เครื่องโปรแกรมชุดไอซี

(Gang Programmer)

โดย

นาย รัฐศาสตร์ อัมฤทธิ์

นายรุ่งโรจน์ พงศาพิชญ์

นายวรพัฒน์ ตุลารักษ์

อาจารย์ที่ปรึกษา

รศ.ดร.มนัส สังวรศิลป์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปริญญานิพนธ์ปีการศึกษาที่ 2543

ภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ผู้จัดทำ

1. นายรัฐศาสตร์ อัมฤทธิ์ เลขประจำตัว 40010640
2. นายรุ่งโรจน์ พงศาพิชญ์ เลขประจำตัว 40010645
3. นายวรพัฒน์ ตุลารักษ์ เลขประจำตัว 40010675

รายงานฉบับนี้ได้ผ่านการตรวจสอบโดยอาจารย์ที่ปรึกษาแล้ว

.....อาจารย์ที่ปรึกษา

(รศ.ดร.มนัส สังวรศิลป์)

บทคัดย่อ

นาย รัฐศาสตร์ อัมฤทธิ์ เลขประจำตัว 40010640
นายรุ่งโรจน์ พงศาพิชญ์ เลขประจำตัว 40010645
นายวรพัฒน์ ตุลารักษ์ เลขประจำตัว 40010675
อาจารย์ที่ปรึกษา รศ.ดร. มนัส สังวรศิลป์
ภาคการศึกษาที่ 2 ปีการศึกษา 2543

โครงงาน "เครื่องโปรแกรมไมโครคอนโทรลเลอร์แบบกลุ่ม" นี้ สร้างขึ้นโดยคำนึงถึงการทำงานในอุตสาหกรรมที่จำเป็นต้องมีการโปรแกรมอุปกรณ์จำพวกไมโครคอนโทรเลอร์, เมมโมรี ซึ่งจำเป็นต้องมีการผลิต, ปรับปรุงเป็นจำนวนครั้งละมากๆ ซึ่งจะเสียเวลามากหากจะต้องโปรแกรมไอซีทีละตัว ดังนั้นโครงงานนี้จึงออกแบบให้เครื่องโปรแกรมนี้สามารถทำการโปรแกรมไอซีเป็นจำนวนครั้งละมากกว่า 1 ตัว เพื่อความสะดวกและรวดเร็วในการทำงาน

ABSTRACT

Mr.Rattasart Ammarit ID: 40010640

Mr.Roongroj Pongsapich ID: 40010645

Mr.Worrapat Turalak ID: 40010675

Assoc. Prof. Dr.Manas Sangworasilp (Adviser)

2nd Semestor, Educational Year 2000

In this Project, The Microcontroller Gang Programmer, is designed to serve the industries that have to update/produce large number of microcontroller each time instead of programming the IC one-by-one, We design this Microcontroller Gang Programmer to help them program more than one IC at the same time.

สารบัญ

	หน้า
บทคัดย่อ	i
Abstract	ii
สารบัญ	iv
บทที่ 1 บทนำ	1
บทที่ 2 ทฤษฎีการทำงาน	2
2.1. วิธีการโปรแกรมเมมโมรีแบบแฟลชของตัวไมโครคอนโทรลเลอร์	2
2.2. การใช้งานพอร์ตอนุกรมของคอมพิวเตอร์	5
2.3. ไมโครคอนโทรลเลอร์ MCS-51	9
บทที่ 3 รายละเอียดโครงสร้าง	12
บทที่ 4 การออกแบบโครงสร้าง	14
บทที่ 5 โปรแกรมควบคุมการทำงานวงจร	19
5.1. หน้าที่ของโปรแกรม	19
5.2. ความสามารถของโปรแกรม	19
5.3. ข้อจำกัดของโปรแกรม	20
5.4. การใช้งานโปรแกรม	20
บทที่ 6 การออกแบบโปรแกรม	25
6.1.การสร้างยูนิต MSCommLib_TLB	25
6.2.คอนโทรล MSComm	27
6.3.ยูนิต FileWorkShop	28
6.4.ยูนิต BurnIT	29
บทที่ 7 การทดลอง	34
บทที่ 8สรุปและวิจารณ์ผลการทดลอง	35
ภาคผนวก	

บทที่ 1
บทนำ

การเขียนโปรแกรมให้แก่ไอซีจำเป็นต้องมีเครื่องโปรแกรม ซึ่งส่วนใหญ่จะเป็นการโปรแกรมทีละตัว สำหรับโรงงานอุตสาหกรรมที่ต้องโปรแกรมแก่ไอซีจำนวนมาก หากใช้เครื่องโปรแกรมที่สามารถโปรแกรมได้เพียงครั้งละตัวนั้นจะทำให้เสียเวลาในการผลิตอย่างมาก จึงจำเป็นต้องใช้เครื่องโปรแกรมชุดไอซี (GANG PROGRAMER) ช่วยเพิ่มความเร็วในการผลิต เนื่องจากสามารถเขียนโปรแกรมให้แก่ไอซีได้พร้อมกันทีละหลายๆ ตัว ไม่ต้องเสียเวลาในการเปลี่ยนตัวไอซีและโปรแกรมลงทุกๆ ครั้ง

โดยทั่วไปเครื่องโปรแกรมที่ใช้กันในโรงงาน เครื่องโปรแกรมแบบเป็นชุดที่ใช้กันจะสามารถโปรแกรมได้เพียง 1 เบอร์ไอซีเท่านั้น ซึ่งจะต้องเปลืองงบประมาณในการซื้อชุดโปรแกรมไอซีเบอร์อื่นๆ เพิ่มอีก ถ้าหากต้องการโปรแกรมไอซีหลายๆ เบอร์ ดังนั้นเครื่องโปรแกรมไอซีที่ได้จัดทำขึ้นนี้ จะเป็นเครื่องโปรแกรมที่มีคุณสมบัติในการโปรแกรมไอซีได้หลายเบอร์และสามารถโปรแกรมได้ทีละหลายๆ ตัวด้วย

