกมส์บอย

: CPU: 8-bit (Similar to the Z80 processsor)

: Main Ram: 8Kbyte

: Video Ram: 8Kbyte

: Screen Size 2.6”

: Resolution: 160*144 (20*18 tiles)

: Max # of sprites : 40

: Max # sprites/line : 10

: Max sprites size : 8*16

: Min sprites size : 8*8

: Clock Speed: 4.194304 Mhz

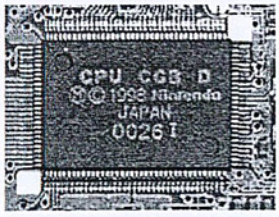
: Horiz Sync : 9198 Hz

: Vert Sync : 59.73 Hz

: power : DC3V 0.7W

2.1.2 หน่วยประมวลผลของเกมส์บอย

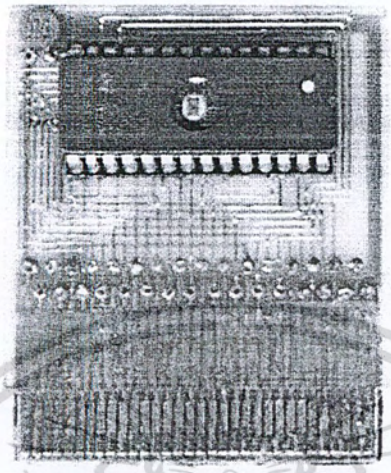
เกมส์บอยใช้หน่วยประมวลผลหรือ CPU ที่มีลักษณะคล้ายกับ CPU ของ zilog z80 ซึ่งโดยโครงสร้างทั่วไปนั้นมีลักษณะที่คล้ายกันแต่คำสั่งที่ใช้บางคำสั่งจะมีรูปแบบที่แตกต่างกันเช่น คำสั่ง LD HL,(word) ของ Z80 จะเป็นคำสั่ง LD A,(HL) ของเกมส์บอย ซึ่งคำสั่งที่แตกต่างกันทั้งหมดนั้นสามารถดูได้จากภาคผนวก



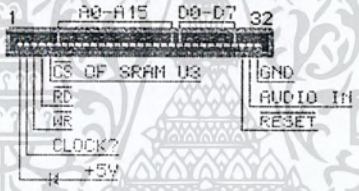
รูปที่2.1 แสดงรูปของCPUที่อยู่ภายในเกมส์บอย

2.1.3 Cartridge Type

ประเภทของ Cartridge จะถูกกำหนดโดย byte ที่ตำแหน่ง 0147



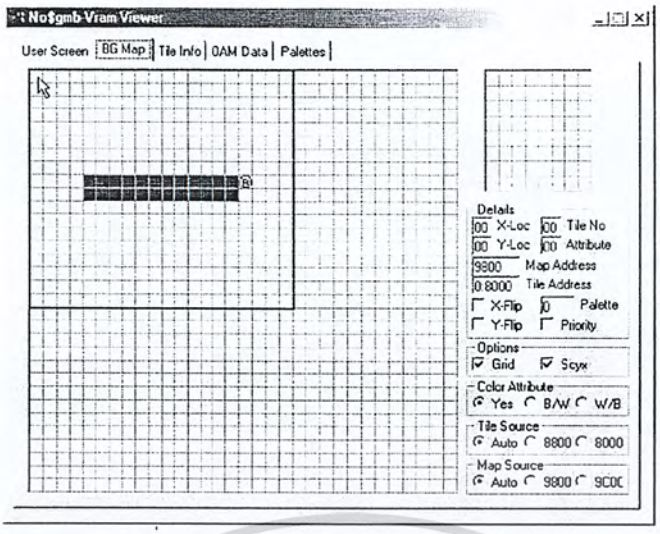
รูปที่2.2 แสดงตัวอย่างของ Cartridge card



รูปที่2.3 แสดง Connector สำหรับใส่ Cartridge บริเวณด้านหลังเกมส์บอย

2.1.4 ส่วนแสดงผล

ในส่วนของการแสดงผลหลักของเกมส์บอยจะมีความละเอียดอยู่ที่ 256*256 pixels หรือ 32*32 tiles (8*8 pixel ในแต่ละ tiles) แต่จะมีอยู่แค่ 160*144 pixel เท่านั้นที่สามารถแสดงผลได้ โดย Registers SCROLLx และ SCOLLy จะใช้ในการแสดงผลพื้นที่ในส่วนของ Background ร่วมกันโดยเริ่มที่มุมซ้ายบนสุดของหน้าจอซึ่ง Background จะครอบคลุมพื้นที่ทั้งหมดของส่วนแสดงผล พื้นที่ของ VRAM ซึ่งก็คือ Background จะบรรจุค่าของ Tile Table ซึ่งจะใช้ในส่วนของการแสดงผลจะมีอยู่ด้วยกัน 32 แถว



รูปที่2.4 แสดงพื้นที่ในส่วนแสดงผลของเกมส์บอยจาก GB Emulator

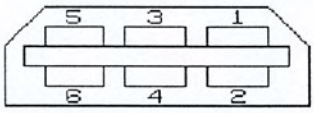
2.1.5 Sprites

Sprites ในเกมส์บอยคือส่วนที่เคลื่อนไหวได้อย่างอิสระซึ่งการเคลื่อนไหวของ Sprites นั้นอาจจะมาจากการโปรแกรมให้ตัวมันเคลื่อนที่เองหรือบังคับจากjoypadของเกมส์บอยก็ได้ ซึ่งในเกมส์บอยนั้นสามารถที่จะมีSpritesได้มากที่สุดคือ40Spritesซึ่ง scale ของ Sprites นั้นอาจเป็น 8*8 pixel หรือ 8*16 pixel ก็ได้ซึ่งpattern ของSpritesตัวนั้นอาจจะสร้างมาจากโปรแกรม tiles editor ก็ได้นั่นหมายความว่าหน้าตาของSpritesและ Tiles Background ถูกสร้างมาจากโปรแกรมเดียวกันซึ่งขึ้นอยู่กับผู้เขียนโปรแกรมว่าจะกำหนดให้ตัวไหนเป็น Spritesและตัวไหนเป็นTiles Background และเมื่อเรากำหนดแล้วว่าตัวไหนเป็น Sprites ชุดเลขฐานสิบหกของTile ตัวนั้นก็จะมาปรากฏอยู่ที่ Sprites pattern table ซึ่งอยู่ที่ 8000-8FFF ซึ่งนอกจากจะมีตาราง Sprites pattern ซึ่งอยู่ที่ 8000-8FFF แล้วนั้นก็ยังจะมีตารางคุณสมบัติของ Sprites (Sprites attribuite table) อยู่ที่ FE00-FE9F อีกด้วย

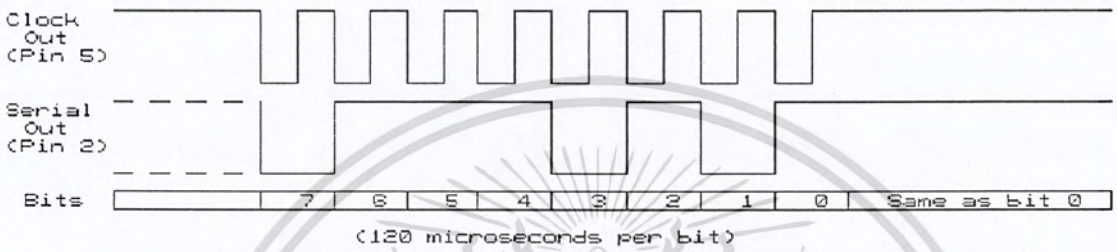
2.1.6 Serial I/O

Serial I/O port ของเกมส์บอยสามารถเปรียบเทียบได้อย่างง่ายว่ามีลักษณะคล้ายกับ port RS-232 ของ IBM PC แต่สิ่งที่ต่างกันคือมาตราฐานการรับส่งข้อมูล ความเร็วในการรับส่งข้อมูลอีกทั้งยังไม่มี start stop bit อีกด้วย ดังนั้น โปรแกรมเมอร์จะต้องสร้างขึ้นมาเองเมื่อเวลาที่ต้องใช้งาน port นี้ และในระหว่างการรับส่งข้อมูล byte จะถูกshiftออกไปในเวลาที่เท่าๆกัน กล่าวคือpluse แต่ละลูกจะมีคาบเวลาที่เท่ากัน ซึ่งอัตราการรับส่งข้อมูลจะถูกกำหนดโดย Clock source ซึ่งสามารถเลือกได้ว่าจะใช้ internal clock หรือ external clock ถ้าเราเลือกใช้ internal clock bit ก็จะถูกส่งออกไปที่ความถี่ 8192 Hz (122 microsecound)perbit ซึ่งบิตที่มีนัยสำคัญสูงสุดจะถูกส่งออกไปก่อน แต่ถ้าเราเลือกใช้ external clock เมื่อเริ่มรับส่งข้อมูล ขา Sout และ Sin จะรอไปเรื่อยๆจนกว่าจะมี

external clock เข้ามา เพื่อให้เกิดการ Synchonization ที่แน่นอนของแต่ละSerial port ซึ่งจากรูป ขา1 คือ +5V ขา2 คือ Serial OUT ขา3 คือ Serial IN ขา4 ยังไม่มีการใช้ในขณะนี้ ขา5 คือ Clock ขา6 คือ ground



รูปที่2.5 แสดง Serial Port ของ Gameboy



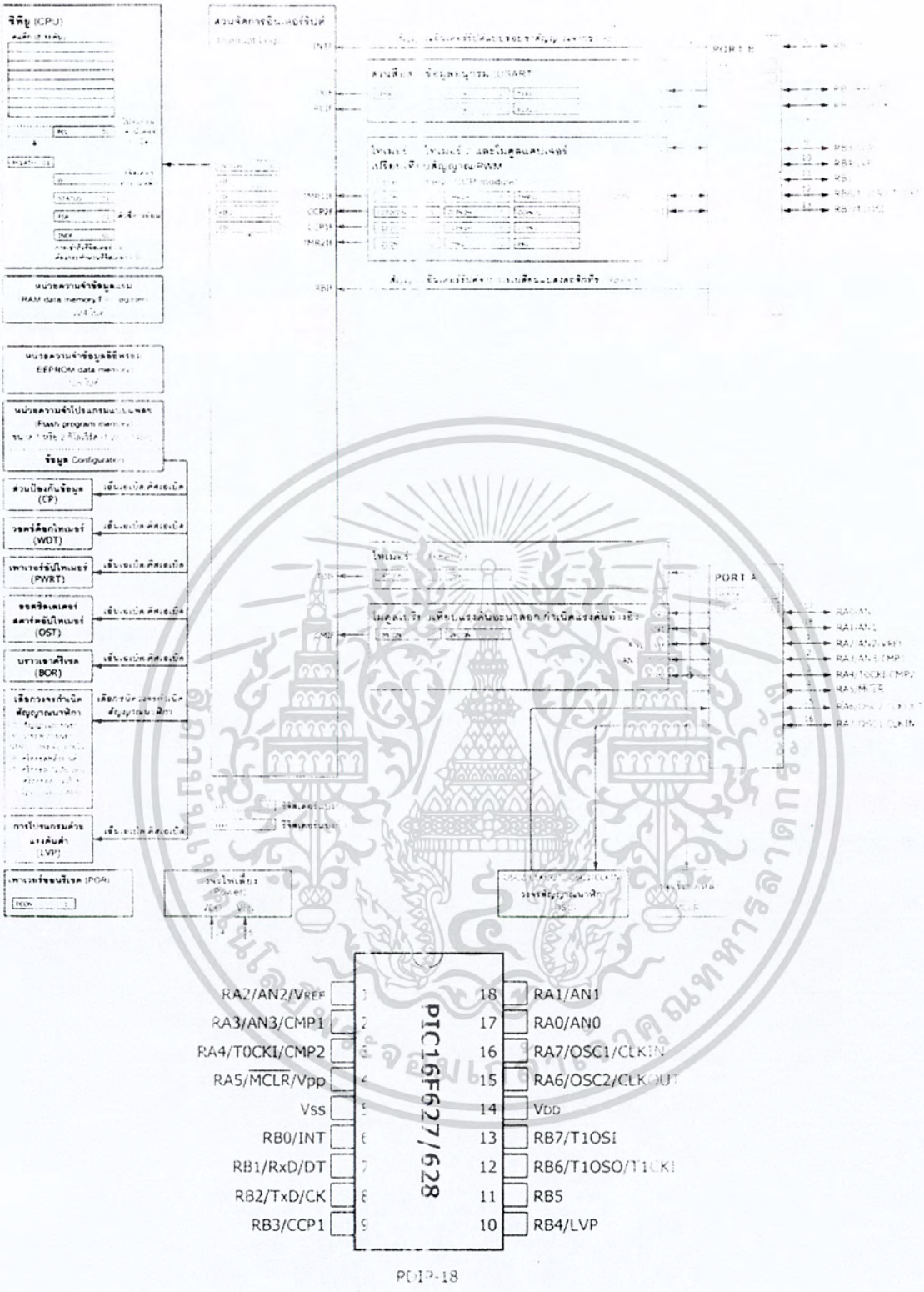
รูปที่2.6 แสดง Protocol การสื่อสารระหว่างเครื่องเกมบอย

2.2 ไมโครคอนโทรเลอร์ตระกูล PIC16F628

ไมโครคอนโทรเลอร์ตระกูล PIC16F628 นี้มีอุปกรณ์สนับสนุนประกอบอยู่ภายในหลายอย่างได้แก่ หน่วยความจำสำหรับเก็บข้อมูล หน่วยความจำสำหรับโปรแกรม ตัวตั้งเวลา/ตัวนับ อุปกรณ์รับส่งข้อมูลแบบอนุกรม เนื่องจากโครงสร้างของไมโครคอนโทรเลอร์มีอุปกรณ์สนับสนุนประกอบอยู่ภายในนี้เอง ทำให้การใช้งานง่ายขึ้นและมีประสิทธิภาพมากขึ้นโดยไม่ต้องมีการเชื่อมต่ออุปกรณ์ภายนอกเพิ่มเติมมากเหมือนกับไมโครโปรเซสเซอร์ทั่วไป นอกจากนี้หากเราต้องการใช้งานไมโครคอนโทรเลอร์ร่มกับอุปกรณ์อื่นเพิ่มเติม เช่น หน่วยความจำภายนอก เราก็ยังสามารถนำมาเชื่อมต่อเพิ่มเติมเข้ากับไมโครคอนโทรเลอร์ได้อีกด้วย

2.2.1 โครงสร้างภายในและคุณสมบัติของไมโครคอนโทรเลอร์ตระกูล PIC16F628

- ความถี่สัญญาณนาฬิกาสูงสุดถึง 20 MHz
- หน่วยความจำข้อมูล 224 ไบต์
- หน่วยความจำข้อมูลอีอีพรอม 128 ไบต์
- เลือกใช้วงจรกำเนิดสัญญาณนาฬิการได้ 6 โหมดหลัก
- ไทเมอร์ 3 ตัว
- มีโมดูลเปรียบเทียบแรงดันอนาล็อก 2 ชุด
- มีโมดูลสร้างแรงดันอ้างอิง
- มีโมดูลสื่อสารข้อมูลอนุกรม USART



รูปที่ 2.7 แสดงโครงสร้างการทำงานและการจัดขาของไมโครคอนโทรเลอร์

2.2.2 หน้าที่และการทำงานของพอร์ตใน PIC16F628

PIC16F628 มีพอร์ทให้ใช้งาน 2 พอร์ทคือพอร์ท A และ พอร์ท B ซึ่งความสามารถในการจ่ายกระแสเอาท์พุทของพอร์ทที่ไฟเลี้ยง +5V คือ +25 mA ต่อขาทั้งกระแสซิงค์และกระแสซอร์สในขณะที่กระแสเอาท์พุทรวมของแต่ละพอร์ทมีค่าสูงสุด 200 mA

2.2.2.1 พอร์ต A

มีทั้งสิ้น 8 ช่องหรือ 8 บิต กำหนดชื่อขาเป็น RA0-RA7 โดยปกติจะใช้งานได้ทันที่ 5 บิตคือ RA0-RA4 ส่วน RA5-RA7 ต้องมีการกำหนดเป็นพิเศษที่ configuration ของไมโครคอนโทรเลอร์ รีจิสเตอร์ที่ใช้ในการเก็บข้อมูล PORTA มีแอดเดรสอยู่ที่ 0x05 (แบงค์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต ส่วนการกำหนดทิศทางของพอร์ทนี้กระทำผ่านรีจิสเตอร์ TRISA ซึ่งมีแอดเดรสอยู่ที่ 0x85 (แบงค์ 1) มีขนาด 8 บิต บิต 0 ของ TRISA ใช้กำหนดทิศทางของขาพอร์ตในบิตใดเป็นอินพุตต้องเขียนข้อมูล “ 1” ไปยังบิตนั้น และในทางตรงข้ามหากต้องการกำหนดให้เป็นขาเอาท์พุต ให้เขียนข้อมูล “0” ไปยังบิตนั้น อย่างไรก็ตามเฉพาะ RA5 สามารถใช้งานเป็นอินพุตได้เพียงอย่างเดียว ในตารางที่ 4-1 แสดงรีจิสเตอร์ที่เกี่ยวข้องกับการกระทำของพอร์ต A

ตารางที่ 1 แสดงรีจิสเตอร์ที่เกี่ยวข้องกับการทำงานของพอร์ต A ทั้งหมด

แอดเดรส	รีจิสเตอร์	บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0	ค่าที่เกิดขึ้นหากเกิดเพาเวอร์ออนรีเซต	ค่าที่เกิดขึ้นหากเกิดรีเซตในแบบอื่น
0x05	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx - 0000	xxxu - 0000
0x85	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	11-1 -1111	11-1 -1111
0x1F	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0000	0000 0000
0x9F	VRCON	VREN	VRON	VRR	-	VR3	VR2	VR1	VR0	000 - 0000	000 - 0000

หมายเหตุ : - หมายถึง ไม่สามารถเขียนได้ อ่านค่าเป็น 0 เสมอ , u หมายถึง ไม่เปลี่ยนแปลง , x หมายถึง ไม่ทราบค่า ส่วนที่แรเงาไม่ใช้งาน ควรกำหนดให้มีค่าเท่ากับ 0

การติดต่อเพื่อกำหนดการทำงานและเขียนข้อมูลมูลไปยังพอร์ต A

ในขั้นแรกต้องทำการเตรียมความพร้อมให้แก่รีจิสเตอร์ PORTA โดยการเลือกแบงก์สำหรับติดต่อกับรีจิสเตอร์ PORTA จากนั้นส่งข้อมูล “0” เพื่อเคลียร์ข้อมูลทั้งหมด และเนื่องจากพอร์ต A ทำงานกับสัญญาณอะนาลอกได้ และค่าเริ่มต้นของพอร์ต A ที่ทำงานกับสัญญาณอะนาลอกจะถูกกำหนดให้เป็นอินพุทอะนาลอกทั้งหมด ดังนั้นหากต้องการใช้งานเป็นพอร์ตดิจิตอลต้องทำการกำหนดขอมูล 0x07 แล้วเขียนลงในรีจิสเตอร์ CMCON เพื่อดิสเอเบิลการทำงานกับสัญญาณอะนาลอก

2.2.2.2 พอร์ต B

มี 8 บิต กำหนดชื่อขาเป็น RB0-RB7 ริจิสเตอร์ที่ใช้ในการเป็นข้อมูลคือ PORTB มีแอดเดรสอยู่ที่ 0x06 (แบงค์ 0) และ (แบงก์ 2) เป็นรีจิสเตอร์ขนาด 8 บิต ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISB ซึ่งมีแอดเดรสอยู่ที่ 0x86 (แบงก์ 1) และ (แบงก์ 3) มีขนาด 8 บิต

เช่นเดียวกับพอร์ต A บิต 0 ของ TRISB ใช้กำหนดทิศทางของขาพอร์ต RB0 ไล่เรียงลำดับจนถึงบิต 7 ของ TRISB ใช้กำหนดทิศทางของขาพอร์ต RB7 หากต้องการกำหนดให้ขาพอร์ตในบิตใดเป็นอินพุตต้องเขียนข้อมูล“1” ไปยังบิตนั้น และในทางตรงกันข้ามหากต้องการให้เป็นขาเอาท์พุต ต้องเขียนข้อมูล “0” ไปยังบิตนั้น นอกจากนั้นพอร์ต B ยังมีความสามารถอื่นรวมอยู่ด้วย

ตารางที่2 แสดงรีจิสเตอร์ที่เกี่ยวข้องกับการทำงานพื้นฐานของพอร์ต B ทั้งหมด

แอดเดรส	รีจิสเตอร์	บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0	ค่าที่เกิดขึ้นหากเกิดเพาเวอร์ออนรีเซต	ค่าที่เกิดขึ้นหากเกิดรีเซตในแบบอื่น
0x006	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	ไม่ทราบค่า	ไม่เปลี่ยนแปลง
0x085	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
0x081	OPTION	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

หมายเหตุ : ส่วนที่แรเงาไม่เกี่ยวข้องกับพอร์ต B

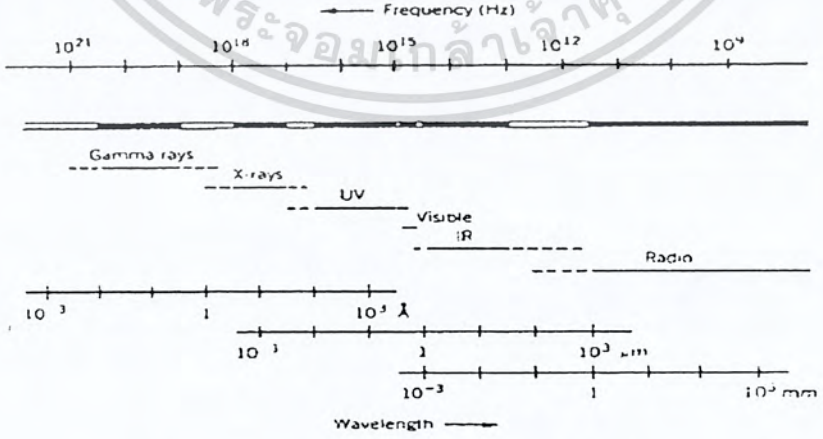
การติดต่อเพื่อกำหนดการทำงานและเขียนข้อมูลไปยังพอร์ต B

เหมือนกับพอร์ต A โดยเริ่มต้นเคลียร์ข้อมูลในรีจิสเตอร์ PORTB จากนั้นกำหนดทิศทางในรีจิสเตอร์ TRISB จากนั้นจึงเลือกแบงก์ใหม่เพื่อติดต่อกับรีจิสเตอร์ TRISB แล้วเขียนข้อมูลเพื่อกำหนดทิศทางของขาพอร์ตตามที่ต้องการลงในตารางรีจิสเตอร์ TRISB

2.3 ทฤษฎีที่เกี่ยวข้องกับการส่งสัญญาณย่านอินฟราเรด

การแผ่รังสีอินฟราเรดเป็นรูปแบบหนึ่งของการแผ่คลื่นพลังงานแม่เหล็กไฟฟ้า และมีคุณสมบัติคล้ายคลึงกับแสงทั่วไป คลื่นวิทยุ รังสีเอ็กซ์ ในความเป็นจริงแล้ว ความแตกต่างระหว่างอินฟราเรด กับ คลื่นแม่เหล็กไฟฟ้าจะต่างกันแค่ในส่วนของความยาวคลื่น

อินฟราเรดมีคุณสมบัติคร่าวๆคือเดินทางเป็นเส้นตรง ไม่สามารถทะลุผ่านโลหะได้เว้นแต่จะเดินทางเป็นเส้นตรง แต่สามารถผ่านผลึกหลายๆอย่างได้เช่น พลาสติก และบรรยากาศโลกเป็นต้น



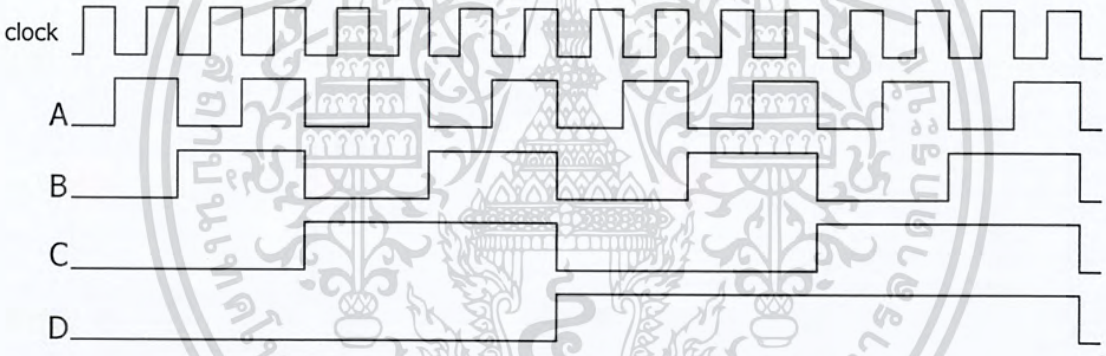
รูปที่2.8 แสดงสเปกตรัมของคลื่นแม่เหล็กไฟฟ้า

การถ่ายเทความร้อน 3 วิธีคือ การแผ่รังสีคลื่นแม่เหล็กไฟฟ้า การนำ เช่น โลหะ การพา เช่น อากาศ การส่งผ่านรังสีมีความสำคัญมากเพราะเครื่องรับอินฟราเรดจะวัดการส่งผ่านของความร้อน (หรือโฟตอน) ที่อุณหภูมิสูงมีผลให้การแผ่รังสีของอินฟราเรดมีกำลังงานมากกว่าที่อุณหภูมิต่ำ ซึ่งตัวรับส่วนใหญ่จะทำงานได้ดีภายใต้อุณหภูมิห้อง

ตัวรับอินฟราเรดเปรียบเสมือนตัวเปลี่ยนสัญญาณ ซึ่งจะเปลี่ยนสัญญาณอินฟราเรดเป็นสัญญาณไฟฟ้า สัญญาณที่รับได้นี้สามารถนำไปแสดงได้ในรูปแบบของความยาวคลื่น, ความถี่, กำลังงาน และการแผ่ของสเปกตรัม ซึ่งอินฟราเรดจะมีความถี่ 10^{14} Hz มีความยาวคลื่นในหน่วยไมโครเมตร

2.3.1 การหารความถี่ (Frequency Division)

จากวงจรหารความถี่เราทราบว่า ความถี่รูปคลื่นเอาท์พุตของ FF แต่ละตัวจะเป็นสัดส่วนกับความถี่สัญญาณคล็อกของอินพุท เช่นสมมุติว่าความถี่ของสัญญาณคล็อกของอินพุต ในรูป เป็น 16 KHz ทำให้เราสามารถหาความถี่รูปคลื่นเอาท์พุตของ FF แต่ละตัวได้



รูปที่ 2.9 แสดงสัญญาณเมื่อถูกหารด้วย 2 แต่ละครั้ง

จากรูปที่ แสดงรูปคลื่นเอาท์พุต(ของFF) A เป็นรูปคลื่นสี่เหลี่ยมที่มีความถี่ 8 KHz รูปคลื่นเอาท์พุต B มีความถี่ 4 KHz, รูปคลื่นเอาท์พุต C มีความถี่ 2 KHz และรูปคลื่นเอาท์พุต D มีความถี่ 1 KHz จะสามารถสังเกตได้ว่า รูปคลื่นเอาท์พุตของ FF D มีความถี่เท่ากับสัญญาณคล็อกของอินพุตหารด้วย 16 ทำให้ทราบว่า ความถี่เอาท์พุตของ FF ตัวสุดท้ายในวงจรนั่นคือความถี่คล็อกของอินพุตหารด้วยจำนวน MOD ของวงจรนับนั้น

2.3.2 การมอดูเลตโดยการเปลี่ยนขนาดของสัญญาณคลื่นพาห์

การมอดูเลตโดยการเปลี่ยนขนาดของสัญญาณคลื่นพาห์นั้น วิธีที่ง่ายที่สุดก็คือ การมอดูเลตให้ได้สัญญาณ AM ออกมา รูปร่างของสัญญาณที่ได้ก็เป็นดังที่ได้แสดงไว้ในรูป สัญญาณตามรูปนี้

โดยทั่วไปเรียกว่า ออนอ๊อฟคียอิง (on-off keying ย่อว่า OOK) เพราะเมื่อนกับการเปิดและปิดสัญญาณคลื่นพาห์ตามสัญญาณดิจิตอลที่เข้ามามอดูเลตสัญญาณ OOK นี้จัดว่าเป็นรูปแบบเฉพาะอันหนึ่งของสัญญาณแบบแอมปริจูดชิปท์คียอิง (amplitude shift keying ย่อว่า ASK)

รูปที่ 2.10 รูปร่างของสัญญาณออนออฟคียอิ้ง(OOK)

