ควบคุมการเปิด-ปิดเอ็มพีสามด้วยไมโครคอนโทรลเลอร์

MP3 MUSIC PLAYER ON TIME CONTROLLING BY USING MICROCONTROLLER

ดนัยวัชร ดาวสุด

ปรเมศวร์ พุงกระโทก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2555

ควบคุมการเปิด-ปิดเอ็มพีสามด้วยไมโครคอนโทรลเลอร์

MP3 MUSIC PLAYER ON TIME CONTROLLING BY USING MICROCONTROLLER

ดนัยวัชร ดาวสุด
ปรเมศวร์ พุงกระโทก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
สาขาวิศวกรรมการวัดคุม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2555

MP3 MUSIC PLAYER ON TIME CONTROLLING BY USING MICROCONTROLLER

DAINAIWAT DAWNSUD

PORAMET PHUNGKRATOK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERRING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUE OF TECHNOLOGY LARDKRABANG

ACADEMIC YEAR 2012

ปริญญานิพนธ์ปีการศึกษา 2555

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

.....

หัวข้อปริญญานิพนธ์	ควบคุมการเปิด-ปิด MP3 ด้วยไมโครคอนโทรลเลอร์ MP3 MUSIC PLAYER ON TIME CONTROLLING BY USING MICROCONTROLLER		
นักศึกษาผู้จัดทำ	นายดนัยวัชร ดาวสุด	**รหัสนักศึกษา**	52010381
	นายปรเมศวร์ พุงกระโทก	**รหัสนักศึกษา**	52010671
ปริญญา	วิศวกรรมศาสตร์บัณฑิต		
สาขาวิชา	วิศวกรรมการวัดคุม		
ปีการศึกษา	2555		

อาจารย์ผู้ควบคุมปริญญานิพนธ์	**ลายมือชื่อ**
รองศาสตราจารย์ทรงชัย วีระทวีมาศ	

หัวข้อปริญญานิพนธ์	ระบบตอบรับความพึงพอใจ			
	CONTENTMENT ACCEPTANCE SYSTEM			
นักศึกษาผู้จัดทำ	นายดนัยวัชร	ดาวสุด	รหัสนักศึกษา	52010381
	นายปรเมศวร์	พุงกระโทก	รหัสนักศึกษา	51010671
อาจารย์ที่ปรึกษา	รองศาสตราจารย์ทรงชัย วีระทวีมาศ			
ปีการศึกษา	2555			

บทคัดย่อ

ปัจจุบันได้มีการนำระบบควบคุมการทำงานมาประยุกต์ใช้ในงานควบคุมอย่างแพร่หลาย การดัดแปลงและประยุกต์ใช้อุปกรณ์บางชนิดให้สามารถทำงานได้ และเพิ่มมูลค่าให้กับอุปกรณ์ชนิดนั้น นับเป็นสิ่งสำคัญเช่นกัน โครงงานนี้ได้พัฒนาอุปกรณ์ที่เรียกว่าโมดูล MP3 เพื่อให้สามารถทำงานได้อย่างอัตโนมัติตามที่เราต้องการ ซึ่งระบบนี้ประกอบด้วยไมโครคอนโทรลเลอร์ประเภท dsPIC ซึ่งใช้เป็นตัวควบคุมการทำงานของโมดูล MP3 สามารถทำงานได้ตามที่ต้องการได้และยังเพิ่มคีย์บอร์ดเข้าไป เพื่อให้สามารถสั่งการผ่านทางคีย์บอร์ดได้ โดยสามารถตรวจสอบการสั่งการผ่านหน้าจอ ไปยังอุปกรณ์ภายในระบบได้ เป็นการเพิ่มประสิทธิภาพให้กับโมดูล MP3 มากยิ่งขึ้น

Thesis Title	CONTENTMENT ACCEPTANCE SYSTEM
Authors	Mr. DAINAIWAT DAWNSUD
	Mr. PORAMET PHUNGKRATOK
Thesis Advisor	Mr. Songchai Weerathaweemas
Year	2012

ABSTRACT

Nowadays,We have control process by automation system more than the past. this is application for lower capital by electronic tools. this project be able to develop Module MP3 for the simple of electronic tools that work on automation system. The Mp3 module make of Ds1307,Keypad ,LCD Display and TDS055 for control MP3 to work automation.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความกรุณาความปรารถนาดีและความอนุเคราะห์จากอาจารย์ที่ปรึกษา ผู้ช่วยศาตราจารย์ทรงชัย วีระทวีนาศ ที่คอยดูแลให้ความเอาใจใส่ สละเวลาอันมีค่าให้คำแนะนำ และแก้ไขปัญหาหรือข้อสงสัยต่างๆให้แก่คณะผู้จัดทำ ทั้งนี้ขอขอบพระคุณอาจารย์สาขาวิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ให้ความรู้ตลอดระยะเวลาที่ทำการศึกษา

สุดท้ายนี้ขอกราบขอบพระคุณบิดามารดาผู้เป็นที่รักยิ่ง คอยเป็นกำลังใจและเป็นแรงผลักดันให้โครงงานนี้ก้าวไปสู่ความสำเร็จ และขอบคุณเพื่อนๆปี4 สาขาวิศวกรรมการวัดคุมทุกคนที่คอยอยู่เคียงข้างกันตลอด 4 ปี

คณะผู้จัดทำหวังเป็นอย่างยิ่งว่าโครงงานนี้จะเป็นประโยชน์แก่ผู้ที่สนใจไม่มากก็น้อย หากมีข้อบกพร่องประการใด คณะผู้จัดทำต้องขออภัยมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

บทที่ 1

บทนำ

1.1 ความสำคัญของปริญญานิพนธ์

การศึกษาในสาขาวิศวกรรมระบบวัดและควบคุม เป็นการศึกษาและประยุกต์ทฤษฎีต่างๆ เพื่อการออกแบบและควบคุมระบบให้มีเสถียรภาพ และมีสมรรถนะตามความต้องการหรือให้เป็นไปตามข้อกำหนด ดังนั้นเพื่อให้เป็นไปตามวัตถุประสงค์ที่ต้องการนี้จึงจำเป็นต้องมีการศึกษาการใช้งานระบบควบคุม การศึกษาวงจรอิเล็กทรอนิกส์รวมถึงศึกษาการเขียนโปรแกรมคอมพิวเตอร์ การศึกษาและเลือกอุปกรณ์วัดและแปลงสัญญาณ ตลอดจนบูรณการเรื่องที่ศึกษาเหล่านี้ในการประยุกต์ใช้กับระบบควบคุมทางกายภาพจริง ซึ่งนับเป็นสิ่งจำเป็นในการศึกษาในสาขาวิชานี้

โครงงานนี้จึงได้เลือกที่จะศึกษาการควบคุมMP3ของบริษัท ศิลา รีเสิร์ท เพื่อให้สามารถเปิดและปิดเพลงได้ตามต้องการโดยใช้ไมโครคอนโทรลเลอร์เป็นหลักและระบบที่มีประโยชน์ในการอำนวยความสะดวกแก่ผู้ใช้งาน เนื่องจากอุปกรณ์MP3ของบริษัท ศิลา รีเสิร์ท สามารถใช้งานในการเก็บเพลงได้เพียงอย่างเดียว เมื่อเล็งเห็นจุดนี้ทำให้สามารถคิดต่อยอดว่าน่าจะสามารถนำมาพัฒนาได้อีก คือ การใช้ไมโครคอนโทรลเลอร์มาควบคุมการเปิดและปิดเพลงของอุปกรณ์MP3 ให้สามารถเปิดและปิดเพลงได้เองตามเวลาที่กำหนด เพื่อเพิ่มมูลค่าให้แก่อุปกรณ์ และสามารถนำมาประยุกต์ใช้ในชีวิตประจำวันได้จริง

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. ศึกษาหลักการทำงานและหลักการควบคุมของไมโครคอนโทรลเลอร์ dsPIC
2. ศึกษาหลักการเขียนและออกแบบวงจรควบคุมการทำงานของMP3 ของบริษัทศิลา รีเสิร์ท
3. พัฒนาอุปกรณ์ MP3 ของ บริษัท ศิลา รีเสิร์ท ให้สามารถควบคุมให้เป็นระบบอัตโนมัติ

1.3 ขอบเขตของปริญญานิพนธ์

1. ออกแบบระบบควบคุม MP3 เพื่อแสดงผล ตรวจสอบและสั่งการ เพื่อควบคุมกระบวนการผ่านทางคีย์บอร์ด และสั่งการอัตโนมัติ
2. เชื่อมต่ออุปกรณ์ MP3 ผ่านพอร์ตอนุกรม โดยมีไมโครคอนโทรลเลอร์ dsPIC เป็นตัวกลาง

3. ศึกษาข้อมูลการออกแบบอุปกรณ์อย่างถูกสุขลักษณะเพื่อให้สามารถทำงานร่วมกันได้

1.4 ขั้นตอนการศึกษา

1. ศึกษาหลักการและการใช้งานอุปกรณ์ MP3 ของ บริษัท ศิลา รีเสิร์ท
2. ศึกษาตัวควบคุมที่สามารถโปรแกรมได้
3. ศึกษาและค้นคว้าข้อมูลการเชื่อมต่ออุปกรณ์เข้ากับระบบ
4. ศึกษาการเขียนโปรแกรมและคำสั่งเพื่อการติดต่อสื่อสารกันระหว่างตัวควบคุมกับอุปกรณ์เพื่อสั่งการอุปกรณ์
5. เรียนรู้หลักการใช้งานเพื่อเลือกใช้อุปกรณ์ควบคุมให้เหมาะสม

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 รูปแบบในการส่งข้อมูล (transmission mode)

การส่งข้อมูลในระบบเครือข่าย สามารถทำได้ 2 ลักษณะ คือ การส่งแบบขนาน และการส่งแบบอนุกรม

การส่งข้อมูลแบบขนาน(Parallel transmission) คือการส่งข้อมูลพร้อมกันทีละหลายๆบิต ในหนึ่งรอบสัญญาณนาฬิกา โดยการส่งจะรวมบิต 0 หรือ 1 หลายๆบิตเข้าเป็นกลุ่มจำนวน n บิต ผู้ส่งส่งครั้งละ n บิต ผู้รับจะรับครั้งละ n บิตเช่นกัน ซึ่งจะคล้ายกับเวลาที่เราพูดคุยเราจะพูดเป็นคำๆไม่พูดทีละตัวอักษร

การส่งข้อมูลแบบอนุกรม(Serial transmission)จะใช้วิธีการส่งทีละ 1 บิตในหนึ่งรอบสัญญาณนาฬิกา ทำให้ดูเหมือนว่าบิตต่างๆเรียงต่อเนื่องกันไป จากอุปกรณ์หนึ่งไปยังอีกอุปกรณ์หนึ่ง

2.2 การสื่อสารของข้อมูล

สามารถแบ่งได้3 แบบ คือ

1. แบบซิมเพล็กซ์(Simplex) เป็นการสื่อสารทางเดียว เมื่ออุปกรณ์หนึ่งส่งข้อมูล อุปกรณ์อีกชุดจะต้องเป็นฝ่ายรับข้อมูลเสมอ
2. แบบฮาล์ฟดูเพล็กซ์(Half-duplex) เป็นการสื่อสารได้ทั้งสองทาง เป็นการเปลี่ยนเส้นทางในการส่งข้อมูลได้ แต่คนละเวลากล่าวคือ ข้อมูลไหลไปในทิศทางเดียว ณ เวลาใดๆ
3. แบบฟลูดูเพล็กซ์(Full-duplex) เป็นการสื่อสารได้ทั้งสองทาง กล่าวคือเป็นผู้รับข้อมูลและผู้ส่งข้อมูล ในเวลาเดียวกันได้

2.3 รูปแบบการสื่อสารข้อมูลแบบอนุกรม

การติดต่อแบบอนุกรมเมื่อแบ่งตามลักษณะของการส่งข้อมูล แบ่งได้2 แบบ คือ

1. การสื่อสารแบบซิงโครนัส (Synchronous)
2. การสื่อสารแบบอะซิงโครนัส(Asynchronous)
3. การส่งข้อมูลแบบไอโซโครนัส (Isochronous Transmission)

2.3.1 การสื่อสารข้อมูลแบบซิงโครนัส (Synchronous)

เป็นการส่งข้อมูลเป็นบล็อก ครั้งละหลายๆไบต์สัญญาณนาฬิกาที่ช่วยให้การทำงานของตัวส่งและตัวรับสอดคล้องกัน อาจจะถูกเข้ารหัสอยู่ในชุดของข้อมูลนั้นหรือแยกอิสระออกเป็นสายต่างหากก็ได้ ตัวอย่างการรับส่งข้อมูลแบบนี้ ได้แก่โปรโตคอล High-Level Data Link Control

โปรโตคอล High-Level Data Link Control (HDLC)

โปรโตคอล High-Level Data Link Control (HDLC) ที่ใช้กับโมเด็มแวน (WAN) ซึ่งมี

F	A	C	I	FCS	F

ลักษณะแฟรม (Fram) ดังนี้

ชื่อฟิลด์	**ขนาด**
Flag Field(F)	8 บิต
Address Field(A)	8 บิต
Control Field(C)	8 or 16 บิต
Information Field(I)	เปลี่ยนแปลงได้ บางแฟรมก็ไม่ได้ใช้
Frame Check Sequence(FCS)	16 or 32 บิต
Closing Flag Field(F)	8 บิต

2.3.2 การสื่อสารข้อมูลแบบอะซิงโครนัส (Asynchronous)

การสื่อสารแบบนี้ใช้มากในเครื่องไมโครคอมพิวเตอร์พีซี

รูปแบบการสื่อสารจะเป็นการรับและส่งข้อมูลครั้งละ1 ไบต์

ตัวอย่าง โปรโตคอลอะซิงโครนัส Serial Commincation - Example Protocols

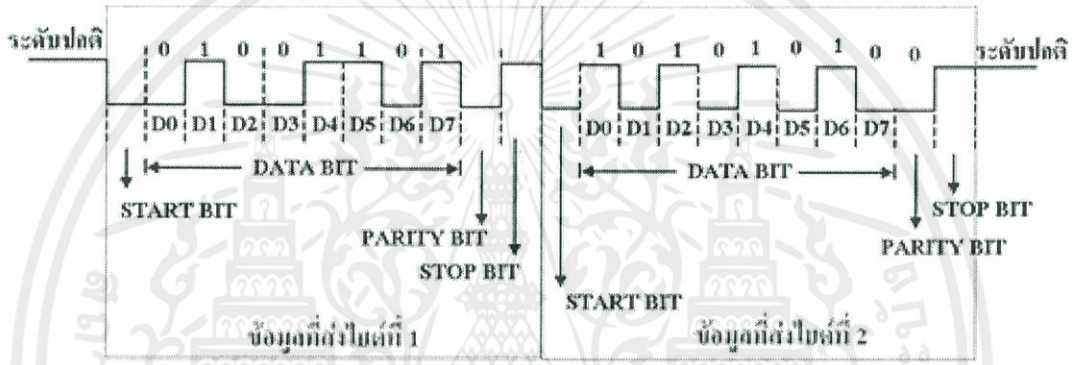
Morse code

- RS-232 - Recommended Standard 232
- RS422, RS-423, RS-485
- I2C - Inter-Integrated Circuit
- SPI - Serial Peripheral Interface
- USB - Universal Serial Bus
- Firewire
- Ethernet

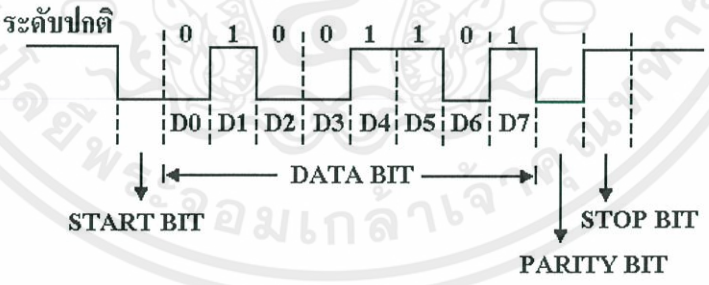
- Serial ATA - Serial Advanced TEchnology Attachment
- Serial Attach SCSI - Serial Attached Small Computer System
- Interface
- SONET - Synchronous Optical Network
- PCI Express - Peripheral Component Interconnect Express

การสื่อสารแบบอะซิงโครนัส (Asynchronous)

การสื่อสารแบบนี้ใช้มากในเครื่องไมโครคอมพิวเตอร์พีซี รูปแบบการส่งข้อมูล จะเป็นการส่งครั้งละ1 ไบต์โดยมีรูปแบบดังนี้



รูปที่ 2.1 การสื่อสารแบบอะซิงโครนัส (Asynchronous)



รูปที่ 2.2 ความหมายของบิต

Start Bit บอกจุดเริ่มต้นข้อมูล มีขนาด1 บิต

Data Bit ค่าข้อมูลมีได้5 ถึง8 บิต

Parity Bit บิตสำหรับใช้ตรวจสอบความผิดพลาดของข้อมูล มีได้ 0 ถึง 1 บิต

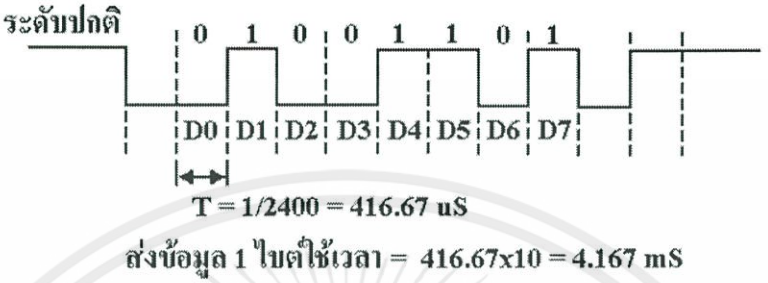
Stop Bit บิตใช้บอกจุดสิ้นสุดข้อมูล มีได้ 11.5และ 2 บิต

ความเร็วในการสื่อสาร

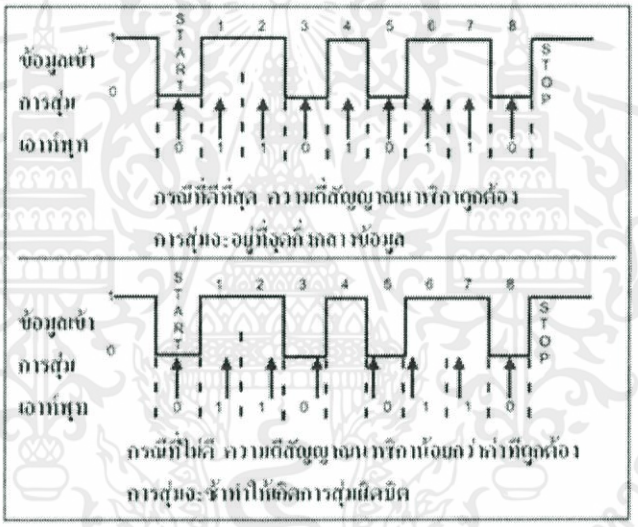
ความเร็วในการสื่อสาร หมายถึงจำนวนบิตที่ใช้รับส่งข้อมูลต่อวินาทีโดยปกติ จะมีค่าเท่ากับ 110 150 300 1200 2400 4800 9600 และ19200 บิตต่อวินาที อัตราความเร็วนี้บางครั้งก็เรียกว่าอัตราบอด(Baud rate) ทั้งตัวส่งและตัวรับต้อง กำหนดให้มีความเร็วในการสื่อสารเท่ากัน

ตัวอย่าง ข้อมูลส่งด้วยความเร็ว2400 บิตต่อวินาทีดังนั้นแต่ละบิตใช้เวลาส่งเท่ากับ

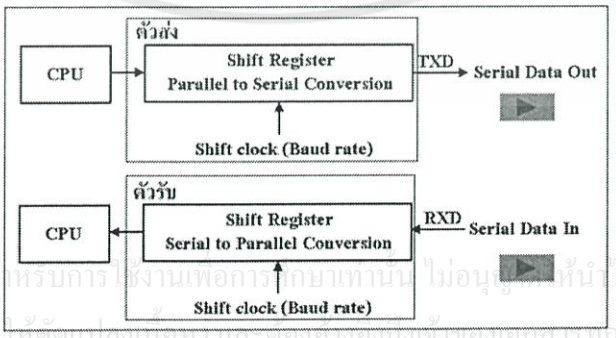
1/2400 = 416.67 ไมโครวินาที



รูปที่ 2.3 ความเร็วในการสื่อสาร



รูปที่ 2.4 การสุ่มข้อมูลแบบอะซิงโครนัส



รูปที่ 2.5 หลักการรับ-ส่งข้อมูลแบบอนุกรม

2.3.3 การสื่อสารแบบข้อมูลแบบไอโซโครนัส (Isochronous Transmission)

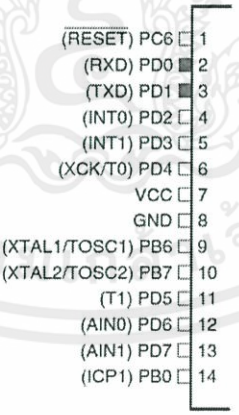
มาจากรากศัพท์ในภาษากรีก 2 คำคือคำว่า iso หมายถึง เท่ากัน และคำว่า chronous ที่หมาย ถึง เวลา เมื่อนำมารวมกันจึงหมายความว่า เวลาที่เท่ากัน สำหรับคุณสมบัติที่สำคัญของการส่งข้อมูลแบบไอโซโครนัสคือการส่งผ่านข้อมูลด้วยความเร็วสูงในอัตราคงที่และรับประกันเวลาในการส่ง

เนื่องจากการส่งข้อมูลแบบเรียลไทม์ เช่น ระบบออดิโอและวิดีโอ จำเป็นต้องส่งข้อมูล ด้วยความเร็วสูง ซึ่งการส่งข้อมูลแบบอะซิงโครนัส (มีการหน่วงเวลาเกิดขึ้นจากช่องว่างระหว่าง เฟรม) และซิงโครนัสก็ยังไม่สามารถรองรับได้ จึงเกิดการส่งข้อมูลแบบไอโซโครนัสขึ้นมา เพื่อใช้งานเรียลไทม์ที่รับประกันข้อมูลที่จะส่งมาถึงด้วยอัตราเร็วคงที่

โดยจะนำการส่งข้อมูลแบบไอโซโครนัสมาใช้เพื่อส่งผ่านข้อมูลบนบัส1394หรือเรียกว่าไฟร์ไวร์ (FireWire) การส่งผ่านข้อมูลของไอโซโครนัสจะตั้งอยู่บนพื้นฐานของแพ็กเก็ต โดยขนาดของแพ็กเก็ตจะส่งผ่านอยู่บนแชนเนลที่ให้ไว้ และสามารถแปรผันจากเฟรมไปยังเฟรมได้ ส่วนขนาดของแพ็กเก็ตจะถูกจำกัดโดยแบนด์วิดธ์เท่าที่มีอยู่

การสื่อสารข้อมูลแบบอนุกรมของไมโครคอนโทรลเลอร์ AVR

ไมโครคอนโทรลเลอร์ AVR สามารถสื่อสารข้อมูลแบบอนุกรมได้โดยใช้โมดูล USART ((Universal Synchronous and Asynchronous serial Receiver and Transmitter) สำหรับ ATMega 16 ขาพอร์ทอนุกรมกำหนดไว้ที่ PD0 Serial input RxD PD1 Serial output TxD



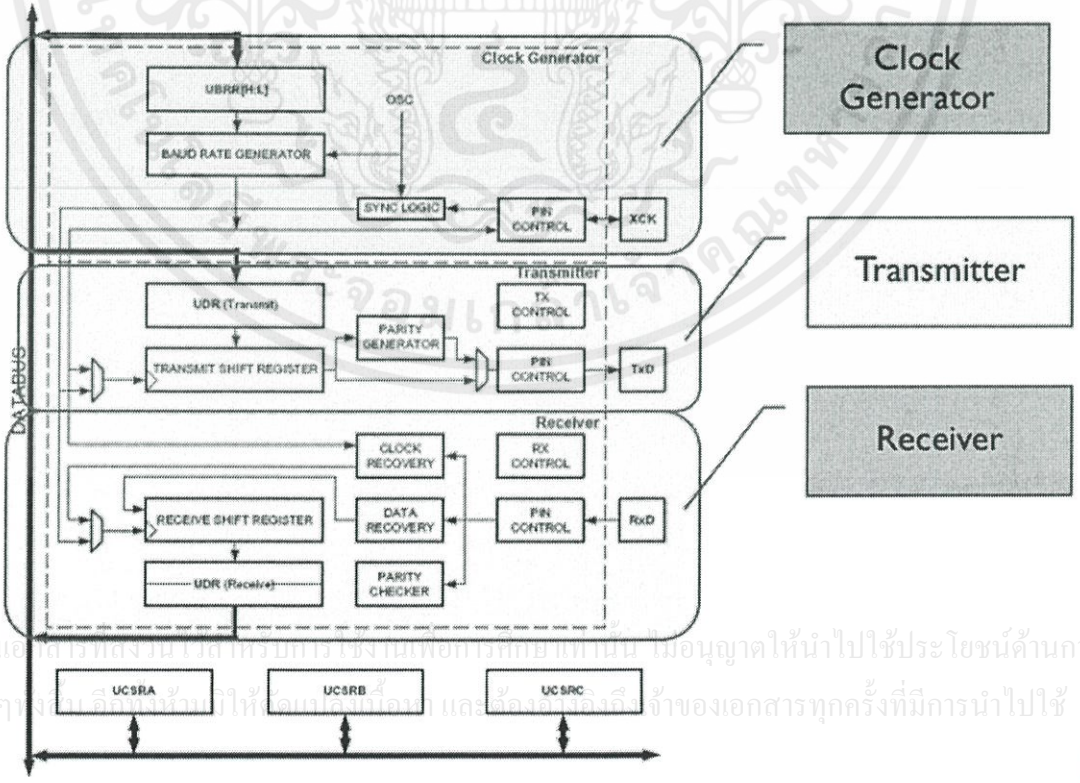
รูปที่ 2.6 ไมโครคอนโทรลเลอร์ AVR

คุณลักษณะของ ATMega168-USART

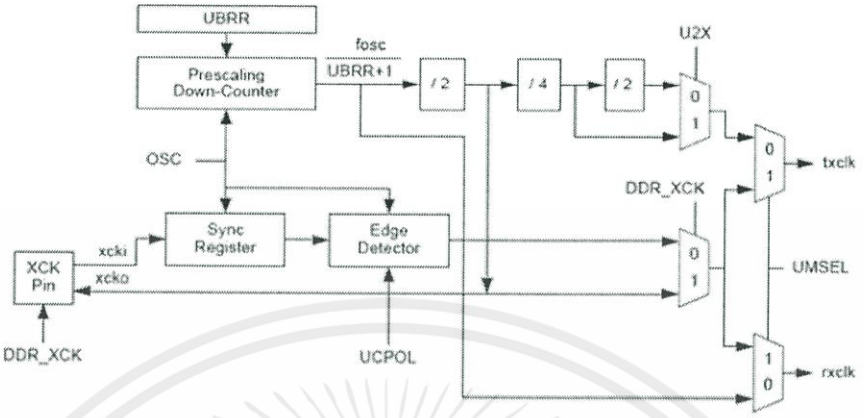
USART - Universal Synchronous Asynchronous Receiver Transmitter.

- ทำงานแบบ Full Duplex (การรับและส่งเป็นอิสระซึ่งกันและกัน)

- ทำงานได้ทั้งแบบ Asynchronous และ Synchronous
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- รองรับการรับส่งข้อมูลแบบ 5, 6, 7, 8, or 9 Data Bits และ 1 หรือ 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Three Separate Interrupts on TX Complete, TX Data Register
- Empty and RX Complete



รูปที่ 2.7 บล็อกไดอะแกรมของพอร์ทอนุกรม ATMega168 – USART



รูปที่ 2.8 Clock Generation Logic, Block Diagram

Signal description:

- txclk Transmitter clock (internal signal).
- rxclk Receiver base clock (internal signal).
- xcki Input from XCK pin (internal signal). Used for synchronous slave operation.
- xcko Clock output to XCK pin (internal signal). Used for synchronous master operation.
- foscSystem clock frequency.