ขอบเขตของโครงงานเครื่องโปรแกรมชุดไอซี

1. สามารถโปรแกรมไอซีเบอร์
   1.1 AT89C51
   1.2 AT89C52
   1.3 AT89S53
   1.4 AT89S8252
   1.5 AT89C1051
   1.6 AT89C2051
   1.7 AT89C4051
2. ใช้เครื่องคอมพิวเตอร์ส่วนบุคคลส่งข้อมูลให้แก่เครื่องโปรแกรม และรับข้อมูลจากเครื่องโปรแกรมได้
3. โปรแกรมไอซีได้ครั้งละ 6 ตัวพร้อมกัน
4. สามารถตรวจสอบความถูกต้องของการโปรแกรมได้
5. อ่านโปรแกรมจากไอซี (ที่ไม่มีการเขียนล็อกไว้) เซฟเป็นไฟล์ HEX เก็บไว้บนเครื่องคอมพิวเตอร์ได้

บทที่ 2
ทฤษฎีการทำงาน

ทฤษฎีที่จำเป็นในการทำงานของวงจร Gang Programmer ที่ควรทราบคือ

1. วิธีโปรแกรมเมมโมรีแบบ Flash ของตัวไมโครคอนโทรลเลอร์
2. การใช้งานพอร์ตอนุกรมของคอมพิวเตอร์
3. การใช้งานไมโครคอนโทรลเลอร์ MCS-51

2.1 วิธีโปรแกรมเมมโมรีแบบ flash ของตัวไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์เบอร์ AT89C51, AT89C52, AT89S53, AT89S8252 นั้นจะมีเมมโมรีบรรจุอยู่ภายใน ซึ่งพร้อมจะให้ใช้งานได้ วิธีการโปรแกรมแฟลชเมมโมรีนั้นมีอยู่ 2 วิธี นั่นคือการโปรแกรมแบบ High Voltage และ แบบ Low Voltage ซึ่งชนิดของการโปรแกรมจะขึ้นอยู่กับตัวไมครคอนโทรเลอร์เอง ซึ่งในที่นี้จะขอกล่าวถึงแต่การโปรแกรมแบบ High Voltage

ขั้นตอนการโปรแกรม

ก่อนที่จะทำการโปรแกรมลงสู่ตัวไมโครคอนโทรลเลอร์นั้น สัญญาณตำแหน่ง (Address), ข้อมูล (Data) และสัญญาณควบคุม ต้องถูกตั้งให้ตรงกับแต่ละโหมดของการโปรแกรม ดังรูปที่ 2.1 และรูป 2.2 โดยขั้นตอนการโปรแกรมจะเป็นไปตามนี้

1. ป้อนค่าตำแหน่งของเมมโมรีที่ต้องการที่สายสัญญาณบอกตำแหน่ง (Address Lines)
2. ป้อนข้อมูลที่ต้องการลงไปยังสายข้อมูล (Data Lines)
3. ป้อนชุดคำสั่งไปยังสายสัญญาณควบคุม (Control Lines)
4. เพิ่มค่าสาย $\overline{EA}/V_{pp}$ เป็น 12 โวลต์ สำหรับการโปรแกรมแบบ High Voltage
5. ป้อนพัลส์ไปตามสาย ALE/$\overline{PROG}$ หนึ่งครั้ง ในการโปรแกรมแต่ละไบต์หรือการโปรแกรมแบบ Lock BIT ช่วงเวลาการเขียน (Write Cycle) ของแต่ละไบต์จะใช้เวลาไม่เกิน 1.5 ms จากนั้นเลื่อนตำแหน่งไปยังตำแหน่งถัดไปแล้วกลับไปทำตามขั้นตอนที่ 1 จนครบทุกตำแหน่ง

การล้างข้อมูลภายในแฟลชเมมโมรี

สามารถล้างข้อมูลภายในเมมโมรีได้ด้วยการป้อนชุดคำสั่งตามรูป 2.1 หรือ 2.2 และดึงให้สาย ALE/$\overline{PROG}$ ลงสถานะ Low นาน 10 ms

การตรวจสอบข้อมูลที่ได้เขียนไปแล้ว

ถ้าหากไม่ได้ทำการโปรแกรมโดยทำการล็อกบิต-1 หรือ บิต-2 แล้ว ข้อมูลที่เขียนลงไปนั้น จะยังสามารถทำการอ่านกลับออกมา เพื่อตรวจสอบได้ผ่านทางสาย สัญญาณตำแหน่งและสายข้อมูล

Flash Programming Modes

Mode		RST	PSEN	ALE/$\overline{PROG}$	$\overline{EA}/V_{PP}$	P2.6	P2.7	P3.6	P3.7
Write Code Data		H	L		H/12V	L	H	H	H
Read Code Data		H	L	H	H	L	L	H	H
Write Lock	Bit - 1	H	L		H/12V	H	H	H	H
	Bit - 2	H	L		H/12V	H	H	L	L
	Bit - 3	H	L		H/12V	H	L	H	L
Chip Erase		H	L	(1)	H/12V	H	L	L	L
Read Signature Byte		H	L	H	H	L	L	L	L

Note: 1. Chip Erase requires a 10 ms $\overline{PROG}$ pulse.