สัญญาณ OOK นี้มีประวัติการใช้งานยาวนานมากขึ้นตั้งแต่สมัยเริ่มใช้คลื่นวิทยุในการสื่อสารใหม่ โดยใช้ในการส่งโทรเลขและความถี่ที่ใช้เป็นย่าน HF (high frequency) ปัจจุบันนี้ ก็มีที่ใช้ในระบบไมโครเวฟบ้าง สำหรับที่ใช้เด่นชัดที่สุดก็คือระบบสื่อสารด้วยเส้นใยแสง ซึ่งการมอดูเลตตัวกำเนิดแสงคือเลเซอร์ไดโอดส่วนใหญ่จะใช้วิธีนี้

2.4 ความรู้เบื้องต้นเกี่ยวกับพอร์ตอนุกรม

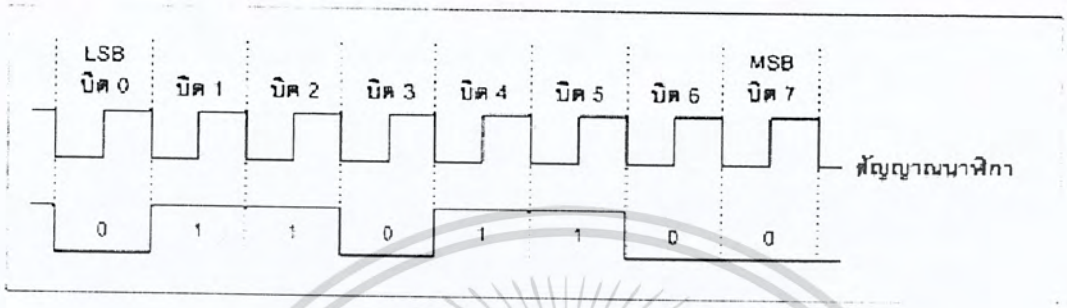
มีทางเลือกอยู่2ทางในการที่จะเคลื่อนย้ายข้อมูลไปยังอุปกรณ์ต่อพ่วงอื่นๆหรือคอมพิวเตอร์ด้วยกันนั่นคือการรับส่งข้อมูลแบบขนานและการรับส่งข้อมูลแบบอนุกรม การรับส่งข้อมูลแบบขนาน จะเป็นการรับส่งข้อมูลคราวละ 4 หรือ 8 บิต ในเวลาเดียวกัน ซึ่งจะการรับและการส่งข้อมูลทำได้ที่ความเร็วสูง ซึ่งก็หมายความว่าจำนวนสายที่ใช้ส่งข้อมูลจะต้องมีมากกว่าหรือเท่ากับจำนวนข้อมูลที่จะส่งด้วย นอกจากนี้ยังจะต้องรวมถึงสายที่ใช้สำหรับการควบคุมและการตรวจสอบการรับส่งข้อมูลด้วย ซึ่งอาจจะต้องใช้สายมากเป็น 2 เท่าของจำนวนบิตข้อมูลที่จะส่งก็ได้ ซึ่งก็เป็นปัญหาในเรื่องราคาของสายที่ใช้ในการเชื่อมต่อแบบขนานมักมีราคาแพง

ในขณะที่การรับส่งข้อมูลแบบอนุกรมเป็นการรับส่งข้อมูลครั้งละ 1 บิต แต่ก็สามารถรับส่งข้อมูลได้คราวละหลายๆบิตได้ หากแต่จะต้องมีการตกลงกันระหว่างตัวส่งและตัวรับว่าจะส่งข้อมูลคราวละกี่บิต ตัวรับจะต้องรอข้อมูลให้ครบทุกบิตเสียก่อนจึงทำการประมวลผล ส่งผลให้การสื่อสารข้อมูลอนุกรมมีความเร็วต่ำกว่าแบบขนาน ในด้านจำนวนสายส่งสัญญาณการรับส่งข้อมูลแบบอนุกรมจะใช้จำนวนสายที่น้อยกว่ามาก อย่างน้อยที่สุดใช้เพียง 2-3 เส้นเท่านั้น อย่างไรก็ตามการรับส่งข้อมูลแบบอนุกรมก็สามารถใช้สายสัญญาณที่มีความยาวมากกว่าแบบขนาน ทำให้ระยะทางในการสื่อสารข้อมูลแบบอนุกรมสามารถทำได้มากกว่า

การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมนั้นจะแบ่งออกได้เป็น 2 แบบคือการสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะสัญญาณนาฬิการ่วมอยู่กับ

การรับและส่งสัญญาณด้วย ตัวอย่างการส่งข้อมูลแบบซิงโครนัสก็คือคีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของสัญญาณนาฬิกา ส่วนอีกเส้นจะเป็นสัญญาณข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือ สัญญาณนาฬิกา, ข้อมูล และกราวด์ **รูป** แสดงให้เห็นถึงไทมิ่งไดอะแกรมของการส่งข้อมูลแบบซิงโครนัส



รูปที่2.11 รูปแบบอย่างง่ายของข้อมูลอนุกรม

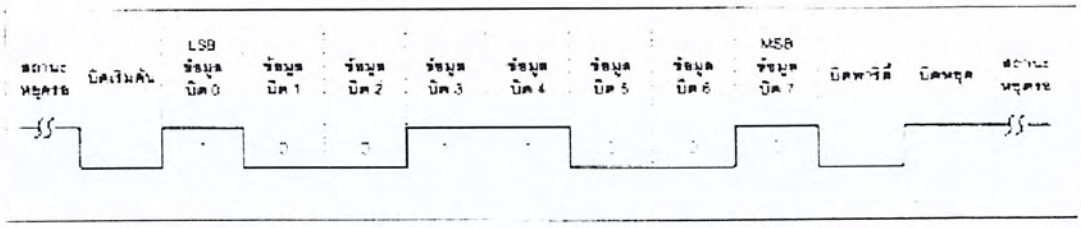
2.4.1การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิโครนัสคือการรับส่งข้อมูลไปในสายโดยไม่จำเป็นต้องมีการใช้สัญญาณนาฬิการ่วมด้วยเหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิการภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับและภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอดเรต (Baudrate) มีหน่วยเป็น บิตต่อวินาที (bit per secone : bps)

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอซิงโครนัสประกอบด้วย 4 ส่วนด้วยกันคือ

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

รูป2.11 แสดงรูปแบบของข้อมูลอนุกรมแบบซิงโครนัส ซึ่งเมื่อไม่มีข้อมูลที่จะส่ง ขา DATA จะมีสถานลอจิก "1" ซึ่งเรียกสถานะนี้ว่าสถานะหยุดรอ (Waiting stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขา DATA มีลอจิก "0" ด้วยช่วงระยะเวลา 1 บิต ซึ่งจะเรียกบิตนี้ว่าบิตเริ่มต้น จากนั้นบิตข้อมูลจะถูกส่งออกไป โดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจมีจำนวนบิต 5,6,7 หรือ 8 บิตก็ได้ จากนั้นจะตามด้วยบิตพาริตี้ ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่จะส่งคือบิตปิดท้าย ซึ่งจะใช้ขาดาต้ามีสถานะลดจิก 1 อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต, 1.5 หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว



รูปที่ 2.12 รูปแบบอย่างง่ายของข้อมูลอนุกรมแบบซิงโครนัส

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับส่งข้อมูลแบบซิงโครนัสถูกเรียกว่ากัน Universal Asynchronous Receiver/Transmitter หรือ UART อัตราความเร็วในการรับส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับส่งข้อมูลบอดเรตมาตรฐานที่ใช้สำหรับพอร์ตอนุกรมRS-232ได้แก่ 110,150, 300, 600, 1200, 2400, 9600 และ19200 บิตต่อวินาที่ และมีค่ามากเพิ่มขึ้นตามเทคโนโลยีของคอมพิวเตอร์ ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดค่าบอดเรตได้สูงถึง 115200 บิตต่อวินาที เนื่องจากบอดเรตคือจำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิตความยาวของข้อมูลที่รับส่งนี้เท่ากับ 10 บิต ถ้าใช้บอดเรตในการส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก็สามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที และถ้ามีการใช้พาริตี้ความเร็วในการรับส่งข้อมูลจะเหลือเป็น 872 ไบต์ต่อวินาที

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นแบบคี่ (odd), แบบคู่ (even) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ การตรวจสอบพาริตี้เป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิก “1” ภายในข้อมูลที่ส่งไป 1 ไบต์ว่ามีจำนวนรวมเป็นเลขคู่หรือเลขคี่ โดยต้องรวมบิตพาริตี้เข้าไปด้วย ยกตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิต และมีค่าเท่ากับ 99 ฐานสิบหก หรือ 10011001 ฐานสอง จะเห็นว่าข้อมูลในไบต์นี้มีจำนวนลอจิกเป็น “1” จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้นถ้ากำหนดค่าพาริตี้เป็นคู่ ค่าในบิตพาริตี้จะมีต้องลอจิกเป็น “0” แต่ถ้าเป็นพาริตี้คี่ ค่าที่บิตพาริตี้จะต้องเป็น “1” เพื่อให้ข้อมูล 1 ไบต์รวมทั้งบิตพาริตี้มีจำนวนลอจิกที่เป็น “1” มีจำนวนรวมกันเป็นเลขคี่ ในตารางแสดงตัวอย่างของบิตพาริตี้ในการรับส่งข้อมูลอนุกรม

ข้อมูล	บิตพาริตี้	บิตพาริตี้
00000000	0	1
00000001	1	0
00000010	1	0
00000011	0	1

00000100	1	0
11111110	0	1
11111111	1	0

ตารางที่3 แสดงบิตพาริตี้ของข้อมูล

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART ซึ่งทางภาครับจะต้องทำการกำหนดคุณสมบัติการตรวจสอบพาริตี้ให้ตรงกันว่าจะตรวจสอบพาริตี้คี่หรือพาริตี้คู่ จากนั้นภาครับของ UART จะทำการตรวจสอบค่าพาริตี้ที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก "1" ทั้งหมดรวมพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแสดงข้อผิดพลาดออกมาให้ผู้ใช้ทราบ นับเป็นการตรวจสอบความผิดพลาดที่เกิดขึ้นในการถ่ายทอดข้อมูลที่ง่ายที่สุด แต่จะเชื่อถือได้เมื่อมีบิตข้อมูลที่ทำการส่งผิดพลาดเพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิตที่ผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งค่าพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี้

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ UART เบอร์ 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT ใช้ UART เบอร์ 8250 UART ชิบเหล่านี้มีระดับแรงดันเป็นแบบทีทีแอล (0 และ +5v) แต่เพื่อให้มีแรงดันเป็นมาตรฐาน RS-232 และเพื่อให้การรับส่งข้อมูลสามารถทำได้ที่ระยะทางไกลมากขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้น โดยลอจิก "0" มีระดับแรงดัน +3v ถึง +12v ในขณะที่ลอจิก "1" มีระดับแรงดัน —12v

2.4.2 มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232

มาตรฐานการเชื่อต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS-232 ในอดีตนั้นถูกออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้สื่อสารผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดซึ่งอยู่ห่างไกลกัน โดยกรรมการที่เรียกว่า *สมาคมอุตสาหกรรมอิเล็กทรอนิกส์* (Electronic Industries Assocition :EIA) ได้วางมาตรฐานที่มีชื่อเรียกกันว่า EIA RS-232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ —3 ถึง —12v แสดงว่ามีข้อมูล (Mark) และ +3 ถึง +12v แสดงว่าเป็นช่องว่าง (Space)

มาตรฐาน RS-232 ได้ทำการกำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equiqment : DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating : DCE) ไว้ว่า อุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัว เช่น ไมโครคอนโทรลเลอร์หรือไมโครคอมพิวเตอร์ ซึ่งมีความสามารถในการสร้างบิตข้อมูลอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่เป็นเพียงตัวรับ

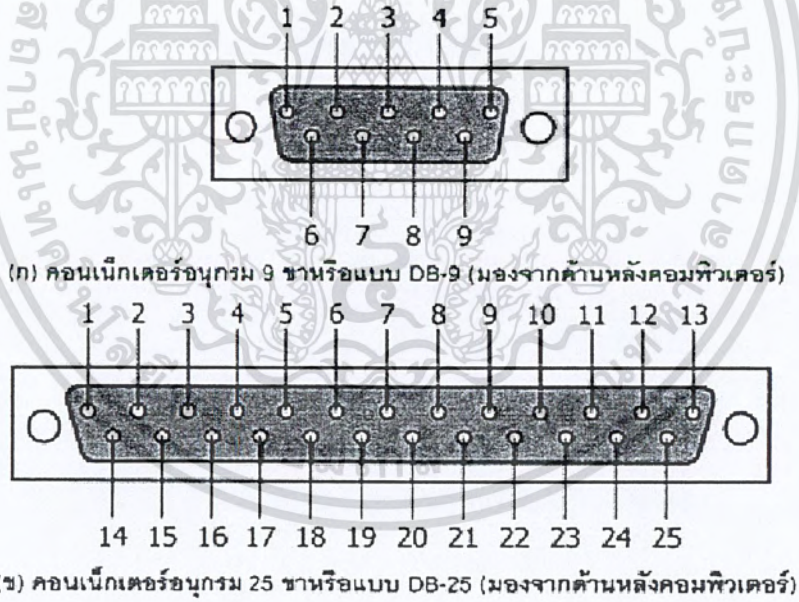
ข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่านมาตรฐาน RS-232

ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อีกอย่างที่เห็นได้ชัดคือ คอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็กเตอร์ที่อยู่ที่โมเด็มจะเป็นแบบ DCE

สำหรับการใช้งานบนคอมพิวเตอร์ พอร์ตอนุกรม RS-232 มักจะถูกใช้เชื่อมต่อโมเด็มหรือเมาส์ โดยสามารถรับส่งข้อมูลได้ที่ความยาวของสายสัญญาณสูงสุด 20 เมตรคอนเน็กเตอร์สำหรับพอร์ต RS-232

2.4.3 คอนเน็กเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้หรือ DB-9 ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับคอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่นๆที่เคยใช้ในอดีต ปัจจุบันมีการใช้งานไม่มากนัก จึงถูกยกเลิกไป โดยแสดงรูปร่างและตำแหน่งขาในรูป2.13



(ก) คอนเน็กเตอร์อนุกรม 9 ขาหรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์)

(ข) คอนเน็กเตอร์อนุกรม 25 ขาหรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์)

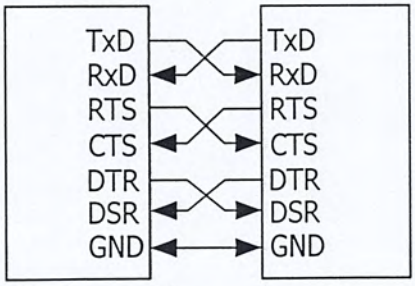
รูปที่2.13 การจัดขาของคอนเน็กเตอร์พอร์ตอนุกรมมาตรฐาน RS-232 DB-25 และDB-9

คอนเน็กเตอร์ DB-9	คอนเน็กเตอร์ DB-25	ชื่อของสายสัญญาณ	ชนิดสายสัญญาณ
1	8	Data Carrier Detect : DCD	อินพุต
2	3	Received Data : RxD	อินพุต
3	2	Data Terminal Data : TxD	เอาต์พุต
4	20	Data Teminal Ready : DTR	เอาต์พุต
5	7	Signal Ground : GND	-
6	6	Data Set Ready : DSR	อินพุต
7	4	ReQuest To send :RTS	เอาต์พุต
8	5	Clear To Send : CTS	อินพุต
9	22	Ring Indicator : RI	อินพุต

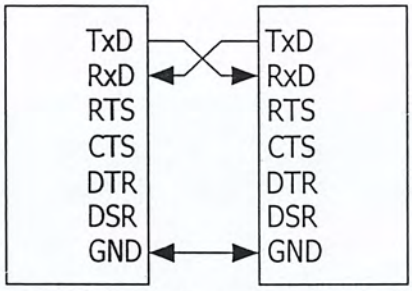
ตารางที่4 เปรียบเทียบ คอนเน็กเตอร์ DB-9 กับ DB-25

สำหรับการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกแสดงในรูป ลูกศรในรูปแสดงถึงทิศทางข้อมูล ในรูป เป็นการเชื่อมต่อแบบ Null modem หรือการเชื่อมต่อโดยตรงโดยไม่จำเป็นต้องผ่านโมเด็ม โดยมีการตรวจสอบหรือแฮนด์เช็กเต็มรูปแบบ ส่วนในรูปเป็นการเชื่อมต่อแบบ Null modem ในลักษณะที่ใช้สายสัญญาณเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูล และเส้นสุดท้ายเป็นกราวด์ สำหรับรายละเอียดการทำงานในแต่ละขาของพอร์ตอนุกรม RS-232 มีดังนี้

- Data Carrier Detect : DCD หรืออาจเรียกว่า Carrier Detect : DC ขานี้จะแอกตีฟเมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูล เช่น โมเด็ม สำหรับการใช้งานปกติ ขานี้จะไม่ได้ถูกใช้งานมากนัก
- Receive Data : RD หรือ RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์ บัฟเฟอร์
- Tranmitted Data : TD หรือ TxD ขานี้ใช้เพื่อส่งข้อมูลออกจากคอมพิวเตอร์ โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลออกไป



(ก) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์ แบบ Null Modem



(ข) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์ แบบ RS-232 โดยใช้สายสัญญาณ 3 เส้น

รูปที่2.14 การต่ออุปกรณ์ภายนอกกับพอร์ตอนุกรมของคอมพิวเตอร์ในลักษณะต่างๆ

- Data Terminal Ready :DTR เป็นขาสัญญาณที่ส่งออกจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรู้ว่า ต้องการติดต่อด้วย โดยขา DTR นี้จะต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์ถ้าใช้การเชื่อมต่อแบบ Null Modem ซึ่งใช้สายในการเชื่อมต่อเพียง 3 เส้น จะต้องต่อขา DTR และ DAR ของตัวมันเองเข้าด้วยกันและต่อต่อกับขา DCD ด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณพาห์
- Signal Ground : GND ขากราวด์ของระบบ
- Data Set Ready : DSR ขานี้จะใช้คู่กับขาก DTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอกซึ่งถูกส่งมาจากขา DTR
- Request To Send : RTS เป็นขาสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทางส่งข้อมูลกลับมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ CTS ในกรณีที่ใช้การเชื่อมต่อแบบ Null modem 3 สาย จะต้องเชื่อมขา RTS และ CTS ของตัวมันเองเข้าด้วยกัน เพื่อจะให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา
- Clear To Send : CTS ขานี้จะคอยรับสัญญาณจากขา RTS เมื่อรับสัญญาณได้ ข้อที่ขา DxT จะถูกส่งออกไป ดังนั้นขานี้จึงถูกใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลหรือไม่
- Ring Indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ปกติในการสื่อสารโดยทั่วไปสายนี้จะไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มและโปรแกรมมีการตรวจสอบสัญญาณนี้เท่านั้น

2.4.4 UART

UART มาจากคำว่า Universal Asynchronous Receiver Transmitter ซึ่งหมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัสนั่งเอง สำหรับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UARTถือเป็นหัวใจสำคัญของการสื่อสารอนุกรม

หน้าที่หลักของ UART คือทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโครนัส แล้วส่งออกไป และทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามายัง UARTให้เป็นแบบขนานก่อนส่งเข้าสู่คอมพิวเตอร์ ซึ่งนอกจาก UARTจะส่งข้อมูลไปยังคอมพิวเตอร์แล้ว ยังแจ้งข้อมูลอื่นๆ ให้คอมพิวเตอร์ทราบด้วย เช่น อัตราเร็วในการรับส่งข้อมูล (บอดเรต) , รูปแบบการส่งข้อมูล, ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอดข้อมูล (ผิดพลาดจาก พาริตี้, เฟรมข้อมูล, โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างบอดเรตแบบโปรแกรมได้ (programamble buadrate generator) โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UARTโดยตัวหารนี้มีขนาด 16 บิต ดังนั้นจึงสามารถกำหนดตัวหารอยู่ในช่วง 1-65535 UARTสามารถรับส่งข้อมูลได้ทั้งแบบฮาล์ฟดูเพล็กซ์(half duplex) และฟูลดูเพล็กซ์ (full duplex) โดยการส่งแบบฮาล์ฟดูเพล็กซ์เป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบฟูลดูเพล็กนั้นสามารถรับและส่งข้อมูลได้ในคราวเดียวกัน

2.4.5 ชนิดของ UART

ในเครื่องคอมพิวเตอร์ทั่วไปมี UARTที่ใช้งานกันอยู่ 2 เบอร์คือ 8250 ซึ่งเป็น UART มาตรฐานที่ใช้กันมายาวนาน UARTเบอร์นี้จะมีบัฟเฟอร์สำหรับรับและส่งข้อมูลตำแหน่งเดียวกัน ทำให้การรับและการส่งข้อมูลถูกจำกัดความเร็วอยู่ที่ 57.6 กิโลบิตต่อวินาทีเท่านั้น แต่ UARTเบอร์นี้ก็ถือว่าเป็นต้นแบบของ UART ที่ใช้ในคอมพิวเตอร์ โดยคอมพิวเตอร์ทุกๆรุ่นจะต้องสนับสนุนการทำงานตามรูปแบบของ UART เบอร์นี้

UART อีกเบอร์หนึ่งคือ 16450 มีความสามารถรับส่งข้อมูลได้ที่ความเร็ว 115200 บิตต่อวินาที และเพิ่มรีจิสเตอร์สำหรับพักข้อมูลสำหรับ UART นอกจากนั้นยังเพิ่มส่วนของชิฟต์รีจิสเตอร์แบบ FIFO(First In First Out) ขนาด 16 ไบต์เข้าไป ทำให้สามารถสนับสนุนการรับส่งข้อมูลที่ 256 กิโลบิตต่อวินาทีได้ โดยคอมพิวเตอร์ปัจจุบันใช้ UARTเบอร์นี้หรือใหม่กว่า เช่น เบอร์ TL16C750 ซึ่งมีรีจิสเตอร์แบบ FIFO ขนาด 64 ไบต์ ทำงานได้ที่ระดับแรงดัน +5v และ —5v และมีโหมดประหยัดพลังงาน สามารถรับส่งข้อมูลได้ที่ความเร็ว 1 เมกะบิตต่อวินาทีเมื่อใช้สัญญาณนาฬิกา 16 Mhz

อย่างไรก็ตาม ความเร็วในการส่งข้อมูลที่มากมายของ UART เบอร์ใหม่ๆ ก็ไม่ได้ช่วยให้การรับส่งข้อมูลของคอมพิวเตอร์เร็วขึ้น เนื่องจากว่าคอมพิวเตอร์ยังใช้ความถี่ของสัญญาณนาฬิกาในการแปลงข้อมูลเพียง 1.8432 Mhz เท่านั้น

2.4.6 วงจรภายในและรีจิสเตอร์ของพอร์ตอนุกรม RS-232

เครื่องคอมพิวเตอร์โดยทั่วไปสามารถต่อพอร์ตอนุกรม RS-232 สูงสุดได้ 4 พอร์ต ซึ่งจะมีชื่อเรียกเป็น COM1, COM2, COM3, และ COM4 ซึ่งพอร์ตอนุกรมแต่ละตัวต่างก็ใช้งาน UARTภายในคอมพิวเตอร์ในการติดต่อกับอุปกรณ์ภายนอกเช่นเดียวกัน

ในรูป2.15แสดงไดอะแกรมการทำงานภายในของพอร์ตอนุกรม ซึ่งประกอบไปด้วยรีจิสเตอร์ขนาด 8 บิต 8 ตัวที่ใช้งานร่วมกับ UART แอดเดรสของรีจิสเตอรภายในพอร์ตอนุกรมสามารถคำนวณได้จากค่ารีจิสเตอร์พื้นฐานของพอร์ตอนุกรม ยกตัวอย่าง พอร์ตอนุกรม COM1 มีแอดเดรสอยู่ที่ 3f8H ตำแหน่งของรีจิสเตอร์ต่างๆจะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H โดยรีจิสเตอร์ที่ใช้งานกับพอร์ตอนุกรมมีดังนี้

00H เป็นรีจิสเตอร์บัฟเฟอร์สำหรับเก็บข้อมูลที่รับเข้ามาหรือเตรียมข้อมูลก่อนที่จะส่งออกไป

01H รีจิสเตอร์อีนาเบิลการอินเตอร์รัปต์ ใช้ในการเซตโหมดการอินเตอร์รัปต์ของพอร์ตอนุกรม

02H รีจิสเตอร์แสดงโหมดการอินเตอร์รัปต์ ใช้เพื่อตรวจสอบโหมดของการอินเตอร์รัปต์เมื่อมีการอินเตอร์รัปต์เกิดขึ้น

03H รีจิสเตอร์กำหนดรูปแบบของข้อมูล

04H รีจิสเตอร์ควบคุมโมเด็ม ใช้ตรวจสอบบิตสำหรับติดต่อกับโมเด็ม เช่น RTS หรือ DTR

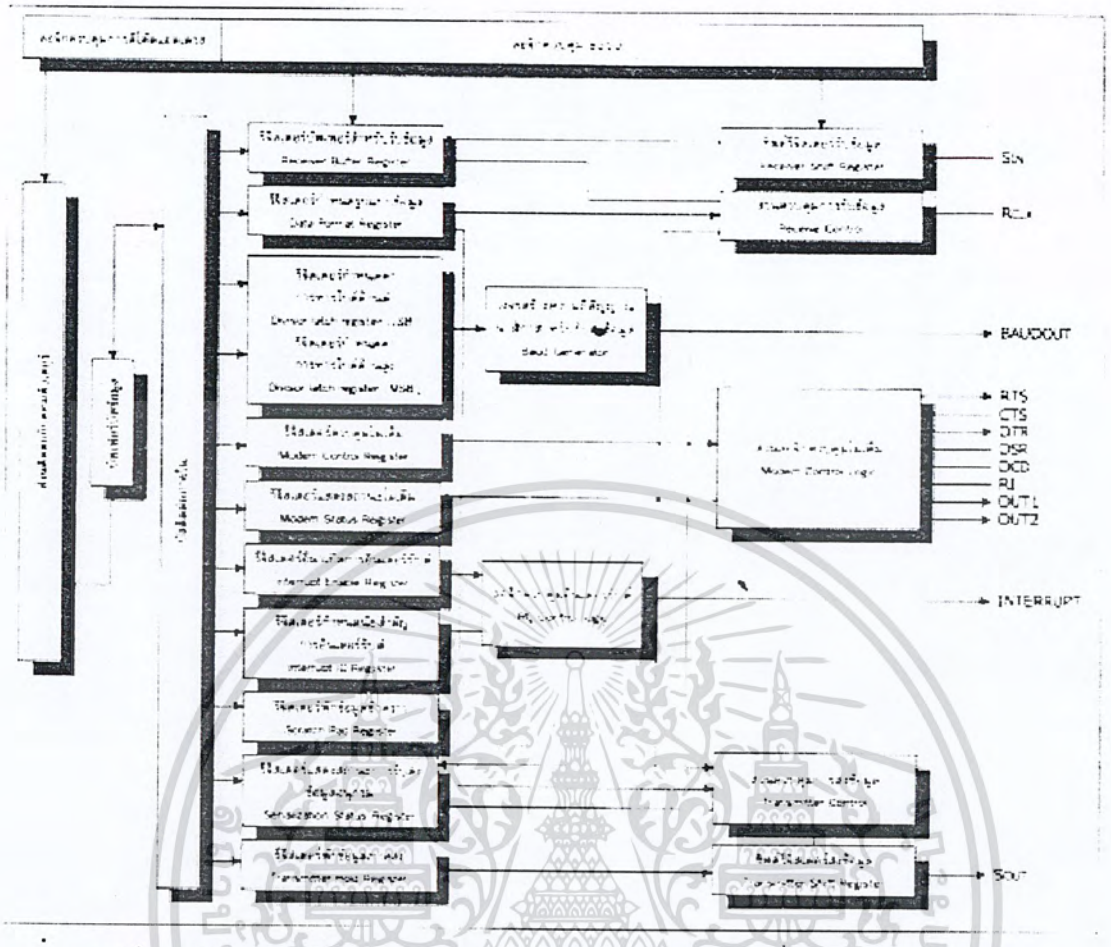
05H รีจิสเตอร์แสดงสถานะการรับและการส่งข้อมูลแบบอนุกรม

06H รีจิสเตอร์แสดงสถานะของโมเด็ม ซึ่งจะแสดงสถานะของขา DCD,RI,DSR และ CTS

07H รีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว

2.4.6.1 รีจิสเตอร์ตำแหน่ง 00H : รีจิสเตอร์บัฟเฟอร์

เป็นรีจิสเตอร์สำหรับเก็บข้อมูลที่รับเข้ามาและข้อมูลที่จะส่งออกไป โดยการติดต่อกับรีจิสเตอร์นี้เพื่อเก็บข้อมูลที่ต้องการจะส่งต้องกำหนดให้บิต DLAB ในรีจิสเตอร์กำหนดรูปแบบข้อมูล (03H) จะต้องมีสถานะเป็น 0 ซึ่งการเขียนข้อมูลมายังแอดเดรสนี้ เป็นการส่งข้อมูลไปยังรีจิสเตอร์ส่งข้อมูลและข้อมูลจะถูกส่งไปแบบอนุกรม สำหรับการส่งรับข้อมูล เมื่อข้อมูลที่รับเข้ามาเรียบร้อยและแปลงเป็นขนานแล้ว ข้อมูลจะถูกส่งมายังรีจิสเตอร์เก็บข้อมูล หลังจากมีการอ่านรีจิสเตอร์นี้ออกไปรีจิสเตอร์จะถูกเคลียร์ และเตรียมพร้อมสำหรับการรับข้อมูลในไบต์ต่อไป