การคำนวณหาอัตราบอต (Baud rate)

โหมดการทำงานอัตราบอตค่ารีจิสเตอร์ UBRR

อะซิงโครนัสปกติ (U2X = 0)

Baud = fosc/(16*(UBRR+1))

UBRR = (fosc/16*Baud)-1

อะซิงโครนัสทวีคูณ (U2X = 1)

Baud = fosc/(8*(UBRR+1))

UBRR = (fosc/8*Baud)-1

มาสเตอร์ซิงโครนัส

Baud = fosc/(2*(UBRR+1))

UBRR = (fosc/2*Baud)-1

UBRRnL and UBRRnH – USART baud rate registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Bit 15:12 – บิตสำรองเพื่อการใช้งาน บิตเหล่านี้ต้องให้เป็น 0

Bit 11:0 – UBRR11:0: USART บิตกำหนดอัตราบอด (baud rate) มีค่าได้ตั้งแต่ 0 -4095

2.4 การใช้งานโมดูล UART กับพอร์ตอนุกรม

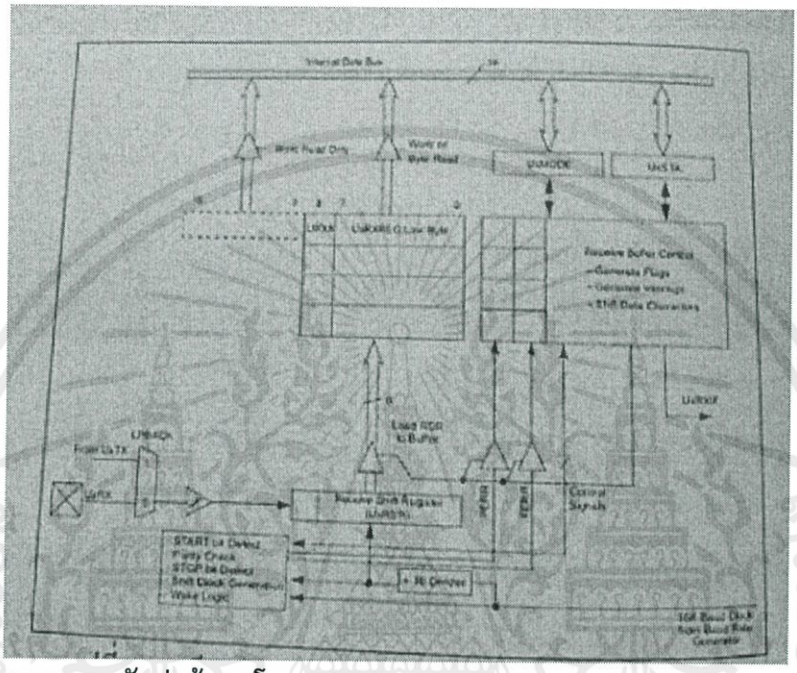
โมดูล Universal Asynchronous Receiver Transmitter(UART) หรือการสื่อสารข้อมูลในรูปแบบอะซิงโครนัส เป็นหนึ่งในโมดูลที่เกี่ยวข้องกับการอินพุต/เอาต์พุตแบบอนุกรมที่มีอยู่ภายใน dsPIC30F โดยโมดูล UART จะเป็นอะซิงโครนัสแบบฟูลดูเพล็ก(full duplex) รับส่งข้อมูลได้พร้อมกัน(สองทิศทาง)สามารถนำมาประยุกต์ใช้งานกับการเชื่อมต่อกับอุปกรณ์ภายนอก เช่น คอมพิวเตอร์ ,RS-232และRS-485เป็นต้น

ไมโครคอนโทรลเลอร์ dsPIC30F2010 มีโมดูล UART ให้ใช้งานได้ 1 UART โดยกำหนดขาพอร์ตสำหรับโมดูล UART ไว้ที่ขาพอร์ต U1RX/RF2 สะหรับรับข้อมูลอนุกรม และU1TX/RF3 สำหรับส่งข้อมูลอนุกรม นอกจากนี้ยังได้เพิ่มเติมพอร์ตอนุกรมเสริม(UART Alternate I/O) ที่ตำแหน่งขาพอร์ตU1ATX/RC13 สำหรับการส่งข้อมูลและ U1ARX/RC14 สำหรับการรับข้อมูลอนุกรม การเลือกใช้ต้องเลือกอย่างใดอย่างหนึ่งเท่านั้น โดยสามารถเลือกการใช้งานผ่านการเซตบิต ALTIO (บิตที่10)ในรีจิสเตอร์UxMODE(UARTx Mode Register)

คุณสมบัติที่สำคัญของโมดูล UART

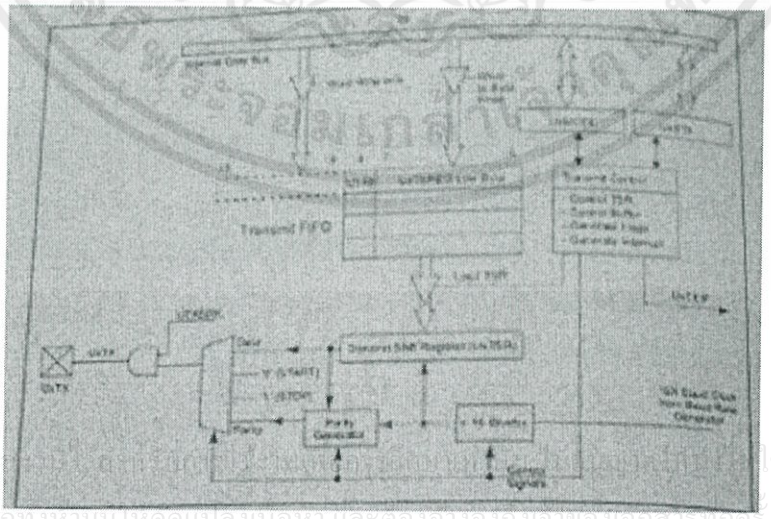
- รับส่งข้อมูลแบบฟูลดูเพล็กซ์
- มีคุณสมบัติการกำหนดข้อมูลการรับส่ง ไม่ว่าจะเป็นจำนวนบิตข้อมูล(8หรือ9บิต),บิตพาริตี้(คู่,คี่,ไม่มี),บิตหยุดข้อมูล(1หรือ2บิต)
- กำหนดอัตราบอต(Baud Rate) พร้อมกับปรีสเกลเลอร์ 16 บิต
- กำหนดอัตรารับส่งข้อมูลได้ตั้งแต่ 29bps ถึง1.875 Mbps ที่ความถี่30MHz
- มีบัฟเฟอร์รับส่งข้อมูล 4 ระดับ
- มีส่วนตรวจสอบข้อผิดพลาด พาริตี้,เฟรมข้อมูล และการซ้อนทับของบัฟเฟอร์ข้อมูล

- สนับสนุนการทานในโหมด 9 บิตข้อมูลพร้อมกับการตรวจจับแอดเดรส(ถ้าบิต9เป็น1จะเป็นแอดเดรส บิตที่ 9เป็น0จะเป็นข้อมูล)
- อินเตอร์รัปต์รับส่งข้อมูล
- โหมดลูปแบล็ค(Loop back) เพื่อการตรวจสอบข้อมูล



บล็อกไดอะแกรมการรับส่งข้อมูลโมดูล UART

รูป 2.9 บล็อกไดอะแกรมการส่งข้อมูลของโมดูล UART



รูป2.10 บล็อกไดอะแกรมการรับข้อมูลของโมดูล UART

2.5 รีจิสเตอร์ควบคุมโมดูล UART

- รีจิสเตอร์ UxMODE(UARTx Mode Register) กำหนดโหมดการทำงานของโมดูล UART เช่น ขาพอร์ตรับส่งข้อมูลที่ต้องการ อัตราการรับส่งข้อมูล,จำนวนบิตข้อมูล พาริตี้บิต และบิตหยุด เป็นต้น
- รีจิสเตอร์ UxSTA(UARTx Status and Control Register) รีจิสเตอร์แสดงสถานะการทำงานของโมดูล UART และบิตควบคุมการทำงานขอ UART เพิ่มเติม
- รีจิสเตอร์ UxRXREG(UARTx Receive Register) รีจิสเตอร์ส่งข้อมูล(เขียนได้อย่างเดียว) ใช้งานเพียง9บิต สำหรับส่งข้อมูลออกทางการเชื่อมต่อพอร์ตอนุกรม
- UxBRG(UARTx Baud Rate Register) รีจิสเตอร์กำหนดหรือความเร็วในการรับส่งข้อมูลของโมดูล UART

2.6 การคำนวณอัตราบอต(Baud Rate)

ก่อนการใช้งานการสื่อสารข้อมูลอนุกรมจะต้องกำหนดอัตราบอตสำหับการสื่อสารข้อมูล โดยคำนวณอัตราบอตได้ด้วยสมการดังนี้

$$BaudRate = \frac{Fcy}{16 \times (UxBEG + 1)}$$

$$UxBRG = \frac{Fcy}{16 \times BuadRate} - 1$$

โดย FCY= ความถี่สัญญาณนาฬิกาหลักหารด้วย 4(Fosc/4)

ตัวอย่าง การคำนวณอัตราบอตที่ 9600 bps กับPLLx1

กำหนดให้ Fosc = 4MHz,Baud Rate= 9600 และค่าPLLxเท่ากับ 4

FCY = (FOSC/4)xPLLx

= (4MHz/4)x4

= 4 MHz

อัตราความเร็วที่ต้องการ (Desired Buad Rate) = FCY/(16(UxBRG+1))

UxBRG = ((FCY/Desired Baud Rate)/16)-1

= ((4000000/9600)/16)-1

= [25.042] = 25

อัตราความเร็วที่คำนวณได้ = 4000000/(16(25+1))

= 9615

ข้อผิดพลาด(Error) = $\frac{9615 - 9600}{9600}$

= 0.16%

จากตัวอย่างข้างต้นอัตราบอตสูงสุดที่เป็นไปได้คือ Fcy/16(UxBRG = 0) และต่ำสุดเท่ากับ Fcy/(16*65536)

การคำนวณอัตราบอตที่9600 bps จะคำนวณหาค่า BRG เพื่อนำไปกำหนดให้กับรีจิสเตอร์ UxBRG ได้ดังนี้

Fcy = (FOSC x PLLx)/4
= (7.3728 M x 4)/4
= 7.3728 MHz
Baud Rate = Fcy/[16 x (UxBRG + 1)]
9600 = 7.3728 M / [16 x (UxBRG + 1)]
UxBRG = [7372800/(16 x 9600)]-1
= 47
Cale Baud Rate = 7372800/(16(47+1))
= 9600
Error = (9600-9600)/9600
= 0%

เพราะฉะนั้น ค่าที่นำมาใช้ในการกำหนดอัตราบอตที่ 9600 คือ 47 นำไปกำหนดให้กับรีจิสเตอร์ UxBRG

2.7 ฟังก์ชันไลบารีโมดูล UART

ฟังก์ชันเกี่ยวข้องกับโมดูล UART หรือการสื่อสารข้อมูลอนุกรม (RS-232)ภายในไฟล์ uart.h มีดังนี้คือ

- ฟังก์ชันติดตั้งการใช้งานโมดูล UART
- ฟังก์ชันอ่านเขียนข้อมูลกับบัฟเฟอร์ โดยสนับสนุนการรับส่งข้อมูลทั้งแบบ 8 บิตและ 9 บิตโดยมีรายละเอียดของฟังก์ชันและมาโครดังนี้

ฟังก์ชัน void ConfigInt UARTx(unsigned int config)
กำหนดคุณสมบัติการใช้งานอินเตอร์รัปต์ของโมดูล UARTx
รูปแบบการใช้งาน : ConfigIntUART1(config);
ConfigIntUART2(config);

อาร์กิวเมนต์:

Config กำหนดคุณสมบัติเปิด/ปิดการใช้งานอินเตอร์รัปต์ UART รายการอาร์กิวเมนต์มีรายละเอียดดังนี้

เปิด/ปิดการใช้งานอินเตอร์รัปต์รับข้อมูล:

UART_RX_INT_EN	เปิดการใช้งานอินเตอร์รัปต์รับข้อมูล
UART_RX_INT_DIS	ปิดการใช้งานอินเตอร์รัปต์รับข้อมูล

กำหนดระดับความสำคัญของอินเตอร์รัปต์รับข้อมูล

UART_RX_INT_PR0	มีระดับความสำคัญสูงสุด
UART_RX_INT_PR1	
UART_RX_INT_PR2	
UART_RX_INT_PR3	
UART_RX_INT_PR4	
UART_RX_INT_PR5	
UART_RX_INT_PR6	
UART_RX_INT_PR7	มีระดับความสำคัญต่ำสุด

เปิด/ปิดการใช้งานอินเตอร์รัปต์ส่งข้อมูล:

UART_RX_INT_EN	เปิดการใช้งานอินเตอร์รัปต์ส่งข้อมูล
UART_RX_INT_DIS	ปิดการใช้งานอินเตอร์รัปต์ส่งข้อมูล

กำหนดระดับความสำคัญของอินเตอร์รัปต์ส่งข้อมูล

UART_RX_INT_PR0	มีระดับความสำคัญสูงสุด
UART_RX_INT_PR1	
UART_RX_INT_PR2	
UART_RX_INT_PR3	
UART_RX_INT_PR4	
UART_RX_INT_PR5	
UART_RX_INT_PR6	
UART_RX_INT_PR7	มีระดับความสำคัญต่ำสุด

ค่าที่ส่งกลับ:

ตัวอย่างเช่น

ConfigIntUART1 (UART_RX_INT_EN & UART_RX_INT_PR5 & UART_TX_INT_EN & UART_TX_INT_PR3);

ฟังก์ชัน void OpenUARTx(unsigned int config 1,unsigned int config2,unsigned int ubrg)

กำหนดคุณสมบัติและเปิดการใช้งานโมดูล UART

รูปแบบการใช้งาน: OpenUART1(config1.config2,ubrg);

OpenUART2(config1.config2,ubrg);

อาร์กิวเมนต์:

Config1 กำหนดคุณสมบัติโหมดการทำงานภายในรีจิสเตอร์ UxMODE รายการอาร์กิวเม้นมีรายละเอียดดังนี้

เปิด/ปิดการใช้งานโมดูล UART:

UART_EN เปิดการใช้งาน UART ขาพอร์ต UART ถูกควบคุมด้วยโมดูล UART

UART_DIS เปิดการใช้งาน UARTขาพอร์ต UARTถูกควบคุมจากบิตภายในรีจิสเตอร์ PORT,LAT และ TRIS

การทำงานโมดูล UART ในโหมดไอเดิล:

UART_IDLE_CON UART ทำงานต่อเนื่องเมื่อเข้าสู่โหมดไอเดิล

UART_IDLE_STOP UARTหยุดทำงานเมื่อเข้าสู่โหมดไอเดิล

กำหนดใช้งานขาTX/RX เสริม:

UART_ALTRX_ALTTX ใช้ขา TX/RX เสริม (UxATX และ UxARX)

UART_RX_TX ใช้ขา TX/RX ปกติ (UxTX และ UxRX)

*ขาTX/RXเสริมสำหรับโมดูล UART ให้ดูเพิ่มเติมใน Data Sheet เบอร์ที่เลือกใช้งานว่ามีหรือไม่

โมดูล UART เริ่มต้นทำงาน(Wake-up)เมื่อพบบิตเริ่มต้น (Start bit) ขณะอยู่ในโหมดสลีป:

UART_EN_WAKE เปิดการใช้งาน Wake-up

UART_DIS_WAKE ปิดการใช้งาน Wake-up

เปิด/ปิดการใช้งานโหมดลูปแบ็ก(Loopback mode):

UART_EN_LOOPBACK เปิดใช้งานโหมดลูปแบ็ก

UART_DIS_LOOPBACK ปิดการใช้งานโหมดลูปแบ็ก

เปิดโมดูลอินพุตจับสัญญาณ(Auto Baud):

UART_EN_ABAUD โมดูลอินพุตจับสัญญาณจากขาUxRX

UART_DIS_ABAUD โมดูลอินพุตจับสัญญาณจากขา ICx

กำหนดบิตพาริตี้(Parity bits)และบิตข้อมูล(Data bits):

UART_NO_PAR_9BIT ไม่มีพาริตี้บิต,9บิตข้อมูล

UART_ODD_PAR_8BIT พาริตี้บิต คี่,8บิตข้อมูล

UART_EVEN_PAR_8BIT พาริตี้บิต คู่,8บิตข้อมูล

UART_NO_PAR_8BIT ไม่มีพาริตี้บิต,8บิตข้อมูล

กำหนดจำนวนบิตหยุด(Stop bits):

UART_2STOPBITS บิตหยุดจำนวน 2 บิต

UART_ISTOPBIT บิตหยุดจำนวน 1 บิต

Config2 กำหนดคุณสมบัติโหมดการทำงานภายในรีจิสเตอร์ UxSTA รายการอาร์กิวเมนต์มีรายละเอียดดังนี้

กำหนดโหมดอินเตอร์รัปต์ส่งข้อมูล UART:

UART_INT_TX_BUF_EMPTY อินเตอร์รัปต์เมื่อรีจิสเตอร์ TXBUF ว่าง

UART_INT_TX อินเตอร์รัปต์ทุกครั้งเมื่อส่งข้อมูลออกไป

เมื่อมีการหยุด(Break)ในขณะที่ส่งข้อมูล:

UART_TX_PIN_NORMAL กำหนดให้ขา UxTX ทำงานตามปกติ

UART_TX_PIN_LOW กำหนดให้ขา UxTX ขับสัญญาณลอจิก “0”

เปิด/ปิดขาส่งข้อมูล(ขาTX):

UART_TX_ENABLE เปิดขาส่งข้อมูล

UART_TX_DISABLE ปิดขาส่งข้อมูล

กำหนดโหมดอินเตอร์รัปต์เมื่อมีการรับข้อมูล:

UART_INT_RX_BUF_FUL อินเตอร์รัปต์เมื่อบัฟเฟอร์เต็ม(มีข้อมูลอักขระ 4 ข้อมูล)

UART_INT_RX_3_4_FUL อินเตอร์รัปต์เมื่อ3/4ของบัฟเฟอร์เต็ม(มีข้อมูลอักขระ 3ข้อมูล)

UART_INT_RX_CHAR อินเตอร์รัปต์ทุกครั้งเมื่อมีการรับข้อมูล

เปิด/ปิดการตรวจจับแอดเดรสของอักขระ(บิตที่8ของข้อมูลเป็น1):

UART_ADR_DETECT_EN เปิดการตรวจจับบิตที่ 8

UART_ADR_DETECT_DIS ปิดการตรวจจับบิตที่ 8

เคลียร์บิตโอเวอร์รัน(Overrun Bit):

UART_RX_OVERRUN_CLEAR กำหนดให้เคลียร์บิตโอเวอร์รัน

Ubrg กำหนดอัตราบอดในรีจิสเตอร์ UxBRG

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่างเช่น

```
Baud = 47;
UMODEvalue = UART_EN & UART_IDLE_CON & UART_DIS_WAKE &
             UART_EN_LOOPBACK & UART_EN_ABAUD &
             &UART_NO_PAR_8BIT & UART_1STOPBIT;
U1STAvalue = UART_INT_TX_BUF_EMPTY &UART_TX_PIN_NORMAL &
             UART_TX_ENABLE & UART_INT_RX_CHAR &
             UART_ADR_DETECT_DIS & UART_RX_OVERRUN_CLEAR
OpenUART1 (U1MODEvalue,U1STAvalue,baud);
```

ฟังก์ชัน void WriteUARTx(unsigned int data) หรือ

void putcUART(unsigned int data)

เขียนข้อมูลไปยังรีจิสเตอร์บัฟเฟอร์ส่งข้อมูล(UxTXREG)

รูปแบบการใช้งาน: WriteUART1(data);หรือ putcUART1(data)

WriteUART2(data);หรือ putcUART2(data)

อาร์กิวเมนต์: data ข้อมูลที่จะส่งออกขนาด 16 บิต

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่างเช่น

WriteUART1 (0xFF);

ฟังก์ชัน void putsUARTx(unsigned int *buffer)

ใช้หลักการเดียวกับฟังก์ชัน WriteUARTx() แต่เป็นการส่งข้อมูลต่อเนื่องจนหมดข้อมูลที่ตัวแปรพอยน์เตอร์ buffer ชี้อยู่

รูปแบบการใช้งาน: putsUART1(*buffer);

putsUART2(*buffer);

อาร์กิวเมนต์: *buffer ข้อมูลที่จะส่งในรูปแบบชุดตัวอักษร(string)

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่างเช่น

Char *TxData_loc = “MPLAB C Compiler”;

putsUART1(Txdata_loc);

ฟังก์ชัน unsigned int ReadUARTx(void) หรือ unsigned int getcUARTx(void)อ่านข้อมูลจากรีจิสบัฟเฟอร์รับข้อมูล(UxRXREG)

รูปแบบการใช้งาน: ReadUART1(); หรือ getcUART1();

ReadUART2(); หรือ getcUART2();

อาร์กิวเมนต์: ไม่มี

ค่าที่ส่งกลับ: ข้อมูลที่อ่านได้จากบัฟเฟอร์รับข้อมูลภายในรีจิสเตอร์ UxRXREG ขนาด 16 บิต

ตัวอย่างเช่น unsigned int RX data;

RX_Data = ReadUART();

ฟังก์ชัน unsigned int getsUARTx(unsigned int length,unsigned int *buffer,unsigned int uart_data_wait)

ใช้หลักการตรวจสอบข้อมูลเช่นเดียวกับฟังก์ชัน ReadUARTx() แต่จะวนอ่านข้อมูลตามจำนวนที่กำหนดไว้ในตัวแปร length หรืออ่านจนกว่าจะหมดเวลาที่กำหนดในตัวแปร data_wait

รูปแบบการใช้งาน: getUART1(length,*buffer,uart_data_wait);
getUART2(length,*buffer,uart_data_wait);

อาร์กิวเมนต์: length ขนาดข้อมูลที่ต้องการอ่าน
*buffer ข้อมูลที่อ่านได้ในแบบชุดอักษรหรือแบบสตริง
Data_wait ระยะเวลาการรอข้อมูล(Time out)

ค่าที่ส่งกลับ: จำนวนข้อมมูลที่อ่านได้

ตัวอย่างเช่น

```
Datarem = getsUART1(6,Rxdata_loc,40)
```

ฟังก์ชัน char BusyUARTx(void)

ตรวจสอบบิต TRMT ในรีจิสเตอร์ U1STA เพื่อตรวจสอบว่ารีจิสเตอร์บัฟเฟอร์ส่งข้อมูลว่างหรือไม่

รูปแบบการใช้งาน BusyUART1();
BusyUART2();

อาร์กิวเมนต์ ไม่มี

ค่าที่ส่งกลับ สถานะของบัฟเฟอร์ส่งข้อมูล ค่าที่ส่งกลับมามีดังนี้
เท่ากับ 1 : แสดงว่ามีข้อมูลในบัฟเฟอร์ส่งข้อมูล
เท่ากับ 0 : แสดงว่าไม่มีข้อมูลในบัฟเฟอร์ส่งข้อมูล

ตัวอย่างเช่น while(BusyUART1());

ฟังก์ชัน char DataRdyUARTx(void)

ตรวจสอบบิต URXDA ในรีจิสเตอร์ U1STA เพื่อตรวจสอบว่ารีจิสเตอร์บัฟเฟอร์รับข้อมูลว่างหรือไม่

รูปแบบการใช้งาน: DataRdyUART1();
DataRdyUART2();

อาร์กิวเมนต์: ไม่มี

ค่าที่ส่งกลับ สถานะของบัฟเฟอร์รับข้อมูล ค่าที่ส่งกลับมามีดังนี้
เท่ากับ 1 : แสดงว่ามีข้อมูลในบัฟเฟอร์รับข้อมูล
เท่ากับ 0 : แสดงว่าไม่มีข้อมูลในบัฟเฟอร์รับข้อมูล

ตัวอย่างเช่น while(DataRdyUART1());

ฟังก์ชัน void CloseUARTx(void)

ปิดการใช้งานโมดูล UARTx

รูปแบบการใช้งาน: CloseUART1();

CloseUART2();

อาร์กิวเมนต์: ไม่มี

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่างเช่น CloseUART1();

มาโคร(Macros)

- EnableIntUxRX เปิดการใช้งานอินเตอร์รัปต์รับข้อมูล
- EnableIntUxTX เปิดการใช้งานอินเตอร์รัปต์ส่งข้อมูล
- DisableIntUxRx ปิดการใช้งานอินเตอร์รัปต์รับข้อมูล
- DisableIntUxTx ปิดการใช้งานอินเตอร์รัปต์ส่งข้อมูล
- SetPriorityIntUxRX กำหนดระดับความสำคัญของอินเตอร์รัปต์รับข้อมูล UART
- SetPriorityIntUxTX กำหนดระดับความสำคัญของอินเตอร์รัปต์ส่งข้อมูล UART

หมายเหตุ X แทนด้วยตัวเลข 1,2,3,4 เป็นต้น

ฟังก์ชัน unsigned char MasterReadI2C(void);

อ่านข้อมูลในรีจิสเตอร์ I2CRCV โดย เซตบิต RECN ในรีจิสเตอร์ I2CCON แล้วรอจนบิต RECNถูกเคลียร์ด้วยฮาร์ดแวร์ จากนั้นเคลียร์บิต I2COV ในรีจิสเตอร์ I2CSTAT แล้วอ่านค่าในรีจิสเตอร์ I2CRCV

รูปแบบการใช้งาน: MasterReadI2C();

อาร์กิวเมนต์: ไม่มี

ค่าที่ส่งกลับ: ข้อมูลที่อ่านได้จากบัส I^2C

ตัวอย่างเช่น unsigned char value;

Value = MasterReadI2C();

ฟังก์ชัน unsigned int MasterReadI2C(unsigned char *wrptr);

ฟังก์ชันเขียนข้อมูลสตริง ไปที่บัส I^2C โดยเรียกใช้งานฟังก์ชัน MasterWriteI2C()

รูปแบบการใช้งาน: MasterputsI2C(*wrptr);

อาร์กิวเมนต์: *wrptr ชุดข้อมูลสตริง

ค่าที่ส่งกลับ: เท่ากับ -3 : แสดงว่ามีการชนกันของข้อมูล

เท่ากับ 0 : พบอักษรนัล(Null Character)ในชุดข้อมูลสตริงที่ส่ง

ตัวอย่างเช่น unsigned char string[] = “MICROCHIP”;

unsigned char *wrptr ;

```
wrptr = string;
MasterputsI2C(wrptr) ;
```

ฟังก์ชัน unsigned char MasterWriteI2C(unsigned char data_out);

เขียนข้อมูลไปที่รีจิสเตอร์ I2CTRN และควรตรวจสอบบิต IWCOL ในรีจิสเตอร์ I2CSTAT ว่ามีการชนกันของข้อมูลหรือไม่

รูปแบบการใช้งาน: MasterWriteI2C(data_out);
อาร์กิวเมนต์: data_out ข้อมูล 1 ตัวอักษร
ค่าที่ส่งกลับ: เท่ากับ 1 : แสดงว่ามีการชนกันของข้อมูล
เท่ากับ 0 : แสดงว่าไม่มีการชนกันของข้อมูล

ตัวอย่างเช่น MasterWriteI2C('a');

ฟังก์ชัน unsigned int SlavegetsI2C(unsigned char *rdptr, unsigned int i2c_data_wait);

ฟังก์ชันอ่านข้อมูลสตริงจากบัส I^2C ในโหมดสเลฟ

รูปแบบการใช้งาน: SlavegetsI2C(*rdptr,i2c_data_wait);
อาร์กิวเมนต์: *rdptr ตัวแปรพอยต์เตอร์ที่ชี้ไปยังข้อมูลที่จะอ่าน
i2c_data_wait ระยะเวลารอข้อมูลที่จะอ่าน
ค่าที่ส่งกลับ: จำนวนข้อมูลที่อ่านได้ขนาด 16 บิต

ตัวอย่างเช่น
```
unsigned char string[12];
unsigned char *rdptr ;
rdptr = string;
i2c_data_wait = 0X11;
SlavegetsI2C(rdptr,i2c_data_wait);
```

ฟังก์ชัน unsigned char SlaveReadI2C(void);

อ่านข้อมูลในโหมดสเลฟ โดยตรวจสอบบิต RBF จนกว่าบิตจะถูกเซต แล้วเคลียร์บิต I2COV ในรีจิสเตอร์I2CSTAT ก่อนที่จะอ่านค่าในรีจิสเตอร์ I2CRCV

รูปแบบการใช้งาน: SlaveReadI2C();
อาร์กิวเมนต์: ไม่มี
ค่าที่ส่งกลับ: จำนวนข้อมูลที่อ่านได้จากบัส I^2C

ตัวอย่างเช่น
```
unsigned char value;
```

```
Value = SlaveReadI2C();
```

ฟังก์ชัน unsigned int SlaveReadI2C(unsigned char *wrptr);

ฟังก์ชันเขียนข้อมูลสตริงจากบัส I^2C ในโหมดสเลฟ

รูปแบบการใช้งาน: SlaveputsI2C(*wrptr);

อาร์กิวเมนต์: *wrptr ข้อมูลสตริงที่จะเขียนไปที่บัส I^2C

ค่าที่ส่งกลับ: เท่ากับ 0 : เมื่อส่งข้อมูลจนถึงอักษรนัล(Null Character)

ตัวอย่างเช่น

```
unsigned char string[] = “MICROCHIP”;
unsigned char *rdptr ;
rdptr = string;
SlaveputsI2C(rdptr) ;
```

ฟังก์ชัน void SlaveWriteI2C(unsigned char data_out);

เขียนข้อมูลไปที่รีจิสเตอร์ I2CTRN แล้วเซตบิต SCLREL ในรีจิสเตอร์ I2CCON

รูปแบบการใช้งาน: SlaveWriteI2C(data_out);

อาร์กิวเมนต์: data_out ข้อมูล 1 ตัวอักษร

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่างเช่น SlaveWriteI2C(‘a’);

ฟังก์ชัน void CloseI2C(void);

ปิดการใช้งานโมดูลระบบบัส I^2C และเคลียร์บิตอินเตอร์รัปต์และแฟล็กซ์ทั้งมาสเตอร์และสเลฟ

รูปแบบการใช้งาน: CloseI2C();

อาร์กิวเมนต์: ไม่มี

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่างเช่น CloseI2C();

มาโคร(Macros)

- EnableInt MI2C เปิดการใช้งานอินเตอร์รัปต์มาสเตอร์ I^2C
- DiableIntMI2C ปิดการใช้งานอินเตอร์รัปต์มาสเตอร์ I^2C
- SetPriorityIntMI2C กำหนดระดับความสำคัญของอินเตอร์รัปต์มาสเตอร์ I^2C
- EnableIntSI2C เปิดการใช้งานอินเตอร์รัปต์สเลฟ I^2C
- DisableIntSI2C ปิดการใช้งานอินเตอร์รัปต์สเลฟ I^2C