รูปที่ 2.1 รูปแบบการโปรแกรมแฟลชเมมโมรีของ AT89C51 และ AT89C52

Flash and EEPROM Parallel Programming Modes

Mode		RST	PSEN	ALE/$\overline{PROG}$	$\overline{EA}/V_{PP}$	P2.6	P2.7	P3.6	P3.7	Data I/O P0.7:0	Address P2.5:0 P1.7:0
Serial Prog. Modes		H	H[1]	H[1]	X						
Chip Erase		H	L	(2)	12V	H	L	L	L	X	X
Write (10K bytes) Memory		H	L		12V	L	H	H	H	DIN	ADDR
Read (10K bytes) Memory		H	L	H	12V	L	L	H	H	DOUT	ADDR
Write Lock Bits		H	L		12V	H	L	H	L	DIN	X
	Bit - 1									P0.7 = 0	X
	Bit - 2									P0.6 = 0	X
	Bit - 3									P0.5 = 0	X
Read Lock Bits:		H	L	H	12V	H	H	L	L	DOUT	X
	Bit - 1									@ P0.2	X
	Bit - 2									@ P0.1	X
	Bit - 3									@ P0.0	X
Read Atmel Code		H	L	H	12V	L	L	L	L	DOUT	30H
Read Device Code		H	L	H	12V	L	L	L	L	DOUT	31H
Serial Prog. Enable		H	L	(2)	12V	L	H	L	H	P0.0 = 0	X
Serial Prog. Disable		H	L	(2)	12V	L	H	L	H	P0.0 = 1	X
Read Serial Prog. Fuse		H	L	H	12V	H	H	L	H	@ P0.0	X

Notes: 1. "h" = weakly pulled "High" internally.
2. Chip Erase and Serial Programming Fuse require a 10 ms $\overline{PROG}$ pulse. Chip Erase needs to be performed first before reprogramming any byte with a content other than FFH
3. P3.4 is pulled Low during programming to indicate RDY/$\overline{BSY}$
4. "X" = don't care

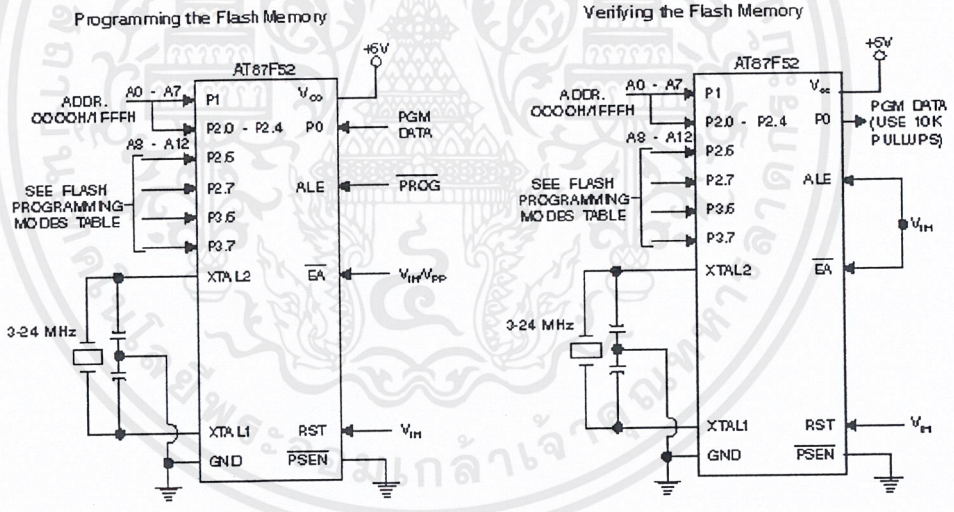
รูปที่ 2.2 รูปแบบการโปรแกรมแฟลชเมมโมรีของ AT89S53 และ AT89S8252

Flash Programming Modes

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

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

รูปที่ 2.3 รูปแบบการโปรแกรมแฟลชเมมโมรีของ AT89C1051, AT89C2051 และ AT89C4051



รูปที่ 2.4 การเชื่อมต่อสายสัญญาณต่างๆ ของ AT89S53 และ AT89S8252

Figure 3. Programming the Flash Memory

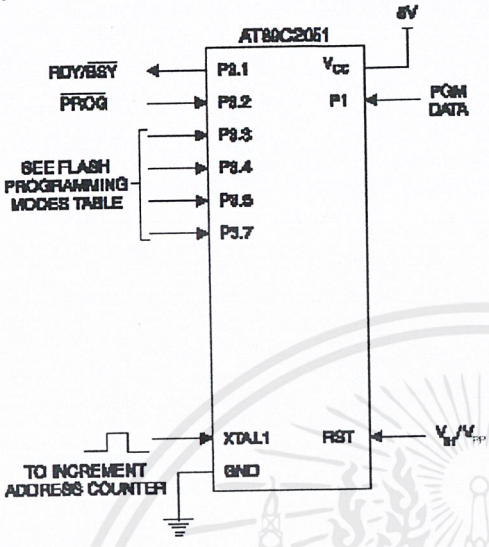
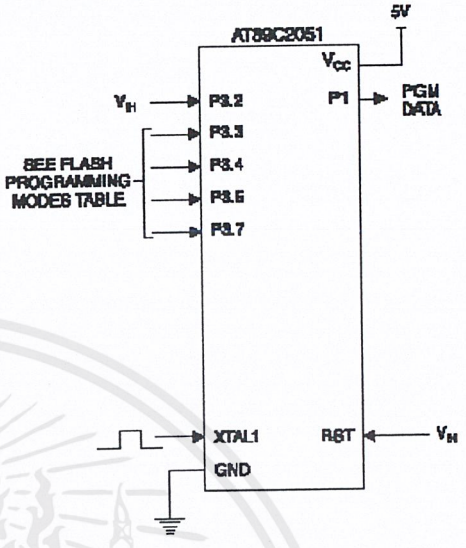
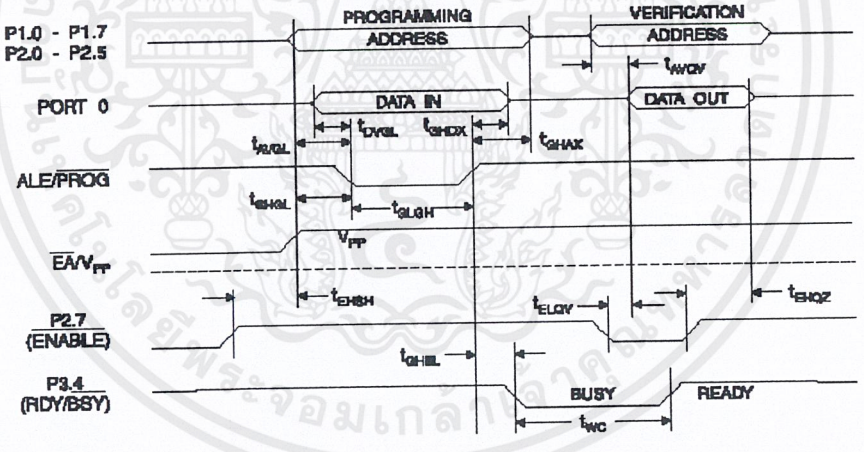