รูปที่ 2.15 ไดอะแกรมการทำงานภายในของพอร์ตอนุกรมของเครื่องคอมพิวเตอร์

2.4.6.2 รีจิสเตอร์ตำแหน่ง 01H : รีจิสเตอร์อีนาเบิลการอินเตอร์รัปต์

เป็นรีจิสเตอร์สำหรับการอีนาเบิลการอินเตอร์รัปต์ ซึ่งเป็นการกำหนดให้ UARTสร้างสัญญาณอินเตอร์รัปต์ขึ้นมา ฟังก์ชั่นการทำงานในแต่ละบิตของรีจิสเตอร์มีดังนี้

บิต 7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	**0**	**0**	**0**	**SINP**	**ERBK**	**TBE**	**RxRD**

บิต 4-7 บิตเหล่านี้ไม่ถูกใช้งาน กำหนดให้เท่ากับ “0”

SINP อีนาเบิลการอินเตอร์รัปต์เนื่องจากเกิดการเปลี่ยนสถานะที่ขาอินพุต CTS,DSR,DCD หรือขา RI

ERBK อีนาเบิลการอินเตอร์รัปต์เนื่องจากเกิดความผิดพลาดขึ้นด้วยสาเหตุจาก พาริตี้, โอเวอร์รัน, เฟรมข้อมูล หรือการเบรกข้อมูล

“1” อีนาเบิลการอินเตอร์รัปต์

“0” ไม่มีการใช้อินเตอร์รัปต์รูปแบบนี้หรือดิสเอเบิล

TBE อีนาเบิลอินเตอร์รัปต์เมื่อรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

"1" อีนาเบิลการอินเตอร์รัปต์

"0" ไม่มีการใช้อินเตอร์รัปต์รูปแบบนี้หรือดิสเอเบิล

RxRD อีนาเบิลการอินเตอร์รัปต์เนื่องจากรีจิสเตอร์บัฟเฟอร์ได้รับข้อมูล

"1" อีนาเบิลการอินเตอร์รัปต์

"0" ไม่มีการใช้อินเตอร์รัปต์รูปแบบนี้หรือดิสเอเบิล

2.3.6.3 รีจิสเตอร์ตำแหน่ง 02H : รีจิสเตอร์แสดงโหมดและสถานะการอินเตอร์รัปต์ มีรายละเอียดของแต่ละบิตดังนี้

บิต 7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	**0**	**0**	**0**	**0**	**ID1**	**ID0**	**PND**

บิต 3-7 ไม่ได้ใช้งาน อ่านค่าได้เท่ากับ "0"

ID1,ID0 ใช้งานร่วมกันเพื่อแจ้งสาเหตุของการเกิดการอินเตอร์รัปต์

"00" เกิดการอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงของขาอินพุทขึ้น การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 4

"01" เกิดการอินเตอร์รัปต์เนื่องจากรีจิสเตอร์บัฟเฟอร์ส่งข้อมูลว่างขึ้น การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 3

"10" เกิดการอินเตอร์รัปต์เนื่องจากข้อมูลถูกเก็บลงในรีจิสเตอร์บัฟเฟอร์สำหรับรับข้อมูลเรียบร้อยแล้ว การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 2

"11" เกิดการอินเตอร์รัปต์เนื่องจากความผิดพลาดในการถ่ายทอดข้อมูลหรือการเบรก (break : เกิดการหยุดถ่ายทอดข้อมูลกระทันหัน) การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 1 หรือมีนัยสำคัญสูงสุด

PND ใช้แสดงสถานะของการเกิดการอินเตอร์รัปต์

"1" แสดงว่าไม่มีการอินเตอร์รัปต์

"0" แสดงว่ามีการอินเตอร์รัปต์เกิดขึ้น

เมื่อมีการสร้างสัญญาณอินเตอร์รัปต์ขึ้น จะต้องมีการเคลียร์ค่าก่อนที่จะให้เกิดการอินเตอร์รัปต์ครั้งต่อไป โดยสามารถทำได้ดังนี้

- ถ้าเกิดอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงของขาอินพุทจะต้องอ่านค่าจากรีจิสเตอร์แสดงสถานะของโมเด็ม (รีจิสเตอร์ตำแหน่ง 06H) หรืออ่านค่ารีจิสเตอร์แสดงสถานะอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 02H)

- ถ้าเกิดอินเตอร์รัปต์เนื่องจากบัฟเฟอร์ส่งข้อมูลว่าง จะต้องเขียนข้อมูลไปยังรีจิสเตอร์บัฟเฟอร์ส่งข้อมูล (รีจิสเตอร์ตำแหน่ง 00H) หรืออ่านค่ารีจิสเตอร์แสดงสถานอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 02H) เพื่อเคลียร์ค่าการอินเตอร์รัปต์
- ถ้าเกิดอินเตอร์รัปต์เนื่องจากการเก็บข้อมูลลงในรีจิสเตอร์บัฟเฟอร์สำหรับรับข้อมูลเรียบร้อย จะต้องเคลียร์ค่าอินเตอร์รัปต์โดยการอ่านข้อมูลจากรีจิสเตอร์บัฟเฟอร์
  ถ้าเกิดอินเตอร์รัปต์เนื่องจากความผิดพลาดในการรับส่งข้อมูลหรือการเบรก จะต้องเคลียร์ค่าอินเตอร์รัปต์โดยการอ่านค่ารีจิสเตอร์แสดงสถานะการรับและส่งข้อมูลแบบอนุกรม

2.4.6.4 รีจิสเตอร์ตำแหน่ง 03H : รีจิสเตอรกำหนดรูปแบบของข้อมูล
มีรายละเอียดหน้าที่ของแต่ละบิตดังนี้

บิต 7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
DLAB	**BRK**	**PAR2**	**PAR1**	**PAR0**	**STOP**	**DAB1**	**DAB0**

DLAB ใช้ในการกำหนดหน้าที่การทำงานของรีจิสเตอร์บัฟเฟอร์ (00H)
"1" เป็นการเข้าสู่โหมดการหารค่าบอดเรต
"0" เป็นการเข้าถึงรีจิสเตอร์บัฟเฟอร์ (รีจิสเตอร์ตำแหน่ง 00H) และรีจิสเตอร์สำหรับอีนาเบิลการอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 01H)เมื่อบิต DLABเป็น "1" รีจิสเตอร์บัฟเฟอร์ (00H) และรีจิสเตอร์อีนาเบิลการอินเตอร์รัปต์(01H)จะใช้สำหรับโหลดค่าการหารความถี่สำหรับกำหนดบอดเรตโดยรีจิสเตอร์00H เก็บค่าบอดเรตสามารถเขียนสมการได้ดังนี้

บอดเรต = 115200/ค่าตัวหาร 16 บิต

ค่าตัวเลข 115200 มาจากความถี่คริสตอลในวงจร UART ภายในเครื่องคอมพิวเตอร์คริสตอลที่ใช้มีความถี่ 1.8432 MHz วงจรภายใน UARTจะทำการหารค่าความถี่นี้ด้วย 16 ทำให้ได้ค่าความถี่115200 Hz ออกมา

ค่าตัวหาร 16 บิต = ข้อมูลในรีจิสเตอร์ 00H+(256xข้อมูลในรีจิสเตอร์ 01H)

สมมุตว่าต้องการค่าบอดเรตเท่ากับ 9600 ค่าตัวหารที่ใช้จะต้องมีค่าเท่ากับ12 ซึ่งค่านี้จะต้องถูกโหลดลงในรีจิสเตอร์ 00H และโหลดค่า 0 ลงในรีจิสเตอร์01Hค่าตัวหารทีทำให้เกิดค่าบอดเรตสูงสุดที่ 115200บิตต่อวินาทีคือค่า 0001 นั่นคือ รีจิสเตอร์ 00H มีค่าเท่ากับ 1 และรีจิสเตอร์01Hมีค่าเท่ากับ 0

BRK ใช้ควบคุมการหยุดถ่ายทอดข้อมูล
"1" สามารถหยุดหรือเบรกได้
"0" ไม่มีการหยุดหรือเบรกได้

PAR2, PAR1, PAR0 ใช้เพื่อกำหนดพาริตี้

“000” ไม่ใช้บิตพาริตี้

“001” กำหนดพาริตี้คี่

“011” กำหนดพาริตี้คู่

“101” มาร์ก (mark)

“111” ช่องว่าง (space)

STOP ใช้กำหนดจำนวนบิตปิดท้าย

“1” มีบิตปิดท้าย 2 บิต

“0” มีบิตปิดท้าย 1 บิต

DAB1, DAB0 ใช้ร่วมกันในการกำหนดจำนวนบิตของข้อมูลที่ต้องการถ่ายทอด

“00” จำนวนบิตข้อมูลเท่ากับ 5 บิต

“01” จำนวนบิตข้อมูลเท่ากับ 6 บิต

“10” จำนวนบิตข้อมูลเท่ากับ 7 บิต

“11” จำนวนบิตข้อมูลเท่ากับ 8 บิต

2.4.6.5 รีจิสเตอร์ตำแหน่ง 04H : รีจิสเตอร์ควบคุมโมเด็ม

มีรายละเอียดหน้าที่ของแต่ละบิตดังนี้

บิต 7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	**0**	**0**	**LOOP**	**OUT2**	**OUT1**	**RTS**	**DTR**

บิต 5-7 ไม่มีการใช้งาน อ่านค่าได้เท่ากับ 0

LOOP “1” อีนาเบิลการส่งค่ากลับ

“0” ดิสเอเบิล

OUT1, OUT2 “1” อีนาเบิลการใช้งานภายใน

“0” ดิสเอเบิล

RST ใช้ควบคุมการทำงานของขา RST (Ready To Send)

“1” อีนาเบิล

“0” ดิสเอเบิล

DTR ใช้ควบคุมการทำงานของขา DTR (Data Terminal Ready)

“1” อีนาเบิล

“0” ดิสเอเบิล

2.4.6.6 รีจิสเตอร์ตำแหน่ง 05H : รีจิสเตอร์แสดงสถานะการรับและส่งข้อมูลอนุกรมของ UART

ใช้งานร่วมกับรีจิสเตอร์แสดงโหมดและสถานะของการอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 02H) เพื่อใช้แสดงสาเหตุของการเกิดอินเตอร์รัปต์ มีรายละเอียดหน้าที่แต่ละบิตดังนี้

บิต 7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	**TXE**	**TBE**	**BREK**	**FRME**	**PARE**	**OVFE**	**RxRD**

TXE (Transmitter Empty) "1" แสดงว่ารีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง
"0" แสดงว่ายังคงมีข้อมูล 1 ไบต์เก็บอยู่ในรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล

TBE (Transmitter Buffer Empty) "1" รีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง
"0" ยังคงมีข้อมูล 1 ไบต์เก็บอยู่ในรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล

BREK (Brek "1" UART ตรวจสอบการเบรก
"0" ไม่มีการเบรก

FRME (Frme Error) "1" UART ตรวจพบความผิดพลาดด้านเฟรมข้อมูล
"0" ไม่พบความผิดพลาดด้ายเฟรมข้อมูล

PARE (Parity Error) "1" UART ตรวจสอบพบความผิดพลาดทางพาริตี้
"0" ไม่พบความผิดพลาดทางพาริตี้

OVRE (Overrun Error) "1" UART ตรวจพบความผิดพลาดแบบโอเวอร์รัน
"0" ไม่พบความผิลพลาดแบบโอเวอร์รัน

RxRD (Received Data Ready) "1" มีการรับข้อมูลเข้ามาเก็บไว้ในบัฟเฟอร์
"0" ไม่มีข้อมูล

2.4.6.7 รีจิสเตอร์ตำแหน่ง 06H : รีจิสเตอร์แสดงสถานะของโมเด็ม

ใช้เพื่อกำหนดสถานะสัญญาณอินพุต ของพอร์ตอนุกรม RS-232 ซึ่งได้แก่สัญญาณ DCD, DSR, CTS และ RI สำหรับการเชื่อมต่อใช้งานเอนกประสงค์ ดังมีรายละเอียดหน้าที่ของแต่ละบิตต่อไปนี้

บิต 7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
DCD	**RI**	**DSR**	**CTS**	**DDCD**	**DRI**	**DDSR**	**DCTS**

DCD ใช้แสดงสภาวะของขา DCD
"1" แสดงว่าที่ขา DCD เป็นลอจิก "1"

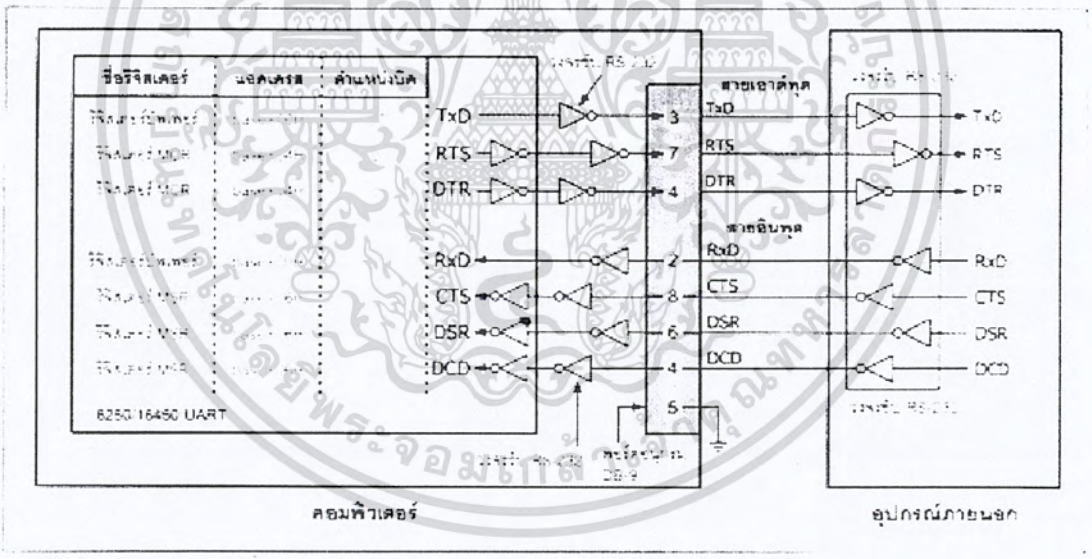
	“0” แสดงว่าที่ขา DCD เป็นลอจิก “0”
RI	ใช้แสดงสภาวะของขา RI
	“1” แสดงว่าที่ขา RI เป็นลอจิก “1”
	“0” แสดงว่าที่ขา RI เป็นลอจิก “0”
DSR	ใช้แสดงสภาวะของขา DSR
	“1” แสดงว่าที่ขา DSR เป็นลอจิก “1”
	“0” แสดงว่าที่ขา DSR เป็นลอจิก “0”
DCTS (Delta Clear To Send)	ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต CTS
	“1” แสดงว่าบิต CTS (Clear To Send) เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว
	“0” แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว
DDSR (Delta Data Set Ready)	ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต DSR
	“1” แสดงว่าบิต DSR (Data Set Ready) เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว
	“0” แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว
DRI (Delta Ring Indicator)	ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต RI
	“1” แสดงว่าบิต RI (Ring Indicator) เกิดการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว
	“0” แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว
DCTS (Delta Data Carrier Detect)	ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต DDCD
	“1” แสดงว่าบิต CTS (Cleat To Sent)เกิดการเปลี่ยนแปลง เมื่อเทียบจากการอ่านค่าครั้งที่แล้ว
	“0” แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว
DCTS (Delta Clear To Sent)	ใช้แสดงสภาวะของขา CTS
	“1’ แสดงว่าที่ขา CTS เป็นลอจิก “1”
	“0” แสดงว่าที่ขา CTS เป็นลอจิก “0”

2.4.6.8 รีจิสเตอร์ตำแหน่ง 07H : รีจิสเตอร์สำหรับเก็บข้อมูลชั่วคราว

ทำหน้าที่เป็นหน่วยความจำแรมขนาด 1 ไบต์ การอ่านและเขียนข้อมูลที่รีจิสเตอร์ตัวนี้ไม่ส่งผลใดๆต่อการทำงานของ UART

2.4.7 ลักษณะสัญญาณอินพุตและเอาต์พุตของพอร์ต RS-232

สัญญาณเอาต์พุตที่ใช้ควบคุม (RTS และ DTR) และสัญญาณแสดงสถานะอินพุต (CTS, DSR และ DCD) ของพอร์ตอนุกรม RS-232 จะถูกกลับสถานะภายในตัว UART ส่วนสัญญาณข้อมูลทั้งภาคส่งและภาครับจะไม่ถูกกลับสถานะ UART จะให้ระดับสัญญาณเอาต์พุตออกมาเป็นทีทีแอลเท่านั้น ดังนั้นเมื่อสัญญาณถูกส่งออกม่าจาก UART จึงต้องส่งเข้าสู่วงจรขับเพื่อปรับระดับแรงดันให้ได้ระดับสัญญาณเป็นไปตามมาตรฐาน RS-232 ก่อนส่งออกไปจากคอมพิวเตอร์สำหรับอุปกรณ์ต่อเชื่อมปลายทางก็จะต้องมีวงจรขับลักษณะนี้เช่นเดียวกัน เพื่อให้ได้ระดับสัญญาณในระดับเดียวกัน แต่วงจรขับที่ใช้ทั้งภายในคอมพิวเตอร์และอุปกรณ์ต่อเชื่อมปลายทางนั้นจะถูกกลับสถานะ ดังแสดงเป็นบล็อกไดอะแกรมในรูป2.16



รูปที่2.16 ไดอะแกรมแสดงโครงสร้างทางฮาร์ดแวร์ของพอร์อนุกรม

2.4.8 แอดแดรสของพอร์ตอนุกรม

แอดเดรสพื้นฐานของพอร์ตอนุกรมมี 4 ตำแหน่งดังนี้คือ

COM1 : 3F8H

COM2 : 2F8H

COM3 : 3E8H

COM4 : 4E8H

เมื่อเริ่มเปิดเครื่องเพื่อใช้งานคอมพิวเตอร์ ไบออสภายในคอมพิวเตอร์จะทำการตรวจสอบแอดแดรสของพอร์ตอนุกรมทั้งหมด ถ้าไบออดตรวจสอบพบแอดเดรสของพอร์ตอนุกรมไบออสจะนำแอดดเดรสที่ตรวจสอบไปเก็บไว้ในหน่วยความจำขนาด 2 ไบต์ สำหรับพอร์ตอนุกรม COM1 จะเก็บไว้ที่แอดเดรส 0000 : 0400H และ 0000 : 4010H ส่วนตำแหน่งอื่นๆมีรายละเอียดดังนี้

COM2 = 0000 : 0402H-0000 : 0403H

COM3 = 0000 : 0404H-0000 : 0405H

COM4 = 0000 : 0406H-0000 : 0407H

นอกจากนี้ที่หน่วยความจำแอดเดรส 0000 : 0411H ยังใช้สำหรับแสดงจำนวนของพอร์ตอนุกรมที่มีอยู่ในคอมพิวเตอร์อีกด้วยโดยมีรายละเอียดแสดงในตารางที่2

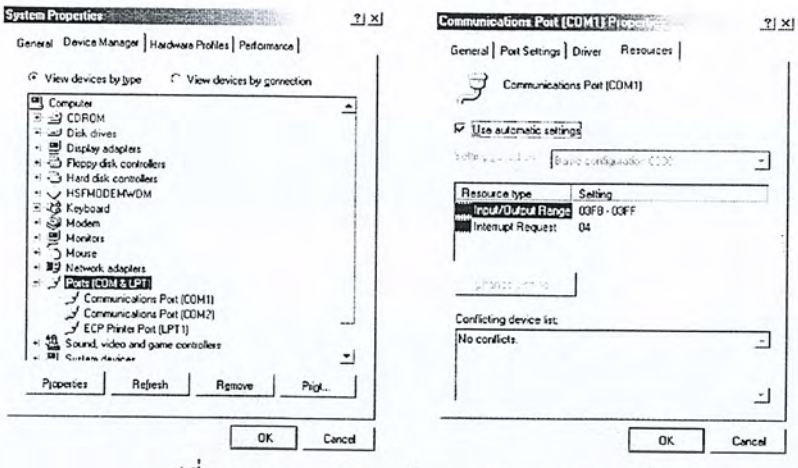
บิต	บิต	บิต	จำนวนพอร์ต
0	0	0	ไม่มีพอร์ตอนุกรม
0	0	1	มีพอร์ตอนุกรม 1 พอร์ต
0	1	0	มีพอร์ตอนุกรม 2 พอร์ต
0	1	1	มีพอร์ตอนุกรม 3 พอร์ต
1	0	0	มีพอร์ตอนุกรม 4 พอร์ต

ตารางที่ 5 แสดงข้อมูลในแอดแดรส 0000:0411H

2.4.9 การหาค่าตำแหน่งแอดเดรสของพอร์ตอนุกรม

การหาค่าตำแหน่งแอดเดรสของพอร์ตอนุกรมสามารถทำได้ดังนี้

สามารถดูได้จากวินโดวส์ โดยไปที่ Control Panel เรียก System $\Rightarrow$ Device Manager $\Rightarrow$ Port(COM & LPT) จากนั้นเลือกพอร์ตอนุกรมที่ต้องการดูค่าดังรูปแล้วเลือก Propertes $\Rightarrow$ Resource ดังแสดงในรูป2.17 ซึ่งในหน้าต่างนี้จะแสดงทั้งตำแหน่งแอดเดรสของพอร์ตอนุกรมนั้นๆ รวมถึงตำแหน่งของอินเตอร์รัปต์ที่ใช้ด้วย



รูปที่ 2.17 แสดงตำแหน่งแอดเดรสของพอร์ทอนุกรม

2.5 คอนโทรลภายในโปรแกรม Visual Basic

Visual Basic เป็นเครื่องมือที่ช่วยพัฒนาแอปพลิเคชั่นสำหรับวินโดวส์ตัวแรกที่ประสบความสำเร็จอย่างมาก ทั้งนี้เนื่องจากแนวความคิดที่จะนำเอาความสามารถของคอนโทรลมาใช้ในการออกแบบโปรแกรม เพราะคอนโทรลเป็นเครื่องมือที่ช่วยลดความซับซ้อนในการเขียนโค้ดโปรแกรมลงได้มาก และนอกจากนี้คอนโทรลยังมีส่วนที่แสดงผลเพื่อสื่อความหมายของการทำงานระหว่างคอนโทรลและผู้ใช้ได้อีกด้วย ส่วนการใช้งานก็ไม่ได้มีความซับซ้อนเพียงแต่ผู้ใช้ทำการเชื่อมต่อคอนโทรลเข้ากับสภาพแวดล้อมของ Visual Basic จากนั้นก็สามารถที่จะนำมาเพิ่มลงในฟอร์มได้ทันที สำหรับ Visual Basic 6.0 ได้มีการแบ่งคอนโทรลออกเป็น 4 กลุ่มหลักๆดังนี้

1. คอนโทรลภายใน (Intrinsic control) เช่น Combox, CommandButton หรือ PictureBox เป็นต้น ซึ่งเป็นคอนโทรลที่ถูกสร้างลงในสภาพแวดล้อมของ vb.exe ดังนั้นทุกครั้งที่ผู้ใช้โหลด Visual Basic สามารถนำคอนโทรลเหล่านี้ออกจากกล่องเครื่องมือได้ทันที ดังนั้นจึงจัดได้ว่าเป็นคอนโทรลมาตรฐาน (Standard Control) กลุ่มหนึ่งของ Visual Basic

2. คอนโทรลมาตรฐาน (Standard Control) เป็นคอนโทรล ActiveX ที่ถูกสร้างเป็นไฟล์ .ocx ที่แยกออกมาต่างหาก เช่น DBGrid (Apex data-bound grid),MSFIexGrid หรือ CommonDialog เป็นต้น ดังนั้นก่อนที่จะสามารถใช้งานคอนโทรลในกลุ่มนี้ได้เราต้องทำการเชื่อมไฟล์ .ocx เหล่านี้เข้ากับสภาพแวดล้อมของ Visual Basic เสียก่อน โดยใช้คำสั่ง Components ในเมนู Project

3. คอนโทรลร่วมวินโดวส์ (Windows Common Control) เป็นคอนโทรล ActiveX ที่ถูกสร้างเป็นไฟล์ .ocx ที่ต้องใช้งานร่วมกับไฟล์ .dll ของวินโดวส์ เช่น RichTextBox, Slider หรือ StatusBar เป็นต้น เช่นเดียวกับคอนโทรลมาตรฐาน เพียงแต่คอนโทรลในกลุ่มนี้ได้ถูกจัดเป็นคอนโทรลพื้นฐานของวินโดวส์ 95 ซึ่งจะติดมากับวินโดวส์ 95 โดยที่คอนโทรลร่วมกับวินโดวส์จะถูกจัดเก็บลงในไฟล์ conctl32.ocx และ conct232.ocx

4. คอนโทรล ActiveX รุ่นมืออาชีพ (Professional ActiveX Control) เป็นคอนโทรล ActiveX ที่ถูกสร้างเป็นไฟล์ .ocx เช่นเดียวกับคอนโทรลมาตรฐาน เช่น MSComm (Communication), MapiMessages (MAPI message) หรือ MMControl (Multimedia MCI) เป็นต้น แต่คอนโทรลในกลุ่มนี้ได้ถูกสร้างและแจกจ่ายมากับ Visual Basic รุ่น Professional และ Enterprise เท่านั้น

2.5.1 คุณสมบัติแสดงค่าของคอนโทรลที่สำคัญๆที่ได้นำมาใช้กับโครงงาน

คอนโทรลทั้งหมดที่มากับ Visual Basic ไม่ว่าจะเป็นคอนโทรลภายในหรือ ActiveX จะมีคุณสมบัติตัวหนึ่งที่ถูกใช้สำหรับการกำหนดค่า (Value) หรืออ่านค่าจากคอนโทรล และคุณสมบัตินี้ได้ถูกกำหนดให้เป็นคุณสมบัติปกติ (Default) ของคอนโทรล โดยในการเขียนโปรแกรมเราสามารถใช้เพียงชื่อของคอนโทรล (Control Name) โดยไม่ต้องกำหนดคุณสมบัติปกติของทุกๆ คอนโทรลได้โดยไม่เกิดข้อผิดพลาด เช่น คอนโทรล TextBox ก็จะมีคุณสมบัติ Text เป็นคุณสมบัติปกติของคอนโทรล

คุณสมบัติปกติที่สำคัญที่ใช้ในโครงงานมีดังตารางต่อไปนี้

คอนโทรล	คุณสมบัติ
CommandButton	Value
Data	Caption
DBGrid (Data-Bound Grid)	Text
Label	Cation
TextBox	Text
Timer	Enable

ตารางที่ 6 แสดงคุณสมบัติของคอนโทรลที่สำคัญ

2.5.2 การแบ่งกลุ่มของคอนโทรลภายใน

เราสามารถแยกคอนโทรลตามวัตถุประสงค์ของการใช้งานได้ทั้งหมด 4 กลุ่ม

1. คอนโทรลภายในทั่วไป ประกอบด้วยคอนโทรลที่แสดงผลในลักษณะของการเลือกตอบหรือเลือกรายการ เช่น CheckBox หรือ ListBox เป็นต้น

2. คอนโทรลภายในด้านระบบไฟล์ ประกอบด้วยคอนโทรลที่ทำหน้าที่ติดต่อหรือแสดงผลระบบไฟล์ (รวมทั้งไดร์ฟ และ ไดเรกทอรี่ด้วย)

3. คอนโทรลภายในด้านกราฟิก ประกอบด้วยคอนโทรลที่ทำหน้าที่ด้านการแสดงผลกราฟิกด้วยวิธีกราฟิกจของคอนโทรลหรือ ฟังก์ชั่นวินโดวส์ API หรือ ไฟล์กราฟิกในรูปแบบต่างๆ เช่น PictureBox, Shape หรือ Image เป็นต้น

4. คอนโทรลภายในด้านเวลา ซึ่งจะมีอยู่คอนโทรลเดียวได้แก่ Timer ซึ่งมีหน้าที่สร้างเหตุการณ์ที่ตอบสนองเป็นครั้งๆตามช่วงเวลาที่กำหนด