- SetPriorityIntSI2C กำหนดระดับความสำคัญของอินเตอร์รัปต์สเลฟ I^2C

2.8 การใช้งานโมดูลสื่อสารข้อมูลอนุกรม I^2C

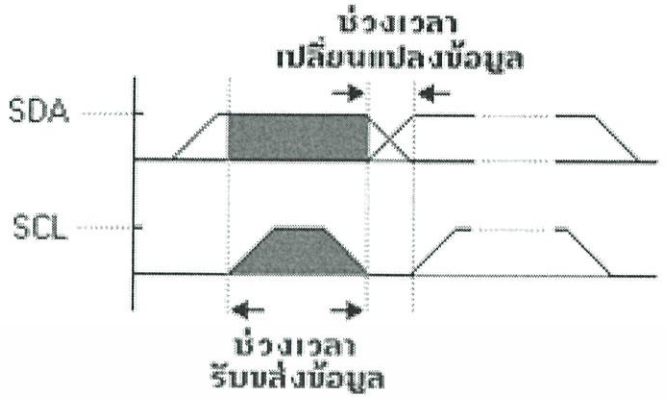
โมดูล Inter-Integrated Circuit(I^2C)หรือโมดูลการสื่อสารข้อมูลอนุกรมด้วยสายสัญญาณ 2 เส้น ประกอบไปด้วย สายข้อมูลอนุกรมหรือ SDA(Serial Data Line) และสายสัญญาณนาฬิกาอนุกรมหรือ SCL(Serial Clock Line) การสื่อสารข้อมูลด้วย I^2C บัส ถูกนำมาใช้ในการติดต่อระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์ภายนอกที่มีบัส I^2C เช่น ไอซี EEPROM แบบอนุกรม อุปกรณ์อินพุตเอาต์พุต และไอซีแปลงสัญญาณอะนาลอกเป็นดิจิตอล เป็นต้น หรือใช้ติดต่อกับไมโครคอนโทรลเลอร์ด้วยกันเอง โดยตัวที่ทำหน้าที่ในการควบคุมจะเรียกว่า มาสเตอร์(Master)และตัวที่ถูกควบคุมจะเรียกว่า สเลฟ (Slave)

2.8.1 คุณสมบัติของระบบบัส I^2C

- มาสเตอร์และสเลฟทำงานแยกกันโดยอิสระ
- รองรับการมีมาสเตอร์มากกว่าหนึ่ง(Multi-Master)
- กำหนดรูปแบบการติดต่อสื่อสารได้ทั้งแบบ 7 บิตและ 10 บิต
- การเรียกแอดเดรสมีรูปแบบเดียวกับโปรโตคอล I^2C บัส
- มีโหมดส่งสัญญาณซ้ำ เพื่อรองรับข้อความทั้งหมดโดยไม่สนใจแอดเดรสของสเลฟ
- หน่วงสัญญาณนาฬิกาอัตโนมัติเพื่อรอข้อมูลตอบรับจากตัวสเลฟ
- รองรับความถี่ของสัญญาณบัสที่ 100 kHz และ 400kHz

2.8.2 โปรโตคอล I^2C บัส

การสื่อสารข้อมูลอนุกรมด้วยบัส 2 เส้น ในรูปแบบ I^2C บัสจะมีการกำหนดรูปแบบการส่งข้อมูลที่เรียกว่า โปรโตคอล I^2C บัส เพื่อใช้ในการติดต่อสื่อสารระหว่างมาสเตอร์และสเลฟ รูปแบบการส่งแสดงดังรูปที่ 2.11



รูปที่ 2.11 แสดงสถานะการทำงานของโปรโตคอล I^2C บัส

2.8.3 รายละเอียดของสัญญาณบนระบบบัส I^2C

- สภาวะเริ่มต้น(S)

  สายสัญญาณ SDA เปลี่ยนจาก High เป็น Low ขณะที่สายสัญญาณ SCL ยังคงค้างสถานะ High เมื่อสายสัญญาณSDA เป็น Low แล้ว สายสัญญาณ SCL จึงเปลี่ยนสถานะจาก High เป็น Low เรียกสภาวะนี้ว่า สภาวะเริ่มต้น(START Condition)

- สภาวะคงอยู่ของข้อมูล(D)

  สายสัญญาณ SDA จะต้องคงสภาวะเดิมไว้ว่าเป็น High (ข้อมูล"1")เป็น Low (ข้อมูล"0") เมื่อสายสัญญาณSCLเปลี่ยนสถานะจาก Low เป็น High สภาวะข้อมูลจึงจะสมบูรณ์ เรียกสภาวะนี้ว่า สภาวะคงอยู่ของข้อมูล(Data Valid)

- สภาวะรอหรือข้อมูลไม่สมบูรณ์(Q)

  สายสัญญาณ SCL มีสถานะ Low ขณะที่สายสัญญาณ SDA อาจมีสถานะ High หรือ Low เรียกสภาวะนี้ว่าสภาวะรอหรือข้อมูลไม่สมบูรณ์ (WAIT/Data Invalid)

- สภาวะรับรู้ข้อมูล(A)or(N)

  สารส่งข้อมูล 1 ไบต์จะต้องมีสัญญาณตอบรับ(ACK)หรือไม่มีสัญญาณตอบรับ(NACK)จากตัวรับ โดยตัวรับจะต้องควบคุมสัญญาณ SDA หากตัวรับทำให้สายสัญญาณ SDA เป็น Low แสดงว่ามีการตอบรับ และหากเป็น High แสดงว่าไม่มีการตอบรับ โดยสัญญาณตอบรับจะแทนด้วยคาบเวลา 1บิต ใช้สัญญาณ SCL 1 สัญญาณนาฬิกา

- สภาวะหยุด(P)

  สายสัญญาณ SDA เปลี่ยนสถานะจาก Low เป็น High ขณะที่สายสัญญาณ SCL เป็น High เรียกสภาวะนี้ว่า สภาวะหยุด (STOP Condition)

- สภาวะบัสว่าง(I)

  สายสัญญาณ SCL และ SDA มีสถานะเป็น High ทั้งคู่ หลังจากสภาวะหยุดและก่อนสภาวะเริ่มต้น เรียกสภาวะนี้ว่า สภาวะบัสว่าง (Bus Idle)

2.8.4 เงื่อนไขการรับ-ส่งข้อมูลบนบัส I^2C

1.การรับส่งข้อมูลจะมีได้เมื่อบัสว่างเท่านั้น

2.ระหว่างรับส่งข้อมูล สายสัญญาณ SDA จะต้องคงสถานะไว้เมื่อสายสัญญาณ SCL เป็น High หากมีการเปลี่ยนแปลงสัญญาณ SDA ขณะที่สายสัญญาณ SCL เป็น High สายสัญญาณจะถูกแปลความว่าเป็นสภาวะเริ่มต้นหรือสภาวะหยุดได้

3.การรับส่งข้อมูลระหว่างอุปกรณ์มาสเตอร์กับอุปกรณ์สเลฟ มีขั้นตอนดังต่อไปนี้

- เริ่มต้นรับส่งข้อมูลด้วย สภาวะเริ่มต้น
- ส่งข้อมูลแอดเดรสติดต่อกับอุปกรณ์โดยอ้างอิงแบบ 7 บิตหรือ 10 บิต
- รับ/ส่ง ข้อมูล
- หยุดรับส่งข้อมูลด้วย สภาวะหยุด

2.9 รีจิสเตอร์โมดูล I^2C บัส

- รีจิสเตอร์ I2CRCV(I2C Receive Register)เป็นรีจิสเตอร์รับข้อมูลขนาด 8 บิต
- รีจิสเตอร์ I2CTRN(I2C Transmit Register)เป็นรีจิสเตอร์ส่งข้อมูลขนาด 8 บิต
- รีจิสเตอร์ I2CBRG(I2C Baud Generator Register)เป็นรีจิสเตอร์กำหนดอัตราเร็วในการรับส่งข้อมูลบนระบบบัส I^2Cขนาด 9 บิต
- รีจิสเตอร์ I2CCON(I2C Control Register)

  จะควบคุมการทำงานหลักของโมดูลระบบบัส I^2C ขนาด 16 บิต
- รีจิสเตอร์ I2CSTAT(I2C Status Register)

  รีจิสเตอร์สถานะของโมดูลระบบบัส I^2C ขนาด 16บิต
- รีจิสเตอร์ I2CADD(I2C Address Register)

  ใช้กำหนดแอดเดรสของอุปกรณ์สเลฟ ขนาด 10 บิต

2.10 ฟังก์ชันไลบารีโมดูล I^2C

เป็นฟังก์ชันที่เกี่ยวข้องกับโมดูลสื่อสารข้อมูลอนุกรมแบบ I^2C จะถูกประกาศอยู่ภายในไฟล์ i2c.h ประกอบด้วย

- ฟังก์ชันติดตั้งการใช้งานโมดูล I^2C ทั้งแบบโหมดมาสเตอร์และโหมดสเลฟ
- ฟังก์ชันอ่านเขียนข้อมูลไบต์เดียวและหลายนไบต์

- ฟังก์ชันอ่านค่าจากรีจิสเตอร์และการกำหนดคุณสมบัติอินเตอร์รัปต์
- มาโครเปิด/ปิดการใช้งานอินเตอร์รัปต์ของโมดูล I^2C

โดยมีรายละเอียดของฟังก์ชันและมาโครดังนี้

ฟังก์ชัน void ConfigIntI2C(unsigned int config);

กำหนดคุณสมบัติการใช้งานอินเตอร์รัปต์ของโมดูล I^2C

รูปแบบการใช้งาน: ConfigIntI2C(config);

อาร์กิวเมนต์:

Config กำหนดเปิด/ปิดและระดับความสำคัญการใช้งานอินเตอร์รัปต์ รายการอาร์กิวเมนต์มีรายละเอียดดังนี้

เปิด/ปิดการใช้งานอินเตอร์รัปต์ I^2Cมาสเตอร์:

MI2C_INT_ON เปิดการใช้งานอินเตอร์รัปต์ของมาสเตอร์

MI2C_INT_OFF ปิดการใช้งานอินเตอร์รัปต์ของมาสเตอร์

เปิด/ปิดการใช้งานอินเตอร์รัปต์ I^2C สเลฟ:

SI2C_INT_ON เปิดการใช้งานอินเตอร์รัปต์ของสเลฟ

SI2C_INT_OFF ปิดการใช้งานอินเตอร์รัปต์ของสเลฟ

กำหนดระดับความสำคัญของอินเตอร์รัปต์ I^2C มาสเตอร์ :

MI2C_INT_PRI_7 มีระดับความสำคัญต่ำสุด

MI2C_INT_PRI_6

MI2C_INT_PRI_5

MI2C_INT_PRI_4

MI2C_INT_PRI_3

MI2C_INT_PRI_2

MI2C_INT_PRI_1

MI2C_INT_PRI_0 มีระดับความสำคัญสูงสุด

กำหนดระดับความสำคัญของอินเตอร์รัปต์ I^2C สเลฟ :

SI2C_INT_PRI_7 มีระดับความสำคัญต่ำสุด

SI2C_INT_PRI_6

SI2C_INT_PRI_5

SI2C_INT_PRI_4

SI2C_INT_PRI_3

SI2C_INT_PRI_2

SI2C_INT_PRI_1

SI2C_INT_PRI_0 มีระดับความสำคัญสูงสุด

ค่าที่ส่งกลับ ไม่มี

ตัวอย่างเช่น configIntI2C(MI2C_INT_ON & MI2C_INT_PRI_3 & SI2C_INT_ON & SI2C_INT_PRI_5);

ฟังก์ชัน void OpenI2C(unsigned int config1,unsigned int config2);

กำหนดรายเอียดและเปิดการใช้งานโมดูล I^2C

รูปแบบการใช้งาน OpenI2C(config1,config2);

อาร์กิวเมนต์

Config กำหนดค่าให้กับรีจิสเตอร์ I2CCON รายการอาร์กิวเมนต์มีรายละเอียดดังนี้

เปิด/ปิดโมดูล I^2C :

I2C_ON เปิดโมดูล I^2C

I2C_OFF ปิดโมดูล I^2C

การทำงานของโมดูล I^2C ในโหมดไอเดิล :

I2C_IDLE_STOP หยุดการทำงานเมื่อเข้าสู่โหมดไอเดิล

I2C_IDLE_CON ทำงานต่อเนื่องเมื่อเข้าสู่โหมดไอเดิล

การควบคุมขา SCL :

I2C_CLK_REL ปล่อยขา SCL ออกจากบัส I^2C

I2C_CLK_HLD คงสถานะขา SCL ไว้

เปิด/ปิดการใช้งานโหมดติดต่อกับอุปกรณ์พิเศษ IPMI(Intelligent Peripheral Management Interface)

I2C_IPMI_EN เปิดการใช้งานโหมด IPMI

I2C_IPMI_DIS ปิดการใช้งานโหมด IPMI

ขนาดแอดเดรสของอุปกรณ์สเลฟ :

I2C_10BIT_ADD แอดเดรสของอุปกรณ์สเลฟขนาด 10 บิต

I2C_7BIT_ADD แอดเดรสของอุปกรณ์สเลฟขนาด 7 บิต

เปิด/ปิดการควบคุมสลูว์เรต(Slew Rate):

I2C_SLW_EN เปิดการใช้งานสลูร์เรต

I2C_SLW_DIS ปิดการใช้งานสลูร์เรต

*สลูว์เรต ถูกใช้งานเมื่อระบบบัส I^2C ทำงานที่ความเร็ว 400 kHz

ระดับสัญญาณอินพุตของบัส SMBus :

I2C_SM_EN ใช้ระดับสัญญาณอินพุตของบัส SMBus

I2C_SM_DIS ไม่ใช้ระดับสัญญาณอินพุตของบัส SMBus

เปิด/ปิดการใช้งานอินเตอร์รัปต์รับค่าแอดเดรส(ทำงานในโหมดสเลฟ) :

I2C_GCALL_EN	เปิดการใช้งานอินเตอร์รัปต์เมื่อมีการรับค่าแอดเดรสที่ รีจิสเตอร์ I2CRSR
I2C_GCALL_DIS	ปิดการใช้งานอินเตอร์รัปต์

การควบคุมขาสัญญาณนาฬิกา SCL (ทำงานในโหมดสเลฟ):

I2C_STR_EN	เปิดการใช้งานด้วยซอฟต์แวร์หรือรับสัญญาณนาฬิกา
I2C_STR_DIS	ปิดการใช้งาน

บิตข้อมูลรับรู้สัญญาณการตอบรับ(ทำงานในโหมดมาสเตอร์):

I2C_ACK	ส่งข้อมูล ACK เพื่อตอบรับสัญญาณรับรู้จากอุปกรณ์สเลฟ
I2C_NACK	ส่งข้อมูล NACK เพื่อตอบรับสัญญาณรับรู้จากอุปกรณ์สเลฟ

บิตควบคุมระดับสัญญาณการตอบรับ(ทำงานในโหมดมาสเตอร์):

I2C_ACK_EN	เปิดการตอบรับสัญญาณ
I2C_ACK_DIS	ปิดการตอบรับสัญญาณ

บิตรับข้อมูล(ทำงานในโหมดมาสเตอร์):

I2C_RCV_EN	เปิดการรับข้อมูล I^2C
I2C_RCV_DIS	ปิดการรับข้อมูล I^2C

บอตหยุดการรับส่งข้อมูล(ทำงานในโหมดมาสเตอร์):

I2C_STOP_EN	เปิดการใช้งานบิตหยุดข้อมูล
I2C_STOP_DIS	ปิดการใช้งานบิตหยุดข้อมูล

บิตสร้างสัญญาณเริ่มต้นใหม่(ทำงานในโหมดมาสเตอร์):

I2C_RESTART_EN	เปิดการใช้งานบิตสร้างสัญญาณเริ่มต้นใหม่
I2C_RESTART_DIS	ปิดการใช้งาน

บิตสร้างสัญญาณเริ่มต้น(ทำงานในโหมดมาสเตอร์):

I2C_START_EN	เปิดการใช้งานบิตสร้างสัญญาณเริ่มต้น
I2C_START_DIS	ปิดการใช้งาน

Config กำหนดค่าอัตราเร็วในการรับส่งข้อมูล(baud rate)

ค่าที่ส่งกลับ ไม่มี

ตัวอย่างเช่น OpenI2C(I2C_on & I2C_IDLE_CON & I2C_CLK_HLD & I2C_IPMI_DIS & I2C_7BIT_ADD & I2C_SLW_DIS & I2C_SM_DIS & I2C_GCALL_DIS &

I2C_STR_DIS & I2C_ACK & I2C_ACK_DIS & I2C_RCV_DIS & I2C_STOP_DIS & I2C_RESTART_DIS & I2C_START_DIS,40)

ฟังก์ชัน void StartI2C(void);

สร้างสัญญาณการเริ่มต้นติดต่อกับบัส I^2C

รูปแบบการใช้งาน: StartI2C();

อาร์กิวเมนต์: ไม่มี

ค่าที่ส่งกลับ ไม่มี

ตัวอย่างเช่น StartI2C();

ฟังก์ชัน void StopI2C(void);

สร้างสัญญาณหยุดการติดต่อกับบัส I^2C

รูปแบบการใช้งาน: StopI2C();

อาร์กิวเมนต์: ไม่มี

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่างเช่น StopI2C();

ฟังก์ชัน void RestartI2C(void);

สร้างสัญญาณเริ่มต้นการติดต่อใหม่กับบัส I^2C

รูปแบบการใช้งาน: RestartI2C();

อาร์กิวเมนต์ ไม่มี

ค่าที่ส่งกลับ ไม่มี

ตัวอย่างเช่น RestartI2C();

ฟังก์ชัน void IdleI2C(void);

สร้างสัญญาณรอจนบัส I^2C ว่าง

รูปแบบการใช้งาน: IdleI2C();

อาร์กิวเมนต์: ไม่มี

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่างเช่น IdleI2C();

ฟังก์ชัน char DataRdyI2C(void);

ส่งค่าสถานะบิต RBF ในรีจิสเตอร์ I2CSTAT กลับมา

รูปแบบการใช้งาน: DataRdyI2C();
อาร์กิวเมนต์: ไม่มี
ค่าที่ส่งกลับ: เท่ากับ 1 : มีข้อมูลในรีจิสเตอร์ I2CRCV
เท่ากับ 0 : ไม่มีข้อมูลในรีจิสเตอร์ I2CRCV
ตัวอย่างเช่น if(DataRdyI2C());

ฟังก์ชัน void AckI2C(void);

ตอบรับสัญญาณระบบบัส I^2C โดยเคลียร์บิต ACKDT และ เซตบิต ACKEN ในรีจิสเตอร์ I2CCON

รูปแบบการใช้งาน: AckI2C();
อาร์กิวเมนต์: ไม่มี
ค่าที่ส่งกลับ: ไม่มี
ตัวอย่างเช่น NotAckI2C();

ฟังก์ชัน unsigned int MastergetsI2C(unsigned int length,unsigned char* rdptr,unsigned int i2c_datr_wait);

ฟังก์ชันอ่านข้อมูลสตริงจากบัส I^2C ในโหมดมาสเตอร์

รูปแบบการใช้งาน: MastergetsI2C(length,*rdptr,i2c_data_wait); อาร์กิวเมนต์:
length จำนวนไบต์ที่ต้องการอ่านจากบัส I^2C
*rdptr ตัวแปรพอยต์เตอร์ที่ชี้ไปยังข้อมูลที่จะอ่าน
I2c_data_wait ระยะเวลารอข้อมูลที่จะอ่าน
ค่าที่ส่งกลับ: จำนวนไบต์ข้อมูลที่จะอ่านได้
ตัวอย่างเช่น

```
unsigned char string[10];
unsigned char *rdptr;
unsigned int length,i2c_data_wait;
length = 9
rdptr = string;
i2c_data_wait = 152;
MastergetsI2C(length,rdptr,i2c_data_wait);
```

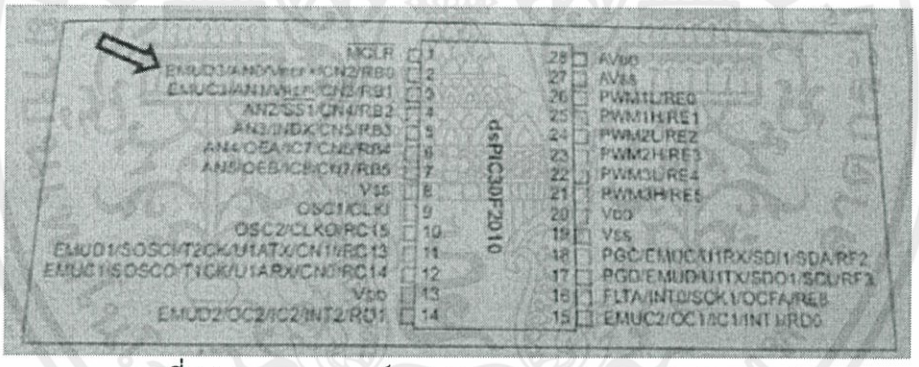
2.11 พอร์ตอินพุตเอาต์พุตดิจิตอล

ขาพอร์ตไมโครคอนโทรลเลอร์ dsPIC ทุกขา(ยกเว้น VDD,Vss,MCLR, และ OSC1/CLK1) สามารถทำงานได้ 2 รูปแบบคือ

1. ขาพอร์ตโมดูลฟังก์ชันสำหรับการติดต่ออุปกรณ์ภายนอก(Peripherals) ทำหน้าที่เป็นขาพอร์ตอินพุตเอาต์พุตให้กับโมดูลฟังก์ชันที่เปิดใช้งาน เมื่อขาพอร์ตถูกโมดูลฟังก์ชันใช้งานแล้วจะไม่สามารถเป็นขาพอร์ต I/O ปกติได้ เช่น โมดูล SPI ,I^2C ,DCI ,UART ,CAN เป็นต้น
2. ขาพอร์ตสำหรับอินพุตเอาต์พุต(I/O Ports) ทำงานที่เป็นขาพอร์ตสำหรับรับและส่งสัญญาณ รวมถึงควบคุมการทำงานของอุปกรณ์ทั่วไปเป็นขาพอร์ตอินพุตเอาต์พุตดิจิตอล ซึ่งเป็นรูปแบบการทำงานพื้นฐานที่สุด เนื่องจากการควบคุมการใช้งานขาพอร์ตเพียงแค่ 2 สถานะ คือ 1.ระดับสัญญาณลิจิก"1"หรือ High และ 2. ลอจิก"0"หรือ Low

ตัวอย่างดังรูป ตำแหน่งขาพอร์ตที่ 2 EMUD3/AN0/Vref+/CN2/RB0 สามารถกำหนดให้ทำงานได้ถึง 5 หน้าที่ด้วยกันคือ

1. EMUD3 ทำหน้าที่เป็นขา ICD4 สำหรับการรับส่งข้อมูล
2. AN0 ทำหน้าที่เป็นขาอะนาลอกช่องที่ 0
3. Vref+ ทำหน้าที่เป็นขารับแรงดันอ้างอิงด้านบวก
4. CN2 ทำหน้าที่เป็นอินเตอร์รัปต์ CN ช่องที่ 2
5. RB0 ทำหน้าที่เป็นขาพอร์ตดิจิตอล



รูปที่2.12 แสดงขาพอร์ต dsPIC30F2011 แบบ SDIP และ SOIC

ด้วยเหตุนี้ขาพอร์ตของ dsPIC จึงสามารถทำงานได้มากกว่าหนึ่งหน้าที่ในขาเดียว โดยการกำหนดการใช้งานพอร์ตผ่านทางรีจิสเตอร์ที่ควบคุมการกำหนดหน้าที่พอร์ต ใน dsPIC30F2010 มีขาพอร์ตให้ใช้งาน 5 พอร์ต รวม 20 ขา ประกอบไปด้วย

- พอร์ต B จำนวนขา 6 ขา RB0-RB5
- พอร์ต C จำนวนขา 3 ขา RC1-RC15
- พอร์ต D จำนวนขา 2 ขา RD0-RD1
- พอร์ต E จำนวนขา 7 ขา RE0-RE5 และ RE8
- พอร์ต F จำนวน 2 ขา RF2-RF3

2.11.1 รีจิสเตอร์ควบคุมการทำงานพอร์ต I/O

การควบคุมการทำงานของขาพอร์ตจะทำผ่านทางรีจิสเตอร์ 3 ตัวด้วยกันคือรีจิสเตอร์

- TRISx(Data Direction register)
- PORTx(I/O Port register)
- LATx(I/O Latch register)

โดยรายละเอียดของรีจิสเตอร์เหล่านี้ได้มีการกำหนดไว้แล้วในเฮดเดอร์ไฟล์ p30f2010.h

รีจิสเตอร์ TRIS

รีจิสเตอร์ TRISx ใช้ในการกำหนดทิศทางการทำงานของขาพอร์ตให้เป็นอินพุตหรือเอาต์พุต พอร์ต หากกำหนดค่า '1' ในบิตใดของรีจิสเตอร์ TRISx จะทำให้บิตที่ตำแหน่งเดียวกันของรีจิสเตอร์ PORTx และ LATx เป็นอินพุต และหากกำหนดค่า '0' จะเป็นเอาต์พุต รีจิสเตอร์ TRISx เช่น TRISA, TRISB, TRISC, TRISD เป็นต้น ตัวอย่างการกำหนดค่ารีจิสเตอร์ TRIS ดังตารางดังนี้

SFR Name	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TRISB	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
ค่าที่กำหนดให้กับTRISB	1	1	0	0	1	1
PORTB	RB5	RB4	RB3	RB2	RB1	RB0
LATB	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0
สถานะของพอร์ต	Input	Input	Output	Output	Input	Input

ตารางที่ 1 การกำหนดค่าให้กับรีจิสเตอร์ TRIsB

รีจิสเตอร์ PORTx ใช้ในการอ่านเขียนข้อมูลกับขาพอร์ต การอ่านข้อมูลด้วยรีจิสเตอร์ PORTx จะเป็นการอ่านข้อมูลจากขาพอร์ตโดยตรง (I/O pin) และหากเป็นการเขียนข้อมูลด้วยรีจิสเตอร์ PORTx จะเป็นการเขียนข้อมูลไปที่แลตช์พอร์ตข้อมูล(port data latch) รีจิสเตอร์ PORTx เช่น PORTA, PORTB, PORTC , PORTD ในบางชุดคำสั่งของ dsPIC เช่นคำสั่ง BSET และ BCLR เมื่อนำมาใช้กับรีจิสเตอร์ PORT ในโหมดการอ่านเขียนข้อมูลต่อเนื่อง อาจทำให้เกิดความผิดพลาดได้ เพราะการนำรีจิสเตอร์ PORT มาใช้ในกระบวนการ อ่าน-แก้ไข-เขียน (read-modify-write)นั้น ขาพอร์จะถูกกำหนดเป็นอินพุตและเอาต์พุตในเวลาที่กระชั้นชิดเกินไป ส่งผลให้เกิดความผิดพลาดของข้อมูลได้

รีจิสเตอร์ LATx ถูกนำมาใช้ในการแก้ไขปัญหาของรีจิสเตอร์ PORTx ในกระบวนการ อ่าน-แก้ไข-เขียนข้อมูลแบบต่อเนื่องกระชั้นชิด โดยการอ่านข้อมูลด้วยรีจิสเตอร์ LATx จะเป็นการอ่าน

ข้อมูลที่ค้างอยู่ในแลตช์พอร์ตเอาต์พุต (port output latches) และการเขียนข้อมูลจะเป็นไปในรูปแบบเดียวกับรีจิสเตอร์ PORTx คือ เขียนจ้อมูลไปที่แลตช์พอร์ตข้อมูล (port data latch) ด้วยวิธีการนี้ทำให้ไม่เกิดความผิดพลาดในการอ่านเขียนข้อมูล

สรุปความแตกต่างระหว่างรีจิสเตอร์ PORT และ LAT มีรายละเอียดดังนี้

- การเขียนข้อมูลไปที่รีจิสเตอร์ PORTx จะเขียนข้อมูลไปที่แลตช์พอร์ต
- การเขียนข้อมูลไปที่รีจิสเตอร์ LATx จะเขียนข้อมูลไปที่แลตช์พอร์ต
- การอ่านข้อมูลจากรีจิสเตอร์ PORTx จะเป็นการอ่านข้อมูลจาก I/O พอร์ตโดยตรง
- การอ่านข้อมูลจาดรีจิสเตอร์ LATx จะเป็นการอ่านข้อมูลที่ค้างอยู่ในแลตช์พอร์ต

2.11.2 การใช้งานพอร์ต

เนื่องจากพอร์ตของ dsPIC สามารถทำหน้าที่ได้มากกว่าหนึ่งหน้าที่ ก่อนการใช้งานจึงต้องตรวจสอบว่าสถานะเริ่มต้นเมื่อไมโครคอนโทรลเลอร์เกิดการรีเซตนั้น แต่ละขาพอร์ตถูกกำหนดให้ทำหน้าที่ใดบ้าง และต้งไม่ลืมที่จะกำหนดสถานะพอร์ตว่าจะใช้งานเป็นอินพุตหรือเอาต์พุตด้วย โดยกำหนดผ่านทางรีจิสเตอร์ TRIS ก่อนที่จะใช้งาน สำหรับการกำหนดหน้าที่ใช้งานเป็นพอร์ตอินพุตเอาต์พุตดิจิตอลแนะนำดังนี้

1. กำหนดขาพอร์ตอินพุตหรือเอาต์พุตด้วยขารีจิสเตอร์ TRIS
2. อ่านค่าข้อมูลจากพอร์ตหรือเป็นอินพุตพอร์ตให้ใช้รีจิสเตอร์ PORT
3. เขียนส่งค่าข้อมูลไปที่พอร์ตหรือเอาต์พุตพอร์ตให้ใช้รีจิสเตอร์ LAT หรือ PORT (แนะนำให้ใช้ LAT)
4. พอร์ตอะนาลอกเมื่อต้องการใช้งานเป็นพอร์ตอินพุตดิจิตอล ต้องเซตบิตในรีจิสเตอร์ ADPCFG(ADC Port Configuration Register) โดยการเซตค่า'1'ที่รีจิสเตอร์ เพื่อปิดอินพุตอะนาลอกควบคู่กับการกำหนดรีจิสเตอร์ TRIS (dsPIC30F2010 ขาพอร์ตอะนาลอกจะอยู่ที่พอร์ต B)
5. หากมีการเปิดใช้งานโมดูลที่เกี่ยวข้องกับขาพอร์ต ขาพอร์ตนั้นจะถูกควบคุมด้วยโมดูลและไม่สามารถที่จะนำมาใช้งานเป็นอินพุตเอาต์พุตดิจิตอลได้