Figure 4. Verifying the Flash Memory



รูปที่ 2.5 การเชื่อมต่อสายสัญญาณต่างๆ ของ AT89C1051, AT89C2051 และ AT89C4051



รูปที่ 2.6 Timing Diagram และรูปคลื่นของการโปรแกรมแบบ High Voltage

2.2 การใช้งานพอร์ตอนุกรมของคอมพิวเตอร์

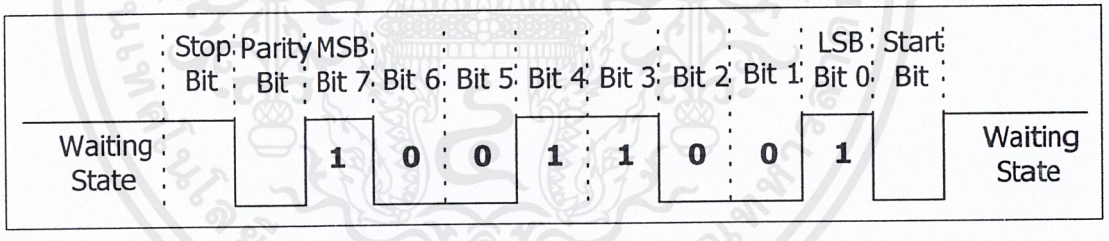
การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัสคือการรับและส่งข้อมูลภายในสายโดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วยเหมือนกับการส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับและภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอดเรต (Baudrate) มีหน่วยเป็น บิตต่อวินาที (Bit per second : bps)

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนด้วยกันคือ

1. บิตเริ่มต้น (Start Bit) ขนาด 1 บิต
2. บิตข้อมูลแบบอนุกรม (Data Bit) ขนาด 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Bit) ขนาด 1 บิต หรือไม่มี
4. บิตปิดท้าย (Stop Bit) ขนาด 1, 1.5 หรือ 2 บิต

รูปที่ 2.7 แสดงรูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส ซึ่งเมื่อไม่มีข้อมูลที่จะส่ง ขา Data จะมีสถานะลอจิก "1" ซึ่งจะเรียกสถานะนี้ว่าสถานะหยุดรอ (Waiting State) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขา Data มีลอจิก "0" ด้วยช่วงเวลา 1 บิต ซึ่งจะเรียกบิตนี้ว่าบิตเริ่มต้น จากนั้นบิตข้อมูลจะถูกส่งออกไป โดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจมีขนาด 5, 6, 7 หรือ 8 บิตก็ได้ จากนั้นจะตามด้วยบิตพาริตี้ เพื่อใช้ตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่จะส่งคือบิตปิดท้าย ซึ่งจะให้ขา Data มีสถานะลอจิก 1 อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต, 1.5 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดการส่งข้อมูลแล้ว



รูปที่ 2.7 รูปแบบอย่างง่ายที่สุดของข้อมูลแบบอะซิงโครนัส

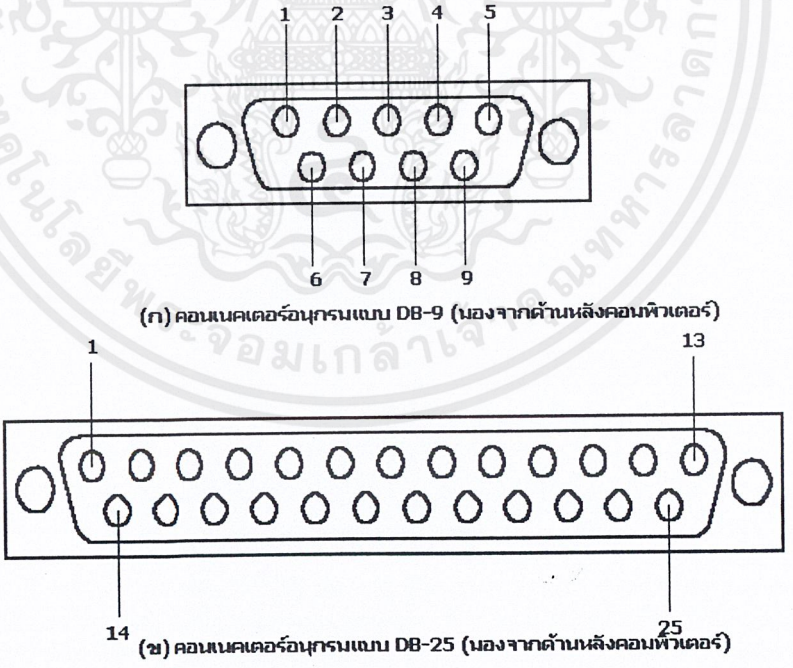
อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับและส่งข้อมูลแบบอะซิงโครนัสเรียกว่า Universal Asynchronous Receiver/Transmitter หรือ UART อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับและส่งข้อมูล บอดเรตมาตรฐานที่ใช้สำหรับพอร์ตอนุกรม RS-232 ได้แก่ 110, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที และมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดบอดเรตได้สูงถึง 115200 บิตต่อวินาที เนื่องจากบอดเรตคือจำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที เช่น ถ้าใช้บอดเรตในการส่งข้อมูลเท่ากับ 9600 bps และไม่มีการใช้พาริตี้บิต ซึ่งจะทำให้ข้อมูล 1 ไบต์จะใช้บิตในการส่งเท่ากับ 10 บิต (ข้อมูล 8 บิต + 1 Start Bit + 1 Stop Bit) ทำให้สามารถรับส่งข้อมูลได้ที่