2.5.2.1 คอนโทรลภายใน

คอนโทรลภายใน ก็จะเป็นคอนโทรลพื้นฐานที่ถูกนำไปใช้งานมากที่สุด เพราะจะเป็นกลุ่มของคอนโทรลที่ช่วยในการสื่อสารแบบสองทางหรือเลือกเงื่อนไขจากผู้ใช้ เช่น ทุกๆแอฟพลิเคชั่นจะใช้คอนโทรล CommandButton สำหรับให้ผู้ใช้เลือกยอมรับ (OK) หรือ ยกเลิก (Cancel) หรืออื่นๆตามข้อกำหนดของแต่ละแอปพิลเคชั่น ซึ่งคอนโทรลภายในทั่วไปจะประกอบไปด้วยคอนโทรลดังต่อไปนี้

2.5.2.2 คอนโทรล CommandButton

คอนโทรล CommandButton จะเป็นคอนโทรลที่ถูกนำไปใช้งานมากที่สุด เพราะในการกำหนดให้ผู้ใช้งานเลือก OK หรือ Cancel นั้น เรามักจะใช้คอนโทรล CommandButton เป็นส่วนใหญ่ ดังนั้นจึงถือว่าเป็นคอนโทรลพื้นฐานที่สุดของ Visual Basic เนื่องจากคอนโทรลนี้เป็นปุ่มสำคัญที่ใช้งานในรูปแบบของการคลิกเพื่อยืนยัน ดังนั้นจึงอาจเรียกคอนโทรล CommandButton ได้อีกอย่างหนึ่งว่า Push Butter ในขณะออกแบบคอนโทรล CommandButton ที่วางลงบนฟอร์มจะมีลักษณะดังในรูปที่

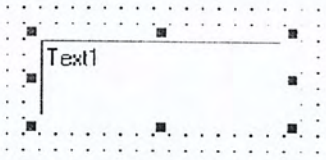
รูปที่2.18 แสดง คอนโทรล CommandButton ขณะวางบนฟอร์ม

เราสามารถแก้ไขข้อความที่แสดงในคอนโทรลนี้ได้ โดยการแก้ไขข้อความในคุณสมบัติ Caption ของคอนโทรลในหน้าต่างคุณสมบัติหรือแก้ไขโค้ดในแอปพลิเคชั่นก็ได้

2.5.2.3 คอนโทรล TextBox

คอนโทรล TextBox มักจะถูกนำไปใช้ทุกๆฟอร์มที่มีการรับการกรอกข้อความจากผู้ใช้ เนื่องจากคอนโทรลนี้ทำหน้าที่แสดงข้อมูล (โดยผ่านทางคุณสมบัติ Text) ในคอนโทรลและยังอนุญาติให้ผู้ใช้สามารถแก้ไขตัวอักษรต่างๆ ของคุณสมบัตินี้ได้อีกด้วย นอกจากนี้แล้วคอนโทรล

TextBox ยังได้รวมเอาความสามารถหลายๆด้านของคอนโทรล Label มาใช้ เช่น สามารถแสดงข้อความได้มากกว่า 1 บรรทัด ความสามารถด้าน DDE (Dynamic Data Exchange) และนอกจากนี้ยังสามารถถูกนำไปใช้ในลักษณะของการกรอกรหัสผ่าน (Password) ได้อีกด้วย แสดงในรูปที่



รูปที่2.19 แสดง คอนโทรล คอนโทรล TextBox ขณะวางบนฟอร์ม

2.5.2.4 คอนโทรล Label

คอนโทรล Label เป็นคอนโทรลในลักษณะของกราฟิกที่ถูกใช้งานด้านการแสดงผลข้อความบนฟอร์ม เหมือนกับผู้เขียนได้นำป้ายข้อความอย่างหนึ่งไปวางไว้บนฟอร์ม เพื่อใช้ในการสื่อของความกับผู้ใช้ และคอนโทรลนี้ผู้ใช้ไม่สามารถแก้ไขได้โดยตรงด้วยวิธีการคีย์หรือใช้เมาส์ในขณะรันแอปพลิเคชั่น นอกเสียจากภายในแอปพลิเคชั่นจะมีการเขียนโค้ดสำหรับแก้ไขข้อความในคอนโทรลโดยการแก้ไขคุณสมบัติ Caption เท่านั้น และนอกจากนี้ Label ยังเป็นคอนโทรลที่มีความสามารถด้าน DDE (Dynamic Data Exchange) ได้อีกด้วย ในขณะออกแบบเราสามารถเพิ่มคอนโทรลลงในฟอร์มหรือตัวบรรจุอื่นๆ ก็จะปรากฏหน้าตาของคอนโทรลดังในรูปที่

รูปที่2.20 แสดง คอนโทรล Label ขณะวางบนฟอร์ม

ในการควบคุมพฤติกรรมของคอนโทรล เราสามารถกระทำได้โดยกำหนดค่าต่างๆ ให้กับคุณสมบัติของคอนโทรล ซึ่งการแก้ไขค่าของคุณสมบัติเราสามารถกระทำได้ในขณะการออกแบบโดยการแก้ไขค่าในหน้าต่างคุณสมบัติ และรันแอปพลิเคชั่นโดยการเขียนโค้ดเพื่อแก้ไขค่าของคุณสมบัติ แต่ก็มีคุณสมบัติบางอย่างที่เราไม่สามารถแก้ไขได้ในขณะรันแอปพลิเคชั่น เช่น Name เป็นต้น

2.5.2.5 คอนโทรล Timer

คอนโทรล Timer เป็นคอนโทรลที่ใช้ในการควบคุมและจัดการเหตุการณ์ด้านเวลา ซึ่งเทียบได้กับประโยค ON TIME GOTO ของ QuickBasic โดยเราสามารถเขียนโค้ดเพื่อทำงานใดๆ เมื่อช่วงเวลาผ่านไปตามค่าที่กำหนด เช่น ทำการปรับการแสดงผลของฟอร์มทุกๆ 1 นาที เป็นต้น โดยที่คอนโทรลนี้จะตอบสนองเหตุการณ์เพียงเหตุการณ์เดียวเท่านั้น แต่เราสามารถกำหนดให้แต่

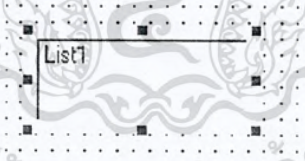
ละฟอร์มมีคอนโทรล Timer ได้มากกว่า 1 คอนโทรล เนื่องจากคอนโทรล Timer เป็นคอนโทรลที่ทำงานตามนาฬิกาของระบบ ดังนั้นมันจึงถูกควบคุมโดยตัวของระบบเอง สำหรับวินโดวส์ 95 และ NT ในทางปฏิบัติจะไม่มีการจำกัดจำนวนของคอนโทรล Timer ในแต่ละฟอร์ม ดังนั้นเราจึงสามารถใช้งานคอนโทรล Timer พร้อมๆกันได้

รูปที่ 2.21 แสดง คอนโทรล Timer ขณะวางบนฟอร์ม

ในขณะออกแบบคอนโทรล Timer ที่วางลงบนฟอร์มให้กับฟอร์ม ก็จะมีลักษณะดังรูปที่ 2.26และเมื่อเรารันแอปพลิเคชั่น คอนโทรลนี้จะไม่ถูกแสดงผล แต่จะมีการทำให้เกิดเหตุการณ์ Timer ทุกครั้งที่ช่วงเวลาครบตามค่าที่กำหนดให้กับคุณสมบัติ Interval ของคอนโทรล Timer

2.5.2.6 คอนโทรล ListView

คอนโทรล ListView เป็นคอนโทรลตัวหนึ่งในชุดคอนโทรลร่วมวินโดวส์ ที่มีความสามารถด้านการแสดงผลได้ในหลายๆรูปแบบ เช่น รูปแบบของ ไอคอนคล้ายๆกับการแสดงผลไอคอนต่างๆของโฟลเดอร์ หรือการแสดงผลข้อมูลจากฐานข้อมูลเป็นต้น ซึ่งเราสามารถที่จะแบ่งการแสดงผลรายการออกเป็นคอลัมน์ได้ตามต้องการ โดยที่รูปแบบที่คอนโทรล ListView แสดงดังรูปที่ 2.27



รูปที่2.22 แสดง คอนโทรล ListView ขณะวางบนฟอร์ม

2.5.2.7 คอนโทรลด้านฐานข้อมูล

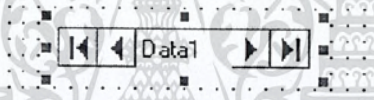
ในโครงงานนี้จะต้องมีการเข้าถึงไฟล์ข้อมูล ดังนั้นตัวแปลภาษาที่เหมาะสมกับการสร้างแอปพลิเคชั่นเหล่านี้ จึงต้องมีเครื่องมือที่สนับสนุนการจัดการกับฐานข้อมูลอย่างง่ายและมีประสิทธิภาพ ดังนั้นจึงเลือกใช้ Visual Basic เป็นตัวแปลภาษาที่มีการสนับสนุนระบบจัดการฐานข้อมูลในรูปแบบของ Microsoft Access โดยอาศัย Jet Database Engine ซึ่งมีเครื่องมือที่ให้โปรแกรมเมอร์สามารถจัดการกับฐานข้อมูลได้ 2 วิธี

2.5.3 Data Control

การเข้าถึงฐานข้อมูลด้วยคอนโทรลด้านฐานข้อมูลจะเป็นวิธีง่ายที่สุดในการเขียนโปรแกรม ซึ่งเพียงแค่กำหนดไฟล์ฐานข้อมูลฟิลด์สำหรับแต่ละคอนโทรลและ คอนโทรล Data เท่านั้น คอนโทรลจะจัดการสร้าง การแสดงผล การแก้ไขฟิลด์ต่างๆของฐานข้อมูลให้โดยอัตโนมัติ หรือตามที่ถูกกำหนดในคุณสมบัติต่างๆ สำหรับความสามารถโดยทั่วไปของคอนโทรลด้านฐานข้อมูลมีรายละเอียดดังต่อไปนี้

- สามารถสร้างตัวแปร Recordest โดยอ้างอิงกับตัวแปร Recordset ที่สนับสนุนโดยคอนโทรลด้านฐานข้อมูล
- แก้ไขโครงสร้างของฐานข้อมูล เช่นการแก้ไขตาราง ฟิลด์ หรือ ดัชนี เป็นต้น
- ค้นหาหรือสืบค้นข้อมูลจากฟิลด์ที่ถูกกำหนดของฐานข้อมูล

คอนโทรลที่สนับสนุนการติดต่อกับฐานข้อมูลนั้นเราสามารถสังเกตได้จากคอนโทรลที่มีคุณสมบัติ Datafield, DataChange หรือ DataSoure เป็นต้น ซึ่งคอนโทรลที่มีคุณสมบัติเหล่านี้จะเป็น

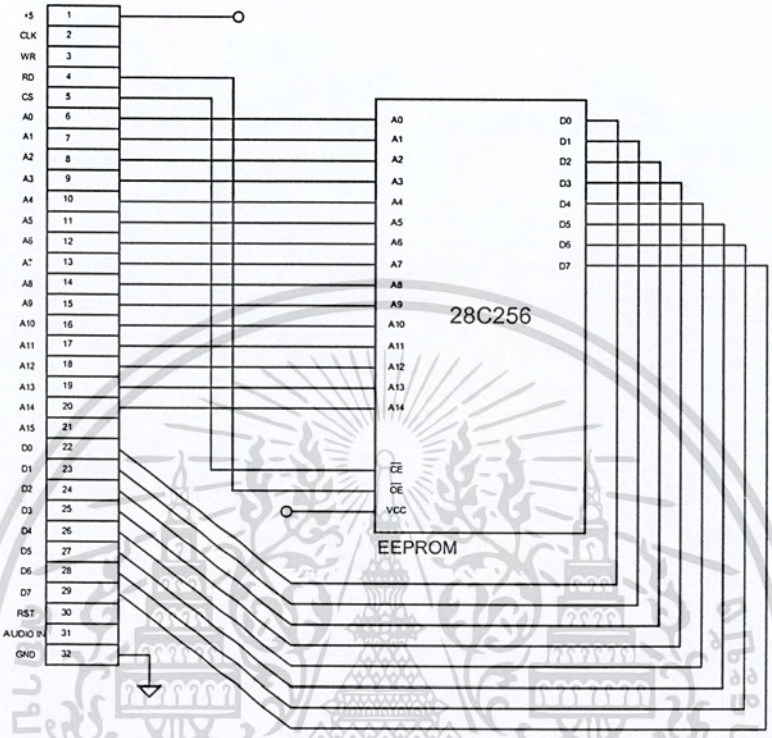


รูปที่2.23 แสดง คอนโทรลดาต้า ขณะวางบนฟอร์ม

คอนโทรลด้านฐานข้อมูลทั้งสิ้น เช่น CheckBox, PictureBox, TextBox, Data, ProgressBar เป็นต้น แสดงในรูปที่ 2.28

บทที่ 3
วงจรที่ใช้ในโครงงาน

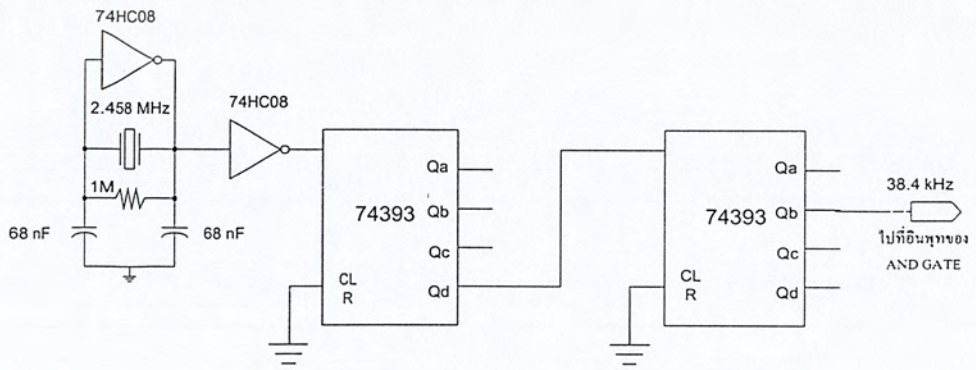
3.1 Cartridge card



รูปที่ 3.1 แสดงวงจรของ Cartridge card

3.2 วงจรกำเนิดสัญญาณความถี่ 38.4 Hz

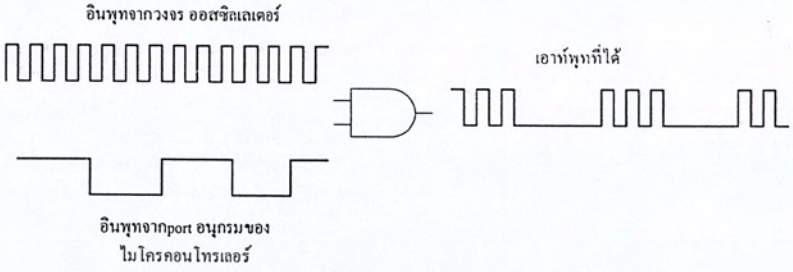
ในการสร้างสัญญาณพาหะ 38.4 kHz เราจะใช้ออสซิลเลเตอร์ความถี่ 2.4576 MHz แล้วนำณฉ 74393 ซึ่งมีคุณสมบัติเป็น Binary counter มาหารความถี่ลงมาให้เหลือ 38.4 kHz ซึ่งค่าที่จะต้องใช้นำมาหารเท่ากับ $2.4576*10^6/38.4*10^3 = 64$ ซึ่งค่า 64 นี้เท่ากับ 2^6 เท่ากับต้องหาร 2 ทั้งหมด 6 ครั้งดังรูป



รูปที่3.2 แสดงวงจรกำเนิดสัญญาณ 38.4 kHz

3.3 การรวมสัญญาณจากเกมบอยเข้ากับสัญญาณ 38.4 kHz

การรวมสัญญาณจากคอมพิวเตอร์เข้ากับสัญญาณ 38.4 kHz จะใช้วิธีการมอดูเลทแบบ OOK โดยใช้ and gate ดังรูป



รูปที่ 3.3 แสดงการมอดูเลทแบบ OOK

3.4 วงจรขับหลอดอินฟราเรด

การออกแบบวงจรทรานซิสเตอร์สวิทช์ เพื่อนำไปขับหลอดอินฟราเรด สามารถกำหนดได้ดังนี้ เมื่อพิจารณาวงจรทรานซิสเตอร์แบบคอมมอนอิมิตเตอร์ จะได้ ic เมื่อปิดวงจร เท่ากับ

$$Ic = \frac{Vcc}{Rl}$$

$$\therefore Rl = \frac{Vcc}{Ic}$$

ถ้าต้องการ ic = 20mA โดยที่ Vcc = 5V.

$$Rl = \frac{5V}{20mA} = 250\Omega$$

เราสามารถหา ib ได้จากความสัมพันธ์

$$Ic = \beta Ib$$

ให้ $\beta = 100$

$$Ib = \frac{20mA}{100} = 0.2mA$$

สามารถหา Rb ได้จากสมการ

$$Vc = ibRb + V_{BE}$$

ให้ $V_C = 5V, V_{BE} = 0.7V$

$$\therefore Rb = \frac{Vc - Vbe}{Ib}$$

$$\therefore Rb = \frac{5-0.7}{0.2mA} = 21500\Omega$$

ในการต่อวงจรทรานซิสเตอร์เราเลือกใช้ Rb = 2kΩ และทรานซิสเตอร์เบอร์ 2SC1815 ที่มี Vbe = 0.7 V, β = 120 เพราะฉะนั้นจะได้ iB เท่ากับ

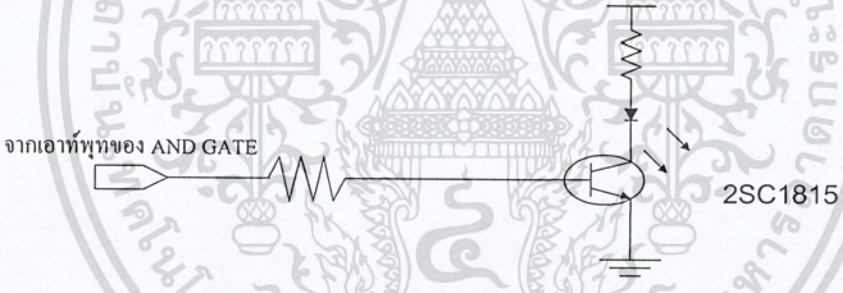
$$\therefore Ib = \frac{5-0.7}{2000} = 2.15mA$$

$$\therefore Ic = \beta Ib = 0.258A$$

เนื่องจากเราจะใช้ทรานซิสเตอร์สวิทช์ในการขับหลอดอินฟราเรด เราจึงนำหลอดอินฟราเรดมาแทนที่ RL ถ้าเราสมมุติว่าหลอดและทรานซิสเตอร์มีความต้านทานต่ ำมาก (ประมาณ 100Ω)เพราะฉะนั้นจะมีกระแส iC เท่ากับ

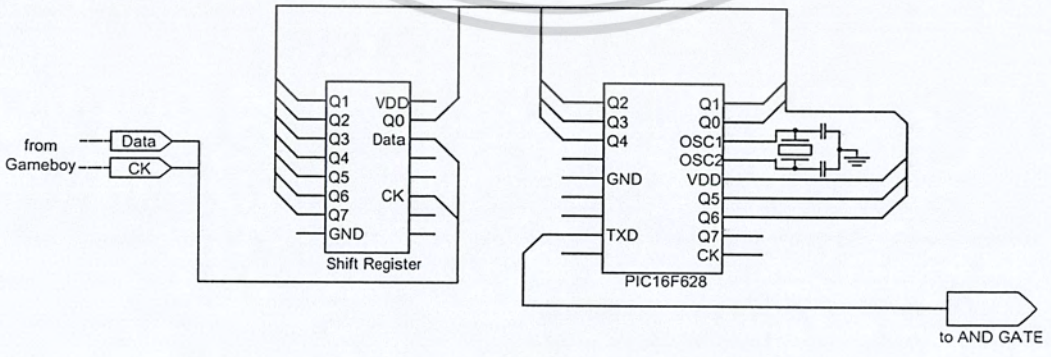
$$\therefore Ic = \frac{5}{100} = 0.05A$$

ซึ่งถ้าใช้ Rb=2000Ω จะให้ ic ได้มากที่สุด = 0.258A เราจึงสามารถใช้วงจรนี้เป็นสวิทช์ขับหลอดอินฟราเรดได้



รูปที่ 3.4 แสดงวงจรขับหลอดอินฟราเรด

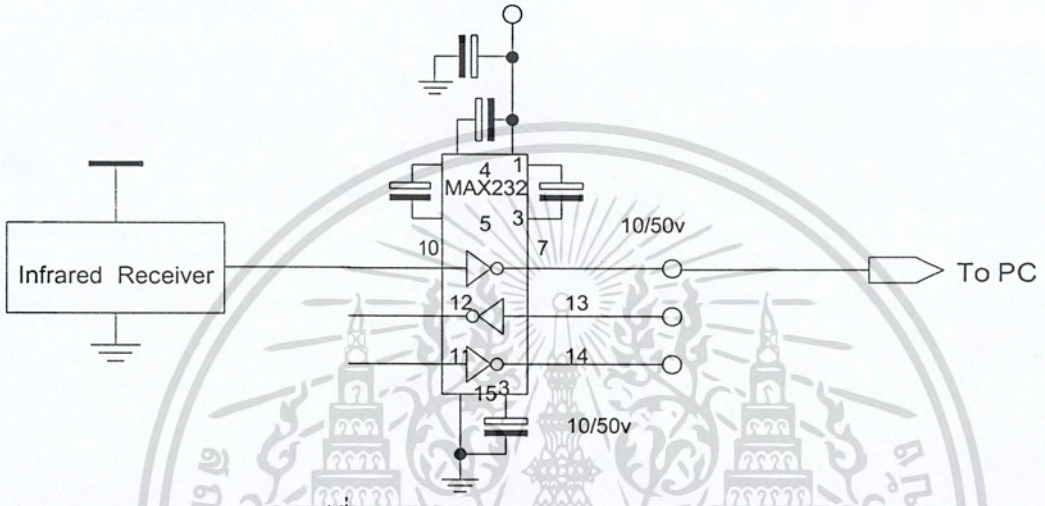
3.5 วงจรส่งข้อมูลแบบอนุกรม



รูปที่ 3.5 แสดงวงจรส่งข้อมูลแบบอนุกรม

หลักการทำงานเริ่มจากการที่ไอซี ชิฟรีจิสเตอร์ได้รับข้อมูลอนุกรมจากเครื่องเกมบอย ก็จะทำการเปลี่ยนข้อมูลจากอนุกรมเป็นแบบขนาน แล้วส่งไปยังไมโครคอนโทรเลอร์ เพื่อให้ไมโครคอนโทรเลอร์ทำการเรียงข้อมูล แล้วส่งออกไปยังขา TX ซึ่งเป็นพอร์ท UART ส่งไปยังวงจรที่ทำหน้าที่มอดูเลทสัญญาณ จากนั้นจึงส่งสัญญาณไปยังวงจรขับเพื่อขับ อินฟราเรด LED เพื่อส่งสัญญาณไปยังภาครับต่อไป

3.6 วงจรภาครับสัญญาณ



รูปที่ 3.6 แสดงวงจรของภาครับสัญญาณ

เมื่อตัว Infrared receiver ได้รับสัญญาณแล้วจะกรองเอาความถี่ 38.4 kHz ออกให้เหลือแต่ข้อมูลที่ส่งมาแล้วจึงส่งข้อมูลมายังไปยังไอซี MAX232 เพื่อแปลงระดับแรงดันให้สามารถสื่อสารกับคอมพิวเตอร์ได้

บทที่ 4
การพัฒนาโปรแกรม

4.1 การใช้โปรแกรม Gameboy Developmentkit (GBDK)

Function ที่เกี่ยวกับ Background

void hide_bkg; // Disable background

เมื่อเราใช้คำสั่งนี้จะเป็นการปิด background tiles ทั้งหมด ยกเว้น sprites ซึ่งคำสั่งนี้จะใช้ไม่ได้ผลกับตัวอีมูเลเตอร์เกมส์บอย

void show_bkg; // Enable Background

เป็นคำสั่งที่จะใช้เพื่อ Turn on Background tiles ทั้งหมด ซึ่งคำสั่งนี้จะใช้หลังจากใช้คำสั่ง hide_bkg(); แล้ว

void scroll_bkg(int x,int y);

เป็นคำสั่งที่จะใช้เพื่อเลื่อน background ในส่วนของ int x และ int y คือตำแหน่งเริ่มต้นของ x และ y ซึ่งมีค่าตั้งแต่ 0-255

void set_bkg_data(int first_tile,int nb_tile,unsinged char*data);

เป็นคำสั่งที่จะใช้เพื่อ load background tile ลงในหน่วยความจำ ในส่วนของ int nb_tile คือ จำนวนของ tile ที่ต้องการจะโหลดเข้ามาไว้ในหน่วยความจำ unsigned char*data คือ ชื่อของ tile data ที่เรากำหนดขึ้นมา

void set_bkg_tile(int x,int y,int w,int h,unsinged char*tile list);

เป็นคำสั่งที่จะใช้เพื่อเป็นการวาง tile data ลงบนหน้าจอ ซึ่งเราจะต้องโหลด tile data ลงในหน่วยความจำก่อนที่จะใช้คำสั่งนี้ โดยที่ค่าของ x มีค่าตั้งแต่ $0 \leq X \leq 31$; $0 \leq Y \leq 31$; $1 \leq W \leq 32-X$; $1 \leq h \leq 32-Y$

Function ที่เกี่ยวกับ ส่วนแสดงผล (Display)

void display_on();

void display_off();

Function ที่เกี่ยวกับ Sprites

void show_sprites();

คำสั่งนี้จะเป็นการ Enable sprites เพื่อให้ sprites ที่กำหนดแสดงผลบนหน้าจอ

void hide_sprites();

คำสั่งนี้จะเป็นการ Disable Sprites เพื่อไม่ให้ Sprites แสดงผลบนหน้าจอ

Void sprites 8*8();

คำสั่งนี้จะเป็นการset ค่าความกว้างของและความยาวของsprites ให้มีขนาด 8*8 pixels

Void sprites 8*16();

คำสั่งนี้จะเป็นการset ค่าความกว้างของและความยาวของsprites ให้มีขนาด 8*16 pixels

Void set_sprites_data(int first_tile,int nb_tiles,unsignd char*data);

คำสั่งนี้เป็นการโหลด sprites data]'.o sprites ram area โดยคำสั่งนี้ sprites จะบรรจุอยู่ใน pattern ที่คุณได้ออกเอาไว้ first tile คือ tile ค่าแรกจาก data ที่ถูกโหลด nb_tile คือจำนวนของ tile ที่เราต้องการจะโหลดและ unsignd char*data คือชื่อของ sprite เรากำหนดขึ้นมา

Void set_sprites_prop(int nb,int prop);

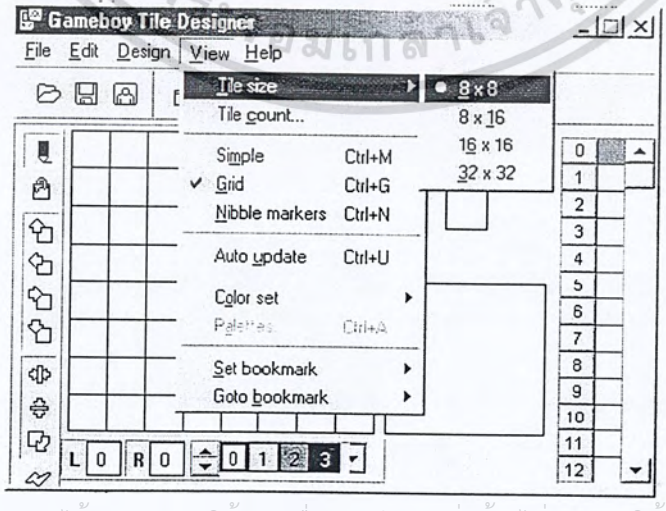
เป็นการเซ็ทคุณสมบัติของแต่ละ sprites ว่าให้มีคุณสมบัติอย่างไร

Void move_ sprites(int nb,int x,int y)

เป็นการเลื่อน sprites ไปมาในตำแหน่งที่กำหนด เมื่อ int nb คือจำนวนหมายเลขของ spritesที่ต้องการเลื่อนส่วน int x คือการกำหนดจุดเริ่มต้นของตำแหน่ง x และ int y คือการกำหนดจุดเริ่มต้นของตำแหน่ง y