2.11.3 การใช้งานพอร์ตกับโมดูล LCD 16X4

ตัวอย่างต่อไปนี้ เป็นการใช้งานพอร์ตควบคุมโมดูล LCD เป็นแบบตัวอักษรขนาด 16 ตัวอักษร 2 บรรทัด(Text LCD 16X2) โดยทำงานร่วมกับคีย์สวิตช์เมตริกซ์ เมื่อกดคีย์สวิตช์เมตริกซ์แล้ว จะแสดงผลตัวเลขบนโมดูล LCD และเมื่อกดคีย์ ENT(#) จะนำตัวเลขดังกล่าวไปหน่วงเวลา (มีหน่วยเป็นมิลลิวินาที) การติดของ LED 1 ดวง หากคีย์ CLR จะเคลียร์ค่าที่กด การใช้งานโมดูล LCD 16X2 และการสแกนคีย์ในตัวอย่างนี้ ผู้เขียนได้สร้างเป็นไฟล์ไลบรารี ทั้งส่วนของโมดูล LCD

(LIB_LCD16X2.c)และ โมดูลการสแกน Keypad (LIB_Scankey.c) ให้สามารถนำมาใช้งานได้ทันที โดยอธิบายเฉพาะส่วนของขาพอร์ต สำหรับใช้ในการเชื่อมต่อกับโมดูลเท่านั้น ดังในโค้ดด้านล่างนี้

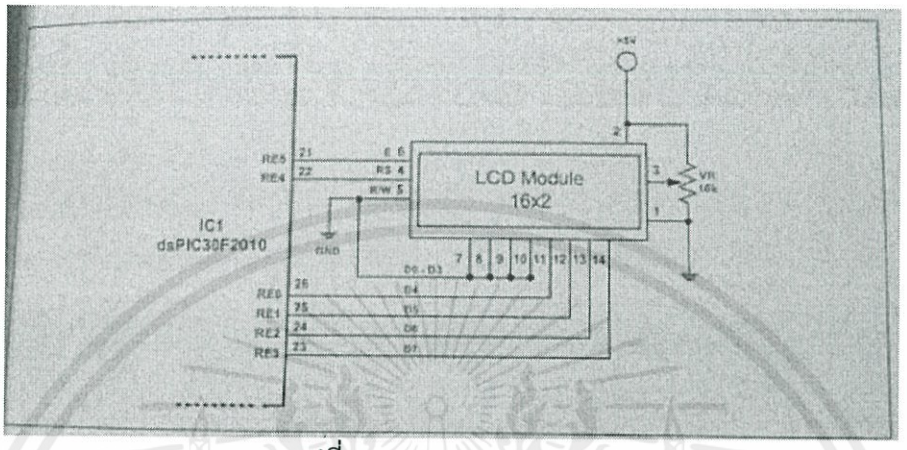
```
//-----------------------------------------------------:Includes
#include <p30fxxxx.h>          // generic header file for dsPIC


//-----------------------------------------------------:Defines
#define LCD_TRIS    TRISE      // Set direction control output
#define LCD_DATA    LATE       // Dataport of LCD-Display (D4..D7)
#define LCD_RS      _LATE4     // Register Select of LCD-Display
#define LCD_E       _LATE5     // Enable of LCD-Display
#define CTRL_REG    0          // Select instruction register
#define DATA_REG    1          // Select data register
#define BLINK       0x01       // Alias for blinking cursor
#define NOBLINK     0x00       // Alias for non blinking cursor
#define SHOW        0x02       // Alias for cursor on
#define HIDE        0x00       // Alias for cursor off
#define ON          0x04       // Alias for display on
#define OFF         0x00       // Alias for display off

// Table to select DD-RAM address (including instruction and address)
// 0x00..0x0F -> Line 1,
// 0x40..0x4F -> Line 2,
// 0x10..0x1F -> Line 3,
// 0x50..0x5F -> Line 4
static unsigned char LOCATION[4][16] = {
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,
0x8F},

{0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,
0xCF},

{0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,
0x9F},

{0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,
0xDF}};

//-----------------------------------------------------:Function Prototypes
static void delay(unsigned int ms);           // Timer dependend delay-
routine
static void LCD_Command(unsigned char cmd);    // Command LCD

// Clear display
void LCDClrscr(void);
// Controls the display
void LCDControl(unsigned char dsp,unsigned char blink,unsigned char cursor);
// Writes a character to display
void LCDPutchar(unsigned char value);
// Prints a text to x/y position
void LCDPrintxy(unsigned char x,unsigned char y, unsigned int dly, unsigned
char *text);
// Sets LCD write position
void LCDGotoxy(unsigned char x,unsigned char y);
// Initialize the LCD display
void LCDInit(void);

//-----------------------------------------------------:delay
// Function : delay_ms
// Parameters : ms - unsigned int
// Returned : nothing
static void delay(unsigned int ms)
{
    unsigned int i;

    for (; ms>0; ms--)
```

```
      //for (i=0; i<128; i++)
      for (i=0; i<728; i++)
        Nop();                    // delay 1 mch cycle
}

//-----------------------------------------------------:LCD command
static void LCD_Command(unsigned char cmd)
{
    delay(20);              // Wait 20ms
    LCD_E  = 0;
    LCD_RS = CTRL_REG;      // Switch to inruction register

    // Set LCD_DATA to high nibble of Display On/Off Control
    LCD_DATA = (LCD_DATA&0xF0)|((cmd&0xF0)>>4);
    LCD_E = 1; LCD_E = 0;   // Write data to display

    // Set LCD_DATA to lower nibble of Display On/Off Control
    LCD_DATA = (LCD_DATA&0xF0)|(cmd&0x0F);
    LCD_E = 1; LCD_E = 0;   // Write data to display

    delay(1);               // Wait 1ms

    return;
}

//-----------------------------------------------------:LCDControl
// Function    : LCDControl(dsp,blink,cursor)
// Input : unsigned char dsp    = ON     -> Display on
//                                OFF    -> Display off
//         unsigned char blink  = BLINK  -> Cursor blinks
//                                NOBLINK-> Cursor not blinks
//         unsigned char cursor = SHOW   -> Cursor visible
//                                HIDE   -> Cursor invisible
//
void LCDControl(unsigned char dsp,unsigned char blink,unsigned char cursor)
{
    unsigned char control;  // variable to generate instruction byte

    control = (0x08 + blink + cursor + dsp); // Cursor control
    LCD_Command(control);

    return;
}

//-----------------------------------------------------:LCDClrscr
void LCDClrscr(void)
{
  LCD_Command(0x01);    // Clear screen
}

//-----------------------------------------------------:LCDGotoxy
// Function    : LCDGotoxy(x,y)
// Description : Sets the write position of the LCD display
//
//                  (1,1)          (16,1)
//                    |              |
//                    ################   -> line 1
//                    ################   -> line 2
//                    ################   -> line 3
//                    ################   -> line 4
//                    |              |
//                  (1,4)          (16,4)
// Input       : unsigned char x    -> x position (horizontal)
//               unsigned char y    -> y position (vertical)
//
void LCDGotoxy(unsigned char x,unsigned char y)
{
    delay(20);              // Wait 20ms
```

```
    LCD_E  = 0;
    LCD_RS = CTRL_REG;       // Switch to inruction register

    // Set LCD_DATA to high nibble of position table value
    LCD_DATA = (LCD_DATA&0xF0)|(((LOCATION[y-1][x-1])&0xF0)>>4);
    LCD_E = 1; LCD_E = 0;        // Write data to display
    // Set LCD_DATA to lower nibble of position table value
    LCD_DATA = (LCD_DATA&0xF0)|((LOCATION[y-1][x-1])&0x0F);
    LCD_E = 1; LCD_E = 0;        // Write data to display

    delay(1);                    // Wait 1ms

    return;
}

//-------------------------------------------------------:LCDPutchar
// Function    : LCDPutchar(value)
// Description : Writes the character value to the display
//
void LCDPutchar(unsigned char value)
{
    LCD_RS = DATA_REG;              // Switch to data register

    // Set LCD_DATA to high nibble of value
    LCD_DATA = (LCD_DATA&0xF0)|((value&0xF0)>>4);
    LCD_E = 1; LCD_E = 0;           // Write data to display
    // Set LCD_DATA to lower nibble of value
    LCD_DATA = (LCD_DATA&0xF0)|(value&0x0F);
    LCD_E = 1; LCD_E = 0;           // Write data to display

    delay(1);                       // Wait 1ms

    return;
}

//-------------------------------------------------------:LCDPrintxy
// Function    : LCDPrintxy(x,y,*text)
// Description : Prints text to position x/y of the display
// Input       : unsigned char x     -> x position of the display
//               unsigned char y     -> y position of the display
//               unsigned char *text -> pointer to text
//                        unsigned int dly      -> delay time
//
void LCDPrintxy(unsigned char x,unsigned char y, unsigned int dly, unsigned
char *text)
{

    LCDGotoxy(x,y);          // Set cursor position

    while( *text )           // while not end of text
  {
    LCDPutchar(*text++);  // Write character and increment position
        delay(dly);           // time delay print text
  }

    return;
}

//-------------------------------------------------------:LCDInit
// Initialize the LCD display
void LCDInit(void)
{
    LCD_TRIS = 0x0000;       // set direction control RB output
    LCD_Command(0x33);       // set 4 bit mode
    LCD_Command(0x32);       // set 4 bit mode
    LCD_Command(0x28);       // 2 line 5x7 dot
  //  LCD_Command(0x2C);       // 4 line 5x7 dot
    LCD_Command(0x01);       // clear screen lcd
```

```
    LCDControl(ON,NOBLINK,HIDE);
}
```

จากโค้ดด้านบนเป็นการใช้งานพอร์ตโมดูล LCD จะใช้ PORTE จำนวน 6 บิตในการควบคุม LCD แบบ 4 บิตโหมด โดยขา R/W ของLCD จะต่อลงกราวด์เนื่องจากเราใช้โมดูล LCD เฉพาะแสดงผลเท่านั้น (ไม่อ่านข้อมูลจาก LCD)

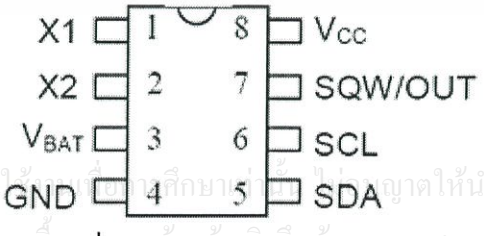


รูปที่ 2.13 แสดงวงจรการทดลอง

2.12 ระบบฐานเวลา

ระบบฐานเวลาเป็นสิ่งสำคัญที่สามารถนำไปใช้ในอุปกรณ์อีเล็กโทรนิกส์ได้หลากหลาย ภายในไมโครคอนโทรเลอร์เองก็มีไทเมอร์เพื่อใช้ในการจับเวลา หรือนำไปใช้เป็นฐานเวลาจริงได้เช่นกัน แต่เนื่องจากไมโครคอนโทรเลอร์สามารถทำงานได้ต่อเมื่อมีไฟเลี้ยงเท่านั้น ดังนั้นการใช้ไทเมอร์ของไมโครคอนโทรลเลอร์ สร้างฐานเวลาจริงจึงไม่ เหมาะสมในบางแอพพลิเคชัน

DS1307 เป็น IC ฐานเวลาของดัลลัสเซมิคอนดักเตอร์ (Dallas Semiconductor) มีบัสรับส่งข้อมูลแบบ I2C ซึ่งเป็นแบบ 2 wire สามารถสื่อสารได้ 2 ทิศทาง (bi-direction bus) ฐานเวลาของ DS1307 นั้นสามารถเก็บข้อมูล วินาที, นาที, ชั่วโมง, วัน, วันที่, เดือน และปี ได้ ระบบเวลาสามารถทำงานโหมดรูปแบบ 24 ชั่วโมง หรือ 12 ชั่วโมง AM/PM ก็ได้ ภายมีระบบตรวจจับแหล่งจ่ายไฟ โดยถ้าแหล่งจ่ายไฟหลักถูกตัดไป DS1307 สามารถสวิตซ์ไปใช้ไฟจากแบตเตอรี่ และทำงานต่อไป โดยที่ยังสามารถรักษาข้อมูลไว้ได้ โครงสร้างมีขาทั้งหมด 8 ขาดังแสดงในรูปที่2.14 และมีรายละเอียดการทำงานของขาต่าง ๆ ดังนี้

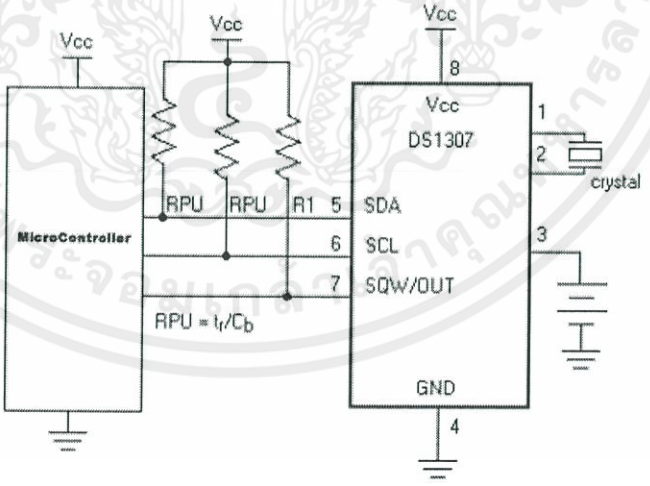


รูปที่ 2.14 แสดงรูปแบบ DS1307

- VCC : ใช้ต่อไฟเลี้ยง +5V
- GND : ใช้ต่อกราวด์
- VBAT: ใช้ต่อกับแบตเตอรี่ 3V เพื่อรักษาการทำงาน ในกรณีที่ไม่มีไฟเลี้ยงจ่าย
- SDA: ขารับส่งข้อมูลด้วยระบบบัส I2C
- SCL: ขาสัญญาณนาฬิกาสำหรับการรับส่งข้อมูลด้วยระบบบัส I2C
- SQW/OUT: ขาเอาต์พุตสัญญาณ Square Wave สามารถเลือกความถี่ได้
- X1,X2:ใช้ต่อกับคริสตอลความถี่มาตรฐาน32.768 kHz เพื่อสร้างฐานเวลาจริงให้กับ IC

ระบบบัสข้อมูลแบบ I2C (Inter-IC Communication) ได้ถูกพัฒนาขึ้นโดยบริษัทฟิลิปส์ (Phillips) การรับส่งข้อมูลใช้สายสัญญาณเพียงแค่ 2 เส้น คือสายสัญญาณข้อมูล SDA (Serial Data line) และสายสัญญาณนาฬิกา SCL (Serial Clock line) มีการทำงานเป็นแบบ Master, Slave โดยอุปกรณ์ที่ทำหน้าที่เป็น Master (ไมโครคอนโทรลเลอร์) จะควบคุมการรับส่งข้อมูล และควบคุมสัญญาณนาฬิกาบน SCL ส่วนอุปกรณ์ Slave (DS1307) นั้นจะทำงานภายใต้การควบคุมของอุปกรณ์ Master

การต่อใช้งานร่วมกับไมโครคอนโทรลเลอร์ด้วยระบบบัส I2C นั้นสามารถทำได้โดยต่อตัวต้านทาน Pull up ดังแสดงในรูปที่ 2.15 ในกรณีที่ต้องการต่อร่วมกับอุปกรณ์ Slave หลายตัว ก็สามารถทำได้โดยต่ออุปกรณ์ Slaveขนานกันไป การติดต่อสื่อสารระหว่างอุปกรณ์ Master กับ Slave แต่ละตัวนั้น จะถูกแยกโดย Address ของอุปกรณ์ Slave ซึ่งจะถูกส่งจากอุปกรณ์ Master ไปยังอุปกรณ์ Slave ก่อนเริ่มการรับส่งข้อมูล



รูปที่ 2.15 แสดงการเชื่อมต่อ DS1307 เข้ากับไมโครคอนโทรลเลอร์ด้วยระบบบัสแบบ I2C

การรับส่งข้อมูลแบบ I2C นั้นมีข้อกำหนดอยู่ 2 ประการด้วยกันคือ

1. การรับส่งข้อมูลจะเริ่มขึ้นได้เมื่อบัสมีสถานะว่างเท่านั้น

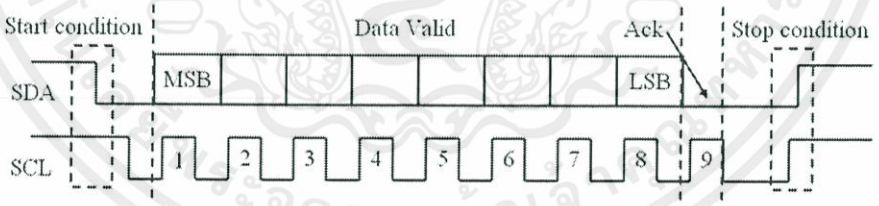
2. ในช่วงที่ทำการรับส่งข้อมูลอยู่ สายสัญญาณ SDA ต้องไม่เปลี่ยนสถานะในช่วงที่ SCL มีสถานะเป็นลอจิก “1” ถ้า SDA มีการเปลี่ยนสถานะในช่วงที่ SCL เป็นลอจิก “1” จะถือว่าเป็นสัญญาณควบคุมการรับส่งข้อมูล

สถานะของการรับส่งข้อมูลแบบ I2C สามารถแบ่งออกได้เป็น 5 สถานะด้วยกันดังแสดงในรูป และมีรายละเอียดดังนี้

1. สถานะว่าง (Bus not busy): สัญญาณ SDA และ SCL มีระดับสัญญาณเป็น High
2. เริ่มส่งข้อมูล (Start data transfer): มีการเปลี่ยนระดับสัญญาณของ SDA จาก High เป็น Low ในขณะที่ SCL มีระดับสัญญาณเป็น High ค้างไว้
3. หยุดส่งข้อมูล (Stop data transfer): มีการเปลี่ยนระดับสัญญาณของ SDA จาก Low เป็น High ในขณะที่ SCL มีระดับสัญญาณเป็น High ค้างไว้
4. รับส่งข้อมูล (Data valid): มีการรับส่งข้อมูลผ่านสายสัญญาณ SDA โดยข้อมูลแต่ละบิตจะถูกส่งในช่วงที่ SCL มีระดับเป็น High โดยในช่วงที่ SCL มีสถานะเป็น High อยู่นั้น SDA จะต้องไม่เกิดการเปลี่ยนระดับสัญญาณ

SDA จะเปลี่ยนระดับของสัญญาณ ในช่วงที่ SCL มีระดับสัญญาณเป็น Low เท่านั้น ตามมาตรฐานการส่งข้อมูล แบบ I2C นี้สามารถส่งข้อมูลด้วยความถี่สัญญาณนาฬิกาสูงสุด 100 kHz ที่โหมดการทำงานธรรมดา และ 400 kHz ที่โหมดการทำงานแบบเร็ว แต่สำหรับ DS1307 สามารถทำงานได้ในโหมดธรรมดาเท่านั้น

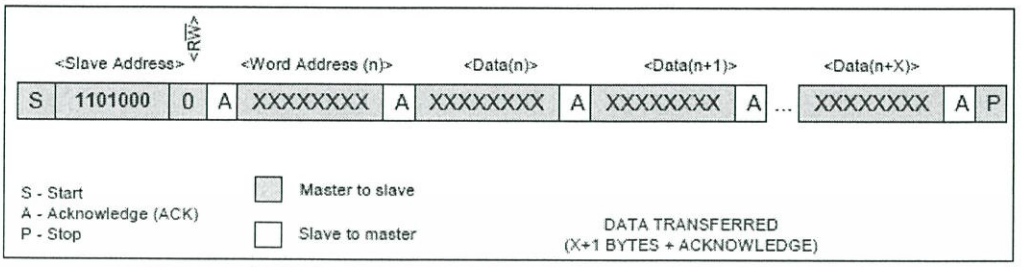
ตอบรับ (Acknowledge): เกิดขึ้นหลังจากที่มีการรับส่งข้อมูลครบแล้ว โดยอุปกรณ์ Master ต้องสร้างสัญญาณ Clock บน SCL เพิ่มอีกลูก อุปกรณ์ที่เป็นตัวรับข้อมูลจะดึงระดับสัญญาณบน SDA ให้เป็น Low เพื่อให้ตัวส่งรับรู้ว่าตัวรับได้รับข้อมูลครบแล้ว



รูปที่ 2.16 แสดงการการรับส่งข้อมูลผ่านบัส I2C

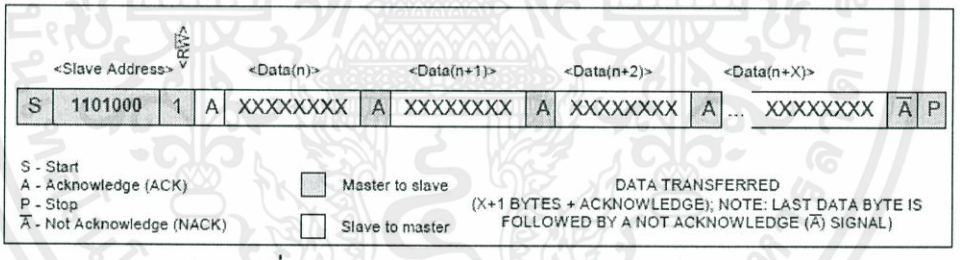
ในการรับส่งข้อมูลผ่านบัส I2C อุปกรณ์ Master จะเป็นผู้สร้างสัญญาณ Clock บน SDA และเป็นตัวควบคุมสถานะ Start และ Stop เพื่อควบคุมการรับส่งข้อมูลทั้งหมด การส่งข้อมูลไปยังอุปกรณ์ DS1307 ดังแสดงในรูปที่ 7 ไมโครคอนโทรลเลอร์ต้องสร้างสภาวะ Start ก่อน จากนั้นต้องส่ง Address ของDS1307 ขนาด 7 บิตซึ่งมีค่าเป็น 1101000 และตามด้วยบิตระบุทิศทางของข้อมูล ในกรณีที่เป็นการเขียนข้อมูลลง DS1307 จะต้องเป็น “0” จากนั้นไมโครคอนโทรเลอร์จะต้องส่งตำแหน่ง Address ภายในรีจิสเตอร์ของ DS1307 ที่ต้องการเขียนข้อมูลลง แล้วจึงค่อยเขียนข้อมูลลง

โดยในการส่งข้อมูลแต่ละไบต์จะต้องรอบิต Ack จาก DS1307 ทุกไบต์ เมื่อส่งจนครบแล้ว ถึงจะสร้างสภาวะ Stop เพื่อกลับสู่สถานะว่าง



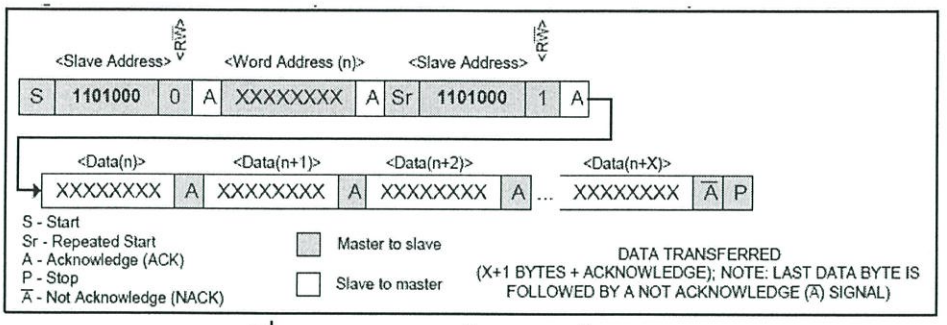
รูปที่ 2.17 การเขียนข้อมูลอุปกรณ์ Slave ผ่านบัส I2C

การรับข้อมูลจากอุปกรณ์ Slave ดังแสดงในรูป เริ่มแรกไมโครคอนโทรลเลอร์ต้องสร้างสภาวะ Start ก่อน จากนั้นต้องส่ง Address ของ Ds1307 ขนาด 7 บิตซึ่งมีค่าเป็น 1101000 และตามด้วยบิตระทิศทางของข้อมูล ในกรณีที่เป็นการอ่านข้อมูลจาก DS1307 จะต้องเป็น"1"จากนั้นจึงค่อยรับข้อมูลจากอุปกรณ์ Slave ทีละไบต์ โดยตำแหน่งที่อ่านเข้ามาจะขึ้นอยู่กับตำแหน่งรีจิสเตอร์พอยท์เตอร์ ซึ่งเป็นตำแหน่งท้ายสุดที่ได้ทำการเขียนข้อมูลไว้ เมื่ออ่านข้อมูลครบแต่ละไบต์อุปกรณ์มาสเตอร์ ต้องส่ง Acknowledge บิตกลับไปให้อุปกรณ์ Slave ด้วย ในกรณีที่เป็นไบต์สุดท้ายอุปกรณ์ Master ต้องส่ง "not acknowledge"กลับไป



รูปที่ 2.18 แสดงการทำงานภายใน DS1307

ภายใน DS1307 มีรีจิสเตอร์ภายในใช้เก็บข้อมูลเวลาขนาด 7 ไบต์ 00H-06H ดังแสดงไว้ในรูป ข้อมูลค่าเวลา และวันที่จะถูกเก็บอยู่ในรูปของเลขฐาน 10 สามารถเลือกได้ว่าให้ทำงานแบบ 12 ชม. หรือ 24 ชม. โดยกำหนดที่บิตที่ 6 ที่แอดเดรส 02H โดยถ้าเป็น"1" จะเป็นการทำงานในโหมด 12 ชม. และเมื่อเลือกแบบ 12 ชม. ที่บิต 5 ในแอดเดรส 02H นั้นจะใช้แสดงค่า AM/PM โดยถ้าบิตนี้เป็น "1" จะเป็น PM ในกรณีที่แสดงแบบ 24 ชม. บิตนี้จะใช้ในการแสดงค่าของหลักสิบในหน่วยของชั่วโมงด้วย



รูปที่ 2.19 แสดงการทำงานภายใน DS1307

	BIT7							BIT0	
00H	CH	10 SECONDS			SECONDS				00-59
	0	10 MINUTES			MINUTES				00-59
	0	12/24	10 HR A/P	10 HR	HOURS				01-12 00-23
	0	0	0	0	0	DAY			1-7
	0	0	10 DATE		DATE				
	0	0	0	10 MONTH	MONTH				01-12
	10 YEAR				YEAR				00-99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	

รูปที่ 2.20 รีจิสเตอร์ภายในไอซีฐานเวลา DS1307

ที่แอดเดรส 07H เป็นรีจิสเตอร์ควบคุมการทำงานของ SQW/OUT โดยมีรายละเอียดดังนี้

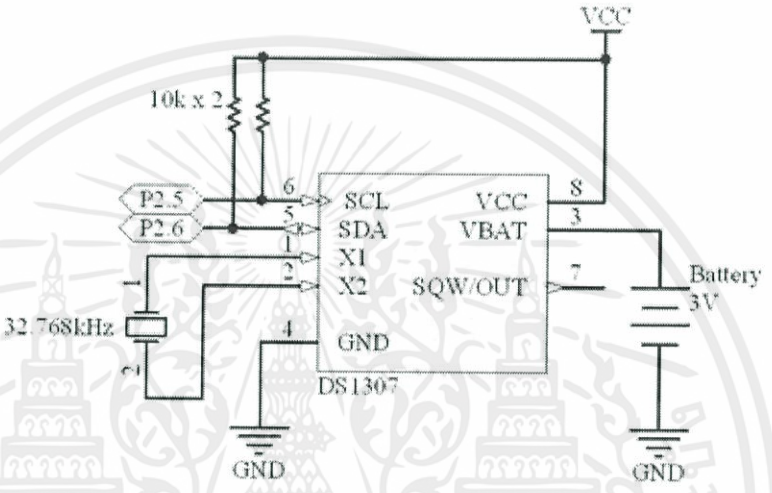
- OUT (Out control) : ใช้ควบคุมเอาต์พุต
- SQWE (Square Wave Enable) : ใช้ควบคุมออสซิลเลเตอร์ภายใน Ds1307 โดยถ้าบิตนี้เป็น “1” จะเป็นการเปิดออสซิลเลอเตอร์
- RS (Rate Select) : ใช้ควบคุมความถี่ของ Square Wave เมื่อเปิดการทำงานของออสซิลเลเตอร์ โดยสามารถปรับความถี่ได้ 4 ความถี่ด้วยกันดังแสดงในตารางที่ 1

RS1	RS0	SQW OUTPUT FREQUENCY
0	0	1 Hz
0	1	4.096 kHz
1	0	8.192 kHz
1	1	32.768 kHz

ตารางที่ 2 การควบคุมความถี่ออสซิลเลเตอร์ด้วยการเซตบิต RS1, RS0

ในการทดลองได้ต่อ DS1307 กับไมโครคอนโทรลเลอร์ P89V51RD2 โดยใช้พอร์ต P2.5 และ P2.6 ของไมโครคอนโทรเลอร์เป็นบัส I2C ต่อกับ SCL และ SDA ของ DS1307 ดังแสดงในรูปที่ 9 ส่วนขาสัญญาณ SQW/OUT นั้นไม่ได้ใช้สร้างสัญญาณให้ไมโครคอนโทรลเลอร์ แต่ใช้การวนลูปคอยตรวจสอบค่าภายในรีจิสเตอร์ของ DS1307 แทน