ความเร็วเท่ากับ 960 ไบต์ต่อวินาที แต่ถ้ามีการใช้พาริตี้ในการส่งด้วย ก็จะสามารถส่งข้อมูลได้ที่ความเร็วเหลือ 872 ไบต์ต่อวินาที

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ UART เบอร์ 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT จะใช้ UART เบอร์ 8250 UART ชิปเหล่านี้มีระดับแรงดันเป็นแบบทีทีแอล (0 และ +5 โวลต์) แต่เพื่อให้มีระดับแรงดันเป็นไปตามมาตรฐาน RS-232 และเพื่อให้การรับส่งข้อมูลสามารถทำได้ที่ระยะทางไกลมากขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้น โดยลอจิก "0" มีระดับแรงดัน +3V ถึง +12V ในขณะที่ลอจิก "1" จะมีระดับแรงดัน –3V ถึง –12V

คอนเนคเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเนคเตอร์แบบ DB-25 และ DB-9 ซึ่งคอนเนคเตอร์แบบ DB-25 นั้นก็มีการต่อขาใช้งานเพียง 9 สาย เช่นเดียวกับ DB-9 เนื่องจากขาอื่นๆ ที่เคยใช้งานในอดีตนั้น ปัจจุบันมีการใช้งานไม่มากนัก จึงถูกยกเลิกไป โดยแสดงรูปร่างและตำแหน่งขาดังในรูป 2.8



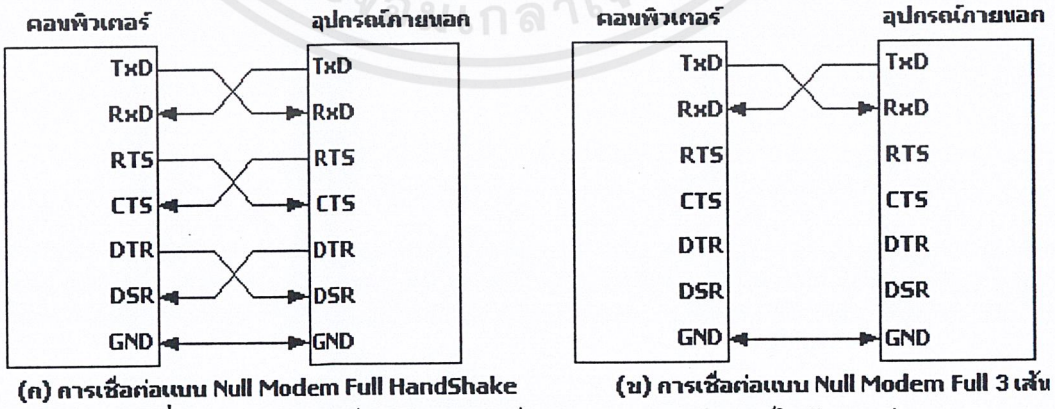
รูปที่ 2.8 คอนเนคเตอร์อนุกรมตามมาตรฐาน RS-232

Connector DB-9	Connector DB-25	Signal Name	Direction
1	8	Data Carrier Detect : DCD	Input
2	3	Received Data : RxD	Input
3	2	Transmitted Data : TxD	Output
4	20	Data Terminal Ready : DTR	Output
5	7	Signal Ground : GND	-
6	6	Data Set Ready : DSR	Input
7	4	Request To Send : RTS	Output
8	5	Clear To Send : CTS	Input
9	22	Ring Indicator : RI	Input

ตาราง 2.1 การจัดขาของคอนเน็กเตอร์พอร์ตอนุกรมตามมาตรฐาน RS-232 ทั้งแบบ DB-9 และ DB-25

สำหรับการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกแสดงดังรูปที่ 2.9 ลูกศรในรูปแสดงถึงทิศทางของข้อมูล รูปที่ 2.9(ก) เป็นการเชื่อมต่อแบบ Null Modem หรือการเชื่อมต่อโดยตรงโดยไม่ต้องผ่านโมเด็ม โดยมีการตรวจสอบหรือแฮนด์เช็กเต็มรูปแบบ ส่วนในรูปที่ 2.9(ข) เป็นการเชื่อมต่อแบบ Null Modem ในลักษณะที่ใช้สายสัญญาณเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูล และเส้นสุดท้ายเป็นกราวด์ ซึ่งโครงงานนี้จะใช้การต่อแบบที่สองซึ่งใช้สาย 3 เส้น สำหรับหน้าที่ของสายสัญญาณมีดังนี้

- Received Data : RD หรือ RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์ โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์บัฟเฟอร์รับข้อมูล
- Transmitted Data : TD หรือ TxD ขานี้ใช้เพื่อส่งข้อมูลออกจากคอมพิวเตอร์ โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลออกไป
- Signal Ground : GND ขากราวด์ของระบบ