4.2 การใช้โปรแกรม Gameboy Tiles Editor

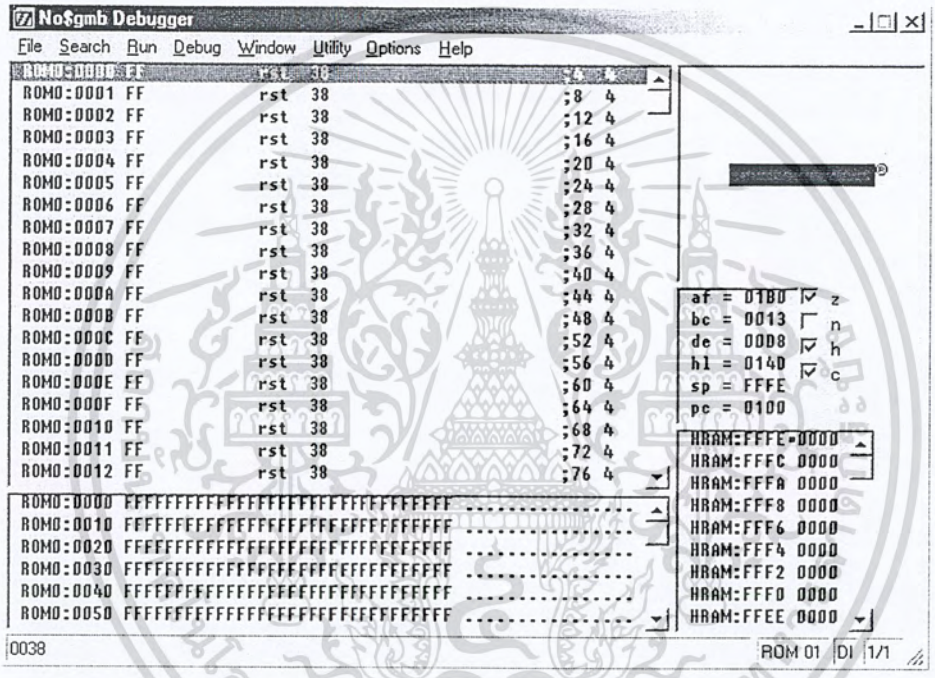
Gameboy Tiles Editor เป็นโปรแกรมที่ช่วยสร้างภาพหรือตัวอักษรในลักษณะต่างๆที่ประกอบกันขึ้นมาเป็นบล็อก ดังที่ได้แสดงในรูป โดยเราสามารถที่จะกำหนดขนาดของแต่ละบล็อกได้เช่น 8*8ม8*16ม16*16 และ 32*32 ตามความต้องการดังแสดงในรูปที่



รูปที่4.1 การกำหนดขนาดของTiles

4.3 การใช้โปรแกรม Gameboy Emulator

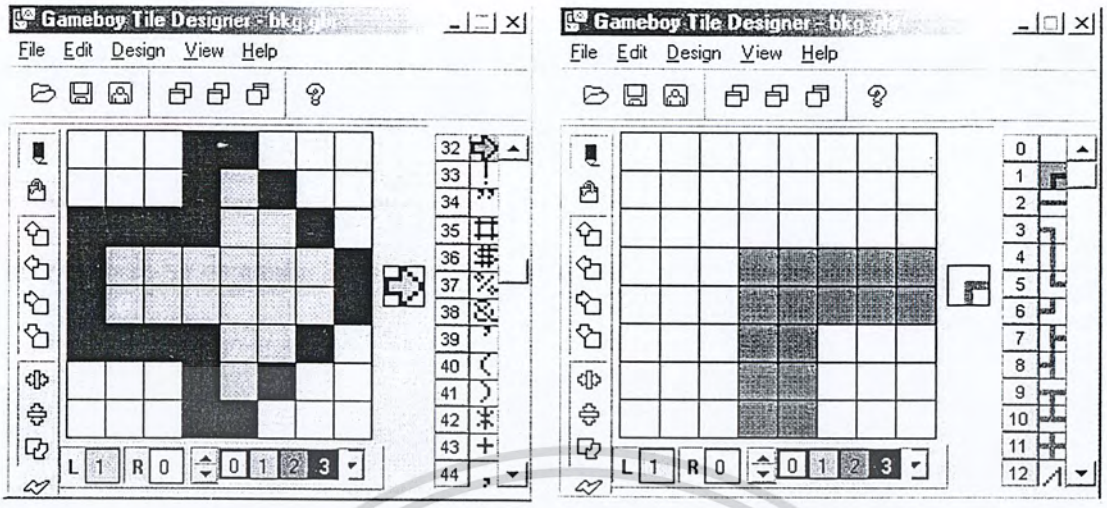
Emulator Program เป็นโปรแกรมที่จำลองการทำงานของเกมส์บอยบนเครื่องPC ซึ่งในโครงงานนี้เราจะใช้โปรแกรม Emulator ที่มีชื่อว่า NOSGMB ซึ่งโปรมแกรมนี้เป็นโปรแกรมเกมส์บอย อีมูเลเตอร์สำหรับเกส์มบอยประเภท pocket gameboy, super gameboy, และ color gameboy และนอกจากนั้นแล้วขณะที่เรากำลังใช้งานโปรแกรมนี้จะให้source code ภาษา assemmble และเลขฐานสิบหกพร้อมกันในคราวเดียว ดังรูปที่ โดยจะเรียงตาม memory address และเรายังสามารถที่จะดูข้อมูลที่ถูกโหลดเข้าสู่memoryที่addressต่างๆได้ในลักษณะ Graphics user interface ซึ่งข้อมูลที่สามารถเรียกดูได้ในลักษณะนี้จะมีตั้งแต่ Tiles data ไปจนถึง sprites data เป็นต้น



รูปที่4.2 แสดงโปรแกรม Gameboy Emulator NO$GMB

4.4 ขั้นตอนการพัฒนาโปรแกรมบนตัวเกมส์บอย

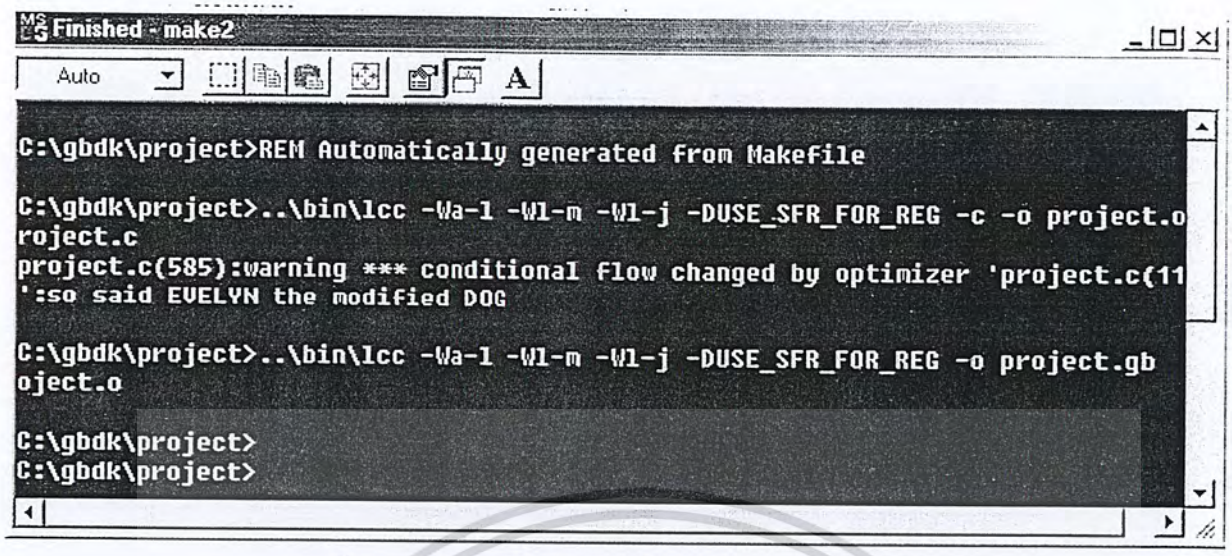
1. ทำการสร้างในส่วนของframe และลูกศรของเมนูด้วยโปรแกรม Gameboy Tiles Editor



รูปที่4.3 แสดงการสร้าง Sprites และ Background

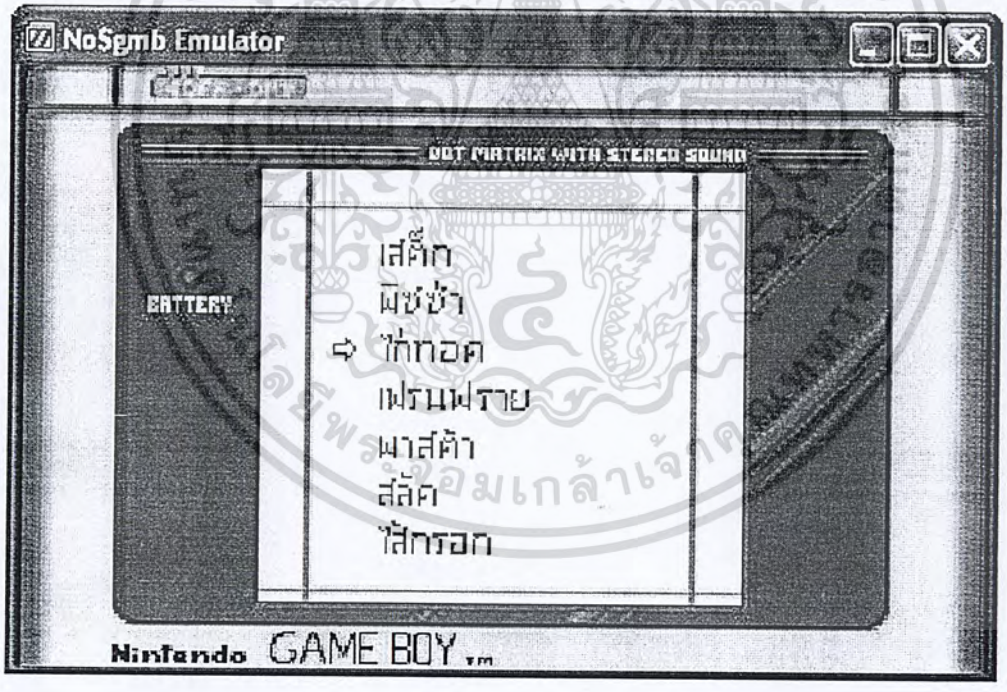
จากนั้นก็ทำการexport file ที่เขียนไว้ไปเป็นไฟล์ในฟอร์แมตของ *.c ซึ่งเป็นรูปแบบที่ตัวคอมไพล์เลอร์ของตัวโปรแกรม GBDK สามารถเข้าใจได้ และเพื่อที่จะได้นำไปประกาศใช้ในโปรแกรมหลักต่อไป

2. ทำการประกาศ Header file ที่จำเป็นต้องใช้ในโปรแกรมหลัก ซึ่ง Header file ที่จำเป็นต้องใช้ได้แก่ gb.h, stdio.h, stdlib.h, console.h (อย่าลืมเขียนอธิบายรายละเอียดไว้ด้วย)
3. หลังจากที่ได้ทำการประกาศ Header file ที่สำคัญๆไว้ในโปรแกรมแล้วก็จะต้องทำการเขียนในส่วนของโปรแกรมหลักซึ่งจะมีเป้าหมายหลักคือ ต้องการให้เกมส์บอยแสดงเมนูที่หน้าจอและสามารถเลือกเมนูต่างๆแล้วส่งค่าออกทางport ให้ได้ ซึ่งจะประกอบไปด้วยfunctionย่อยต่างๆ เช่น ส่วนของfunctionที่ใช้ในการเลื่อนลูกศร(ซึ่งก็คือการ control sprites) ส่วนของฟังก์ชั่นที่ใช้เปรียบเทียบตำแหน่งของลูกศรกับลำดับของเมนู หรือจะเป็นส่วนของfunctionที่ใช้ในการส่งค่าออก หรือส่วนของfunction joypad และส่วนของโปรแกรมวนลูปเพื่อให้frameที่เขียนขึ้นนั้นได้มาเรียงในตำแหน่งที่ต้องการ
4. ทำการคอมไพล์โปรแกรมที่เขียนขึ้นมาด้วย complier ของโปรแกรม GBDK (Gameboy Development kit) ซึ่งผลที่ได้จากการคอมไพล์คือไฟล์ *.gb ซึ่งมีรูปแบบเป็น binary file และก็ยังเป็นไฟล์ที่ใช้ในการโปรแกรมข้อมูลลง eprom อีกด้วย



รูปที่4.4 แสดงการใช้ Compiler ของโปรมแกรม GBDK

5. ทดสอบโปรแกรมที่คอมไพล์ โดยใช้โปรแกรมทดสอบประเภท emulator ในที่นี้จะใช้โปรแกรมที่มีชื่อว่า no$gmb



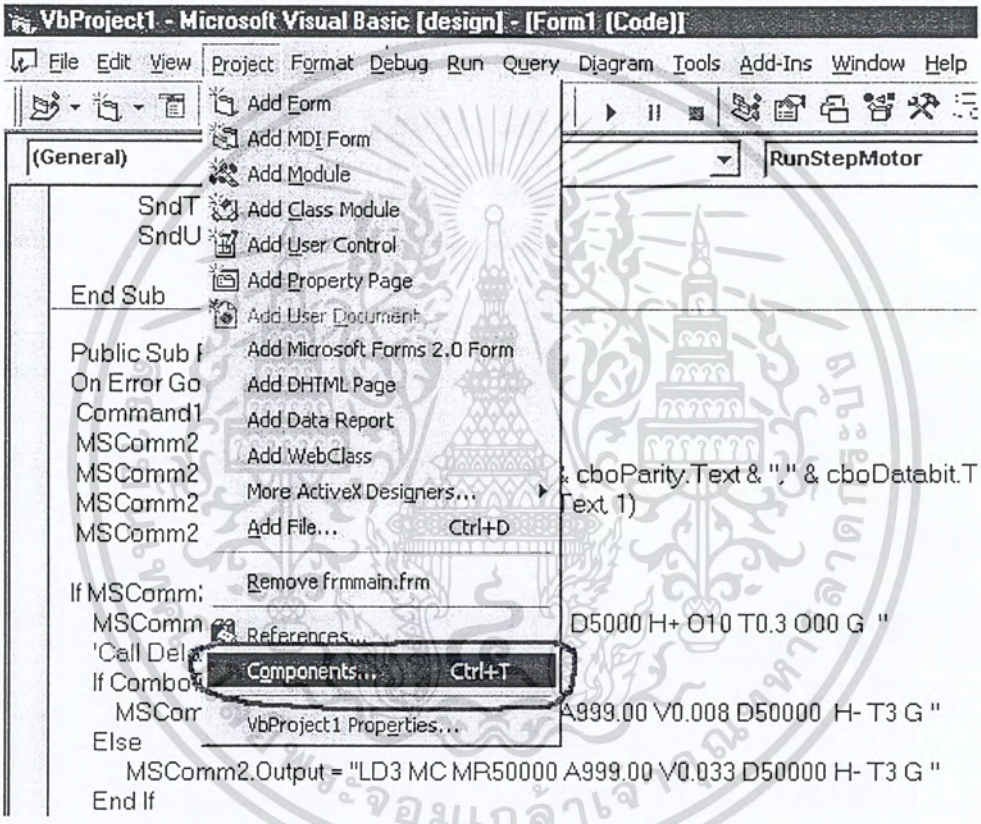
รูปที่ 4.5 แสดงการใช้โปรแกรม No$GMB ทดสอบโปรแกรมที่คอมไพล์ได้

6. ทำการโปรแกรม binary file ที่คอมไพล์ได้ลงใน eprom

4.5 ขั้นตอนการพัฒนาโปรแกรมคอมพิวเตอร์

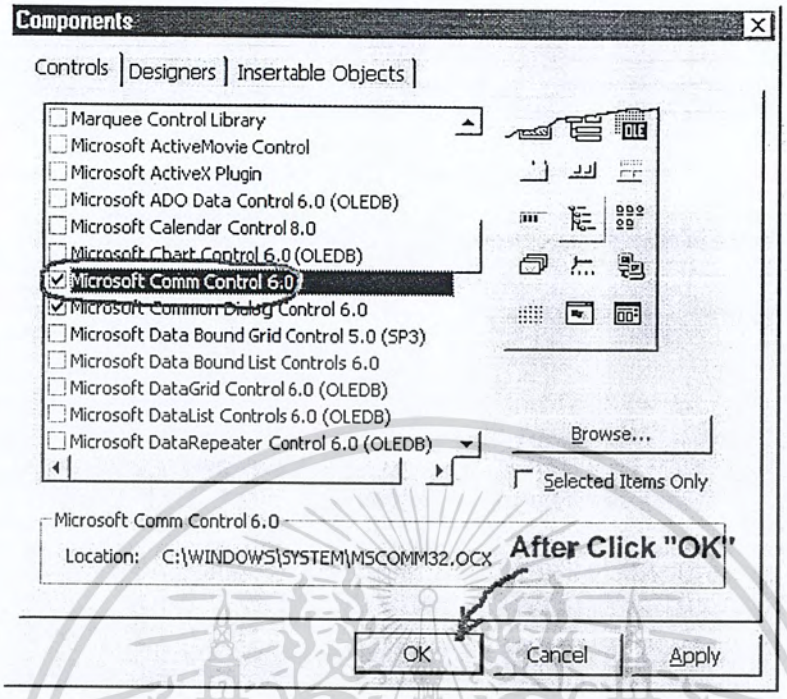
ในโครงงานนี้จะใช้ภาษา Visual Basic ทำการเขียนโปรแกรมควบคุม ซึ่งการที่เลือกภาษา Visual Basic เพราะว่าง่ายต่อการออกแบบ GUI (grific user interface) และมีความสามารถติดต่อกับพอร์ตต่างๆได้ดีไม่แพ้ภาษาคอมพิวเตอร์ตัวอื่นๆ โดยการที่จะติดต่อกับพอร์ตอนุกรมได้นั้นจะต้องใช้ความสามารถของ VB Control ที่ชื่อว่า MSComm ซึ่งเป็น Control ที่ใช้สำหรับการติดต่อกับพอร์ตอนุกรมโดยเฉพาะแต่ ซึ่ง Control ตัวนี้สามารถนำออกมาใช้งานได้โดยทำตามขั้นตอนต่อไปนี้

1. ขั้นที่ตอนแรก เลือกที่เมนูบาร์ด้านบนของโปรแกรม Visual Basic ดังรูปด้านล่าง



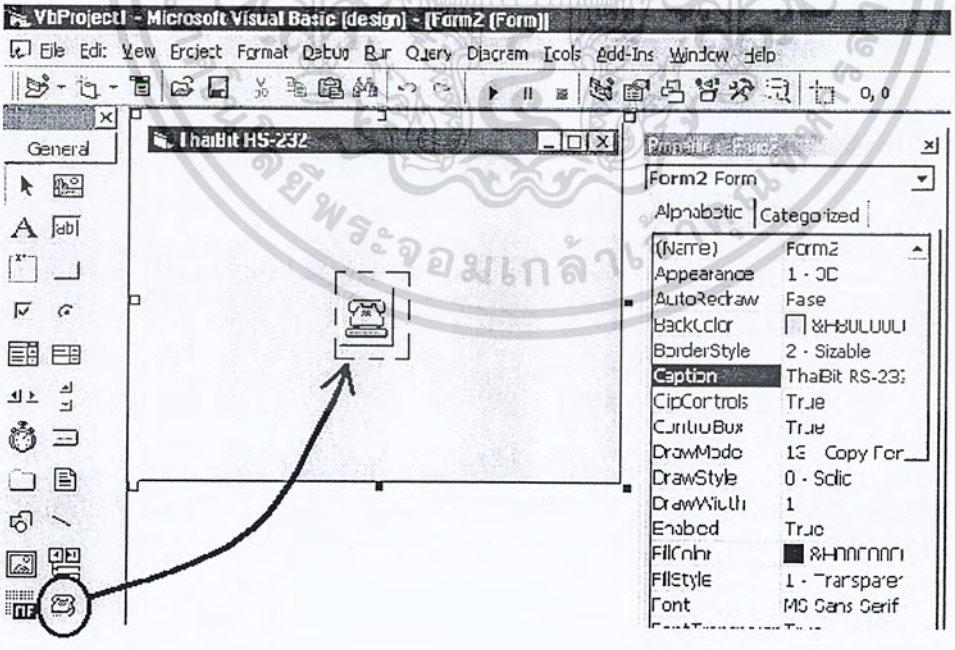
รูปที่4.6 การเรียก MSComm Control ขั้นตอนที่1

2. ขั้นที่สอง เลือกชื่อ Control ชื่อ Microsoft Comm Control 6 ดังรูปด้านล่าง



รูปที่4.7 การเรียก MSComm Control ขั้นตอนที่2

3. ขั้นที่สาม ลากControlชื่อMicrosoft Comm จากToolBox มาไว้บนFormดังรูปด้านล่าง



รูปที่4.8 การเรียก MSComm Control ขั้นตอนที่3

ในตัวคอนโทรล MSComm มี Event ที่ใช้เพียง Event เดียวเท่านั้นเอง ก็คือ OnComm Event ซึ่งจะใช้ในการติดต่อแบบอินเตอร์รัพต์ การเขียนโปรแกรมติดต่อ Serial Port แบบธรรมดาจะใช้ comEvent เพียง comEvReceive,comEvSend ถ้าเป็นการติดต่อสื่อสารแบบ โมเด็มจะใช้หลายตัวในการตรวจสอบสัญญาณ ซึ่งการใช้งานรูปแบบนี้จะมีคำแนะนำอยู่ใน Help Visual Basic อยู่แล้ว

4.5.1 การเขียนโปรแกรมติดต่อกับ Serial Port สามารถทำได้ 2 วิธี คือ

4.5.1.1การติดต่อแบบอินเตอร์รัพต์

ขบวนการอินเตอร์รัพท์ อุปกรณ์รอบข้างเกือบทุกชิ้นจะต้องปฏิบัติงานอยู่เพื่อส่งสัญญาณไปให้แก่ซีพียูเสมอ ถ้าอุปกรณ์นั้นพร้อมที่จะรับส่ง ที่เคยเจอจากการทำโครงงานอุปกรณ์ จะส่งเป็นรหัสแอสกี้ เราจะเขียนโปรแกรมอินเตอร์รัพต์ โดยเมื่อที่ข้อมูลเข้ามาก็จะทำให้มี CommEvent กับ OnComm Event

4.5.1.2การติดต่อแบบโพลลิ่ง

ในระบบพีซี การโพลมีบ้างที่ใช้การส่งผ่านข้อมูลระหว่าง Terminal กับ CPU กรณีข้อมูลเป็นประเภทไบท์ที่ส่งจากคีย์บอร์ด โดยวิธีการนี้จะตรวจสอบ คีย์บอร์ดว่ามีข้อมูลส่งมาหรือเปล่า โดยจะตรวจสอบตลอดเวลา การทำงานกับข้อมูลที่รับเข้ามาจะตรวจสอบด้วยความเร็วที่สูงกว่าอัตราความเร็วข้อมูลที่ส่งเข้ามาทาง คีย์บอร์ด การที่ CPU ส่งสัญญาณออกไปตรวจสอบพบว่ามีข้อมูลที่ต้องส่งเข้ามา เรียกว่า"Wet Poll" ซึ่งจะเสียช่วงเวลา 90 เปอร์เซ็นต์ คาบเวลาที่เสียไปนั้น เราเลี่ยงไปใช้เทคนิค การโพลแบบ "Round Robin" แทน แต่ในVB เราจะใช้การตรวจสอบข้อมูลที่มาจาก Serial Port ตลอด โดยจะใช้ Control Timer เข้ามาช่วยในการเขียนโปรแกรมซึ่งสามารถตรวจสอบได้ถึงระดับ 1 มิลลิวินาที หรือจะใช้ Do....Loop ก็ได้

4.5.2 องค์ประกอบในการใช้ MSComm

การตั้งค่าติดต่อกับพอร์ต

- ComPort คือ เราต้องกำหนดหมายเลข Port ที่ใช้ต่อRS-232 (Com1,Com2)รายละเอียดดูในเมนูด้านซ้าย Serial Port Detail

- Setting คือ เราต้องกำหนดอัตรา Baud,Parity,Data(จำนวนบิต),Stop ตัวอย่าง 1200,n,8,1 เป็นต้น

- HandShaking คือ เราจะกำหนดได้ 4 แบบ 1.comNone 2.comXonXoff 3. comRTS 4.comTRSXonXoff

4.5.3 การใช้ Buffer ในการรับส่งข้อมูล

-InBuffersize คือ การกำหนด Buffer ในการรับข้อมูลเข้ามา

-OutBuffersize คือ การกำหนด Buffer ในการส่งข้อมูลออกไป

- Rthreshold คือ การที่เรากำหนดการเกิด Event-driven ในการรับข้อมูลเข้ามา

- Sthreshold คือ การที่เรากำหนดการเกิด Event-driven ในการรับข้อมูลออกไป

- Inputlen คือ จำนวนของข้อมูลที่ไปอ่านใน Buffer รับข้อมูล

- EOFEnable คือ การที่บอกว่าสิ้นสุดของไฟล์(EOF) End of File

ด้านฮาร์ดแวร์

- ParityReplace คือ ค่าของคาเลกเตอร์ที่จะแทนในเมื่อเกิด Parity Error

- NullDiscard คือ การกำหนดให้รับหรือไม่รับ NULL CHARACTER

- RTSEnableคือ ทำให้มีสัญญาณ RTS (Request To Send)

- DTSEnableคือ ทำให้มีสัญญาณ DTR(Data Terminal Ready)

4.5.4 การกำหนดคุณสมบัติของ MSComm Control ให้สามารถติดต่อกับพอร์ตได้

1. Property ชื่อ CommPort คือ เลือกคอมพอร์ตที่เราจะต่อใช้งาน

ตัวอย่าง MSComm1.CommPort=1

ในที่นี้เลือกจะใช้ Com1อยู่ที่ด้านหลังเครื่องคอมฯ

2. Property ชื่อ Settingsคือ การตั้งค่าของการรับส่งข้อมูล ซึ่งจะต้องรู้ด้วยว่าอัตราบอด ของอุปกรณ์ที่จะติดต่อด้วยเป็นเท่าไร โดยมีรายละเอียดการใส่ต่างๆค่าดังนี้

MSComm1.Settings="Baud(อัตราการรับส่งข้อมูล),Parity(ถ้าไม่ใช้ใส่ N,จำนวนบิตข้อมูล, บิตสต๊อบ"

ตัวอย่าง MSComm1.Settings="1200,N,8,1"

3.Property ชื่อ InputLenคือ กำหนดขนาดขณะที่มีข้อมูลเข้ามาให้ไปอ่านข้อมูลทั้งหมดที่อยู่ในบัพเฟอร์

ตัวอย่าง MSComm1.InputLen=1

4. Property ชื่อ PortOpen คือ จะเปิดให้พอร์ตใช้งานหรือไม่ ถ้าเปิด =True ถ้าปิด =False

ตัวอย่าง MSComm1.PortOpen=True

5. Property ชื่อ Rthresholdคือ ทำให้เกิดการกระตุ้นด้วย Event-driven เมื่อมีข้อมูลในบัฟเฟอร์รับข้อมูล(Comport)มันทำให้เกิดCommEvent ใน OnComm Event

ตัวอย่าง MSComm1.Rthreshold =1

จากรายละเอียดที่กล่าวมา เราจะมาเขียนใน โพรซีเดอร์ VB ซึ่งจะไว้ที่ Sub Form_Load() หรือจะสร้าง Sub ขึ้นใหม่ในกรณีที่จะเรียกใช้ภายหลัง

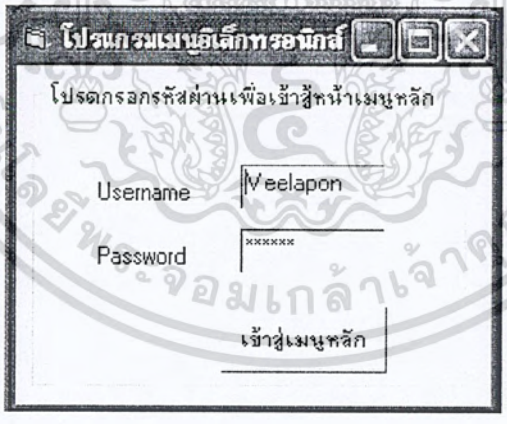
```
Private Sub Form_Load()
    MSComm1.Settings="1200,N,8,1"
    MSComm1.CommPort=1
    MSComm1.InputLen=1
    MSComm1.PortOpen=True
    MSComm1.Rthreshold =1
End Sub
```

จากวิธีเขียนโค๊ดด้านบนเป็นการกำหนดค่าเริ่มต้นให้กับคอมพอร์ตและเปิดใช้การรับและส่งของพอร์ต RS-232 ดังนั้นก็สามารถจะรับและส่งข้อมูลทางพอร์ตได้ โดยใช้ Property ดังนี้