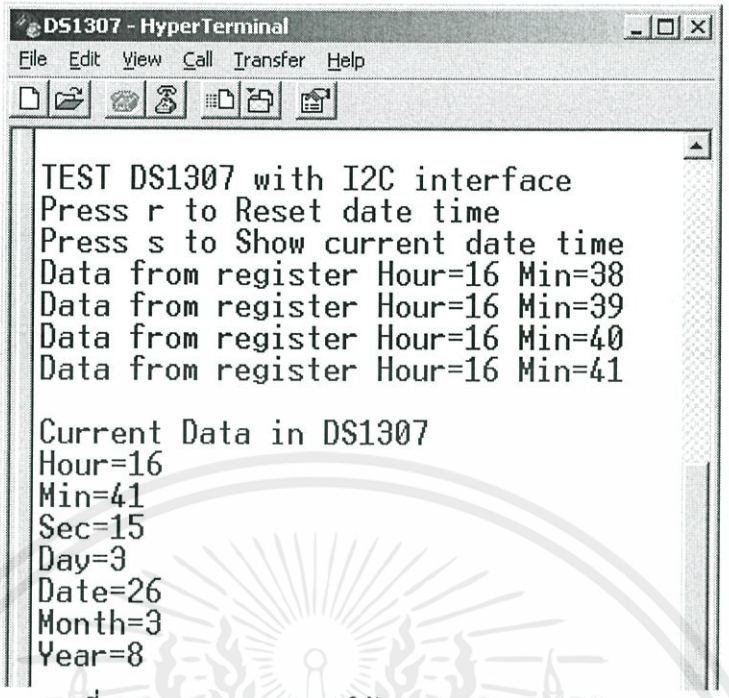
ในการควบคุมการทำงานของโปรแกรม และแสดงผล ได้ใช้โปรแกรม HyperTerminal เป็นโปรแกรมติดต่อผ่านพอร์ตอนุกรมด้วยอัตราข้อมูล 9600 bps ดังแสดงในรูปที่ 9



รูปที่ 2.21 วงจรใช้งาน DS1307 ที่ใช้ในการทดลอง

ในส่วนโปรแกรมที่ใช้ในการทดสอบ ได้เขียนให้ไมโครคอนโทรลเลอร์คอยตรวจสอบพอร์ตอนุกรม และรีจิสเตอร์ภายใน DS1307 ในกรณีที่มีข้อมูล “r” เข้ามาทางพอร์ตอนุกรม ไมโครคอนโทรลเลอร์จะเขียนข้อมูลเวลา, วันที่, เดือน และปี ที่เก็บอยู่ใน Flash memory ลงในรีจิสเตอร์ของ DS1307 รวมทั้งตั้งให้ DS1307 ทำงานในโหมด 24 ชั่วโมง ส่วนในกรณีที่มีข้อมูล “s” เข้ามาทางพอร์ตอนุกรม ไมโครคอนโทรลเลอร์จะแสดงข้อมูลเวลา, วันที่, เดือน และปี ออกมาทางพอร์ตอนุกรมดังแสดงในรูปที่ 10

ไมโครคอนโทรลเลอร์จะคอยวนตรวจสอบรีจิสเตอร์ภายใน DS1307 ที่ตำแหน่ง 01H ซึ่งใช้เก็บค่าเวลาหน่วยนาที เมื่อค่าภายในรีจิสเตอร์นี้เปลี่ยนไป ไมโครคอนโทรลเลอร์จะอ่านข้อมูลเวลาชั่วโมง และนาที ภายในรีจิสเตอร์ของ DS1307 ขณะนั้นออกมา และส่งข้อมูลนั้นออกมาแสดงผลทางพอร์ตอนุกรมดังแสดงในรูปที่ 10



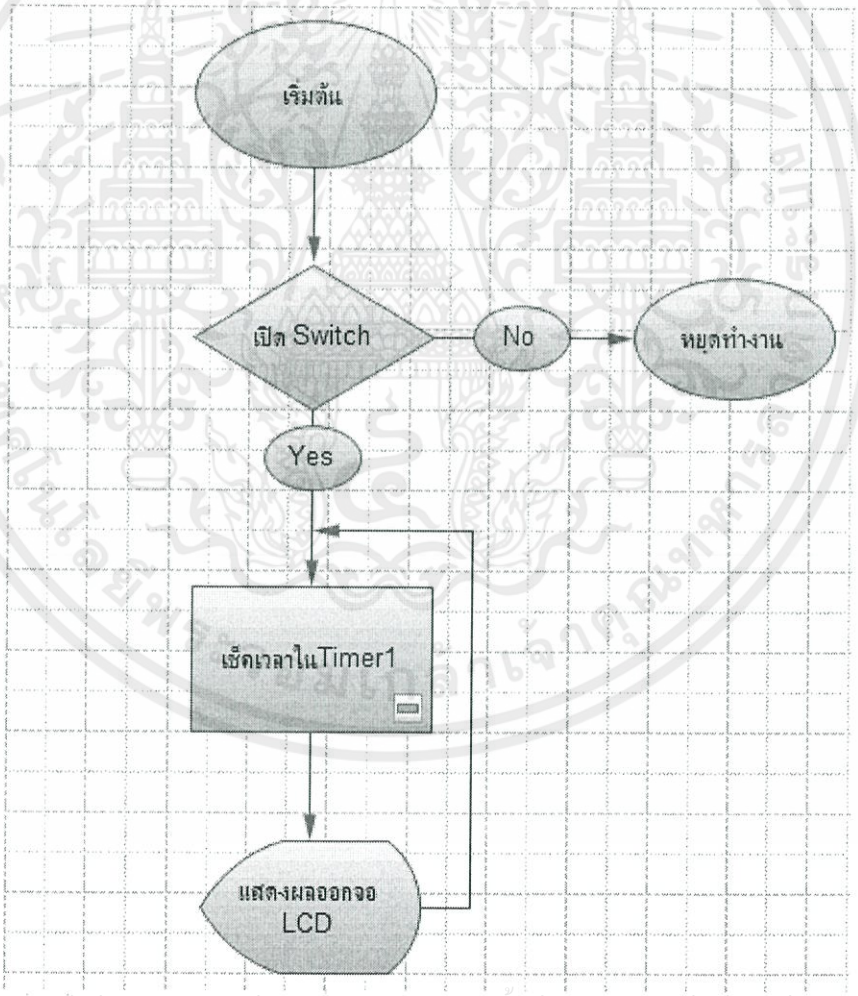
รูปที่ 2.22 การทดสอบการใช้งาน DS1307 ผ่านบัส I2C

บทที่ 3
ขั้นตอนการดำเนินงาน

ในการดำเนินงานสร้างอุปกณ์อิเล็กทรอนิกส์โดยทั่วไปนั้น จะต้องค่อยๆดำเนินการไปที่ละส่วน เพื่อไม่ให้เกิดความสับสนในตัวโปรแกรมโดยในการออกแบบตัวควบคุม MP3 นี้จะมีลำดับขั้นตอนในการออกแบบคือ

3.1 การแสดงผลโปรแกรมนาฬิกาออกจอ LCD โดยใช้ Timer1

ขั้นแรกคือการทดลอง การแสดงผลทางจอLCD โดยใช้ Timer1 มาเป็นตัวนับเวลาโดยเขียนโปรแกรมขึ้นมาให้เป็นโปรแกรมนาฬิกาโดยมีFlow Chart



โดยมีโปรแกรมที่เขียนขึ้นมาทดสอบดังนี้

1. #include <p30fxxxx.h>

```
2.  #include <timer.h>
3.  #include <stdio.h>
4.  _FOSC(CSW_FSCM_OFF & XT_PLL4);
5.  _FWDT(WDT_OFF);
6.  #include "LIB_EX05_04_Time.C"
7.  #include "LIB_LCD16x4.C"
8.  #define TRUE 1
9.  int main(void)
10. {
11.   char buf[4];
12.
13.   Timer1_Init();
14.   LCDInit();
15.   LCDClrscr();
16.
17.   Timer_Set(10,58,59);
18.
19.   LCDPrintxy(1,1,0, "Timer1 Module..");
20.   LCDPrintxy(1,2,0, "TIME: ");
21.
22.   while (TRUE)
23.    {
24.     if (dsp)
25.     {
26.       if (time_t1.hr <= 9)
27.      {
28.          sprintf(buf,"0%d:",time_t1.hr);
29.       }
30.     else
31.      {
32.          sprintf(buf,"%d:",time_t1.hr);
33.       }
34.       LCDPrintxy(7,2,0, buf);
```

```
35.
36.       if (time_t1.min <= 9)
37.       {
38.         sprintf(buf,"0%d:",time_t1.min);
39.       }
40.     else
41.       {
42.           sprintf(buf,"%d:",time_t1.min);
43.        }
44.       LCDPrintxy(10,2,0, buf);
45.
46.       if (time_t1.sec <= 9)
47.       {
48.           sprintf(buf,"0%d",time_t1.sec);
49.       }
50.     else
51.       {
52.           sprintf(buf,"%d",time_t1.sec);
53.        }
54.       LCDPrintxy(13,2,0, buf);
55.
56.       dsp = 0;
57.     }
58.   }
59.
60.   return 0;
61.     }
```

รายละเอียดโค้ดโปรแกรม

1.บรรทัดที่ 3

ผนวกไฟล์ stdio.h เพื่อเรียกใช้งานฟังชั่นก์ sprint() ฟังชั่นพิมพ์ข้อมูลลงบัฟเฟอร์สตริง โดยทำหน้าที่เปลี่ยนชนิดข้อมูลตัวเลขสตริง

2.บรรทัดที่ 6,7

ไฟล์ไลบรารี LIB_EX05_04_Time ควบคุมการทำงานของไทเมอร์ 1 เพื่อใช้ในการสร้างนาฬิกา ไฟล์ไลบรารี LIB_LCD16x4ควบคุมการแสดงผลบนโมดูล LCD 16x4 แบบตัวอักษร

3.บรรทัดที่ 11

ตัวแปรอะเรย์ buf[4] ใช้ในการเก็บข้อมูลถูกเรียกใช้งานร่วมกับฟังก์ชั่น sprint()

4.บรรทัดที่ 13,14

ติดตั้งการใช้งานโมดูลไทเมอร์ 1 และติดตั้งการใช้งานโมดูลLCD 16x4

5.บรรทัดที่ 17

กำหนดค่าเริ่มต้นของนาฬิกา โดยกำหนดที่เวลา 10:58:59

6.บรรทัดที่ 24

ตัวแปร dsp เป็นตัวแปรที่ใช้ในการกำหนดรอบเวลาในการแสดงผลของโมดูล LCD เพื่อไม่ให้โมดูล LCD แสดงผลตลอดเวลา จะแสดงเฉพาะเมื่อเวลาเปลี่ยนเท่านั้น โดยตัวแปร dsp จะถูกเซตเป็น 1 ในฟังชั่นก์อิรเตอร์รัปต์ของไทเมอร์ 1 เมื่อแสดงเวลาเสร็จแล้วจะถูกเคลียร์ในบรรทัดที่ 78 เพื่อรอการเซตครั้งต่อไป

7.บรรทัดที่ 26,36,46

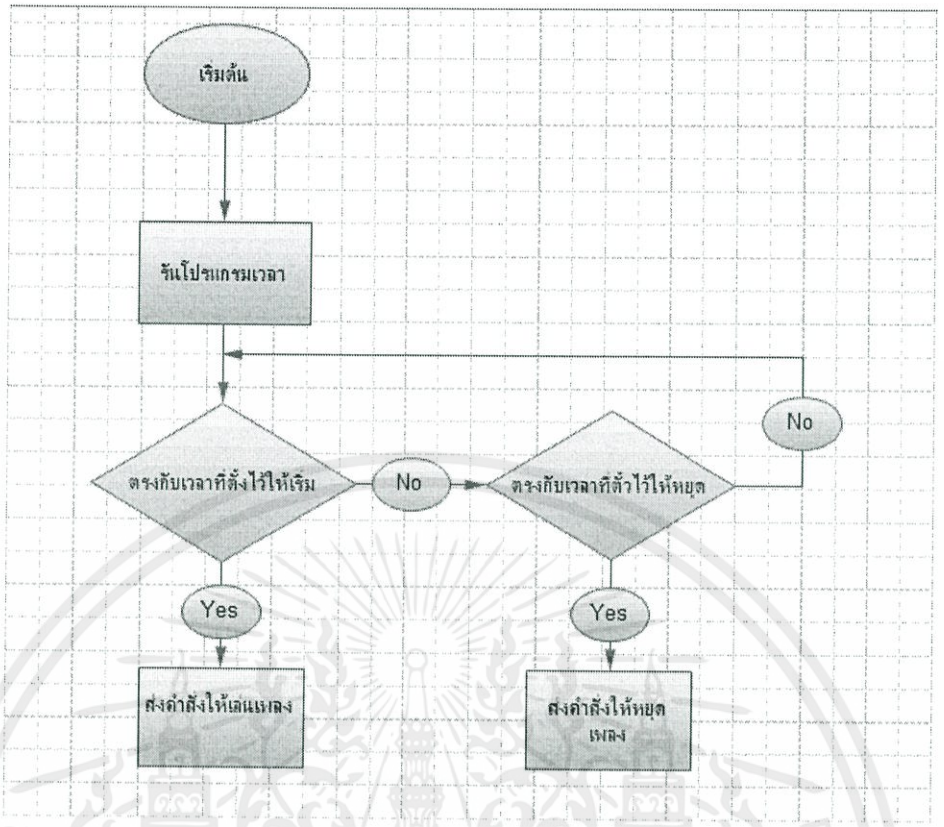
คำสั่งเงื่อนไข if() ตรวจสอบค่าเวลา หากเวลาอยู่ในช่วง 0 ถึง 9 จะเริ่มต้นเปลี่ยนข้อมูลตัวเลขเป็นสตริงพร้อมกับเพิ่มตัวเลข “0” เข้าไปข้างหน้าตัวเลขเวลาดังกล่าวเพื่อให้สามารถแสดงผลในรูปแบบ 10:05:09 เป็นต้น หากเวลาไม่ได้อยู่ในช่วง 0 ถึง 9 ก็จะเปลี่ยนตัวเลขเป็นสตริงเท่านั้น

8.บรรทัดที่ 34,44,54

แสดงผลเวลาไปที่โมดูล LCD เริ่มต้นแสดงเวลา ชั่วโมง,นาที และวินาที ตามลำดับ

3.2 การควบคุม Module MP3 โดยใช้โปรแกรมนาฬิกา

ลำดับต่อมาคือการควบคุม Module MP3 โดยใช้โปรแกรมในข้อ 3.1 โดยมีFlow Chart ดังนี้



โดยมีโปรแกรมที่เขียนขึ้นมาทดสอบดังนี้

```
1.  #include <p30fxxxx.h>
2.  #include <uart.h>
3.  #include <timer.h>
4.  #include <stdio.h>
5.  _FOSC(CSW_FSCM_OFF & XT_PLL4);
6.  _FWDT(WDT_OFF);
7.  #include "LIB_EX05_04_Time.C"
8.  #include "LIB_LCD16x2.C"
9.  #define Fcy 4000000.0
10. #define BAUD_RATE 9600.0
11. #define BAUD_RATE_GEN (Fcy/(16.0*BAUD_RATE))-1
12. #define TRUE 1
13. void Uart1_Init(void)
14. {
15. unsigned int configU1MODE, configU1STA, BaudRate;
16. CloseUART1();
17. configU1MODE = UART_EN &
```

```
        i.  UART_IDLE_CON &
        ii. UART_RX_TX &
        iii. UART_DIS_WAKE &
        iv. UART_DIS_LOOPBACK &
        v. UART_DIS_ABAUD &
        vi. UART_NO_PAR_8BIT &
        vii. UART_1STOPBIT;
18. configU1STA = UART_INT_TX_BUF_EMPTY &
        i. UART_TX_PIN_NORMAL &
        ii. UART_INT_RX_CHAR &
        iii. UART_ADR_DETECT_DIS &
        iv. UART_RX_OVERRUN_CLEAR;
19. BaudRate = BAUD_RATE_GEN;
20. OpenUART1(configU1MODE, configU1STA, BaudRate);
21. }
22. int main(void)
23. {
24. char buf[4];
25. Timer1_Init();
26. LCDInit();
27. LCDClrscr();
28. Timer_Set(0,0,0);
29. LCDPrintxy(1,1,0, "Timer1 Module..");
30. LCDPrintxy(1,2,0, "TIME: ");
31. while (TRUE)
32. {
33. if (dsp)
34. {
35. if (time_t1.hr <= 9)
36. {
        a. sprintf(buf,"0%d:",time_t1.hr);
37. }
38. else
```

```
39. {
        a.  sprintf(buf,"%d:",time_t1.hr);
40. }
41. LCDPrintxy(7,2,0, buf);
42. if (time_t1.min <= 9)
43. {
        a.  sprintf(buf,"0%d:",time_t1.min);
44. }
45. else
46. {
        a.  sprintf(buf,"%d:",time_t1.min);
        b.  }
47. LCDPrintxy(10,2,0, buf);

48. if (time_t1.sec <= 9)
49. {
        a.  sprintf(buf,"0%d",time_t1.sec);
50. }
51. else
52. {
        a.  sprintf(buf,"%d",time_t1.sec);
        b.  }
53. LCDPrintxy(13,2,0, buf);
54. if(time_t1.hr==0 && time_t1.min==0 && time_t1.sec==5)
55. {
        a.  Uart1_Init();
        b.  _TRISD0 = 0;
        c.  _LATD0 = 0;
        d.  WriteUART1(0x02);
        e.  WriteUART1(0xA2);
56. }
57. if(time_t1.hr==0 && time_t1.min==0 && time_t1.sec==10)
58. {
```

```
        a. Uart1_Init();
        b. _TRISD0 = 0;
        c. _LATD0 = 0;
        d. WriteUART1(0x02);
        e. WriteUART1(0xA0);
59. }
60. dsp = 0;
61. }
62. }
63. return 0;
64. }
```

บทที่ 4
การทดลองและผลการทดลอง

4.1 จุดประสงค์ในการทดลอง

1. เพื่อทดสอบการทำงานของโมดูล MP3 ให้สามารถทำงานตามที่ต้องการได้

2. เพื่อเช็คเวลาจริงกับเวลาที่ประมวลผลมาจากไมโครคอนโทรลเลอร์ว่ามีค่าใกล้เคียงกันหรือไม่

เวลาที่ตั้งเปิดMP3 (ชั่วโมง/นาที/วินาที)	เวลาที่ตั้งปิดMP3 (ชั่วโมง/นาที)	MP3 เล่นเพลง	MP3 ไม่เล่นเพลง
00.05.00		X	
	00.05.30		X
01.00.00		X	
	01.00.30		X
08.00.00		X	
	08.00.30		X
16.15.00		X	
	16.15.30		X

ตารางที่ 3 แสดงการทำงานของโมดูล MP3

18.00.00		X	
	18.00.30		X

ตารางที่ 4 แสดงการทำงานระหว่าง Keypad และโมดูล MP3

การตั้งเวลา	เวลาเล่นเพลง	เวลาเพลงหยุด	หมายเหตุ
00.05.00			ตั้งเวลาไว้ที่ 00.05.00
	#00.08.00#	00.08.30	ต้องการให้เล่นเพลงเวลา 00.08.00 และหยุดเพลงเวลา 00.08.30
01.00.00			ตั้งเวลาไว้ที่ 01.00.00
	#01.30.30#	01.31.00	ต้องการให้เล่นเพลงเวลา 01.03.30 และหยุดเพลงเวลา 01.31.00
08.00.00			ตั้งเวลาไว้ที่ 08.00.00
	#08.45.00#	08.45.30	ต้องการให้เล่นเพลงเวลา 08.45.00 และหยุดเพลงเวลา 08.45.30
16.00.00			ตั้งเวลาไว้ที่ 16.00.00
	#16.00.00#	16.00.30	ต้องการให้เล่นเพลงเวลา 16.00.00 และหยุดเพลงเวลา 16.00.30
18.00.00			ตั้งเวลาไว้ที่ 18.00.00
	#18.00.00#	18.00.30	ต้องการให้เล่นเพลงเวลา 18.00.00 และหยุดเพลงเวลา 18.00.30

Note “ * ” เป็นการเซตค่าเพื่อใช้ในการตั้งเวลา

“ # ” เป็นการเซตค่าในการตั้งเวลาเพื่อให้เล่นเพลงตามต้องการ

ตารางที่ 5 ทดสอบเวลาจริงกับเวลาได้จาดการอ่านจากหน้าจอ LCD

เวลาจริง	เวลาที่ได้จากจอหน้าจอ LCD
00.01.00	00.01.13
00.02.00	00.02.17
00.03.00	00.03.12
00.05.00	00.05.17

00.15.00	00.15.16
00.30.00	00.30.13
00.45.00	00.45.14
01.00.00	01.00.12
05.00.00	05.00.18
08.00.00	08.00.12
12.00.00	12.00.17
15.00.00	15.00.13

ค่าความผิดพลาดที่เกิดขึ้น =

$$\frac{\text{เวลาที่ทดลอง} - \text{เวลาจริง}}{\text{เวลาจริง}} X\ 100\%$$

ตัวอย่าง

ค่าความผิดพลาดที่เกิดขึ้น =

$$\frac{1.13 - 1.00}{1.00} X\ 100\%$$

$$= \quad 13\%$$

4.2 สรุปผลการทดลอง

1. จากการทดสอบตามโปรแกรมที่ได้ออกแบบไว้ ปรากฏว่าระบบสามารถแสดงผลได้ตามที่ออกแบบ โดยเมื่อเซตค่าเวลาที่ต้องการเข้าไปเพื่อให้โมดูล MP3 ทำงาน ไมโครคอนโทรลเลอร์ก็จะสั่งการมาให้ MP3 เปิดเพลงตามเวลาที่ตั้งไว้

2. จากการทดสอบการตั้งค่าจากคีย์บอร์ดให้เล่นเพลงเวลาที่ต้องการ โมดูล MP3 ก็สามารถทำงานได้ตามเวลาที่ตั้งไว้

บทที่ 5
สรุปและแนวทางการพัฒนา

5.1 สรุปผลงาน

โครงงานนี้จัดทำขึ้นเพื่อพัฒนาองค์ความรู้เกี่ยวกับการควบคุมการทำงานของอุปกรณ์ในระบบ เพื่อให้สามารถสั่งการการทำงานให้เป็นไปตามต้องการจะเห็นได้ว่าในโครงงานได้นำไมโครคอนโทรลเลอร์มาใช้ในการควบคุมโมดูล MP3 ให้สามารถสั่งการได้ โดยใช้การสื่อสารในรูปแบบ UART และยังสามารถใช้คีบอร์ดในการสั่งการด้วยมือ โดยใช้การสื่อสารแบบ I^2C และในการควบคุมยังใช้ อุปกรณ์รักษาเวลาที่เรียกว่า DS1307 มาใช้ในการรักษาเวลาให้ดำเนินต่อไป จากโครงงานจะเห็นได้ว่าเป็นการนำโมดูลMP3มาพัฒนาเพื่อให้เกิดมูลค่า และยังสามารถนำมาต่อยอดได้อีกเช่น นำไปเป็นนาฬิกาปลุก หรือแม้กระทั่งเครื่องวิทยุกระจายเสียงในโรงเรียน เป็นต้น

5.2 ข้อเสนอแนะและแนวทางพัฒนา

1.จากโครงงานที่ผ่านมาเป็นการพัฒนาโมดูล MP3 ให้สามารถทำงานได้อย่างอัตโนมัติ หรือสั่งการผ่านคีย์บอร์ดได้ แต่การทำงานของโมดูล MP3 นั้น ยังสามารถเพิ่มฟังก์ชั่นการทำงานให้เกิดการสั่งการได้อีกมากมายหลายอย่าง เช่น ให้เพิ่ม-ลดเสียงของโมดูล หรือแม้กระทั่งควบคุมการทำงานโดยวิธีเลือกใช้เพลงที่ต้องการ

เอกสารอ้างอิง

หนังสือ dsPIC30F MPLAB C (dsPIC30F Programming with MPLAB C Compiler)

http://www.silaresearch.com/manual/m_tds055.pdf

http://www.wvshare.com/datasheet/DALLAS_PDF/DS1307.PDF

ภาคผนวก

maxim integrated™

DS1307
64 x 8, Serial, I²C Real-Time Clock

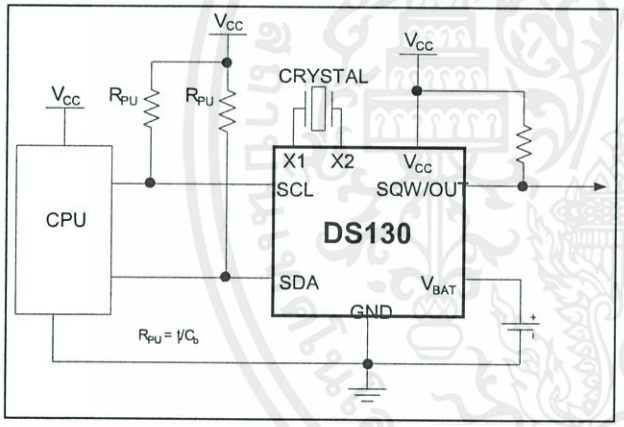
GENERAL DESCRIPTION

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I²C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

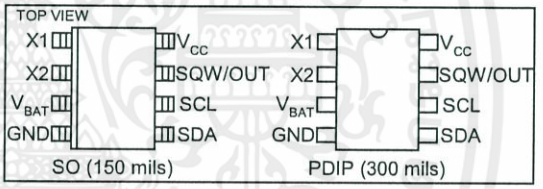
FEATURES

- Real-Time Clock (RTC) Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the week, and Year with Leap-Year Compensation Valid Up to 2100
- 56-Byte, Battery-Backed, General-Purpose RAM with Unlimited Writes
- I²C Serial Interface
- Programmable Square-Wave Output Signal
- Automatic Power-Fail Detect and Switch Circuitry
- Consumes Less than 500nA in Battery-Backup Mode with Oscillator Running
- Optional Industrial Temperature Range: -40°C to +85°C
- Available in 8-Pin Plastic DIP or SO
- Underwriters Laboratories (UL) Recognized

TYPICAL OPERATING CIRCUIT



PIN CONFIGURATIONS



ORDERING INFORMATION

PART	TEMP RANGE	VOLTAGE (V)	PIN-PACKAGE	TOP MARK*
DS1307+	0°C to +70°C	5.0	8 PDIP (300 mils)	DS1307
DS1307N+	-40°C to +85°C	5.0	8 PDIP (300 mils)	DS1307N
DS1307Z+	0°C to +70°C	5.0	8 SO (150 mils)	DS1307
DS1307ZN+	-40°C to +85°C	5.0	8 SO (150 mils)	DS1307N
DS1307Z+T&R	0°C to +70°C	5.0	8 SO (150 mils) Tape and Reel	DS1307
DS1307ZN+T&R	-40°C to +85°C	5.0	8 SO (150 mils) Tape and Reel	DS1307N

+Denotes a lead-free/RoHS-compliant package.
**A "+" anywhere on the top mark indicates a lead-free package. An "N" anywhere on the top mark indicates an industrial temperature range device.*

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim's website at www.maximintegrated.com.

REV: 100208

ABSOLUTE MAXIMUM RATINGS

Voltage Range on Any Pin Relative to Ground ..... -0.5V to +7.0V
Operating Temperature Range (Noncondensing)
Commercial ..... 0°C to +70°C
Industrial ..... -40°C to +85°C
Storage Temperature Range ..... -55°C to +125°C
Soldering Temperature (DIP, leads) ..... +260°C for 10 seconds
Soldering Temperature (surface mount) ..... Refer to the JPC/JEDEC J-STD-020 Specification.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to the absolute maximum rating conditions for extended periods may affect device reliability.

RECOMMENDED DC OPERATING CONDITIONS

(T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Supply Voltage	V_{CC}		4.5	5.0	5.5	V
Logic 1 Input	V_{IH}		2.2		V_{CC} + 0.3	V
Logic 0 Input	V_{IL}		-0.3		+0.8	V
V_{BAT} Battery Voltage	V_{BAT}		2.0	3	3.5	V

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 4.5V to 5.5V; T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Input Leakage (SCL)	I_{LI}		-1		1	µA
I/O Leakage (SDA, SQW/OUT)	I_{LO}		-1		1	µA
Logic 0 Output (I_{OL} = 5mA)	V_{OL}				0.4	V
Active Supply Current (f_{SCL} = 100kHz)	I_{CCA}				1.5	mA
Standby Current	I_{CCS}	(Note 3)			200	µA
V_{BAT} Leakage Current	I_{BATLKG}			5	50	nA
Power-Fail Voltage (V_{BAT} = 3.0V)	V_{PF}		1.216 x V_{BAT}	1.25 x V_{BAT}	1.284 x V_{BAT}	V

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 0V, V_{BAT} = 3.0V; T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
V_{BAT} Current (OSC ON); SQW/OUT OFF	I_{BAT1}			300	500	nA
V_{BAT} Current (OSC ON); SQW/OUT ON (32kHz)	I_{BAT2}			480	800	nA
V_{BAT} Data-Retention Current (Oscillator Off)	I_{BATDR}			10	100	nA

WARNING: Negative undershoots below -0.3V while the part is in battery-backed mode may cause loss of data.

AC ELECTRICAL CHARACTERISTICS

(V_{CC} = 4.5V to 5.5V; T_A = 0°C to +70°C, T_A = -40°C to +85°C.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
SCL Clock Frequency	f_{SCL}		0		100	kHz
Bus Free Time Between a STOP and START Condition	t_{BUF}		4.7			µs
Hold Time (Repeated) START Condition	$t_{HD:STA}$	(Note 4)	4.0			µs
LOW Period of SCL Clock	t_{LOW}		4.7			µs
HIGH Period of SCL Clock	t_{HIGH}		4.0			µs
Setup Time for a Repeated START Condition	$t_{SU:STA}$		4.7			µs
Data Hold Time	$t_{HD:DAT}$		0			µs
Data Setup Time	$t_{SU:DAT}$	(Notes 5, 6)	250			ns
Rise Time of Both SDA and SCL Signals	t_R				1000	ns
Fall Time of Both SDA and SCL Signals	t_F				300	ns
Setup Time for STOP Condition	$t_{SU:STO}$		4.7			µs

CAPACITANCE

(T_A = +25°C)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Pin Capacitance (SDA, SCL)	$C_{I/O}$				10	pF
Capacitance Load for Each Bus Line	C_B	(Note 7)			400	pF

Note 1: All voltages are referenced to ground.

Note 2: Limits at -40°C are guaranteed by design and are not production tested.

Note 3: I_{CCS} specified with V_{CC} = 5.0V and SDA, SCL = 5.0V.