รูปที่ 2.9 การต่ออุปกรณ์ภายนอกกับพอร์ตอนุกรมของคอมพิวเตอร์ในลักษณะต่างๆ

2.3 ไมโครคอนโทรลเลอร์ mcs-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 เป็นอุปกรณ์ที่ออกแบบมาสนองความต้องการของผู้ใช้คือมีสายอินพุทเอาท์พุทภายในตัวเอง พอร์ทอินพุทและพอร์ทเอาท์พุทบัฟเฟอร์อินเตอร์เฟสและสายควบคุมอื่นๆที่ใช้สำหรับแยกข้อมูลกับปแอดเดรส และยังมีชุดคำสั่งเพิ่มขึ้นเป็นพิเศษเพื่อจัดการข้อมูล และนอกจากนี้ยังมีวงจรตั้งเวลาและวงจรนับด้วย MCS-51 มีอยู่ด้วยกันหลายเบอร์แต่ละเบอร์ก็มีความสามารถพิเศษแตกต่างกันไป ผู้ใช้สามารถดูได้จากคู่มือของ MCS-51 และเลือกใช้ได้ตามสะดวก

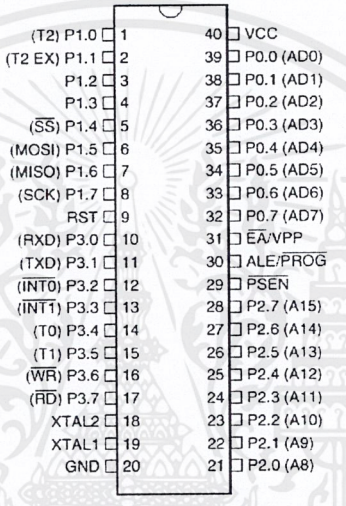
2.3.1 คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ MCS-51

- คุณสมบัติทั่วไปที่สำคัญของไมโครคอนโทรลเลอร์ ตระกูล MCS-51 มีดังนี้
- เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิท
- มีวงจรออสซิเลเตอร์และวงจรผลิตสัญญาณนาฬิกาภายในตัว
- มีขาสัญญาณอินพุทและเอาท์พุท จำนวน 32 บิท
- สามารถเชื่อมต่อหน่วยความจำข้อมูลภายนอก (External Data Memory) โดยอ้างตำแหน่งแอดเดรสได้ถึง 64 กิโลไบต์
- สามารถเชื่อมต่อหน่วยความจำโปรแกรมภายนอก (External Program Memory) โดยอ้างตำแหน่งแอดเดรสได้ถึง 64 กิโลไบต์
- มีหน่วยความจำโปรแกรมภายในตัว (On-chip Program Memory) ขนาด 4 กิโลไบต์ โดยเฉพาะเบอร์ 8052 จะมีหน่วยความจำในส่วนนี้ถึง 8 กิโลไบต์ สำหรับเบอร์ 8031 และ M8032 จะไม่มีหน่วยความจำในส่วนนี้
- มีหน่วยความจำข้อมูลภายในตัว (On-chip Data Memory) ขนาด 128 กิโลไบต์ โดยเฉพาะเบอร์ 8032 และ 8052 จะมีหน่วยความจำในส่วนนี้ถึง 256 ไบต์
- หน่วยความจำข้อมูลภายในบางส่วน สามารถเข้าถึงข้อมูลระดับบิทได้ด้วย ทำให้การควบคุมหรือการตรวจสอบสถานะบิททำได้ง่าย ส่งผลให้การเขียนโปรแกรมทำได้ง่ายมากขึ้น
- มีไทม์เมอร์/เคาน์เตอร์ (Timer/Counters) ขนาด 16 บิท จำนวน 2 ตัว โดยเฉพาะเบอร์ 8032 หรือ 8052 จะมีไทม์เมอร์/เคาน์เตอร์จำนวน 3 ตัว
- การอินเตอร์รัปต์ได้จาก 6 แหล่งกำเนิด โดยการอินเตอร์รัปต์ยังสามารถจัดระดับความสำคัญได้เป็น 2 ระดับ
- มีพอร์ทสื่อสารอนุกรมภายในตัวเอง ซึ่งทำงานเป็นแบบฟูลเพล็กซ์ (Full Duplex)
- มีคำสั่งในการคำนวณทางคณิตศาสตร์และทางตรรกศาสตร์

- คำสั่งโดยส่วนใหญ่ใช้เวลาการทำงานเพียง 1 ไมโครวินาทีเมื่อใช้คริสตอลความถี่ 12 เมกะเฮิร์ท
- ต้องการแหล่งจ่ายไฟ 5 โวลต์ เพียงชุดเดียว

2.3.2 โครงสร้างของไมโครคอนโทรลเลอร์ ตระกูล mcs-51

โครงสร้างของไมโครคอนโทรลเลอร์ ตระกูล MCS-51 ทุกเบอร์จะมีตำแหน่งขาพื้นฐานที่เหมือนกัน สำหรับตัวอย่างนี้จะใช้เบอร์ 8031 ซึ่งมีโครงสร้างภายนอก ดังรูปที่ 2.10



รูปที่ 2.10 แสดงสถาปัตยกรรมภายนอกและการจัดตำแหน่งขาต่างๆของไมโครคอนโทรลเลอร์ 8031

ตำแหน่งหน้าที่การใช้งานของแต่ละขาของไมโครคอนโทรลเลอร์ 8031 มีดังนี้

- **ขาพอร์ท 0** (Port 0)