Output =ซึ่งจะเป็นการส่งข้อมูลไปที่พอร์ต

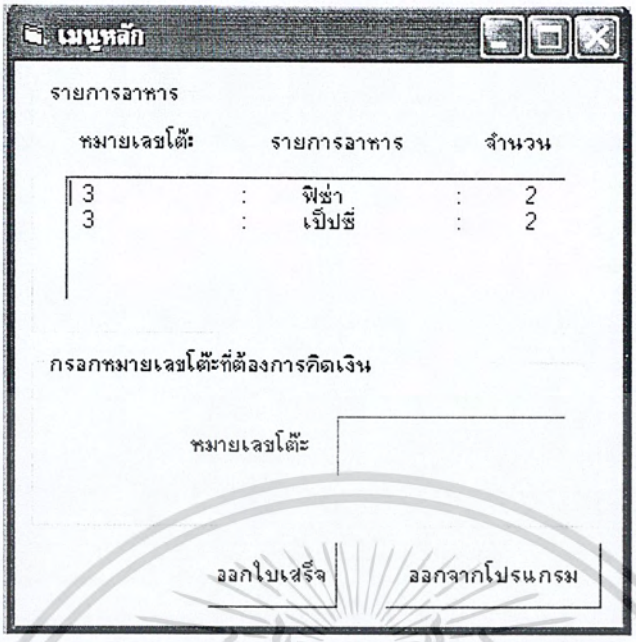
Input =เป็นส่วนของการรับข้อมูลจากพอร์ต แต่ในส่วนนี้จะต้องนำคำสั่งไปเขียนที่ Event Property OnComm จะอยู่ใน Sub MSComm_OnComm ซึ่ง จะอ่านข้อมูลเข้ามาจากทางพอร์ต RS232 นั้นเอง

หลังจากที่กำหนดในส่วนของการติดต่อกับพอร์ต RS232 เรียบร้อยแล้ว ก็จะมาทำในส่วนของ User Interface และการเขียนโปรแกรมติดต่อกับฐานข้อมูล โดยเริ่มจากสร้างฟอร์ม โดยใช้คอนโทรลมาตรฐานต่างๆดังรูปที่



รูปที่ 4.9 แสดงการใส่ Password ก่อนเข้าสู่โปรแกรม

หลังจากนั้นก็เขียนโค้ด ให้เมื่อกดปุ่ม "เขาสู่เมนูหลัก" ก็จะปรากฏเมนูหลักดังรูปที่ ซึ่งเมนูหลักก็จะใช้คอนโทรลมาตรฐานเช่นเดียวกัน ซึ่งในการเขียนโปรแกรมติดต่อกับฐานข้อมูลจะนำค่าที่ได้จากพอร์ตอนุกรมมาทำการเปรียบเทียบค่าจากตารางฐานข้อมูลแล้วนำข้อมูลที่ได้มาแสดงใน Textbox โดยใช้เครื่องมือ Data คอนโทรลในการเชื่อมต่อกับฐานข้อมูลที่เตรียมไว้



รูปที่4.10 แสดงรายการอาหารที่เปรียบเทียบกับฐานข้อมูลและแสดงออกหน้าจอ

ในส่วนของการออกใบเสร็จจะเขียนโค้ด ให้มีการคำนวนราคาอาหารทั้งหมดและให้ข้อมูลทั้งหมดแสดงใน ใบเสร็จดัง ซึ่งใช้เครื่องมือในการสร้างรายงานที่เรียกว่า Data Report

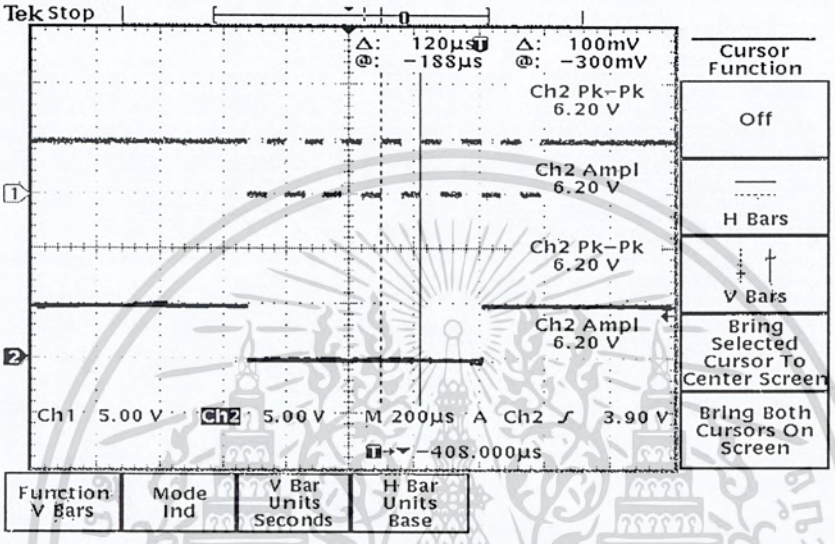
บทที่ 5

ผลการทดลอง

5.1 ผลการทดลองวัดสัญญาณเอาท์พุทของเกมส์บอย

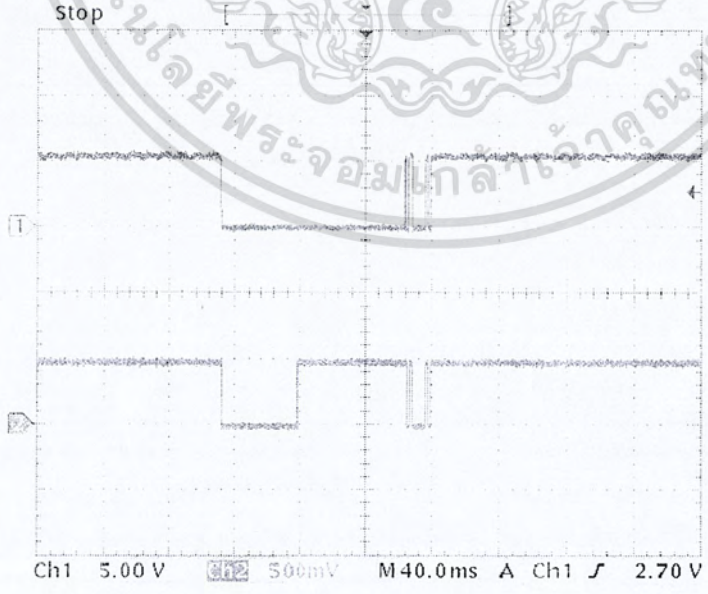
จากรูปข้างล่างเป็นตัวอย่างการวัดสัญญาณจากเอาท์พุทของเครื่องเกมบอยบางตัวอย่าง เมื่อเลือกส่งเมนูที่ต้องการออกทางเอาท์พุทพอร์ท

5.1.1 ผลการทดลองเมื่อเลือกส่งค่า 0x03

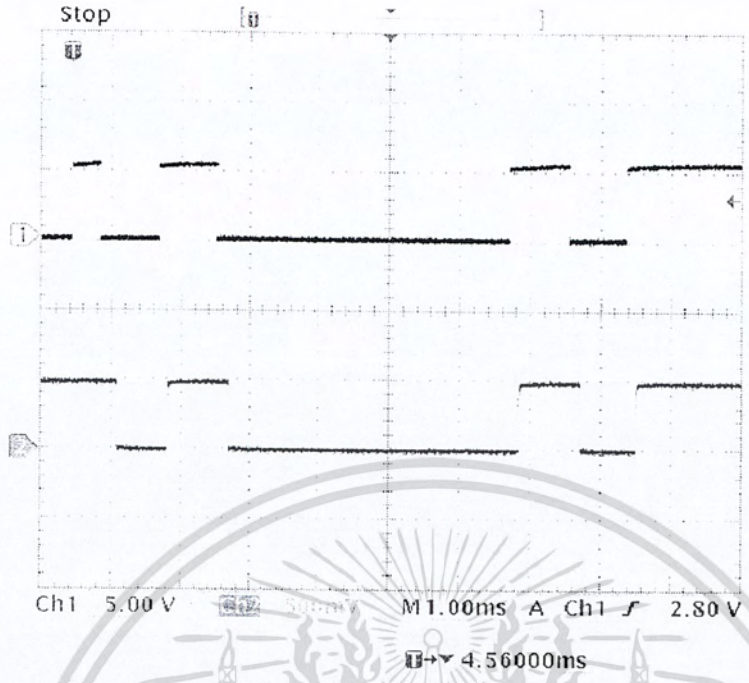


รูปที่5.1 แสดงรูปสัญญาณเมื่อส่งค่า 0x03 จากเกมบอย

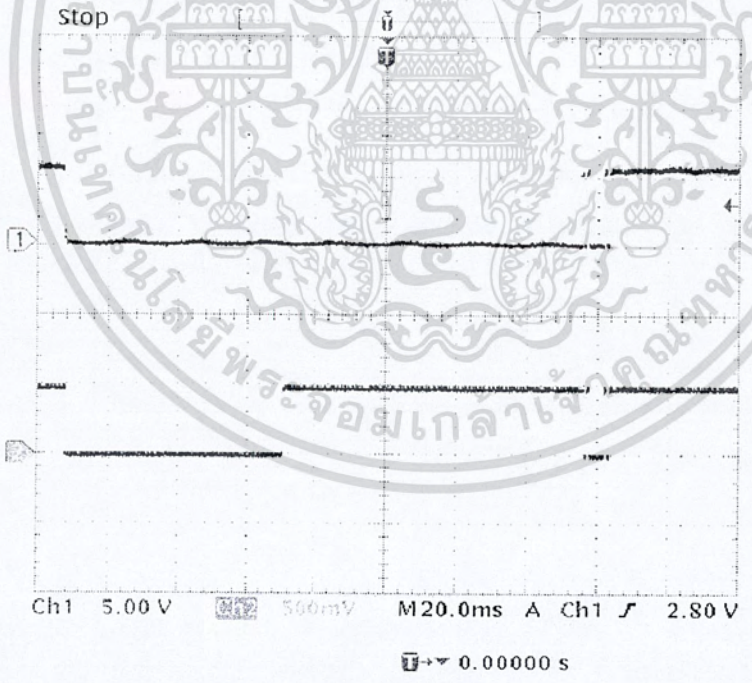
5.2 ผลการทดลองการส่งข้อมูลที่ความเร็วต่างๆ



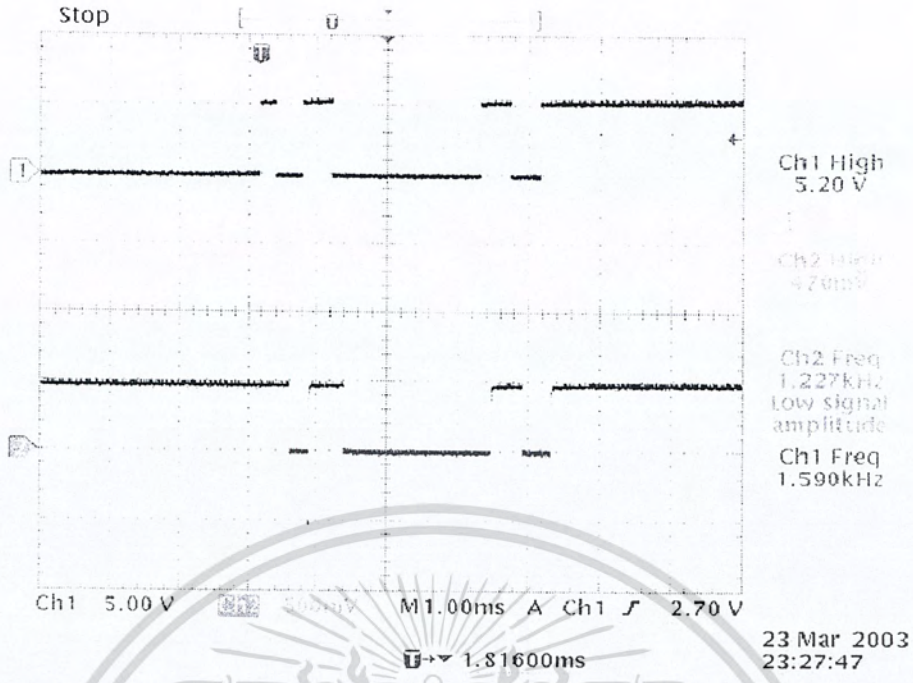
รูปที่ 5.2.1 แสดงรูปสัญญาณอักขระ A ที่ความเร็ว 600 bpsที่ Time Div = 40 ms



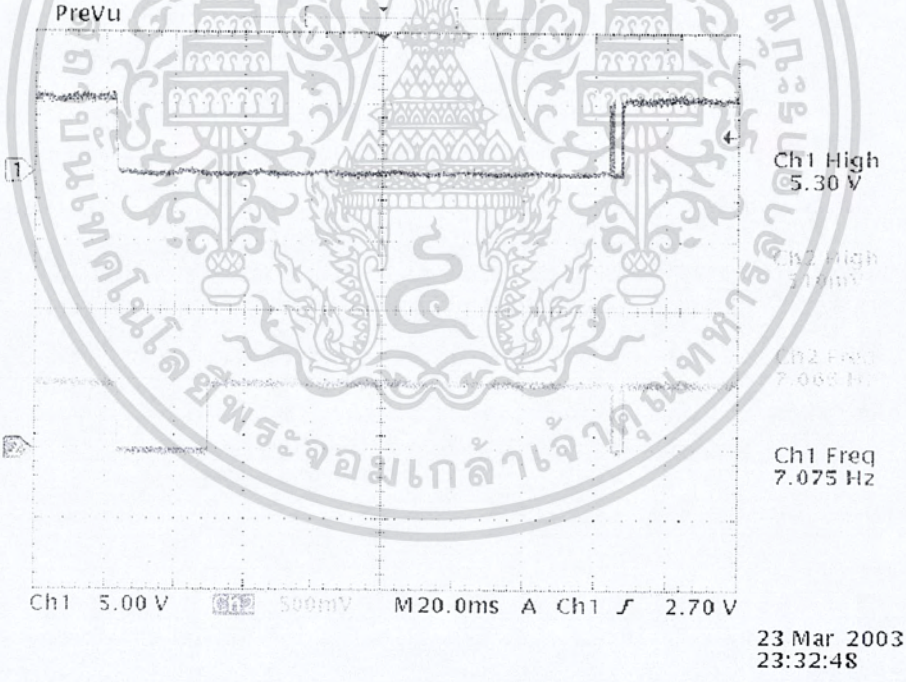
รูปที่ 5.2.2 แสดงรูปสัญญาณอักขระ A ที่ความเร็ว 1200 bps ที่ Time Div = 1 ms



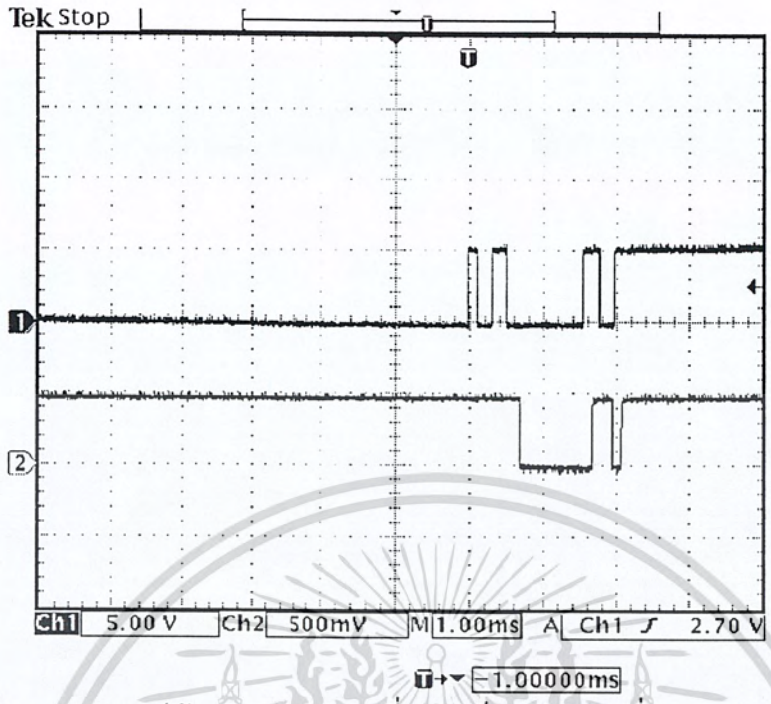
รูปที่ 5.2.3 แสดงรูปสัญญาณอักขระ A ที่ความเร็ว 1200 bps ที่ Time Div = 20 ms



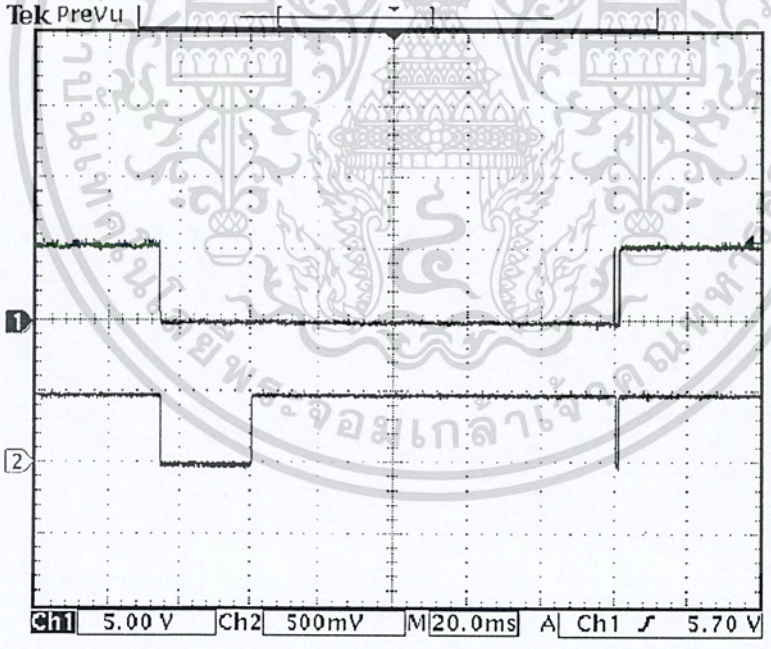
รูปที่ 5.2.4 แสดงรูปสัญญาณอักขระ A ที่ความเร็ว 2400 bps ที่ Time Div = 1 ms



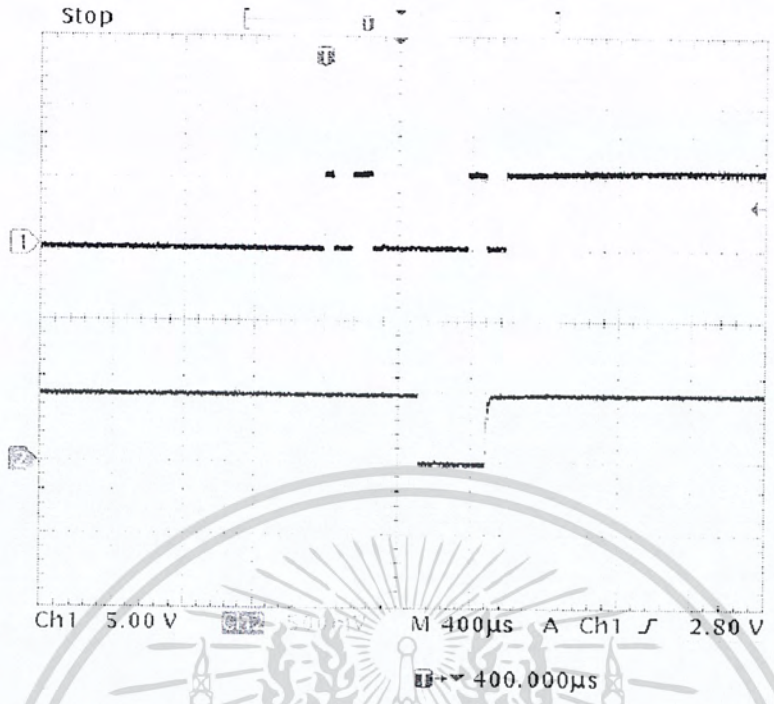
รูปที่ 5.2.5 แสดงรูปสัญญาณอักขระ A ที่ความเร็ว 2400 bps ที่ Time Div = 20 ms



รูปที่ 5.2.6 แสดงรูปสัญญาณอักขระ A ที่ความเร็ว 4800 bps ที่ Time Div = 1 ms

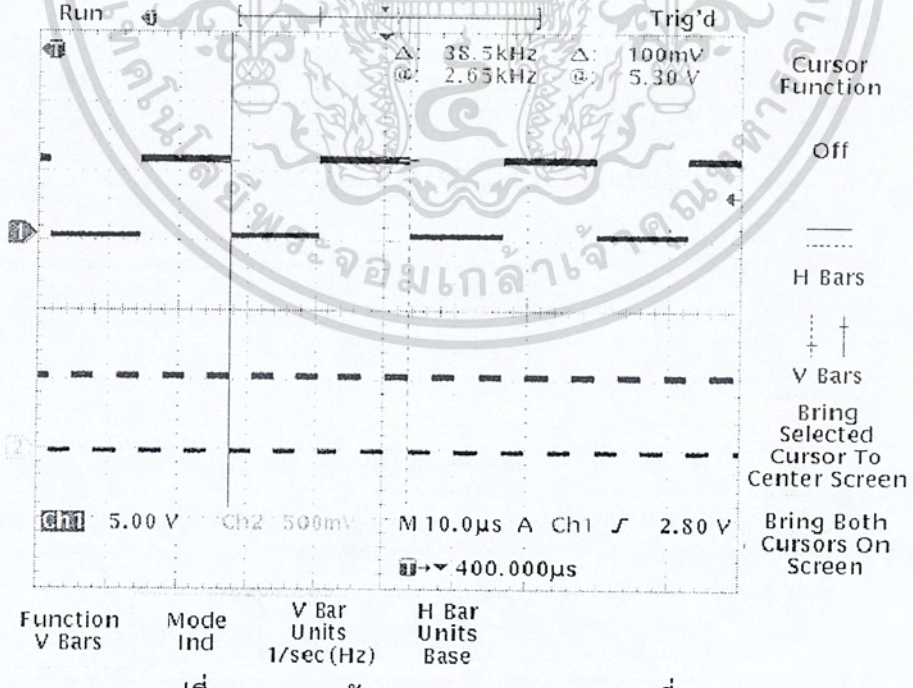


รูปที่ 5.2.7 แสดงรูปสัญญาณอักขระ A ที่ความเร็ว 4800 bps ที่ Time Div = 20 ms



รูปที่ 5.2.8 แสดงรูปสัญญาณอักขระ A ที่ความเร็ว 9600 bps ที่ Time Div = 400 µs

5.3 ผลการทดลองวัดสัญญาณจากวงจรหารความถี่

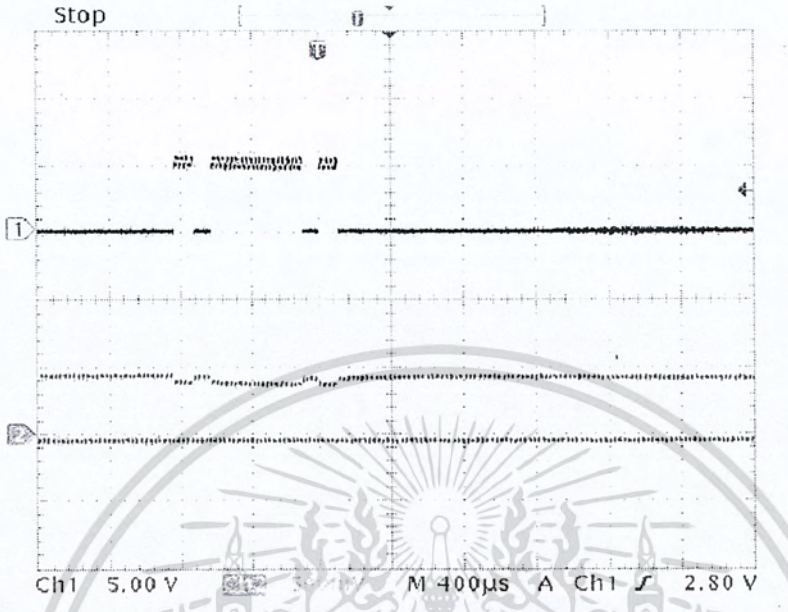


รูปที่ 5.3.1 แสดงสัญญาณจากวงจรหารความถี่

Ch 1 แสดงสัญญาณที่ได้จากการหาร 2 ครั้งที่ 4 ความถี่ 160kHz

Ch 2 แสดงสัญญาณที่ได้จากการหาร 2 ครั้งที่ 6 ความถี่ 38.4kHz

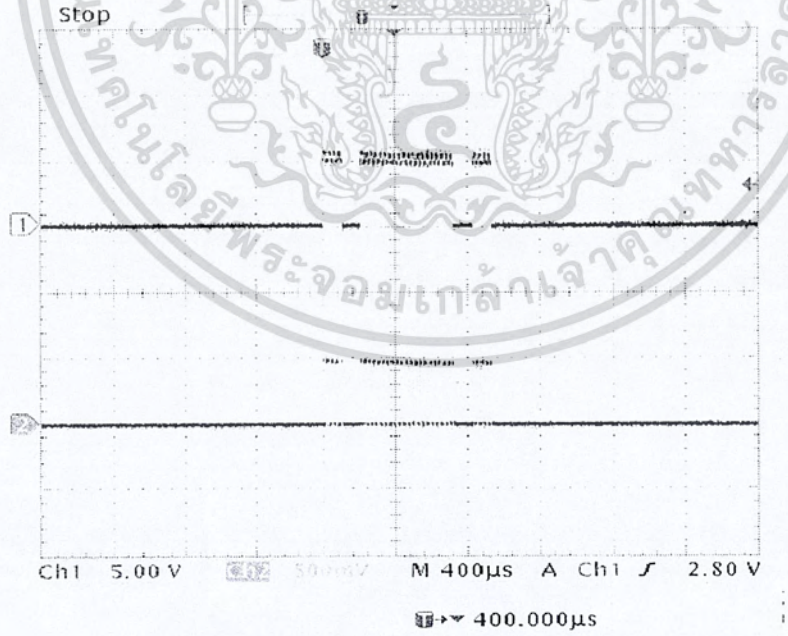
5.4 ผลการทดลองวัดสัญญาณจากการมอดูเลท



รูปที่ 5.4.1 แสดงสัญญาณก่อนการมอดูเลท

Ch 1 แสดงสัญญาณที่เป็นข้อมูล

Ch 2 แสดงสัญญาณที่ได้จากวงจรกำเนิด ความถี่ 38.4kHz

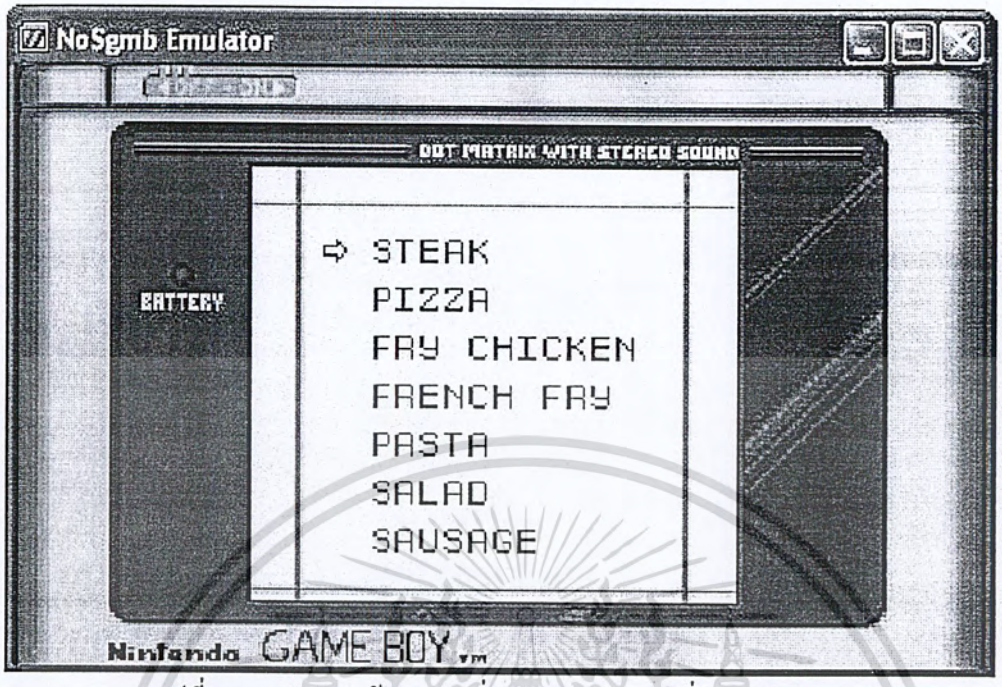


รูปที่ 5.4.1 แสดงสัญญาณที่ถูกมอดูเลทแล้ว

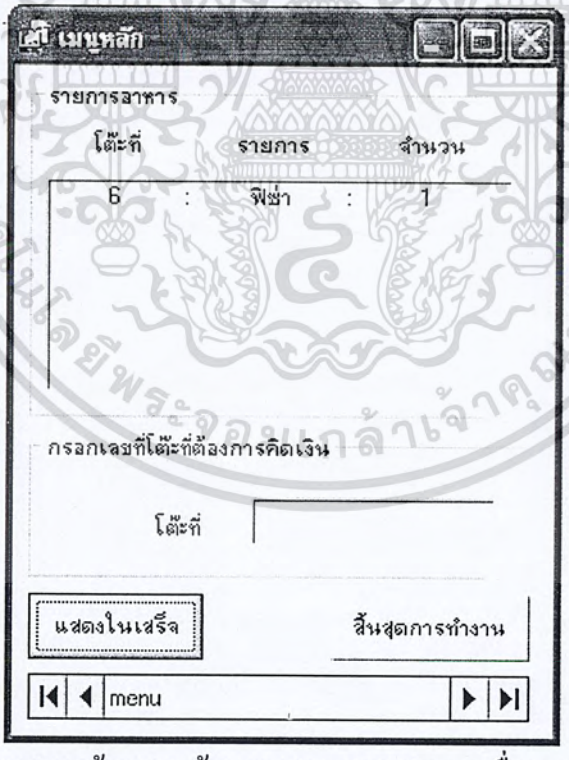
Ch 1 แสดงสัญญาณที่เป็นข้อมูลก่อนการมอดูเลท