Note 4: After this period, the first clock pulse is generated.

Note 5: A device must internally provide a hold time of at least 300ns for the SDA signal (referred to the $V_{IH(MIN)}$ of the SCL signal) to bridge the undefined region of the falling edge of SCL.

Note 6: The maximum $t_{HD:DAT}$ only has to be met if the device does not stretch the LOW period (t_{LOW}) of the SCL signal.

Note 7: C_B—total capacitance of one bus line in pF.

TIMING DIAGRAM

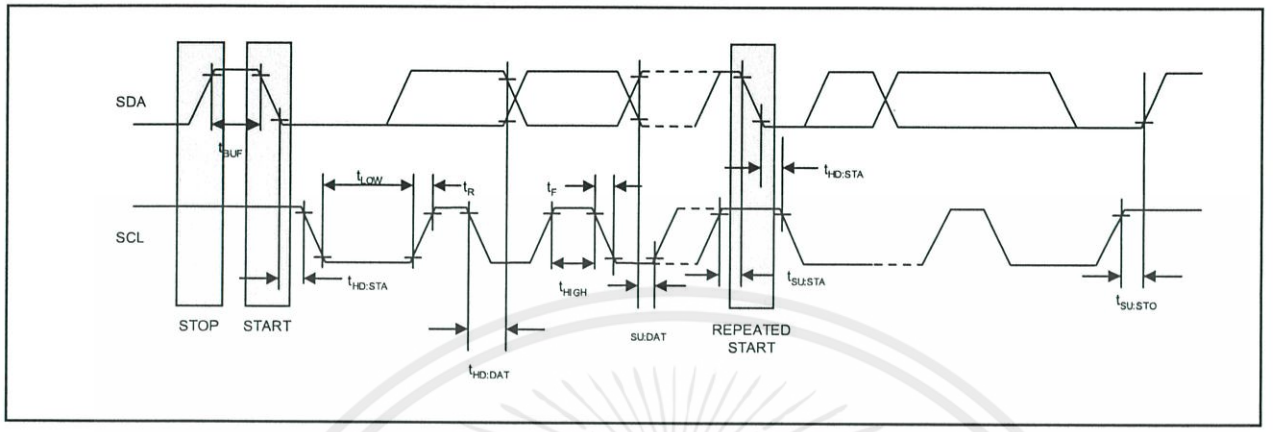
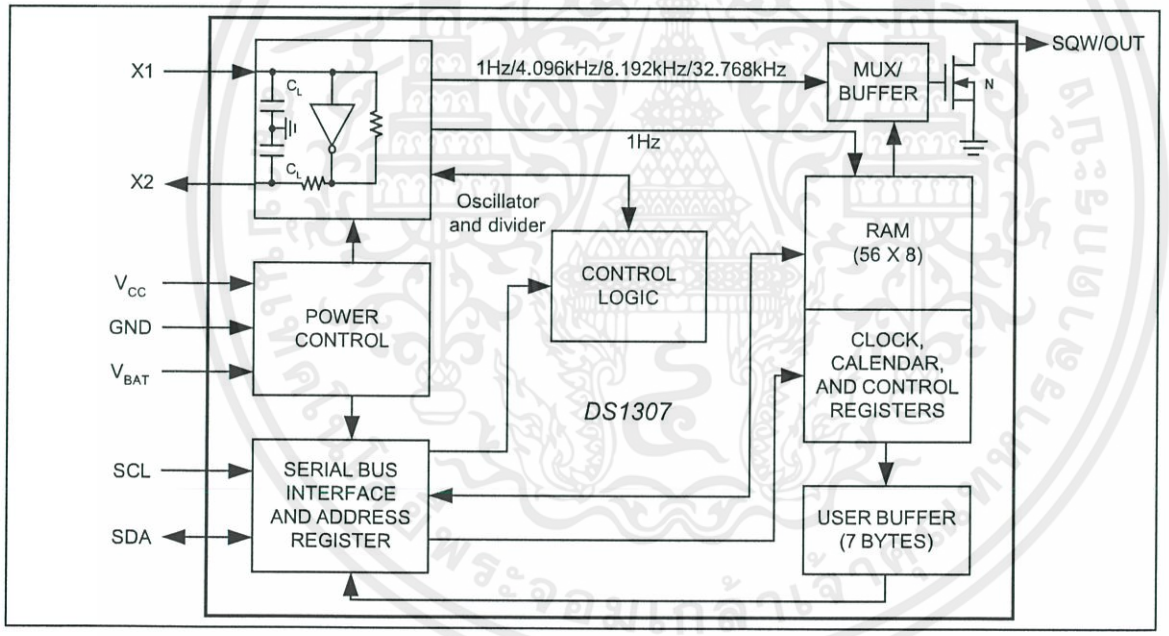
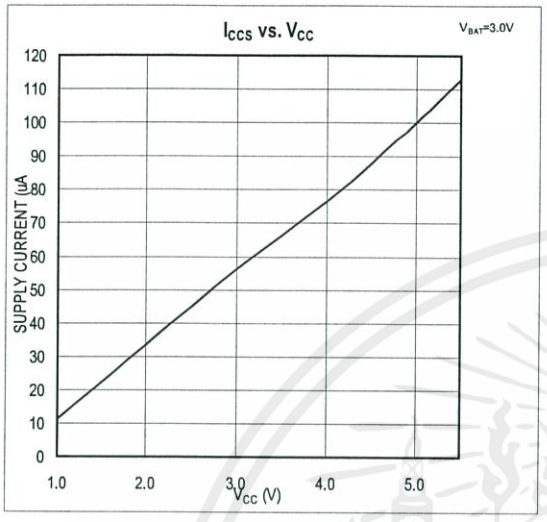


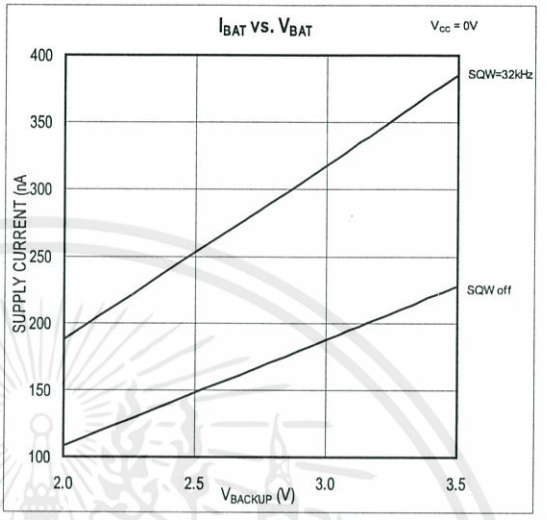
Figure 1. Block Diagram

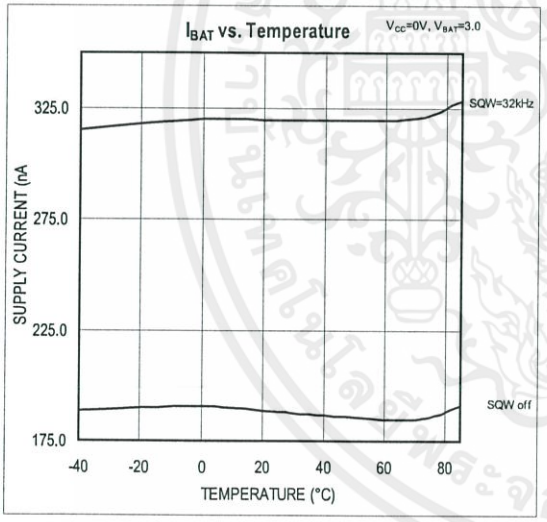


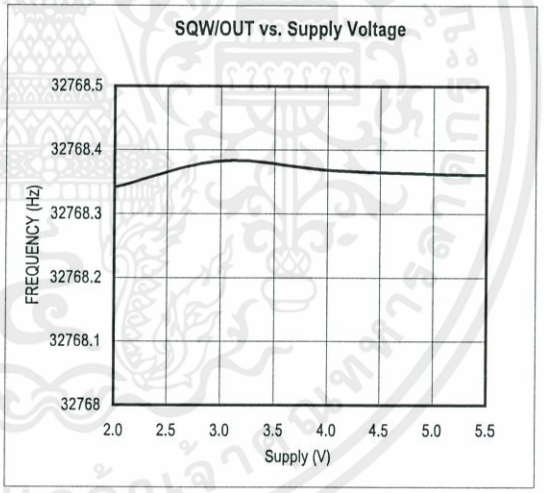
TYPICAL OPERATING CHARACTERISTICS

(V_{CC} = 5.0V, T_A = +25°C, unless otherwise noted.)









PIN DESCRIPTION

PIN	NAME	FUNCTION
1	X1	Connections for Standard 32.768kHz Quartz Crystal. The internal oscillator circuitry is designed for operation with a crystal having a specified load capacitance (C_L) of 12.5pF. X1 is the input to the oscillator and can optionally be connected to an external 32.768kHz oscillator. The output of the internal oscillator, X2, is floated if an external oscillator is connected to X1. **Note:** For more information on crystal selection and crystal layout considerations, refer to *Application Note 58: Crystal Considerations with Dallas Real-Time Clocks.*
2	X2	
3	V_{BAT}	Backup Supply Input for Any Standard 3V Lithium Cell or Other Energy Source. Battery voltage must be held between the minimum and maximum limits for proper operation. Diodes in series between the battery and the V_{BAT} pin may prevent proper operation. If a backup supply is not required, V_{BAT} must be grounded. The nominal power-fail trip point (V_{PF}) voltage at which access to the RTC and user RAM is denied is set by the internal circuitry as 1.25 x V_{BAT} nominal. A lithium battery with 48mAh or greater will back up the DS1307 for more than 10 years in the absence of power at +25°C. UL recognized to ensure against reverse charging current when used with a lithium battery. Go to: www.maxim-ic.com/qa/info/ul/.
4	GND	Ground
5	SDA	Serial Data Input/Output. SDA is the data input/output for the I^2C serial interface. The SDA pin is open drain and requires an external pullup resistor. The pullup voltage can be up to 5.5V regardless of the voltage on V_{CC}.
6	SCL	Serial Clock Input. SCL is the clock input for the I^2C interface and is used to synchronize data movement on the serial interface. The pullup voltage can be up to 5.5V regardless of the voltage on V_{CC}.
7	SQW/OUT	Square Wave/Output Driver. When enabled, the SQWE bit set to 1, the SQW/OUT pin outputs one of four square-wave frequencies (1Hz, 4kHz, 8kHz, 32kHz). The SQW/OUT pin is open drain and requires an external pullup resistor. SQW/OUT operates with either V_{CC} or V_{BAT} applied. The pullup voltage can be up to 5.5V regardless of the voltage on V_{CC}. If not used, this pin can be left floating.
8	V_{CC}	Primary Power Supply. When voltage is applied within normal limits, the device is fully accessible and data can be written and read. When a backup supply is connected to the device and V_{CC} is below V_{TP}, read and writes are inhibited. However, the timekeeping function continues unaffected by the lower input voltage.

DETAILED DESCRIPTION

The DS1307 is a low-power clock/calendar with 56 bytes of battery-backed SRAM. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The DS1307 operates as a slave device on the I^2C bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When V_{CC} falls below 1.25 x V_{BAT}, the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out-of-tolerance system. When V_{CC} falls below V_{BAT}, the device switches into a low-current battery-backup mode. Upon power-up, the device switches from battery to V_{CC} when V_{CC} is greater than V_{BAT} +0.2V and recognizes inputs when V_{CC} is greater than 1.25 x V_{BAT}. The block diagram in Figure 1 shows the main elements of the serial RTC.

OSCILLATOR CIRCUIT

The DS1307 uses an external 32.768kHz crystal. The oscillator circuit does not require any external resistors or capacitors to operate. Table 1 specifies several crystal parameters for the external crystal. Figure 1 shows a functional schematic of the oscillator circuit. If using a crystal with the specified characteristics, the startup time is usually less than one second.

CLOCK ACCURACY

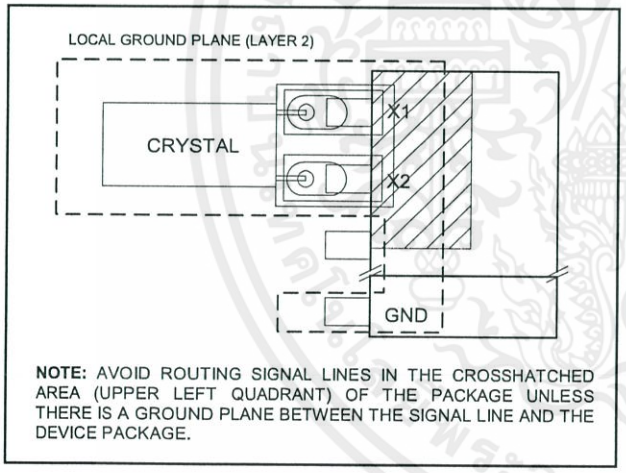
The accuracy of the clock is dependent upon the accuracy of the crystal and the accuracy of the match between the capacitive load of the oscillator circuit and the capacitive load for which the crystal was trimmed. Additional error will be added by crystal frequency drift caused by temperature shifts. External circuit noise coupled into the oscillator circuit may result in the clock running fast. Refer to Application Note 58: *Crystal Considerations with Dallas Real-Time Clocks* for detailed information.

Table 1. Crystal Specifications*

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS
Nominal Frequency	f_O		32.768		kHz
Series Resistance	ESR			45	kΩ
Load Capacitance	C_L		12.5		pF

**The crystal, traces, and crystal input pins should be isolated from RF generating signals. Refer to* Application Note 58: Crystal Considerations for Dallas Real-Time Clocks *for additional specifications.*

Figure 2. Recommended Layout for Crystal



RTC AND RAM ADDRESS MAP

Table 2 shows the address map for the DS1307 RTC and RAM registers. The RTC registers are located in address locations 00h to 07h. The RAM registers are located in address locations 08h to 3Fh. During a multibyte access, when the address pointer reaches 3Fh, the end of RAM space, it wraps around to location 00h, the beginning of the clock space.

CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. Table 2 shows the RTC registers. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the BCD format. The day-of-week register increments at midnight. Values that correspond to the day of week are user-defined but must be sequential (i.e., if 1 equals Sunday, then 2 equals Monday, and so on.) Illogical time and date entries result in undefined operation. Bit 7 of Register 0 is the clock halt (CH) bit. When this bit is set to 1, the oscillator is disabled. When cleared to 0, the oscillator is enabled. On first application of power to the device the time and date registers are typically reset to 01/01/00 01 00:00:00 (MM/DD/YY DOW HH:MM:SS). The CH bit in the seconds register will be set to a 1. The clock can be halted whenever the timekeeping functions are not required, which minimizes current (I_{BATDR}).

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12-hour or 24-hour mode-select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10-hour bit (20 to 23 hours). The hours value must be re-entered whenever the 12/24-hour mode bit is changed.

When reading or writing the time and date registers, secondary (user) buffers are used to prevent errors when the internal registers update. When reading the time and date registers, the user buffers are synchronized to the internal registers on any I^2C START. The time information is read from these secondary registers while the clock continues to run. This eliminates the need to re-read the registers in case the internal registers update during a read. The divider chain is reset whenever the seconds register is written. Write transfers occur on the I^2C acknowledge from the DS1307. Once the divider chain is reset, to avoid rollover issues, the remaining time and date registers must be written within one second.

Table 2. Timekeeper Registers

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

CONTROL REGISTER

The DS1307 control register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	0	0	SQWE	0	0	RS1	RS0

Bit 7: Output Control (OUT). This bit controls the output level of the SQW/OUT pin when the square-wave output is disabled. If SQWE = 0, the logic level on the SQW/OUT pin is 1 if OUT = 1 and is 0 if OUT = 0. On initial application of power to the device, this bit is typically set to a 0.

Bit 4: Square-Wave Enable (SQWE). This bit, when set to logic 1, enables the oscillator output. The frequency of the square-wave output depends upon the value of the RS0 and RS1 bits. With the square-wave output set to 1Hz, the clock registers update on the falling edge of the square wave. On initial application of power to the device, this bit is typically set to a 0.

Bits 1 and 0: Rate Select (RS[1:0]). These bits control the frequency of the square-wave output when the square-wave output has been enabled. The following table lists the square-wave frequencies that can be selected with the RS bits. On initial application of power to the device, these bits are typically set to a 1.

RS1	RS0	SQW/OUT OUTPUT	SQWE	OUT
0	0	1Hz	1	X
0	1	4.096kHz	1	X
1	0	8.192kHz	1	X
1	1	32.768kHz	1	X
X	X	0	0	0
X	X	1	0	1

I²C DATA BUS

The DS1307 supports the I²C protocol. A device that sends data onto the bus is defined as a transmitter and a device receiving data as a receiver. The device that controls the message is called a master. The devices that are controlled by the master are referred to as slaves. The bus must be controlled by a master device that generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions. The DS1307 operates as a slave on the I²C bus.

Figures 3, 4, and 5 detail how data is transferred on the I²C bus.

- Data transfer can be initiated only when the bus is not busy.
- During data transfer, the data line must remain stable whenever the clock line is HIGH. Changes in the data line while the clock line is high will be interpreted as control signals.

Accordingly, the following bus conditions have been defined:

Bus not busy: Both data and clock lines remain HIGH.

START data transfer: A change in the state of the data line, from HIGH to LOW, while the clock is HIGH, defines a START condition.

STOP data transfer: A change in the state of the data line, from LOW to HIGH, while the clock line is HIGH, defines the STOP condition.

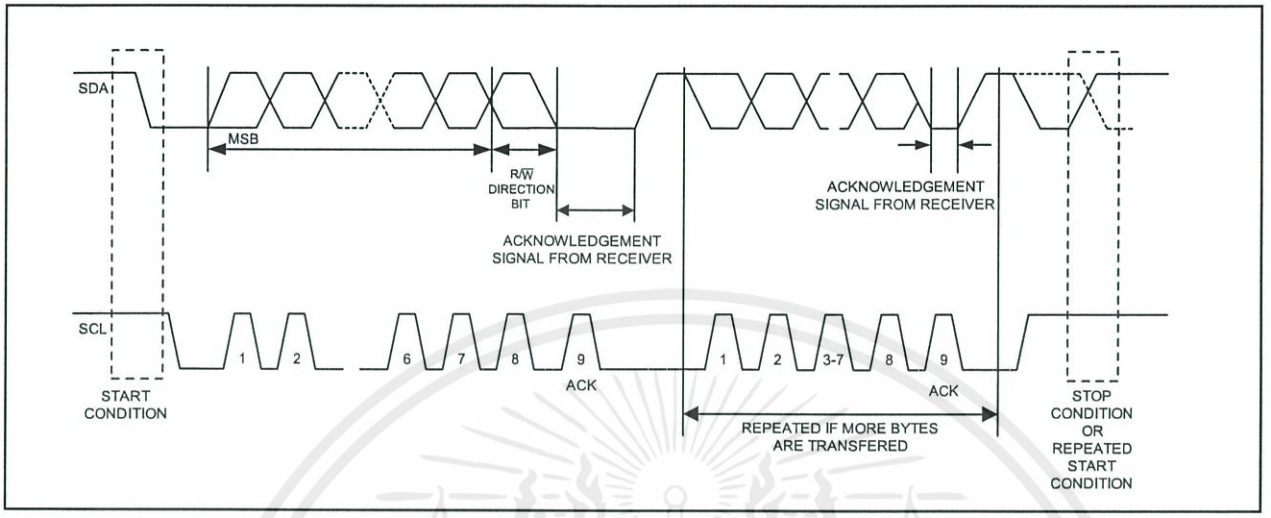
Data valid: The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the HIGH period of the clock signal. The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data.

Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes transferred between START and STOP conditions is not limited, and is determined by the master device. The information is transferred byte-wise and each receiver acknowledges with a ninth bit. Within the I²C bus specifications a standard mode (100kHz clock rate) and a fast mode (400kHz clock rate) are defined. The DS1307 operates in the standard mode (100kHz) only.

Acknowledge: Each receiving device, when addressed, is obliged to generate an acknowledge after the reception of each byte. The master device must generate an extra clock pulse which is associated with this acknowledge bit.

A device that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse. Of course, setup and hold times must be taken into account. A master must signal an end of data to the slave by not generating an acknowledge bit on the last byte that has been clocked out of the slave. In this case, the slave must leave the data line HIGH to enable the master to generate the STOP condition.

Figure 3. Data Transfer on I²C Serial Bus



Depending upon the state of the R/$\overline{W}$ bit, two types of data transfer are possible:

1. **Data transfer from a master transmitter to a slave receiver.** The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte. Data is transferred with the most significant bit (MSB) first.

2. **Data transfer from a slave transmitter to a master receiver.** The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. This is followed by the slave transmitting a number of data bytes. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a "not acknowledge" is returned.

   The master device generates all the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the bus will not be released. Data is transferred with the most significant bit (MSB) first.

The DS1307 can operate in the following two modes:

1. **Slave Receiver Mode (Write Mode):** Serial data and clock are received through SDA and SCL. After each byte is received an acknowledge bit is transmitted. START and STOP conditions are recognized as the beginning and end of a serial transfer. Hardware performs address recognition after reception of the slave address and direction bit (see Figure 4). The slave address byte is the first byte received after the master generates the START condition. The slave address byte contains the 7-bit DS1307 address, which is 1101000, followed by the direction bit (R/$\overline{W}$), which for a write is 0. After receiving and decoding the slave address byte, the DS1307 outputs an acknowledge on SDA. After the DS1307 acknowledges the slave address + write bit, the master transmits a word address to the DS1307. This sets the register pointer on the DS1307, with the DS1307 acknowledging the transfer. The master can then transmit zero or more bytes of data with the DS1307 acknowledging each byte received. The register pointer automatically increments after each data byte are written. The master will generate a STOP condition to terminate the data write.

2. **Slave Transmitter Mode (Read Mode):** The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit will indicate that the transfer direction is reversed. The DS1307 transmits serial data on SDA while the serial clock is input on SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer (see Figure 5). The slave address byte is the first byte received after the START condition is generated by the master. The slave address byte contains the 7-bit DS1307 address, which is 1101000, followed by the direction bit (R/$\overline{W}$), which is 1 for a read. After receiving and decoding the slave address the DS1307 outputs an acknowledge on SDA. The DS1307 then begins to transmit data starting with the register address pointed to by the register pointer. If the register pointer is not written to before the initiation of a read mode the first address that is read is the last one stored in the register pointer. The register pointer automatically increments after each byte are read. The DS1307 must receive a Not Acknowledge to end a read.

Figure 4. Data Write—Slave Receiver Mode

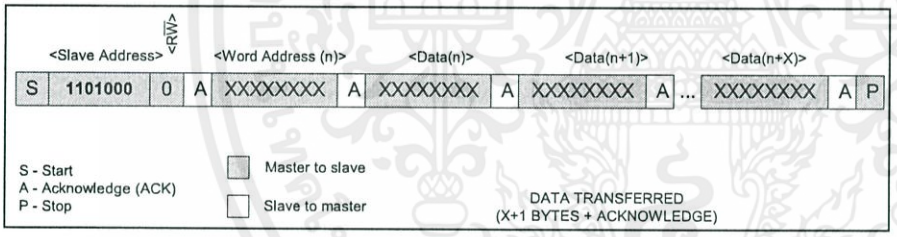


Figure 5. Data Read—Slave Transmitter Mode

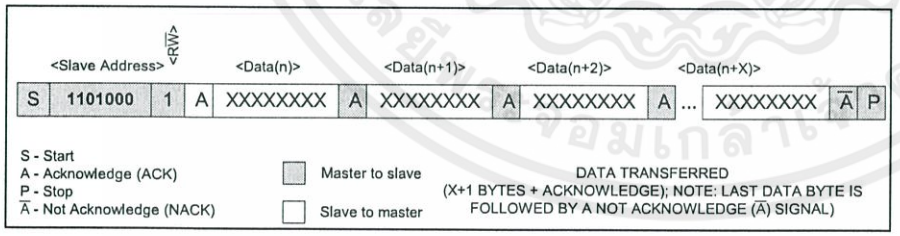
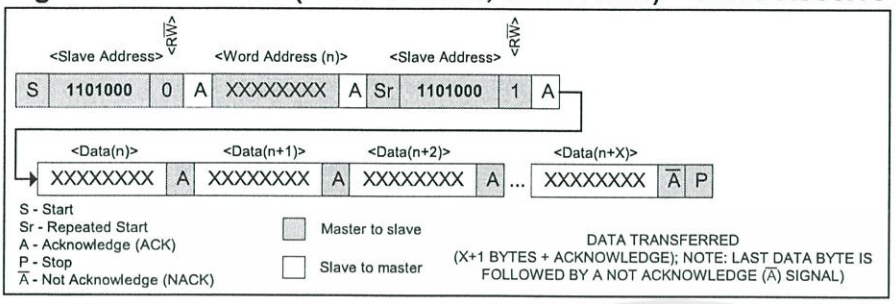


Figure 6. Data Read (Write Pointer, Then Read)—Slave Receive and Transmit



PACKAGE INFORMATION

For the latest package outline information and land patterns, go to **www.maxim-ic.com/packages**.

PACKAGE TYPE	PACKAGE CODE	DOCUMENT NO.
8 PDIP	—	**21-0043**
8 SO	—	**21-0041**

REVISION HISTORY

REVISION DATE	DESCRIPTION	PAGES CHANGED
100208	Moved the *Typical Operating Circuit* and *Pin Configurations* to first page.	1
	Removed the leaded part numbers from the *Ordering Information* table.	1
	Added an open-drain transistor to SQW/OUT in the block diagram (Figure 1).	4
	Added the pullup voltage range for SDA, SCL, and SQW/OUT to the *Pin Description* table and noted that SQW/OUT can be left open if not used.	6
	Added default time and date values on first application of power to the *Clock and Calendar* section and deleted the note that initial power-on state is not defined.	8
	Added default on initial application of power to bit info in the *Control Register* section.	9
	Updated the *Package Information* section to reflect new package outline drawing numbers.	13

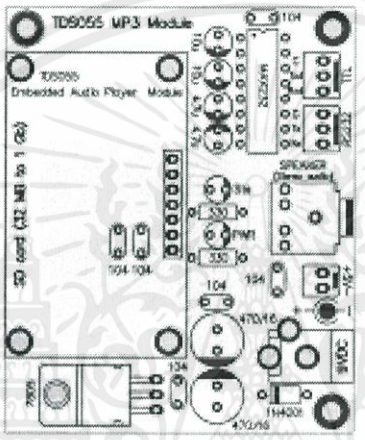
TDS055 MP3 PLAYER MODULE

(แผ่น PCB ฐานรอง V1.0)

ส่วนประกอบสินค้า ... บอร์ด, สาย L232PC, แผ่น CD-Rom, ชีทคู่มือ
Option บนบอร์ด ... ไม่มี
อาจต้องซื้อเพิ่ม ... ภาคจ่ายไฟ, SD-Card

D-TDS055 เป็นโมดูลที่ใช้สำหรับเล่นไฟล์เสียง MP3 โดยการใช้งานจะสั่งผ่าน RS-232 Baudrate 9600,N,8,1 เก็บข้อมูลโดยใช้ SD-Card ประยุกต์ใช้งานได้หลากหลายที่เกี่ยวกับไฟล์เสียง MP3 โดยทางบริษัทศิลารีเสิร์ช ได้นำตัวโมดูล TDS055 มาทำเพิ่มเติมในส่วนของ Power Supply, RS-232 เพื่อสะดวกในการนำไปใช้งาน

ภาพบอร์ด



คุณสมบัติ

- Power Supply 9 Vdc หรือ 5 Vdc
- ใช้ได้เฉพาะไฟล์ MP3 เท่านั้น
- Stereo Audio Output
- รองรับ SD-Card (32 Mb to 1 Gb)
- ความเร็วในการสื่อสาร 9600,N,8,1
- มีพอร์ตสื่อสาร RS232 ต่อผ่านขั้ว 3 PIN
- มีขั้วต่อสำหรับ ADAPTER 9 Vdc หรือเลือกใช้ขั้วเสียบ 5 VOLT (DC) แบบ 2 PIN

แนวทางนำไปใช้งาน

- SD Card ที่จะนำมาใช้งานต้อง Format แบบ FAT32 (not FAT16)
- Folder ที่ใช้สำหรับเก็บไฟล์ MP3 จะมี 2 แบบ คือ "SONG" และ "ADVERT"
- SONG Floder จะเล่นไฟล์เสียง MP3 ทันที เมื่อมีการจ่ายไฟเข้าบอร์ด
- ADVERT Floder จะเล่นไฟล์เสียง MP3 เฉพาะไฟล์ที่ต้องการได้ โดย folder "ADVERT" จะต้องตั้งชื่อเป็น ADVERT01 ,ADVERT02 สามารถตั้งได้จนถึง ADVERT99 และชื่อไฟล์ใน folder "ADVERT" จะต้องตั้งเป็น 001.mp3 , 002.mp3 ได้จนถึง 999.mp3
- โปรแกรมเทอร์มินอลที่ใช้สำหรับส่งค่าเพื่อสั่งงานโมดูลชื่อ sscom32E.exe อยู่ในแผ่น CD-Rom ที่ให้มาพร้อมกับสินค้า
- การใช้งานชุดคำสั่งดูได้จาก DATASHEET ที่ให้มา

บริษัท ศิลารีเสิร์ช จำกัด 1108/41 ถนนสุขุมวิท แขวงพระโขนง เขตคลองเตย กรุงเทพฯ 10110
TEL. 02-712-2850-2 FAX. 02-381-1447 www.silaresearch.com

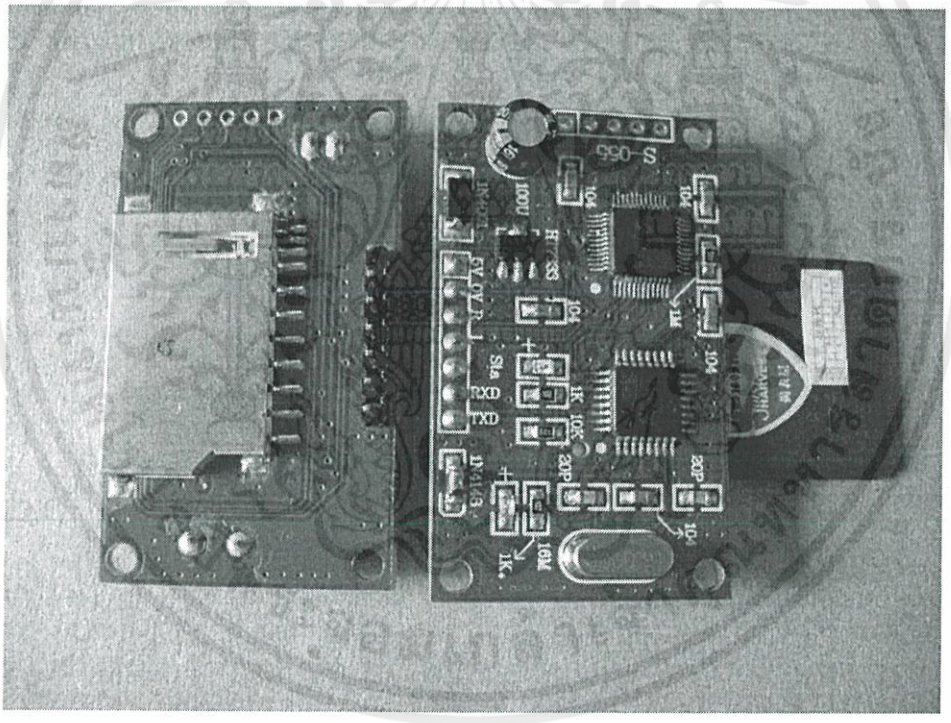
Embedded Audio Player Module

TDS055 MP3 Module

Ⅰ.Features

>Power supply DC 5V to DC 9V
>Dimension 50mm*30mm
>Support mp3 audio format
>Stereo audio output with good sound quality
>Memory type SD card (32Mb to 1Gb)
>File System FAT16
>For simple application,
> Means of communication : RS232
>9600bit/s 1 start bit+ 8 data bit+1 end bit None verify （TTL level）
>Play mp3 files in "SONG" folder in SD card once give module power .
>External MCU control volume and play exact files in the "ADVERT" folders.