มี 8 ขาได้แก่ขา P0.0 – P0.7 เป็นขาพอร์ทอินพุทเอาท์พุทแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นอินพุทพอร์ทต้องทำการเขียนค่า 1 ไปยังแต่ละบิทของพอร์ท เพื่อกำหนดให้ขาพอร์ทเหล่านั้นอยู่ในสถานะปล่อยลอย ซึ่งในสถานะนี้เองที่สามารถนำมาใช้เป็นพอร์ทอินพุทอิมพีแดนซ์สูงได้ นอกจากพอร์ทนี้จะใช้งานเป็นพอร์ทอินพุทเอาท์พุทแล้วมันยังถูกใช้งานในการติดต่อกับหน่วยความจำภายนอกด้วย โดยทำหน้าที่ในการกำหนดตำแหน่งแอดเดรสไบท์ต่ำ (A0-A7) ส่วนตำแหน่งแอดเดรสไบท์สูงจะอยู่ที่พอร์ท 2

- **ขาพอร์ท 1** (Port 1)

มี 8 ขา ได้แก่ขา P1.0 – O1.7 เป็นขาพอร์ทอินพุทเอาท์พุทแบบ 2 ทิศทาง สำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นอินพุทพอร์ทต้องทำการเขียนค่า 1 ไปยังแต่ละพอร์ทเพื่อกำหนดให้เป็นพอร์ทอินพุท

- **ขาพอร์ท 2** (Port 2)

มี 8 ขา ได้แก่ขา P2.0 – P2.7 เป็นขาพอร์ทอินพุทพอร์ทเอาท์พุทแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นอินพุทพอร์ทต้องทำการเขียนค่า 1 ไปยังแต่ละบิทของพอร์ท เพื่อกำหนดให้เป็นพอร์ทอินพุท นอกจากพอร์ทนี้จะใช้งานเป็นพอร์ทอินพุทเอาท์พุทแล้วมันยังถูกใช้งานในการติดต่อกับหน่วยความจำภายนอกด้วย โดยทำหน้าที่ในการกำหนดตำแหน่งแอดเดรสไบท์สูง (A8 - A15)

- **ขาพอร์ท 3** (Port 3)

มี 8 ขา ได้แก่ขา P3.0 – P3.7 เป็นขาพอร์ทอินพุทเอาท์พุทแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้เป็นอินพุทพอร์ทต้องทำการเขียนค่า 1 ไปยังแต่ละบิทของพอร์ท เพื่อกำหนดให้เป็นพอร์ทอินพุท นอกจากพอร์ทนี้จะใช้งานเป็นพอร์ทอินพุทเอาท์พุทแล้วมันยังถูกใช้งานในหน้าที่พิเศษต่างๆดังแสดง ในตารางที่ 2.2

Pin	Description
P3.0	RXD (Serial Input Port)
P3.1	TXD (Serial Output Port)
P3.2	INT0 (External Interrupt 0)
P3.3	INT1 (External Interrupt 1)
P3.4	T0 (Timer 0 External Input)
P3.5	T1 (Timer 1 External Input)
P3.6	WR (External Data Memory Write Strobe)
P3.7	RD (External Data Memory Read Strobe)