Ch 2 แสดงสัญญาณที่ถูกมอดูเลทแล้ว

5.5 ผลการทดลองแสดงหน้าจอเมนูสั่งอาหาร



รูปที่ 5.5.1 แสดงหน้าจอเมนูสั่งอาหารทางจอเครื่องเกมบอย



รูปที่ 5.5.2 แสดงหน้าจอเมนูข้อมูลรายการอาหารทางเครื่องคอมพิวเตอร์

บทที่6
สรุปผลและวิจารณ์

จากการที่ได้ศึกษาและทดลองในโครงงานนี้ ในส่วนของการเขียนโปรแกรมคอนโทรลเครื่องเกมบอยจะพบว่า ในภาคของการแสดงผลออกทางหน้าจอ สามารถทำได้ในระดับหนึ่งเท่านั้น โดยทำการส่งรายการอาหารได้เพียงครั้งละ 1 รายการอาหารเท่านั้น ยังไม่สามารถเลือกหลายๆเมนูแล้วเก็บค่าเอาไว้ดูได้เนื่องจาก การเขียนโปรแกรมสำหรับคอนโทรลเครื่องเกมบอยนั้น ใช้ภาษา C ในการเขียน แต่การประกาศใช้ฟังก์ชั่นต่างๆนั้น มีความแตกต่างจากภาษา C ที่ใช้เขียนสำหรับคอมพิวเตอร์เป็นอย่างมาก อีกทั้งเอกสารอ้างอิงก็ยังไม่มี สามารถหาได้จากในอินเตอร์เน็ตเท่านั้น แต่ก็เป็นส่วนน้อย ดังนั้นโปรแกรมในส่วนของการแสดงผลเมนูทางหน้าจอจึงยังไม่สามารถทำได้ดีเท่าที่ควร

ในส่วนของภาคส่งจะพบว่าสามารถส่งข้อมูลได้ถูกต้อง ระยะทางประมาณ 8 เมตร แต่มีข้อเสียคือ ต้องส่งเป็นเส้นตรง โดยที่ไม่มีสิ่งกีดขวาง และถ้าส่งในระดับความสูงที่แตกต่างกันมากเกินไปก็ทำให้ไม่สามารถรับได้ แต่ก็มีข้อดีคือ ชิ้นงานมีขนาดไม่ใหญ่มากนัก สามารถลดการใช้สายลงไปได้

ในส่วนของโปรแกรมในภาครับ จะพบว่าไม่สามารถเขียนโปรแกรมให้รับข้อมูลที่ส่งติดต่อกันมากกว่า 1 ชุดได้ ซึ่งในตอนที่เขียนโปรแกรมช่วงแรกจะทำการส่งข้อมูลต่อเนื่องกัน 2 ชุด และจะนำข้อมูลทั้งสองชุดมาเก็บไว้ในตัวแปร 2 ตัวเพื่อนำไปเปรียบเทียบกับฐานข้อมูล แต่จากการทดลองปรากฏว่าข้อมูลที่เข้ามาในบับเฟอร์รับข้อมูลของพอร์ต RS-232 มูลชุดสุดท้ายได้จะเข้ามาทับข้อมูลชุดแรก ซึ่งถ้าส่งข้อมูลติดต่อกัน 2 ชุด ข้อมูลที่รับได้จะเป็นข้อมูลชุดที่ 2 เช่น ส่งข้อมูลชุดแรกเป็นหมายเลขของโต๊ะและชุดข้อมูลที่ 2 เป็นรายการอาหาร (ข้อมูลแต่ละชุดมีขนาด 8 บิต) ข้อมูลทั้งสองชุดห่างกัน 20 mS พบว่าข้อมูลที่รับได้จะเป็นข้อมูลของรายการอาหารเท่านั้น ซึ่งทำให้ไม่สามารถส่งข้อมูลมาหลายชุดติดต่อกันได้เพราะไม่สามารถนำมาเปรียบเทียบกับฐานข้อมูลที่เตรียมไว้ในคอมพิวเตอร์ได้จึงต้องส่งข้อมูลเพียง 1 ชุด เป็นสาเหตุให้สั่งอาหารได้ 1 โต๊ะเท่านั้น

บรรณานุกรม

1. Jeff Frohwein , www.derv.com/gb

2. อรรถพล บุญยโภคา และคณะ "ปฏิบัติการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกผ่านพอร์ตอนุกรม" อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด กรุงเทพฯ 2540