Ⅱ.Product picture



Ⅲ.Applications

a. Public place announcement system
b. Entertainment device sound sources
c.Tourism guide device
d.High class gift and toys.
e.Announcer,alarm,adverting, background music systems
f. Selfservice audio navigation system in dining room, hotel,bank…
g.And other products need high quality and long duration sound.

Ⅳ.Pins descriptions

Pins	Desriptions
5V	Power positive DC5V to DC9V
0V	GND
R	R audio output
L	L audio output
Sta	Playing indicator (Low level normally , when playing it is High Level)
RXD	Serial interface receive pin (TTL level)
TXD	Serial interface transmit pin(TTL level)

Ⅴ.Communication format

Item	Function	Byte A	Byte B	Byte C	Byte D	Byte E	Byte F	Byte G
1	Play song	02H	A0H	-	-	-	-	-
2	Pause song	02H	A1H	-	-	-	-	-
3	Stop song	02H	A2H	-	-	-	-	-
4	Next song	02H	A3H	-	-	-	-	-
5	Previous song	02H	A4H	-	-	-	-	-
6	Volume control	03H	A5H	Volume(0-8)	-	-	-	-
7	Play ad.	07H	A6H	Folder name tens	Folder name digits	File name digits	File name tens	File name hundreds
8	Pause ad.	02H	A7H	-	-	-	-	-
9	Continue ad.	02H	A8H	-	-	-	-	-
10	Stop ad.	02H	A9H	-	-	-	-	-

1.Play song （02H/A0H）

Note: In pause or stop state ,send these codes will make it into playing state ..

Byte A: 02H

Byte B :A0H

Receive:

Receive correctly and return: 'ok'

2. Pause song (02H/A1H)

Note: In playing state, send these codes will make it into pause state.

Byte A: 02H

Byte B :A1H

Receive:

Receive correctly and return: 'ok'

3. Stop song (02H/A2H)

Note: In playing state, send these codes will make it into stop state.

Byte A: 02H

Byte B :A2H

Receive:

Receive correctly and return: 'ok'

4. Stop song (02H/A3H)
Note: In playing ,pause or stop state, send these codes will make it into next song.
Byte A: 02H
Byte B :A3H
Receive:
Receive correctly and return: 'ok'

5. Stop song (02H/A4H)
Note: In playing ,pause or stop state, send these codes will make it into previous song.
Byte A: 02H
Byte B :A4H
Receive:
Receive correctly and return: 'ok'

6. Volume control (02H/A5H)
Note: For volume adjustment
Byte A: 03H
Byte B :A5H
Byte C: Volume level (0 – 8)
Receive:
Receive correctly and return: 'ok'

7.Play ad(07H/A6H/)
Note: Play advertisement
Byte A: 07H
Byte B:A6H
Byte C:Folder name tens (ASCII value)
Byte D:Folder name digits(ASCII value)
Byte E: File name hundreds (ASCII value)
Byte F: File name tens(ASCII value)
Byte G : File name digits (ASCII value)
Receive correctly and return: 'ok'

8.Pause ad (02H/A7H)
Note:In advertisement playing state, send these codes to pause the it .
Byte A: 02H
Byte B: A7H
Receive correctly and return: 'ok'

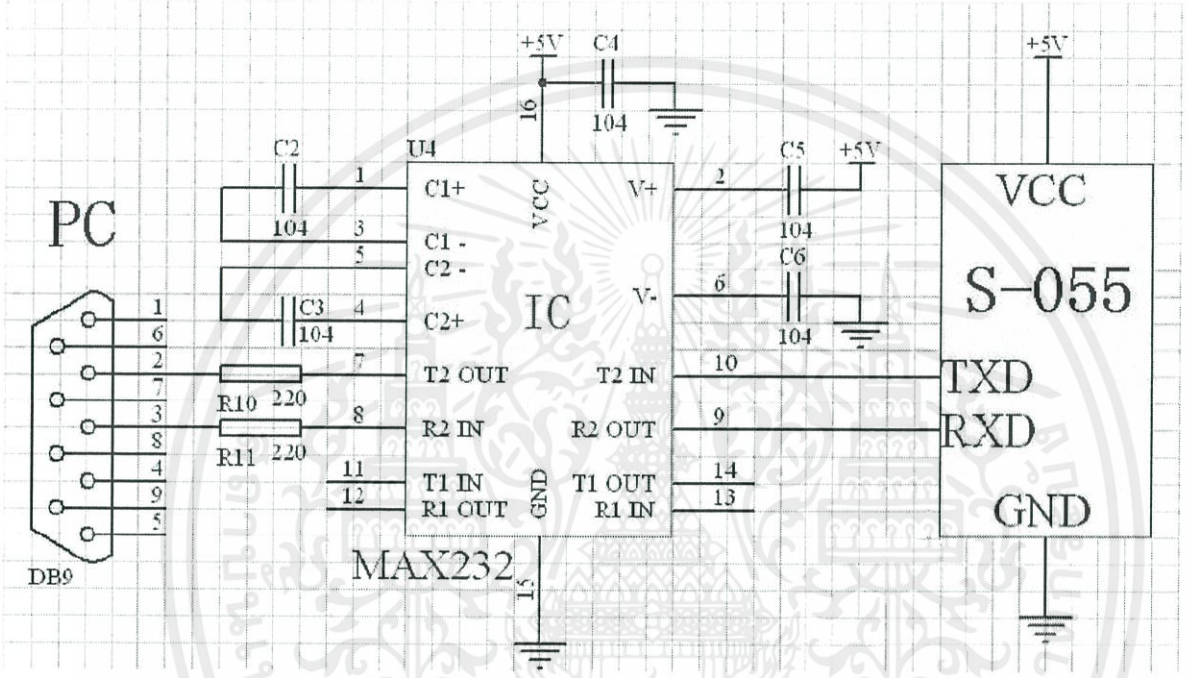
9.Pause ad (02H/A7H)
Note:In advertisement pause state, send these codes to continute it .
Byte A: 02H
Byte B: A8H
Receive correctly and return: 'ok'

10.Stop ad (02H/A9H)
Note:In advertisement pause state, send these codes to continute it .

Byte A: 02H
Byte B: A9H
Receive correctly and return: 'ok'

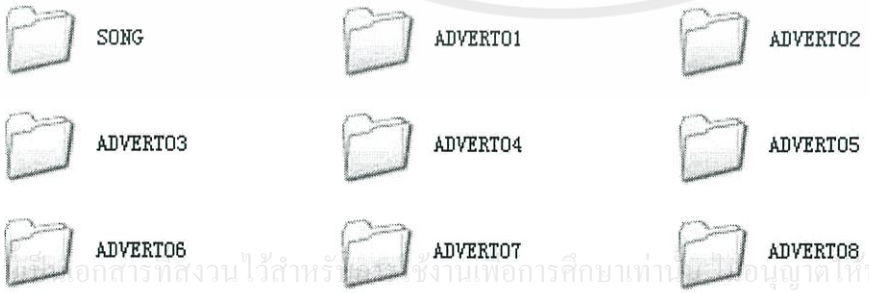
VI. RS232 Serial Interface Test

1. Set up a Max232 circuit as follow schematic



2. Format the SD card to "FAT" (not FAT32),then new "SONG" folder and ADVER folders.
ADVERT folders' name should b ADVERT01,ADVERT02,ADVERT03,.....ADVERT99 , 99 folders maximum.
And mp3 files in each ADVERT folder should be 001.mp3 ,002.mp3,003.mp3 .....999.mp3 ,
999 mp3 files in each folder maximum.

Folders in SD card



MP3 files in SONG folder


big big world.mp3


my love.mp3


seasons in the sun.mp3


say you say me.mp3


yesterday once more.mp3


love .mp3


my heart will go on .mp3


take me to your heart .mp3

MP3 files in ADVERT folders


001.mp3


002.mp3


003.mp3


004.mp3


005.mp3

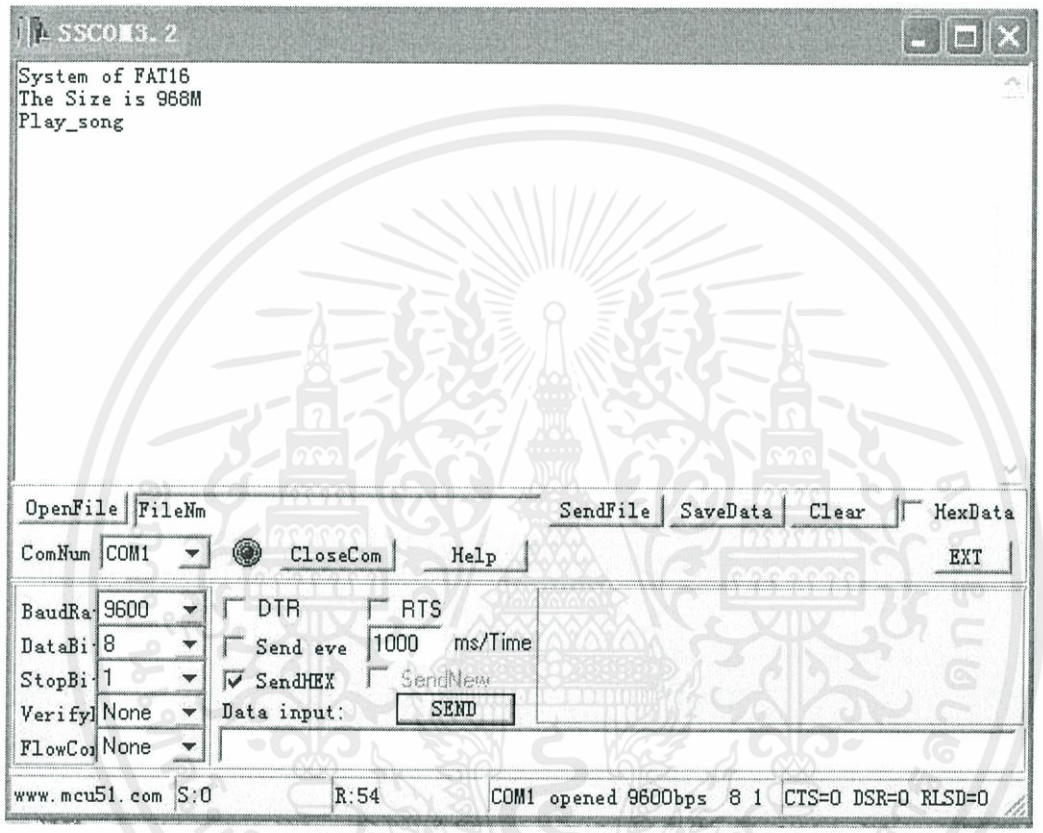

006.mp3


007 .mp3


008 .mp3

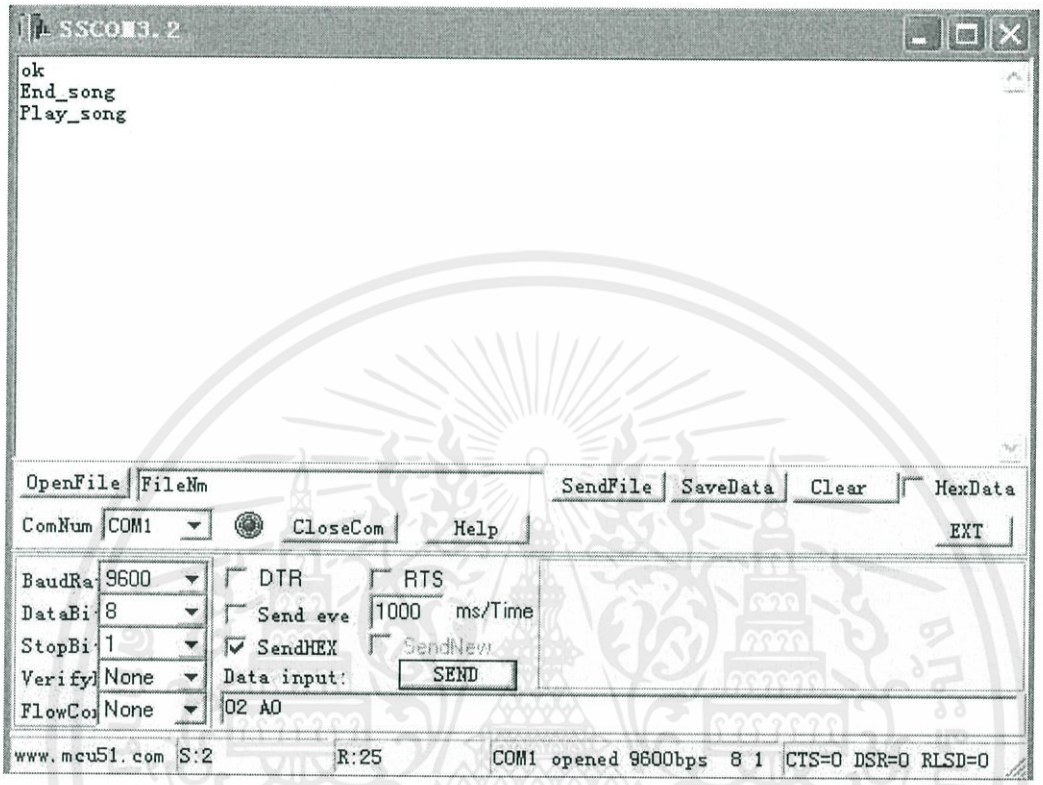
3. Connect computer to Max232 circuit and module , open the SSCOM 3.2 or other serial interface assistant software download from Internet

Click "Send Hex"

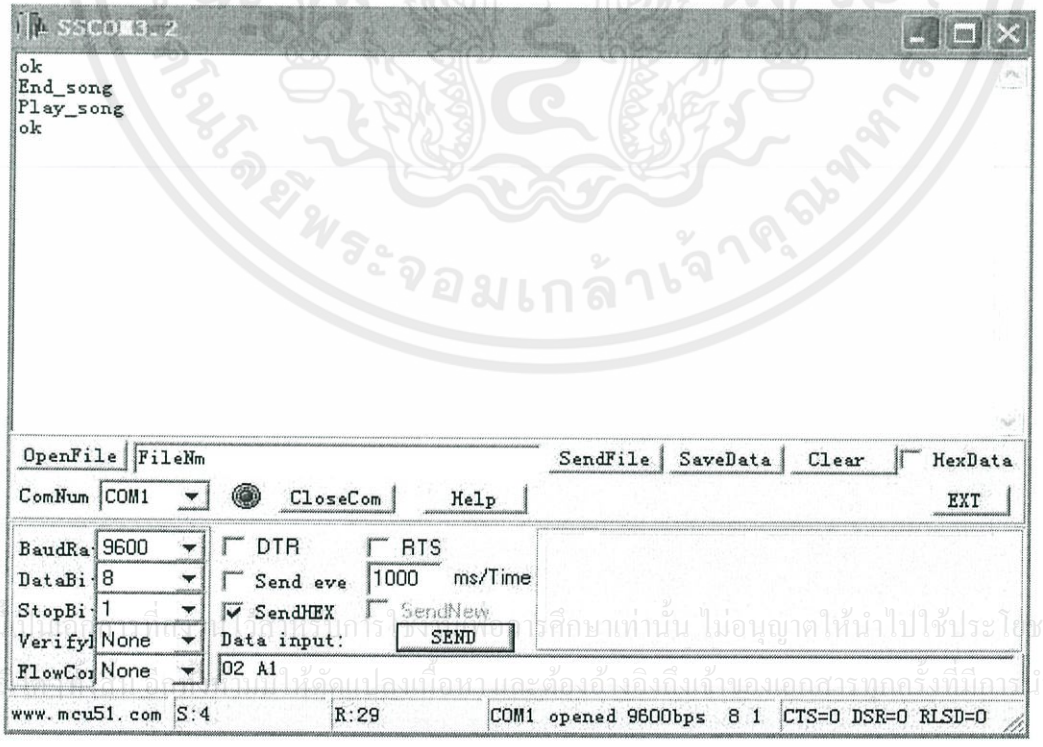


Control Examples

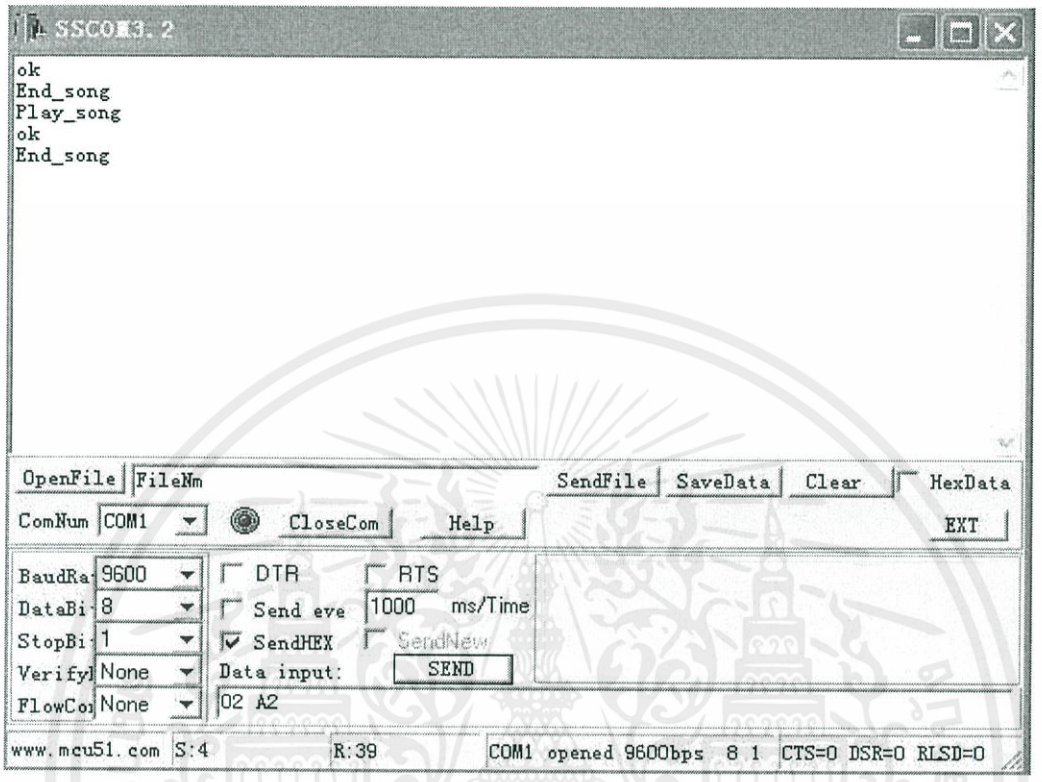
Send : 02 A0 to play song.



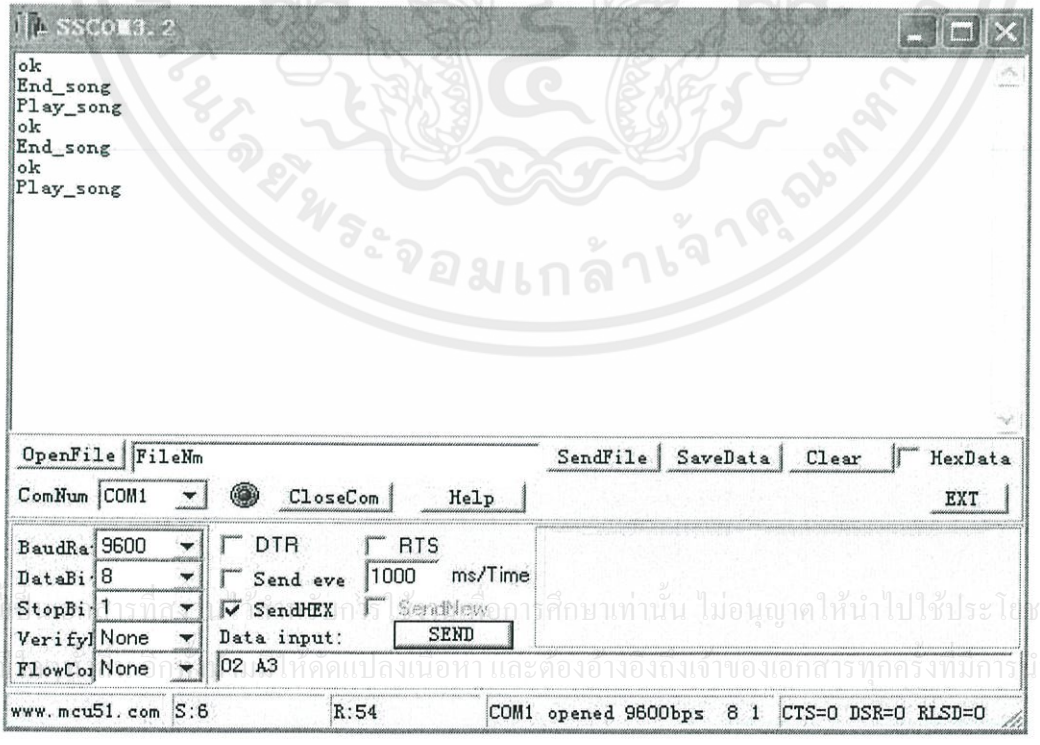
Send : 02 A1 to plause song.



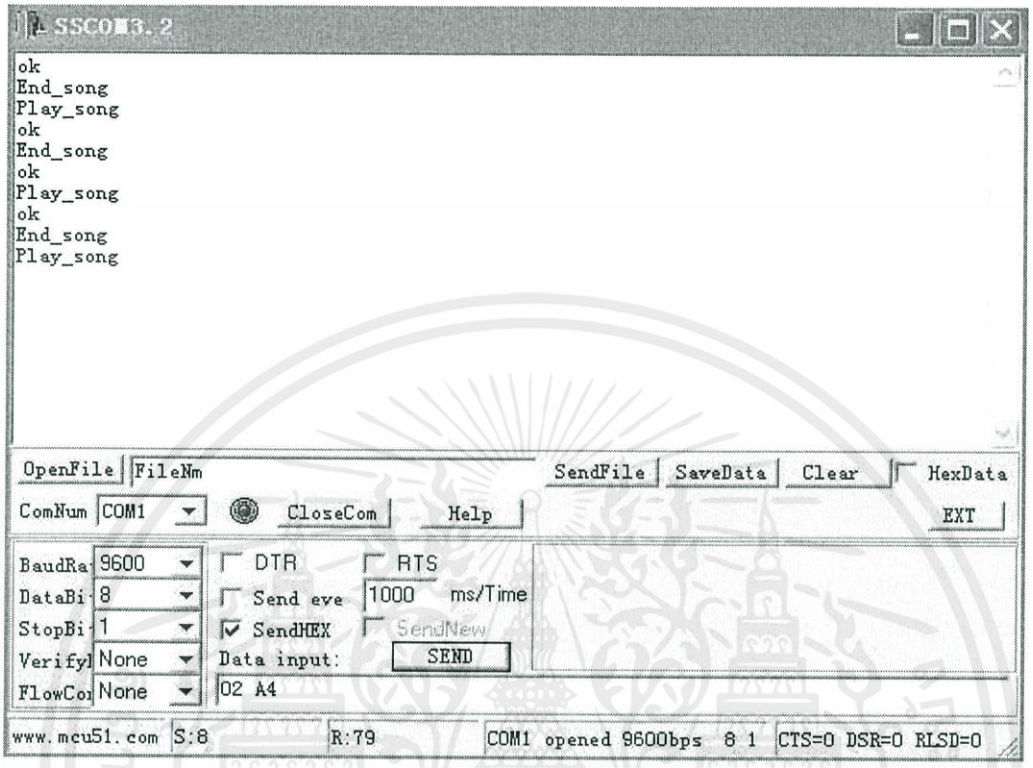
Send : 02 A2 to stop song.



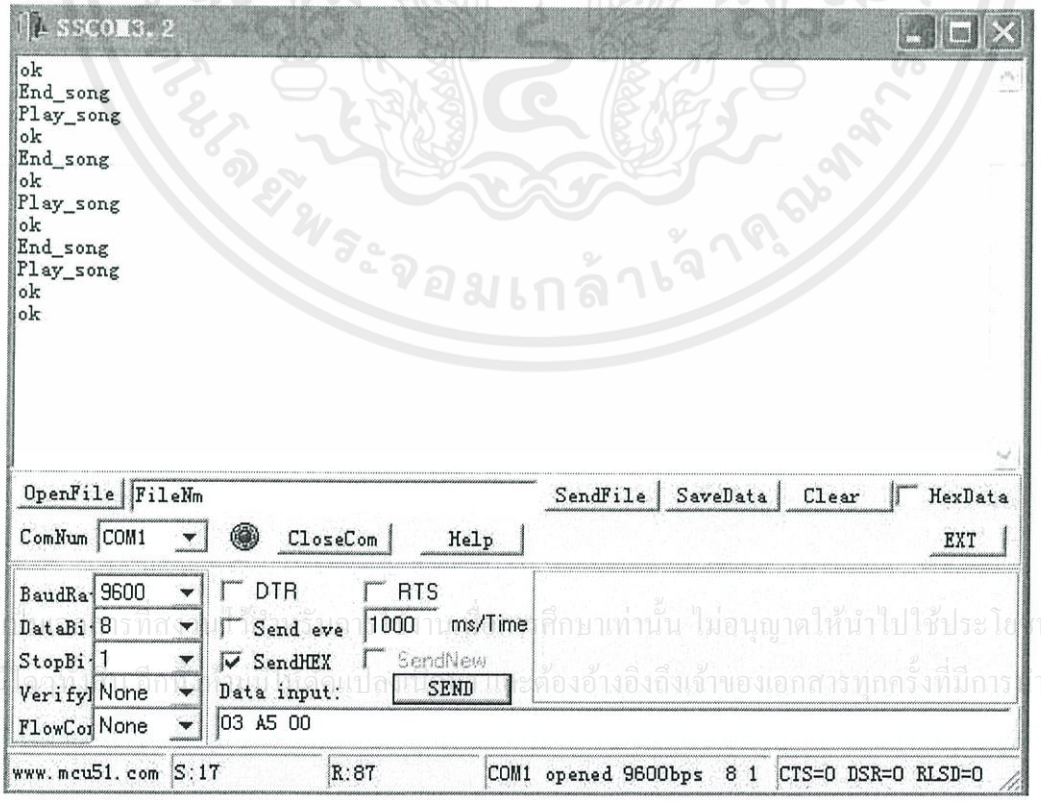
Send : 02 A3 to Play Next song.



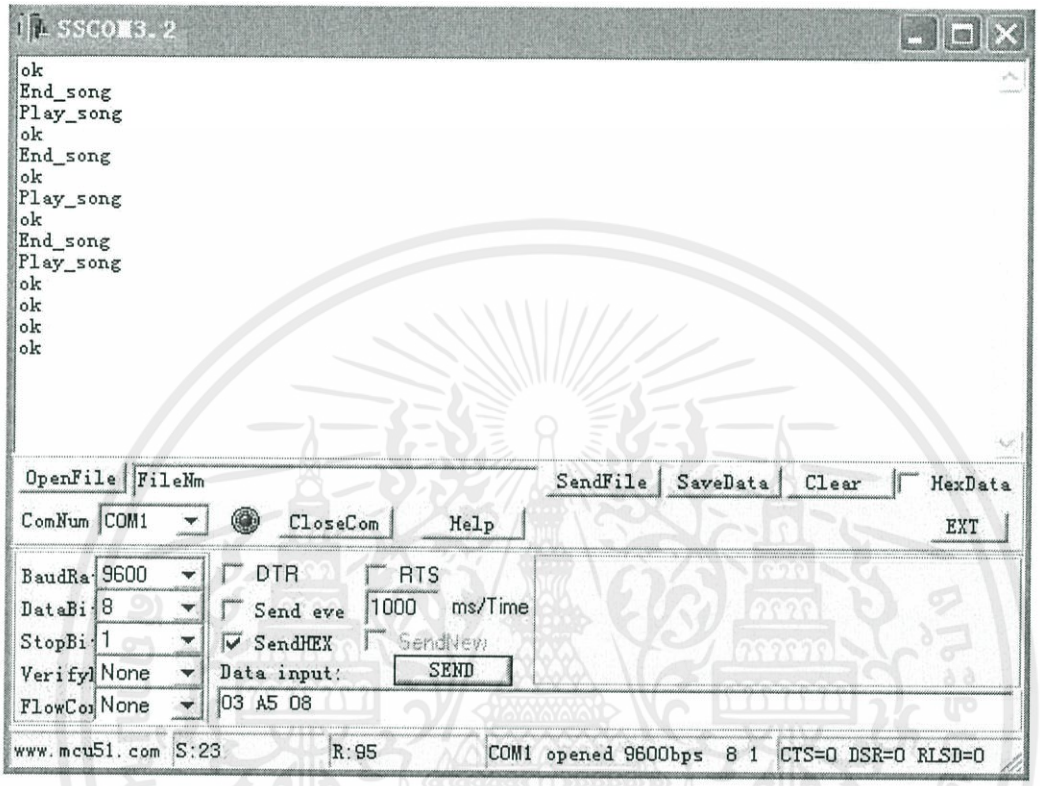
Send : 02 A4 to Play Previous song.