3. ทวีชัย ภูรีทิพย์ "ไขปัญหา RS232 ซีเอ็ดยูเคชั่น กรุงเทพฯ 2538

4. อธิคม ฤกษบุตร "รีโมต เครื่องควบคุมไร้สาย" ซีเอ็ดยูเคชั่น กรุงเทพฯ 2538

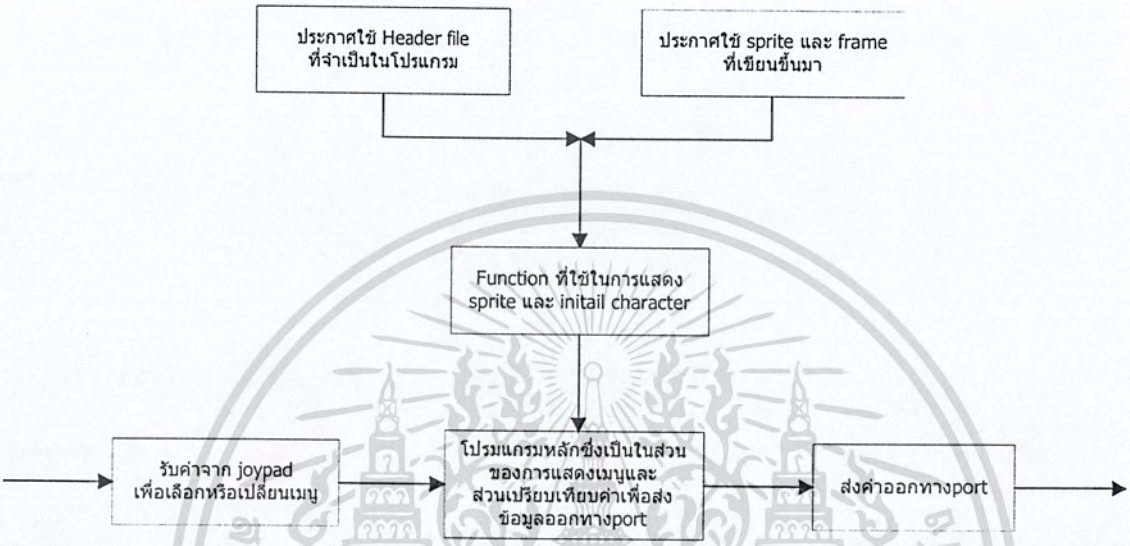
ภาคผนวก

ภาคผนวก ก) รูปแสดง Flow Chart ของโครงงาน

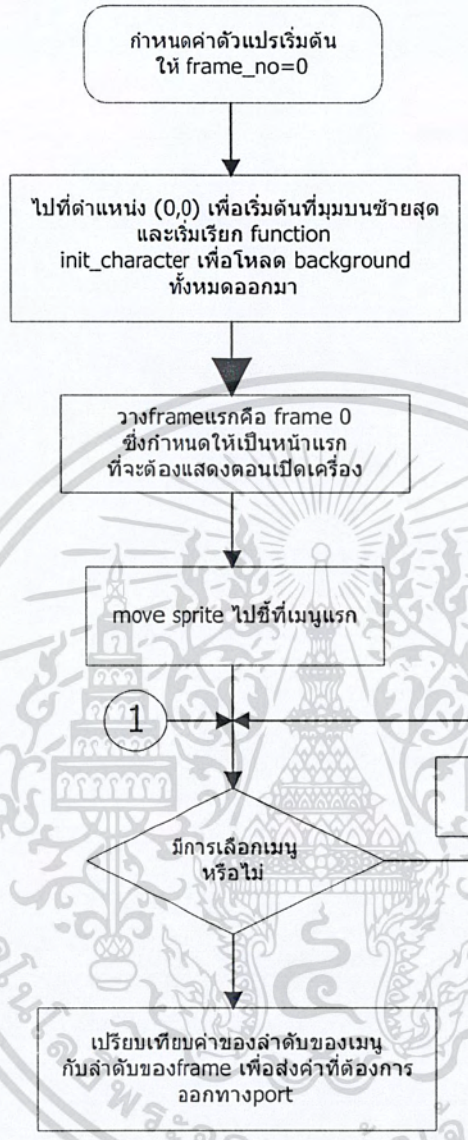
โฟลว์ชาร์ตของโปรแกรม

โฟลว์ชาร์ตของโปรแกรมในส่วนของตัวเกมส์บอย

โฟลว์ชาร์ตของโปรแกรมหลัก

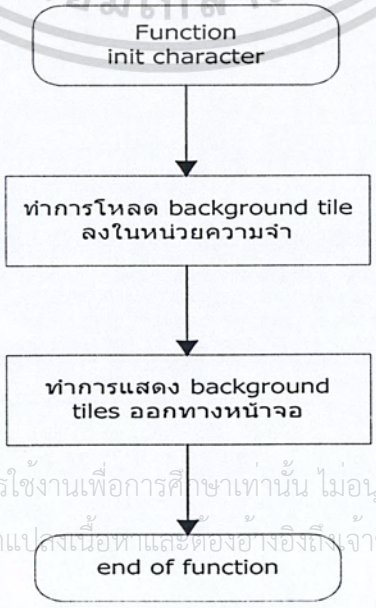


โฟลว์ชาร์ตของ main function

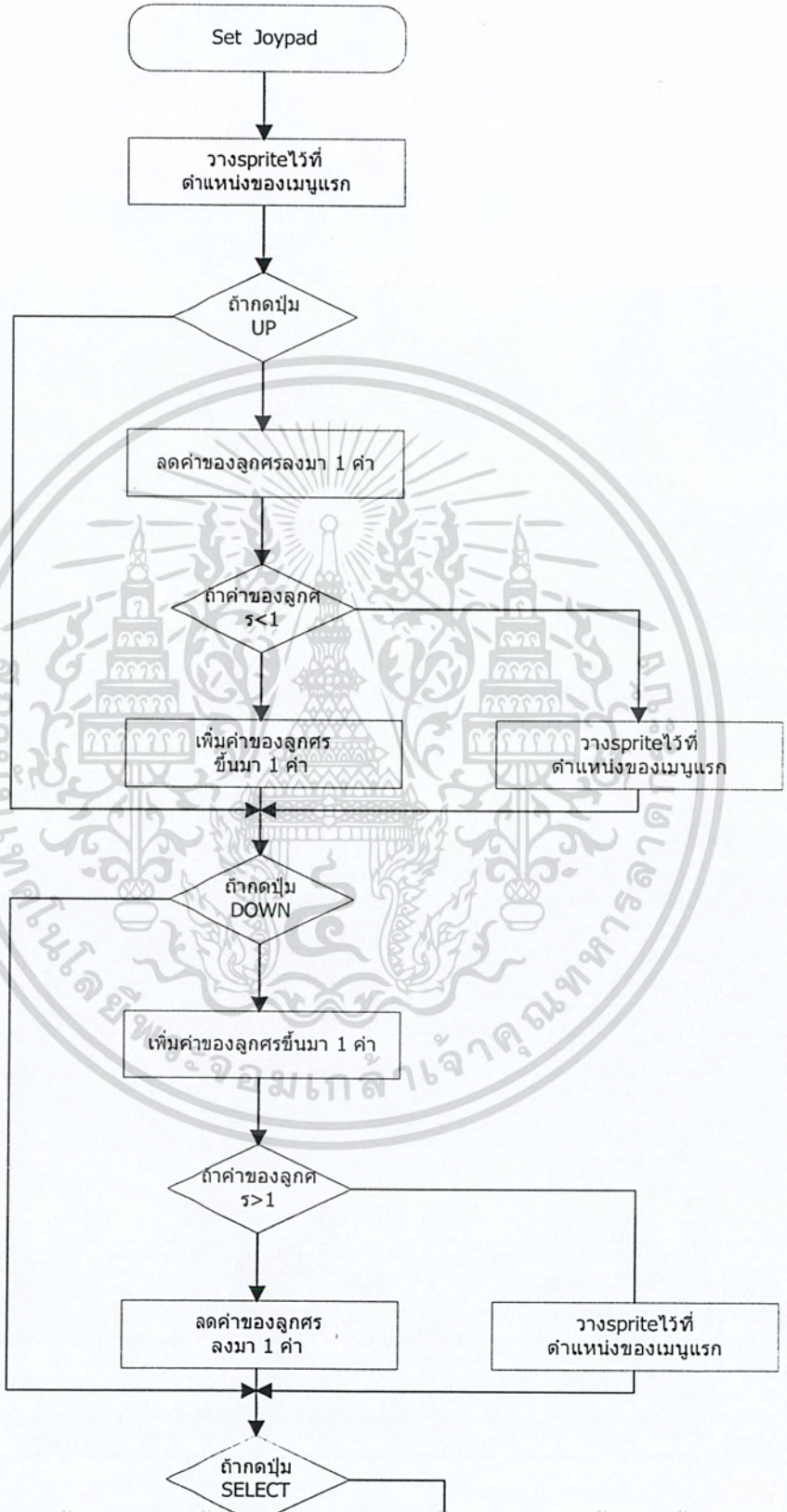


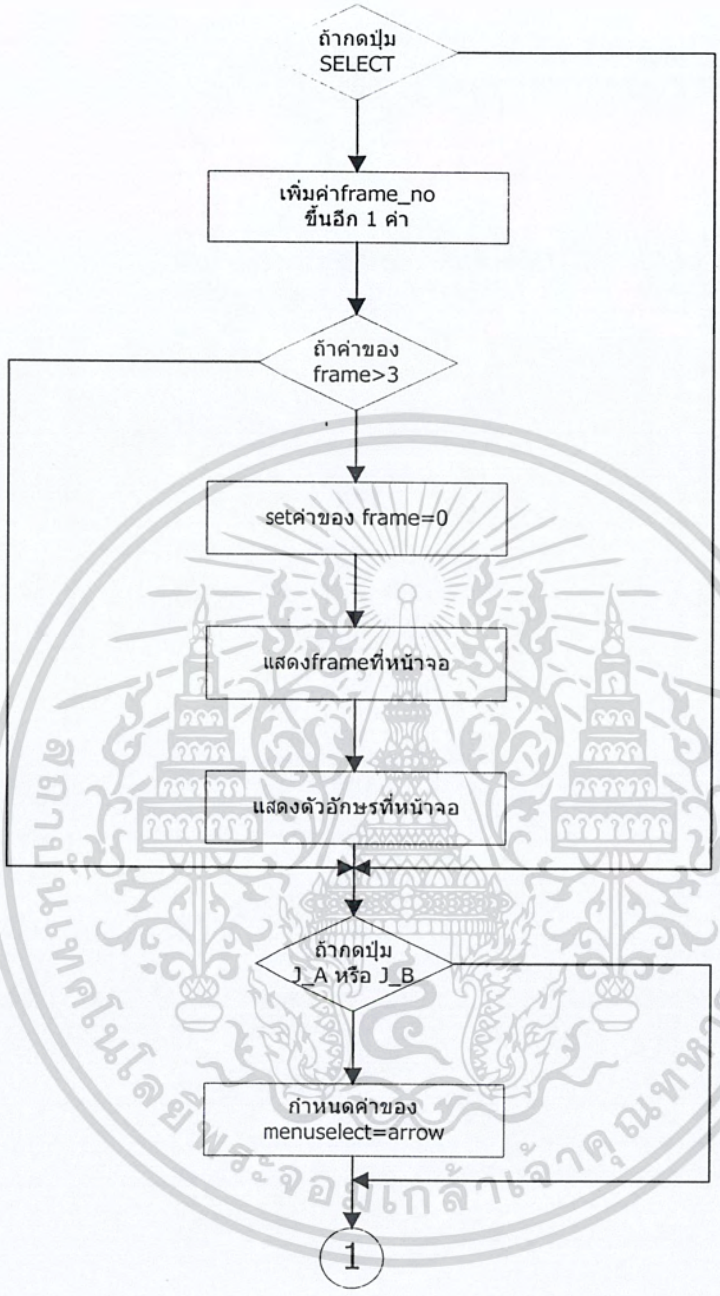
โฟลว์ชาร์ตของ display frame function

โฟลว์ชาร์ตของฟังก์ชั่น init_character

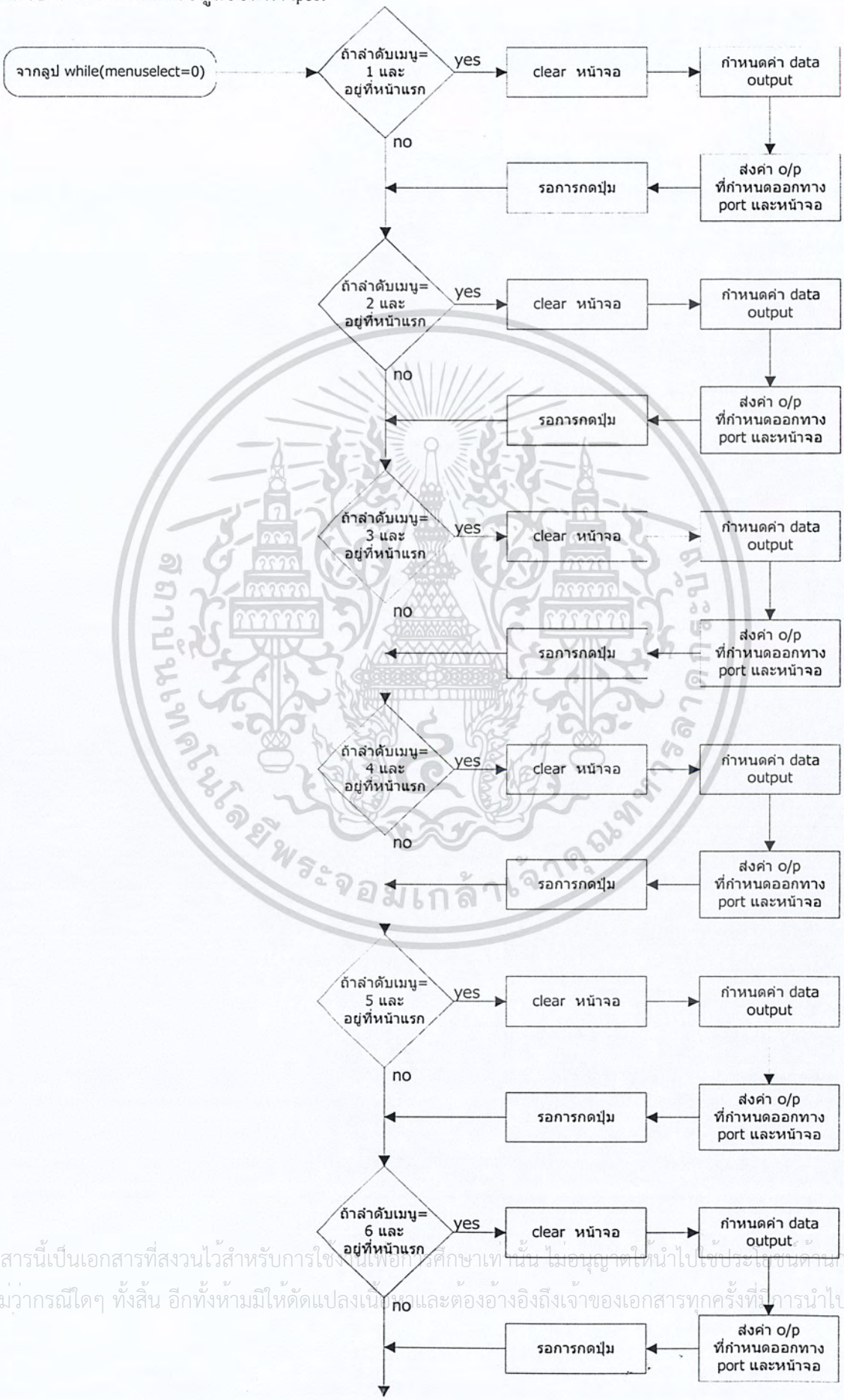


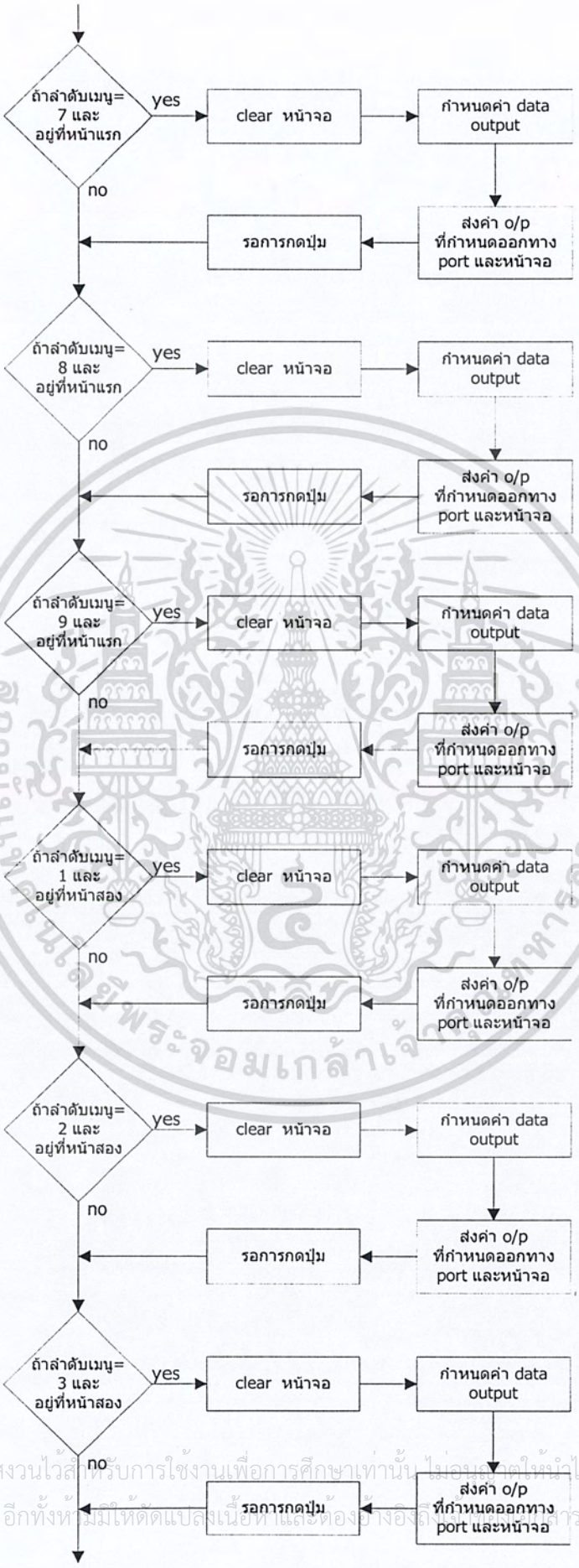
โฟลว์ชาร์ตของ joypad

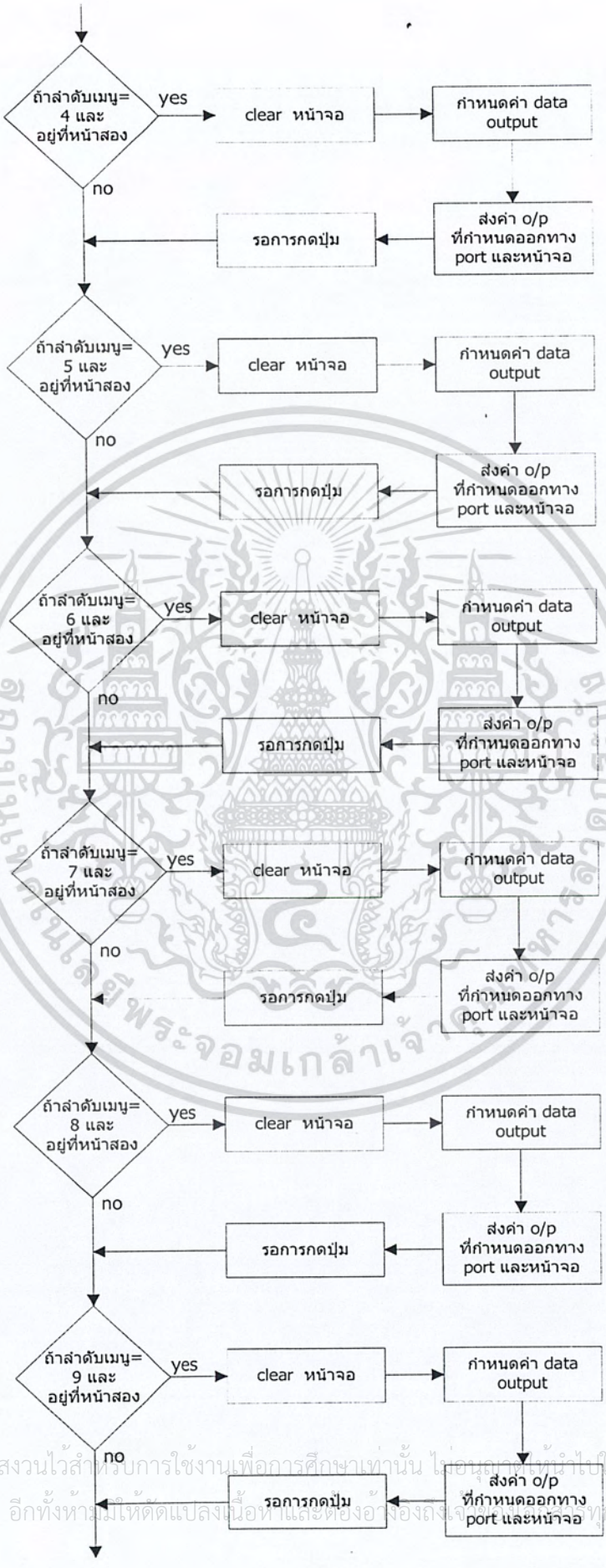


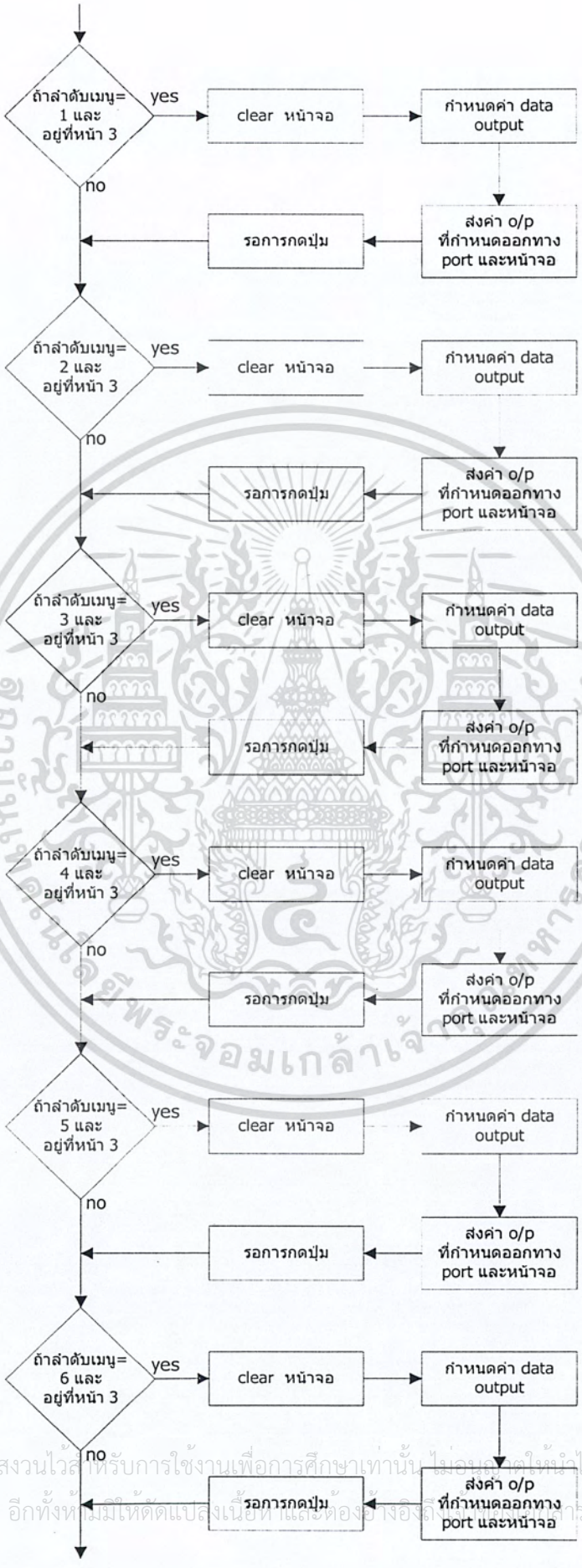


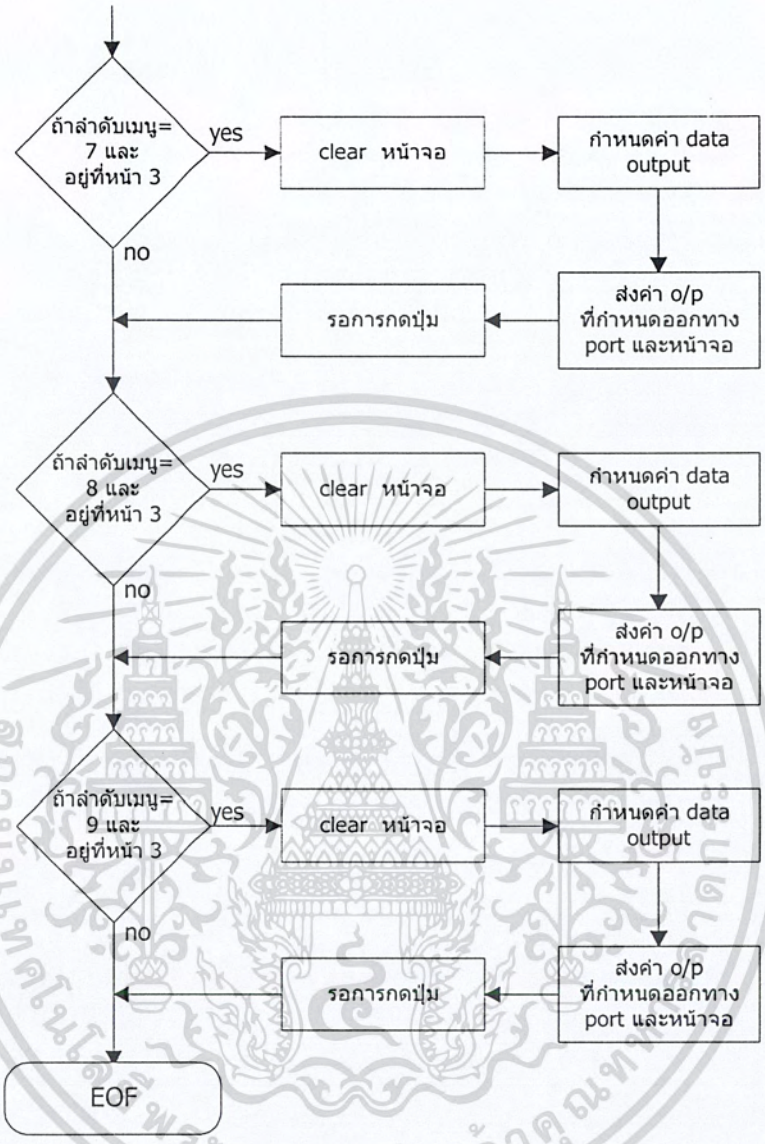
โฟลว์ชาร์ตของการส่งข้อมูลออกทางport



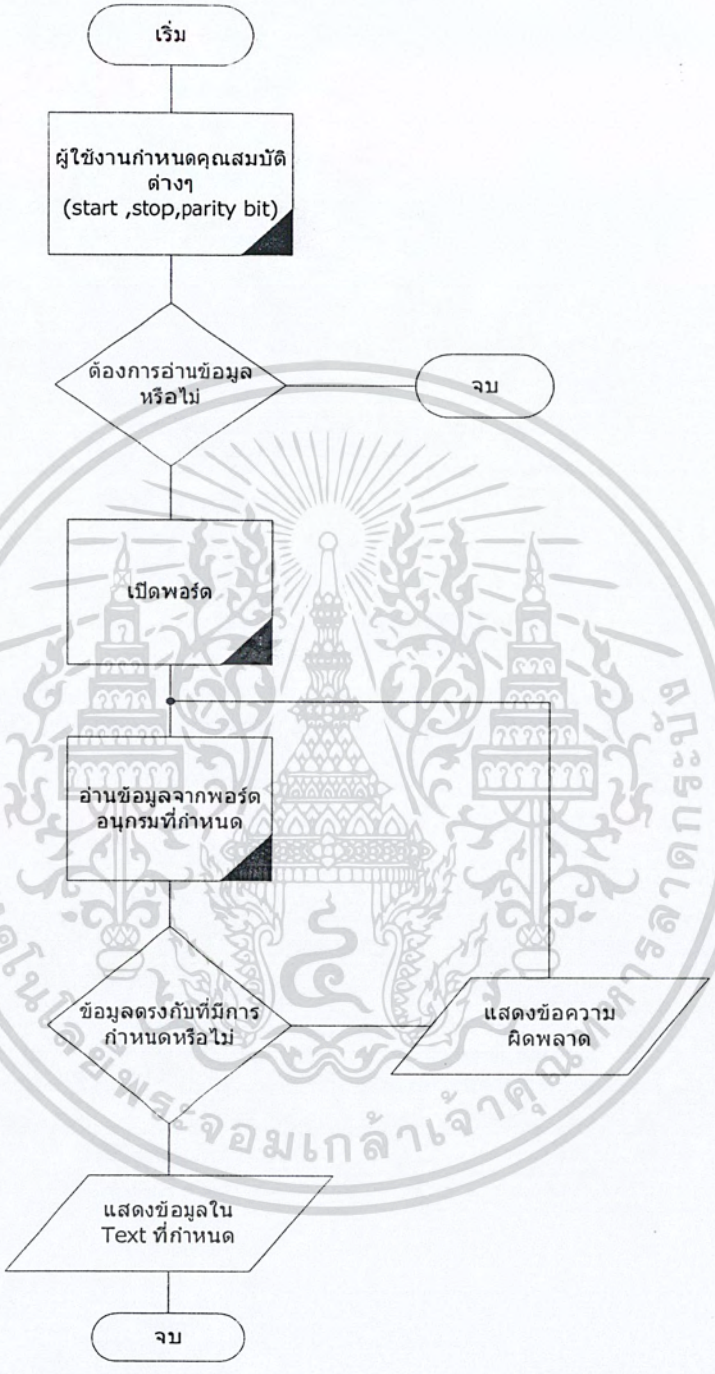








โฟล์ชาร์ตโปรแกรมที่ใช้ในการรับค่าจาก Serial Port



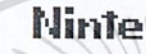
ภาคผนวก ข) คู่มือการใช้งานเมนูอิเล็กทรอนิกส์

คู่มือการใช้งานเมนูอิเล็กทรอนิกส์

ข้อจำกัดในการใช้งานเมนูอิเล็กทรอนิกส์

- สามารถเลือกส่งรายการอาหารที่ต้องการได้เพียงครั้งละ 1 รายการ
- ในฝั่งของคอมพิวเตอร์ผู้รับไม่สามารถระบุโต๊ะที่เลือกได้
- ระยะห่างระหว่างตัวส่งกับตัวรับมีประมาณ 7-8 เมตร

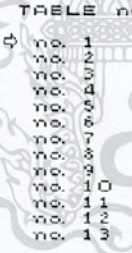
เมื่อทำการเปิดเครื่องเกมบอย ทางหน้าจอจะแสดงข้อความดังรูป



รูปที่ ข-1 แสดงหน้าจอเครื่องเกมบอยขณะเริ่มเปิดเครื่อง

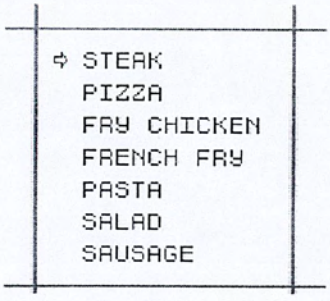
การใช้งานเมนูอิเล็กทรอนิกส์มีวิธีการดังต่อไปนี้

1. หลังจากเปิดเครื่องเกมบอยแล้วจะปรากฏเมนูให้เลือกหมายเลขโต๊ะ โดยการกดปุ่ม A



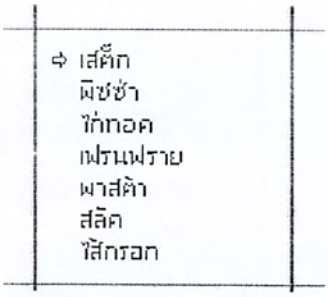
รูปที่ ข-2 แสดงรายการหมายเลขโต๊ะ

2. เมื่อเลือกโต๊ะแล้วจะเปลี่ยนหน้าจอไปที่รายการอาหาร



รูปที่ ข-3 แสดงรายการอาหารภาษาอังกฤษ

3. ทำการสลับหน้าจอรายการอาหารจากภาษาอังกฤษไปเป็นภาษาไทย โดยการกดปุ่ม A หรือ B

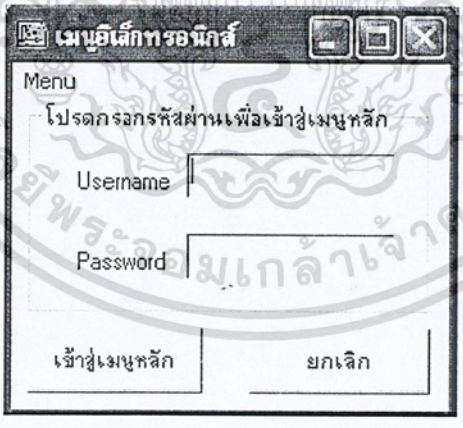


รูปที่ ข-4 แสดงรายการอาหารภาษาไทย

4. เมื่อต้องการเลือกเมนูเพื่อสั่งอาหาร ให้ทำการกดปุ่ม start จะปรากฏหน้าจอดังรูป

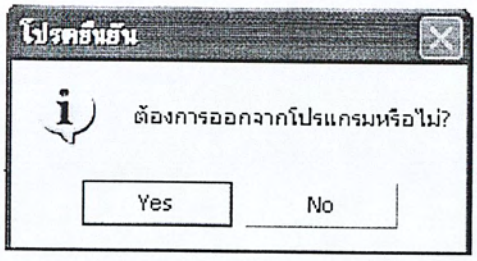
รูปที่ ข-5 แสดงหน้าจอขณะทำการส่งข้อมูลให้กับฝั่งผู้รับ

5.เมื่อเปิดโปรแกรมทางฝั่งผู้รับก็จะพบกับไดอะล็อกดังรูป เพื่อให้ผู้ใช้งานกรอกชื่อผู้ใช้งานและรหัสผ่าน



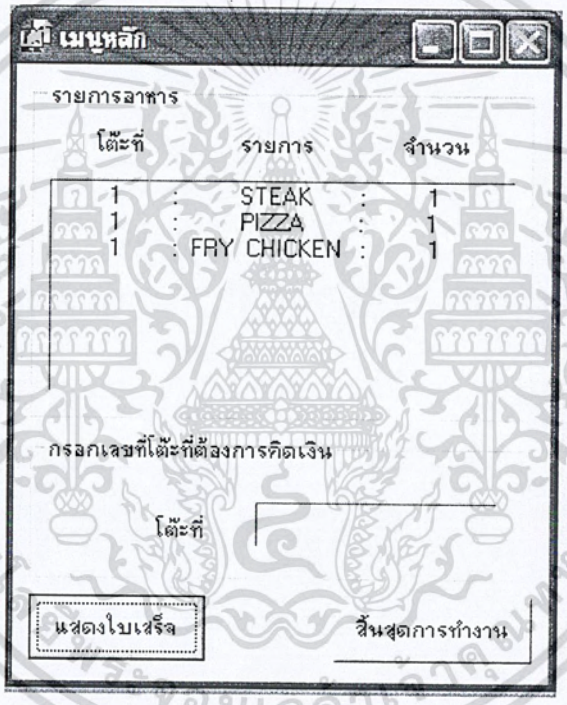
รูปที่ ข-6 แสดงหน้าจอการป้อนรหัสผ่าน

6.เมื่อป้อนชื่อผู้ใช้และรหัสผ่านแล้วจากนั้นถ้ากดปุ่ม “ยกเลิก” ก็จะปรากฏไดอะล็อกดังรูป เพื่อต้องการให้ผู้ใช้งานยืนยันว่าจะออกจากโปรแกรมหรือไม่



รูปที่ ข-7 แสดงหน้าจอการยืนยัน

หรือถ้าผู้ใช้กดปุ่ม“เข้าสู่เมนูหลัก”ก็จะปรากฏไดอะล็อกเมนูหลักดังรูป แสดงว่าโปรแกรมพร้อมใช้งานแล้ว ซึ่งถ้ามีการส่งข้อมูลรายการอาหารมาก็จะปรากฏออกในช่องแสดงผลดังรูป



รูปที่ ข-8 แสดงหน้าจอรายการอาหารที่ได้รับ

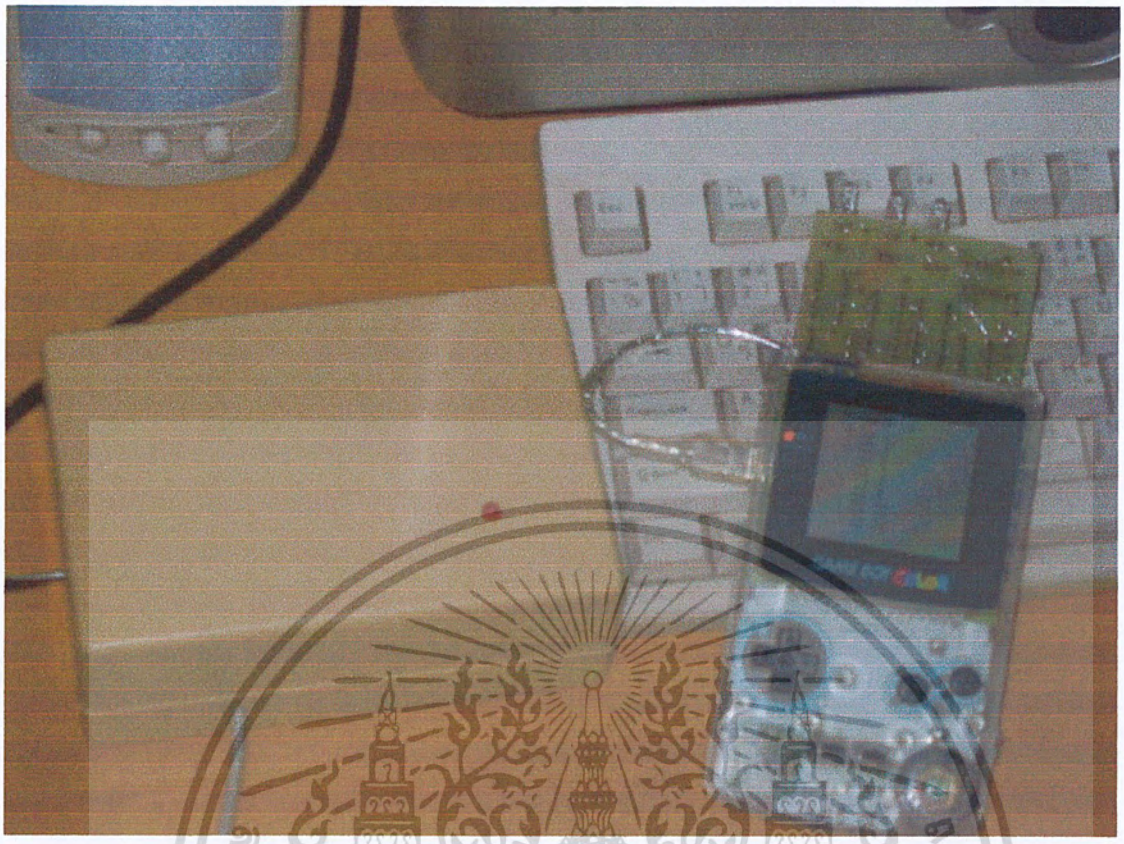
7.ถ้าต้องการคิดเงินก็สามารถกรอกเลขที่โต๊ะและกดปุ่ม “แสดงใบเสร็จ” ก็จะปรากฏใบเสร็จดังรูป

ใบเสร็จ

โต๊ะที่	รายการ	ราคา	จำนวน	ราคารวม
1	STEAK	10	1	10
1	STEAK	10	1	10
1	FRENCH FRY	40	1	40
1	SALAD	60	1	60
			ราคาทั้งหมด	120

รูปที่ ข-6 แสดงใบเสร็จรับเงิน

8.ถ้าต้องการออกจากโปรแกรมก็สามารถกดปุ่ม “สิ้นสุดการทำงาน” จะปรากฏไดอะล็อกให้ยืนยันว่าจะออกจากโปรแกรมหรือไม่อีกครั้งหนึ่ง



รูปที่ ข-10 แสดงเมนูอิเล็กทรอนิกส์

ภาคผนวก ค) Source Code ของโปรแกรม

Source Code สำหรับ Gameboy

```
#include <gb.h>
#include <stdio.h>
#include <console.h>
#include "sprites.c"
#include "blank.c"
#include "blank.h "
#include "table.c "
#include "table.h"
 #include "pmap.c"
#include "pmap.h"
#include "ptd.c"
#include "ptd.h"

void main()
{
/*
UBYTE arrow,page,tableno  = 1;
UBYTE a,b = 0;
*/
int i,menuselect,notstart,button,tables,menuno,x;
int arrow,page,tableno,a,b;
page=1;
menuno = 0;
menuselect = 0;
arrow = 1;
a=0;
b=0;
tableno=1;
tables = 0;
notstart=1;
SHOW_BKG;
set_sprite_data(0,11,sprites);
for(x=0;x<12;x++)set_sprite_tile(x,x);
SHOW_SPRITES;
move_sprite(1,(1*8)+24,(2*8)+24);
```

```
while(tables == 0)
{
        button = joypad();
                set_bkg_data(0,18,table);
                for( i=0; i<1; i++ )
                {
                set_bkg_tiles(i,0,20,18,blank);
                }
                 if(button & J_UP)
                {
                        tableno=tableno-1;
                        if(tableno < 1)
                        tableno=tableno+14;
                        else
                {
                move_sprite(1,(1*8)+24,((2+tableno)*8)+16);
                delay(200);
                }
                }
                if(button & J_DOWN)
                {
                        tableno=tableno+1;
                        if(tableno > 14)
                tableno=tableno-14;
                        else
                {
                        move_sprite(1,(1*8)+24,((2+tableno)*8)+16);
                delay(200);
                }
                }
                if(button & J_A)
                {
                tables = tableno ;
                delay(500);
                }
```

```
}
while(notstart == 1)
 {
 waitpadup();
move_sprite(1,(14*8)+24,(0*8)+16);
 while(menuselect == 0)
        {
        set_bkg_data(0,121,ptd);
        set_bkg_tiles(0,0,32,32,pmap);
        move_sprite(1,(1*8)+24,((2+arrow)*8)+16);
        button = joypad();
        if(button & J_UP)
                {
                        {
                        arrow=arrow-2;
                        }
                        if(arrow < 1)
                        {
                        move_bkg(0,0);
                        arrow=arrow+14;
                        page=1;
                        }
                        else
                        {
                        move_sprite(1,(1*8)+24,((2+arrow)*8)+16);
                        delay(200);
                        }
                }
        if(button & J_DOWN)
                {
                        {
                        arrow=arrow+2;
                        }
                        if(arrow > 14)
                        {
                        move_bkg(0,128);
```

```
                arrow=arrow-14;
                page=2;
                }
                else
                {
                move_sprite(1,(1*8)+24,((2+arrow)*8)+16);
                delay(200);
                }
        }
if(button & J_A)
        {
        HIDE_BKG;
        scroll_bkg(128,0);
        delay(200);
        SHOW_BKG;
        }
if(button & J_B)
        {
        HIDE_BKG;
        scroll_bkg(-128,0);
        delay(200);
        SHOW_BKG;
        }
/*
if(button & J_SELECT)
        {
        waitpadup();
        }
*/
if(button & J_SELECT )
        {
        /*move_bkg(0,0);*/
        if(arrow == 3&&page==1)
                {
                b = b++;
                a = b;
```

```
                    move_sprite(5,(1*8)+148,(2*8)+48);
                    move_sprite(6,(1*8)+148,(2*8)+50);
                    }
if(arrow == 9&&page==2)
                    {
                    b = b++;
                    a = b;
                    move_sprite(2,(1*8)+148,(2*8)+24);
                    move_sprite(3,(1*8)+148,(2*8)+40);
                    }
          }
if(button & J_START)
          {
          _io_out = tableno ;
          send_byte();
          while(_io_status == IO_SENDING && joypad() == 0);
          delay(20);

          _io_out = page;
          send_byte();
          while(_io_status == IO_SENDING && joypad() == 0);
          delay(20);

          _io_out = arrow;
          send_byte();
          while(_io_status == IO_SENDING && joypad() == 0);
          delay(20);

          HIDE_BKG;

          move_sprite(1,200,200);
          move_sprite(2,(1*8)+48,(2*8)+48);
          move_sprite(3,(1*8)+56,(2*8)+48);
          move_sprite(4,(1*8)+64,(2*8)+48);
          move_sprite(5,(1*8)+72,(2*8)+48);
          move_sprite(6,(1*8)+80,(2*8)+48);
```

```
                move_sprite(7,(1*8)+88,(2*8)+48);
                move_sprite(8,(1*8)+96,(2*8)+48);
                move_sprite(9,(1*8)+104,(2*8)+48);
                move_sprite(10,(1*8)+112,(2*8)+48);
                delay(1500);
                move_sprite(2,200,200);
                move_sprite(3,200,200);
                move_sprite(4,200,200);
                move_sprite(5,200,200);
                move_sprite(6,200,200);
                move_sprite(7,200,200);
                move_sprite(8,200,200);
                move_sprite(9,200,200);
                move_sprite(10,200,200);
                SHOW_BKG;
                /*
                waitpadup();
                while(!joypad()) {;}
                */
                /*menuselect = 1;*/
                }
        }
move_sprite(1,200,200);
}
}
```

Source code สำหรับ PIC16F628

```
LIST    P=16F628,W=-302
TRISA   EQU     0X85
PORTA   EQU     0X05
TRISB   EQU     0X86            ; Bank 1
PORTB   EQU     0X06    ; Bank 0
CMCON   EQU     0X1F
STATUS  EQU     0X03
PIR1    EQU     0x0C                            ; Bank 0
RCSTA   EQU     0x18
TXREG   EQU     0x19
TRISB   EQU     0x86                            ; Bank 1
TXSTA   EQU     0x98
SPBRG   EQU     0x99
TXIF    EQU     4                               ; Bit
TXEN    EQU     5
SYNC    EQU     4
BRGH    EQU     2
SPEN    EQU     7
CREN    EQU     4
RA      EQU     0X72
RB      EQU     0X73
RCV     EQU     0X74
SENDMAX         EQU     0X75
COUNT1          EQU     0X76
TX1     EQU     0X77
TX2     EQU     0X78
RX1     EQU     0X79
RX2     EQU     0X7A
RCVMAX          EQU     0X7B
X       EQU     0x70
Y       EQU     0x71

        __CONFIG        0X3F61
```

```
        ORG     0X00
        MOVLW.7
        MOVWFCMCON
        BANKSEL         TRISA
        MOVLWB'11111111'
        MOVWFTRISA
        BANKSEL         PORTA
        BANKSEL         TRISB
        MOVLWB'11111111'
        MOVWFTRISB
        BANKSEL         PORTB
READSW          BTFSS   PORTB,4
        GOTO    LOOP
        GOTO    READSW
LOOP    CLRF    RA
        CLRF    RB
        CLRF    RCV
        CALL    DELAY
        MOVF    PORTA,W
        MOVWFRA
        MOVF    PORTB,W
        MOVWFRB
        RRF     RA,1
        RRF     RCV,1
        RRF     RA,1
        RRF     RCV,1
        RRF     RA,1
        RRF     RCV,1
        RRF     RA,1
        RRF     RCV,1
        RRF     RA,1
        RRF     RCV,1
        RLF     RB,1
        RRF     RCV,1
        RLF     RB,1
        RRF     RCV,1
```

```
        RLF     RB,1
        RRF     RCV,1
        GOTO    UART                    ;***************
DELAY   CLRF    X
        MOVLW0x02
        MOVWFY
LOOP2   DECFSZ X,1
        GOTO    LOOP2
        DECFSZ Y,1
        GOTO    LOOP2
        RETURN
UART
        BANKSEL         TRISB                   ; Bank 1
        BSF     TRISB,2                 ; RB2 as input
        MOVLW.104
        MOVWFSPBRG                      ; Baudrate = 2400 bps
        BSF     TXSTA,BRGH              ; Hi-speed
        BSF     TXSTA,TXEN              ; TX enable
        BANKSEL         RCSTA                   ; Bank 0
        BSF     RCSTA,SPEN              ; Serial port enable
        MOVF    RCV,W                           ; Load ASCII data
        CALL    SEND_DATA               ; Send data
        GOTO    READSW                          ; Endless loop
SEND_DATA
        BTFSS   PIR1,TXIF               ; Skip if TXREG empty
        GOTO    $-1                             ; Not empty then wait
        MOVWFTXREG                      ; Send data to RS-232
        RETURN
        END
```

Source code สำหรับ โปรแกรม Visual Basic

```
Dim a  As Variant
Private Sub command1_Click()
   If Text2.Text <> "1" Then
      MsgBox “เลขที่โต๊ะไม่ถูกต้อง.”
   Else
      DataReport1.Show
   End If
End Sub

Private Sub command2_Click()
   If MsgBoxต้องการออกจากโปรแกรมหรือไม่?", 68, "โปรดยืนยัน'") = 6 Then
      End
   End If
End Sub

Private Sub Form_Load()
   MSComm1.CommPort = 1
   MSComm1.Settings = "2400,n,8,1"
   MSComm1.PortOpen = True
      End Sub

Private Sub Text2_Change()
   command1.SetFocus
End Sub

Private Sub Timer1_Timer()
   If MSComm1.InBufferCount Then
       Call reci
   End If
End Sub
```

```
Private Sub reci()
a = Asc(MSComm1.Input)
  If a = 2 Then
      List1.AddItem "     1     :     STEAK     :     1"
      menu.Recordset.AddNew
      menu.Recordset!TableId = 1
      menu.Recordset!odername = "STEAK"
      menu.Recordset!price = 10
      menu.Recordset!quantity = 1
      menu.Recordset!total = ((menu.Recordset!price) * (menu.Recordset!quantity))
      menu.Recordset.Update
      Beep
   ElseIf a = 3 Then
      List1.AddItem "     1     :     PIZZA     :     1"
      menu.Recordset.AddNew
      menu.Recordset!TableId = 1
      menu.Recordset!odername = "PIZZA"
      menu.Recordset!price = 20
      menu.Recordset!quantity = 1
      menu.Recordset!total = ((menu.Recordset!price) * (menu.Recordset!quantity))
      menu.Recordset.Update
      Beep
   ElseIf a = 6 Then
      List1.AddItem "     1     : FRY  CHICKEN  :     1"
      menu.Recordset.AddNew
      menu.Recordset!TableId = 1
      menu.Recordset!odername = "FRY  CHICKEN "
      menu.Recordset!price = 30
      menu.Recordset!quantity = 1
      menu.Recordset!total = ((menu.Recordset!price) * (menu.Recordset!quantity))
      menu.Recordset.Update
      Beep
```

```
ElseIf a = 7 Then
    List1.AddItem "      1      : FRENCH FRY   :      1"
    menu.Recordset.AddNew
    menu.Recordset!TableId = 1
    menu.Recordset!odername = "FRENCH FRY"
    menu.Recordset!price = 40
    menu.Recordset!quantity = 1
    menu.Recordset!total = ((menu.Recordset!price) * (menu.Recordset!quantity))
    menu.Recordset.Update
    Beep
ElseIf a = 10 Then
    List1.AddItem "      1      :      PASTA      :      1"
    menu.Recordset.AddNew
    menu.Recordset!TableId = 1
    menu.Recordset!odername = "PASTA "
    menu.Recordset!price = 50
    menu.Recordset!quantity = 1
    menu.Recordset!total = ((menu.Recordset!price) * (menu.Recordset!quantity))
    menu.Recordset.Update
    Beep
ElseIf a = 11 Then
    List1.AddItem "      1      :      SALAD      :      1"
    menu.Recordset.AddNew
    menu.Recordset!TableId = 1
    menu.Recordset!odername = "SALAD  "
    menu.Recordset!price = 60
    menu.Recordset!quantity = 1
    menu.Recordset!total = ((menu.Recordset!price) * (menu.Recordset!quantity))
    menu.Recordset.Update
    Beep
ElseIf a = 14 Then
    List1.AddItem "      1      :      SAUSAGE   :      1"
```

```
        menu.Recordset.AddNew
        menu.Recordset!TableId = 1
        menu.Recordset!odername = "SAUSAGE "
        menu.Recordset!price = 70
        menu.Recordset!quantity = 1
        menu.Recordset!total = ((menu.Recordset!price) * (menu.Recordset!quantity))
        menu.Recordset.Update
        Beep
        End If
End Sub
```