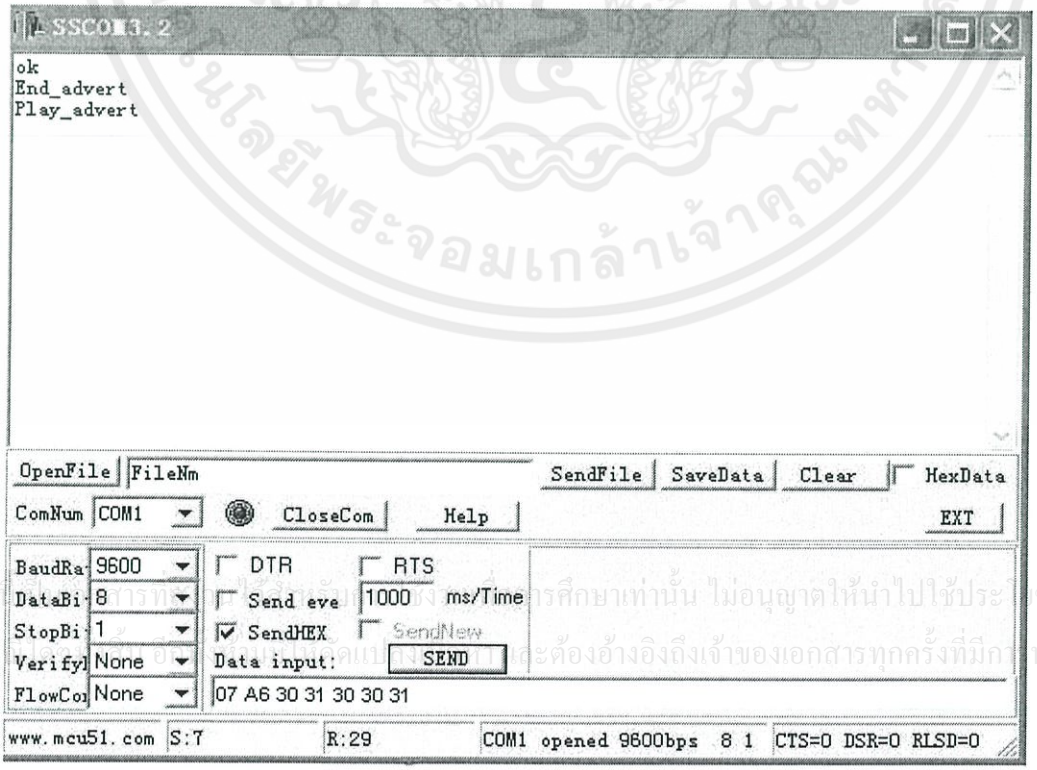
Send: 03 A5 00 for minimum volume (mute)



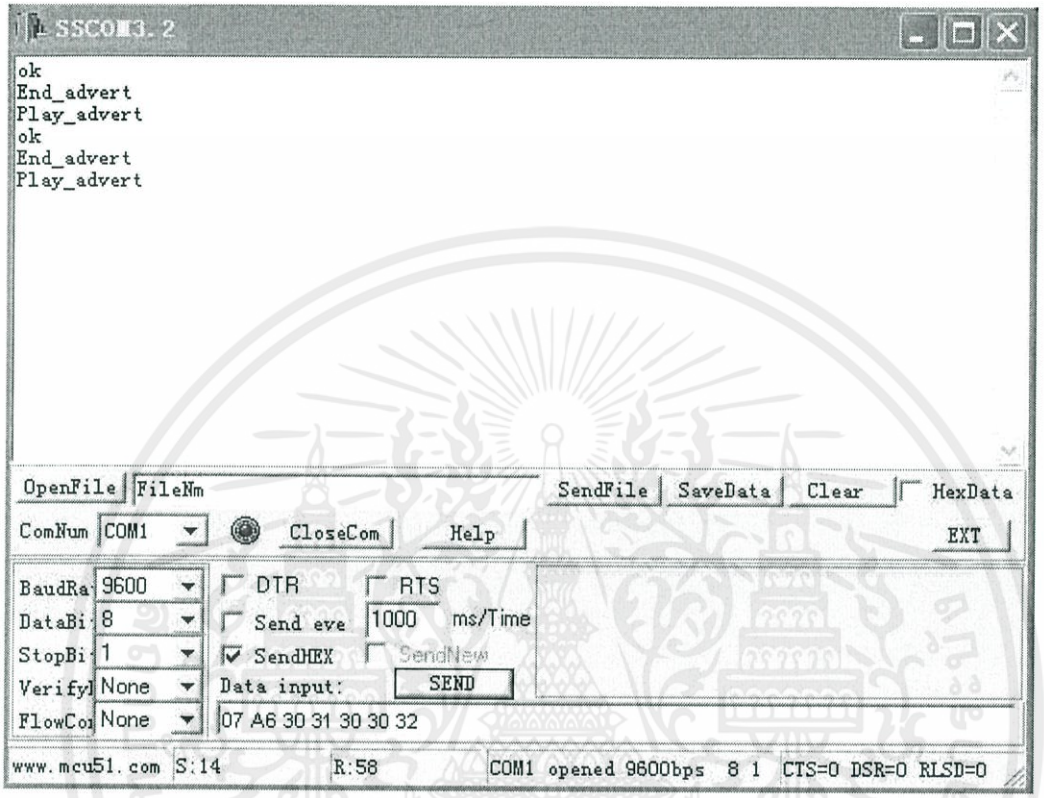
Send: 03 A5 00 for maximum volume , volume can be adjustable from 00 ,01,02,.....08 for SONG or ADVERT



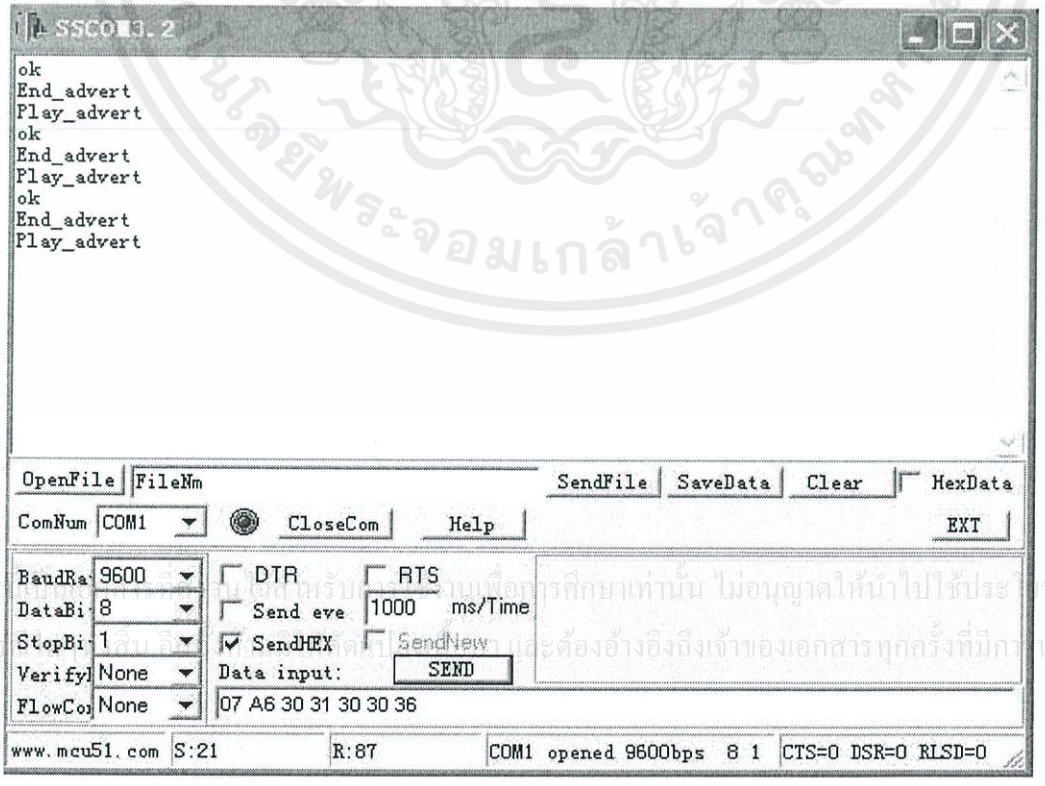
Send :07 A6 30 31 30 30 31 to play ad. (001.mp3 in ADVERT01)



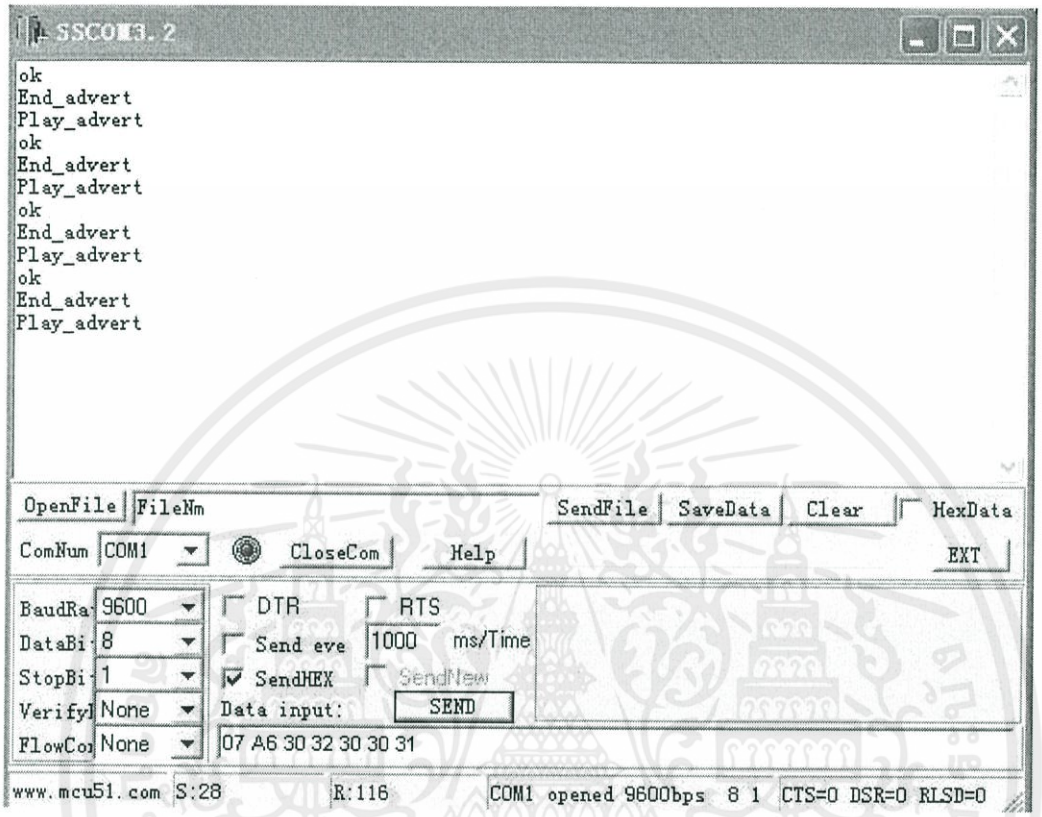
Send :07 A6 30 31 30 30 32 to play ad. (002.mp3 in ADVERT01)



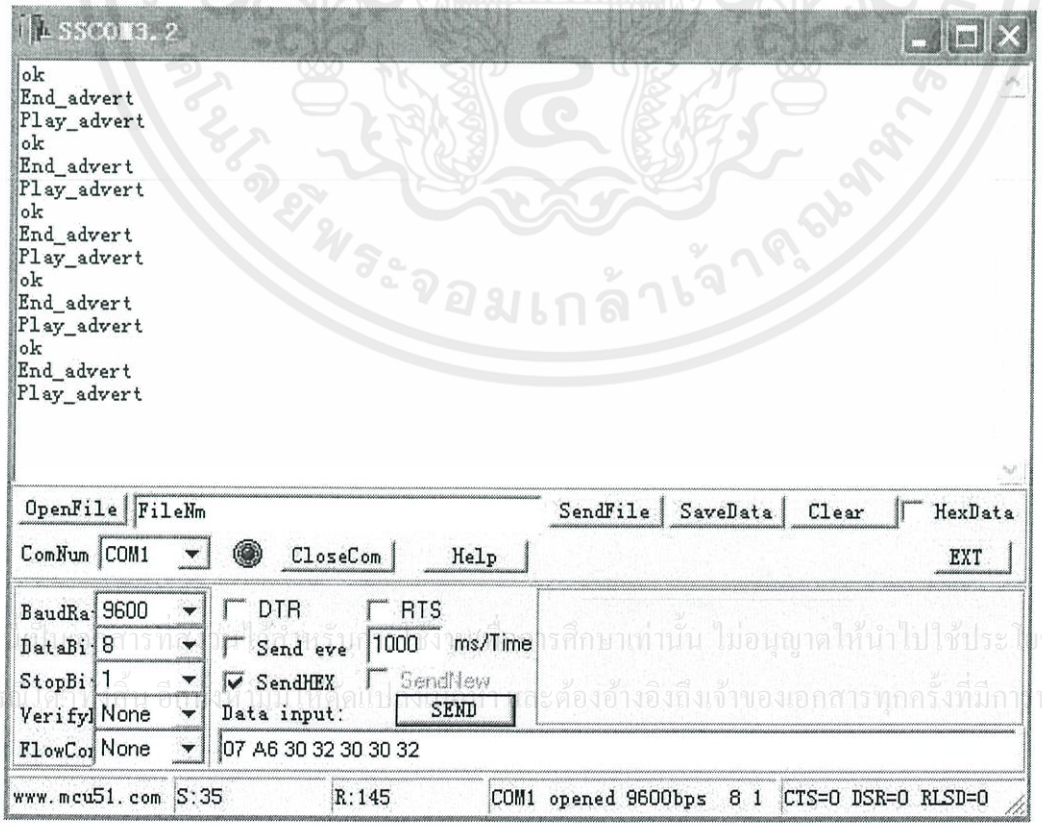
Send :07 A6 30 31 30 30 36 to play ad. (006.mp3 in ADVERT01)



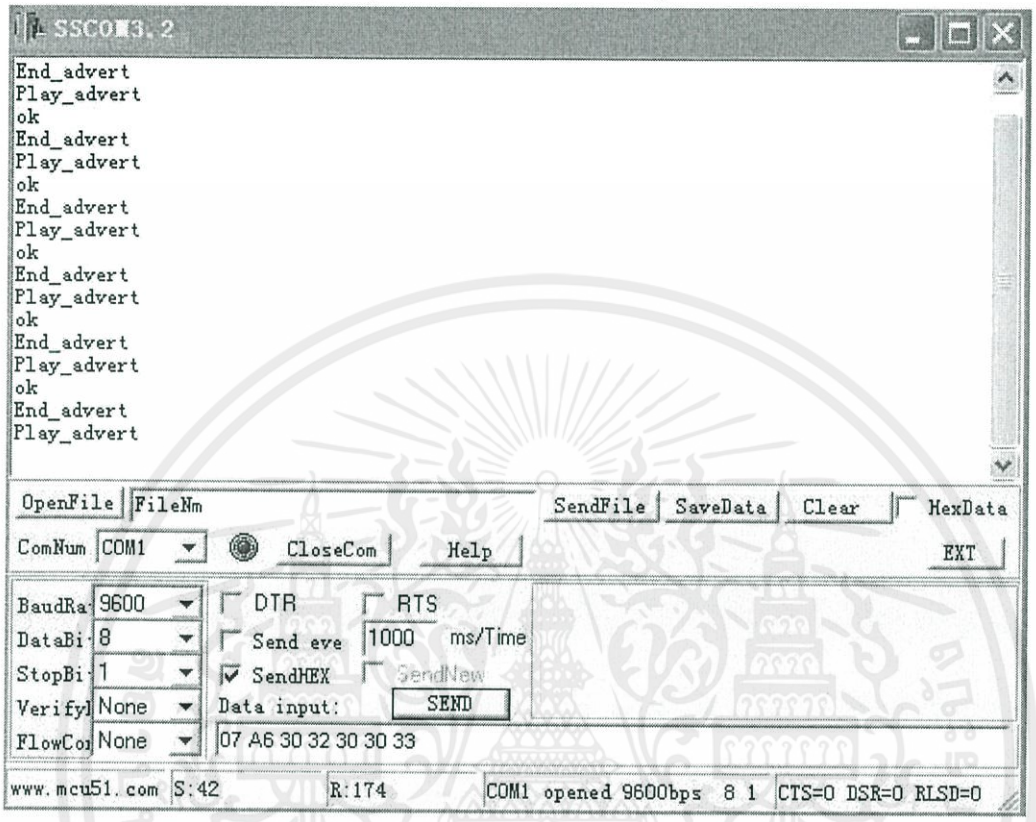
Send :07 A6 30 32 30 30 31 to play ad. (001.mp3 in ADVERT02)



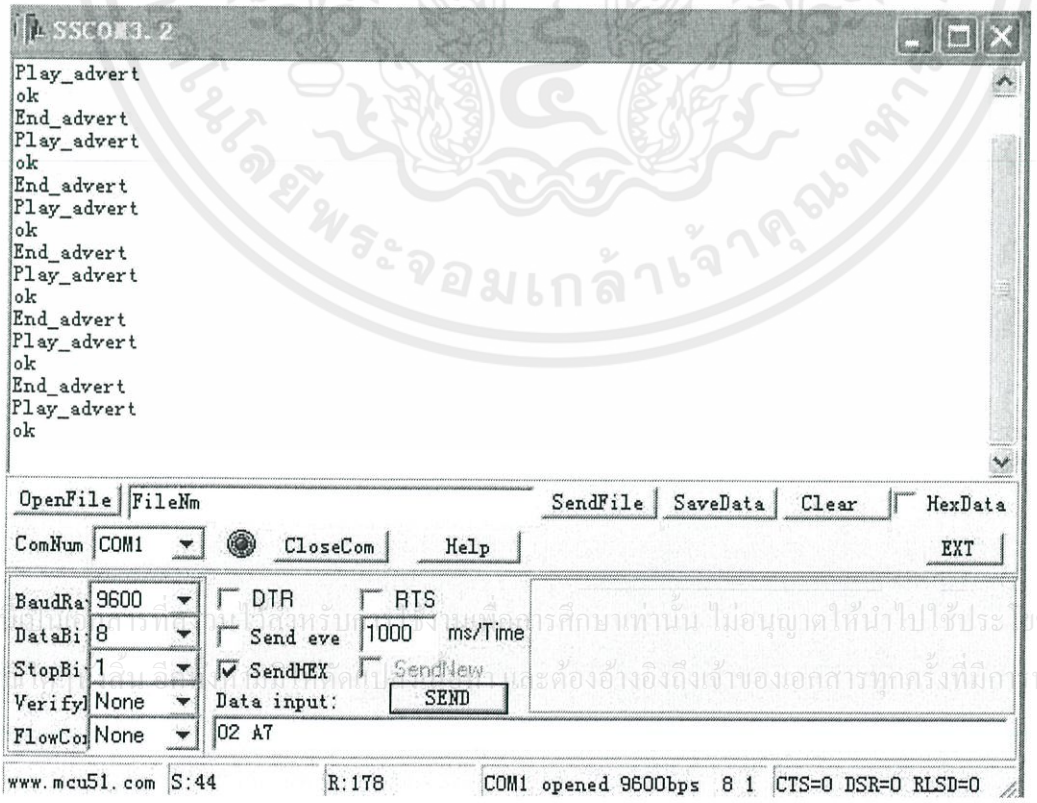
Send :07 A6 30 32 30 30 32 to play ad. (002.mp3 in ADVERT02)



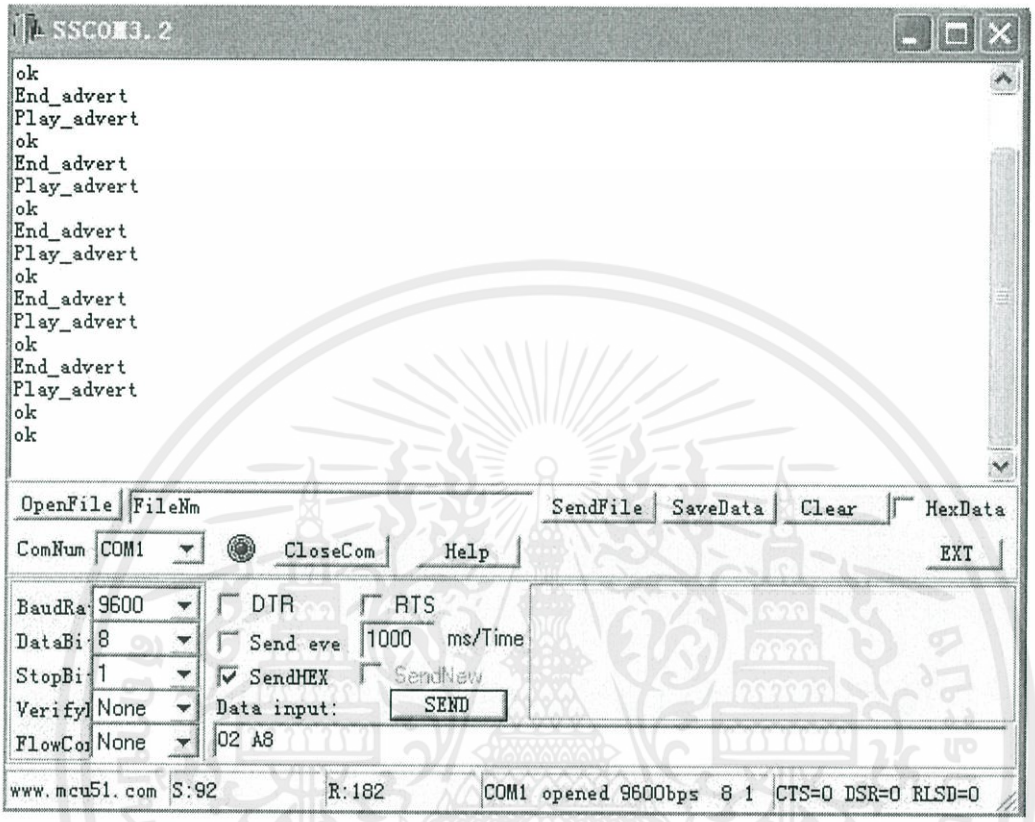
Send :07 A6 30 32 30 30 33 to play ad. (003.mp3 in ADVERT02)



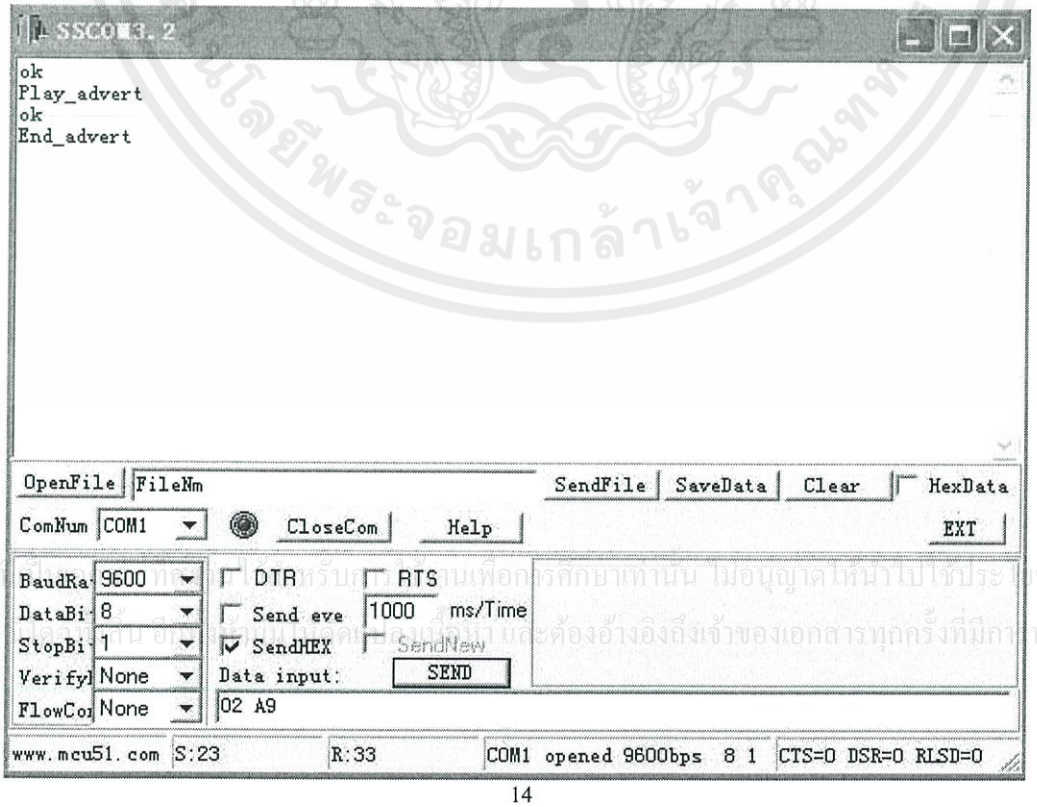
Send :02 A7 to pause ad.



Send :02 A8 to continue ad.



Send :02 A9 to stop ad.



V. Versions

Date	Version	Description
25th Jan.,2007	V1.0	Chinese Original version
14th Jun.,2008	V1.1	English version
1st Oct.,2009	V1.2	English Revised version

VI.Contact information

Tenda Electronics Limited
Add: No.15 Street,Guang Cong North Road,Taiping Economic and Technological

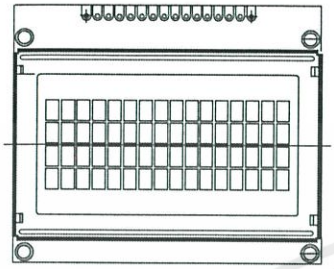
Development Zone,Guangzhou,China

Contact Person : Mr Keith / Miss Lynn / Miss Lisa

Tel: 86-20-22100510 Fax:86-20-37921106

Email:tendaelectronic@gmail.com web:www.tendaelectronics.com

16 x 4 Character LCD



FEATURES

- 5 x 8 dots includes cursor
- Built - in controller (KS 0066 or Equivalent)
- + 5V power supply (Also available for + 3V)
- 1/16 duty cycle
- B/L to be driven by pin 1, pin 2, or pin 15, pin 16 or A and K
- N.V. optional for + 3V power supply

MECHANICAL DATA

ITEM	STANDARD VALUE	UNIT
Module Dimension	70.6 x 60.0	mm
Viewing Area	60.0 x 32.6	mm
Mounting Hole	65.6 x 50.0	mm
Character Size	2.95 x 4.75	mm

ABSOLUTE MAXIMUM RATING

ITEM	SYMBOL	STANDARD VALUE			UNIT
		MIN.	TYP.	MAX.	
Power Supply	VDD-VSS	- 0.3	–	7.0	V
Input Voltage	VI	- 0.3	–	VDD	V

NOTE: VSS = 0 Volt, VDD = 5.0 Volt

ELECTRICAL SPECIFICATIONS

ITEM	SYMBOL	CONDITION	STANDARD VALUE			UNIT
			MIN.	TYP.	MAX.	
Input Voltage	VDD	VDD = + 5V	4.7	5.0	5.3	V
		VDD = + 3V	2.7	3.0	5.3	V
Supply Current	IDD	VDD = + 5V	–	1.65	–	mA
Recommended LC Driving Voltage for Normal Temp. Version Module	VDD - V0	- 20 °C	5.0	5.1	5.7	V
		0°C	4.6	4.8	5.2	
		25°C	4.1	4.5	4.7	
		50°C	3.9	4.2	4.5	
		70°C	3.7	3.9	4.3	
EL Power Supply Current	IEL	Vel = 110VAC; 400Hz	–	–	5.0	mA

DISPLAY CHARACTER ADDRESS CODE:

Display Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DD RAM Address	00	01														0F
DD RAM Address	40	41														4F
DD RAM Address	10	11														1F
DD RAM Address	50	51														5F

LCD-016M004B

Vishay

16 x 4 Character LCD

PIN NUMBER	SYMBOL	FUNCTION
1	Vss	GND
2	Vdd	+ 3V or + 5V
3	Vo	Contrast Adjustment
4	RS	H/L Register Select Signal
5	R/$\overline{W}$	H/L Read/Write Signal
6	E	H →L Enable Signal
7	DB0	H/L Data Bus Line
8	DB1	H/L Data Bus Line
9	DB2	H/L Data Bus Line
10	DB3	H/L Data Bus Line
11	DB4	H/L Data Bus Line
12	DB5	H/L Data Bus Line
13	DB6	H/L Data Bus Line
14	DB7	H/L Data Bus Line
15	A/Vee	+ 4.2V for LED (RA = 0Ω)/Negative Voltage Output
16	K	Power Supply for B/L (0V)

DIMENSIONS in millimeters