ตารางที่ 2.2 แสดงหน้าที่พิเศษของแต่ละขาของพอร์ท 3

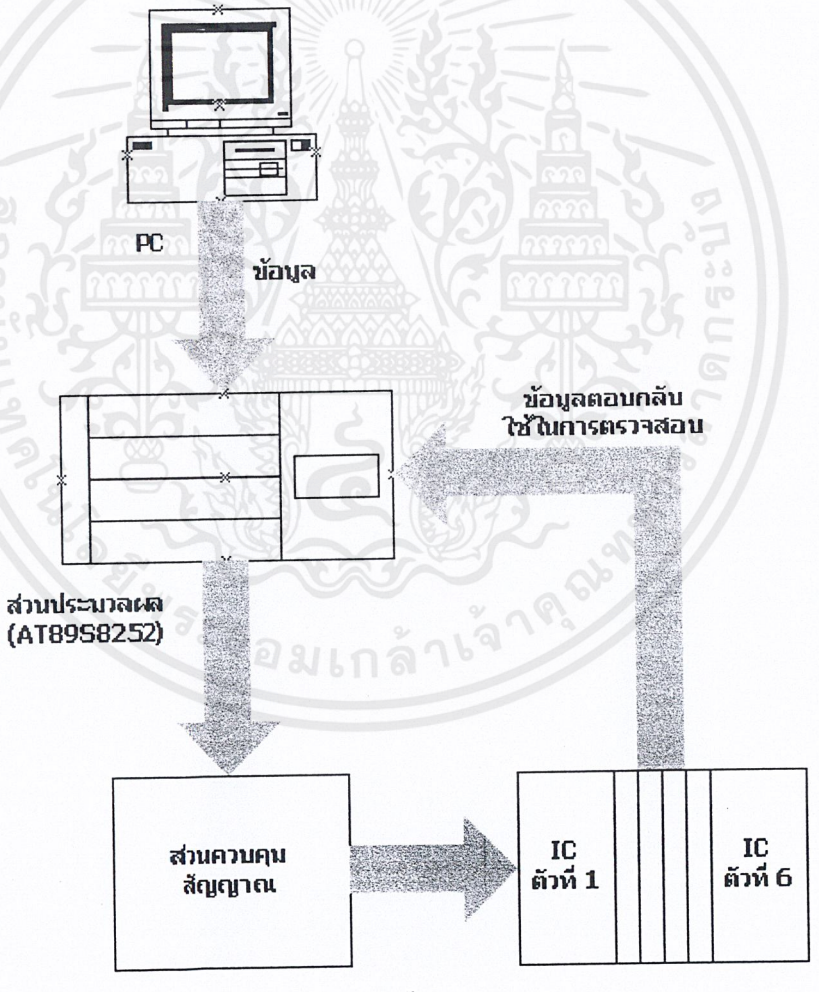
บทที่ 3
รายละเอียดโครงสร้าง

3.1 คำนำ

บทนี้จะกล่าวถึงการทำงานตามโครงสร้างของระบบเพื่อจะนำไปใช้ออกแบบในบทต่อไป

3.2 หลักการทำงาน

การทำงานของเครื่องโปรแกรมชุดไอซี มีหลักการทำงานดังรูป 3.1



รูปที่ 3.1

สามารถอธิบายหลักการทำงานได้ดังนี้

1. ตัวส่งข้อมูลจะส่งข้อมูลมายังส่วนโปรแกรมไอซี โดยส่วนโปรแกรมไอซีจะประมวลผลว่าทำหน้าที่อะไร เบอร์อะไร จะได้ส่งข้อมูลการโปรแกรมได้ถูกต้อง
2. ส่วนโปรแกรมไอซีส่งข้อมูลและการควบคุมไปยังไอซีที่นำมาโปรแกรม
3. ส่วนโปรแกรมไอซีตรวจสอบข้อมูลที่ได้รับจากไอซีทั้งสอง นำมาประมวลผลแล้วเก็บค่าไว้
4. ส่วนส่งข้อมูล ส่งข้อมูลการสิ้นสุดการโปรแกรมให้แก่ส่วนโปรแกรมไอซี
5. ส่วนโปรแกรมไอซีตรวจสอบความถูกต้อง และแสดงผลออกทาง LED

ขั้นตอนการดำเนินงานตามโครงสร้าง

ขั้นตอนที่ 1 - ศึกษารายละเอียดของระบบการโปรแกรมของไอซีเบอร์นั้นๆ
- ศึกษาการเขียนโปรแกรมทั้งจาก MCS51 และในส่วนของ PC

ขั้นตอนที่ 2 - ออกแบบวงจรที่ใช้ในการทำงานในโครงสร้าง 3 ส่วนที่ได้กล่าวไว้ คือ ส่วนส่งข้อมูล ส่วนโปรแกรมไอซี และส่วนตรวจสอบความถูกต้อง

ขั้นตอนที่ 3 - สร้างวงจรตามที่ได้ออกแบบไว้ในขั้นตอนที่ 2

ขั้นตอนที่ 4 - ทดสอบวงจรในแต่ละส่วนตามที่ได้สร้างไว้จากขั้นตอนที่ 3

ขั้นตอนที่ 5 - เขียนโปรแกรมเพื่อประมวลผลข้อมูล ให้แก่ ส่วนส่งข้อมูล และส่วนโปรแกรมไอซี

ขั้นตอนที่ 6 - ทดสอบการทำงานของเครื่องโปรแกรมชุดไอซี

ขั้นตอนที่ 7 - สรุปผลการทำโครงงาน ปัญหาในการทำงาน และพิมพ์รายงาน

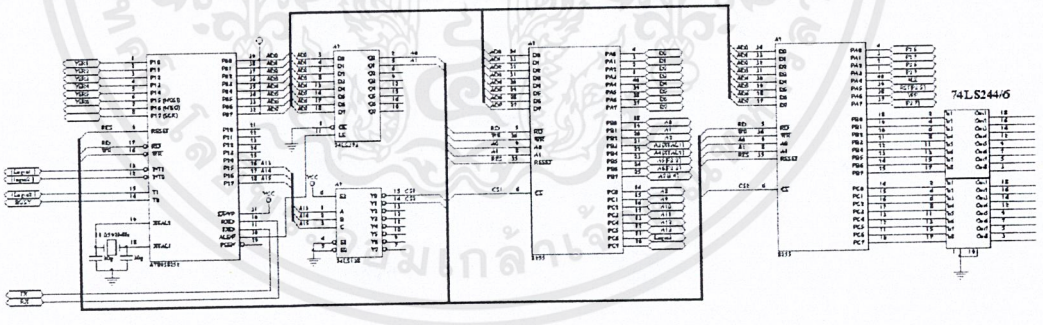
บทที่ 4
การออกแบบโครงสร้าง

โครงสร้างของเครื่องโปรแกรมชุดไอซี ประกอบด้วย 3 ส่วนหลักๆ คือ

1. ส่วนประมวลผล
2. ส่วนตรวจสอบข้อมูล
3. ส่วนสร้างสัญญาณโปรแกรม (Vpp)

โครงสร้างส่วนที่ 1

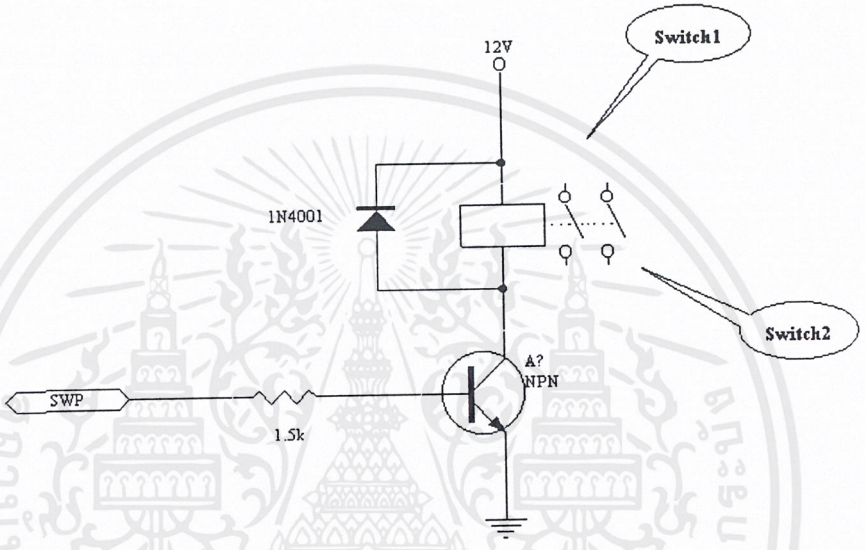
ส่วนประมวลผลมีหน้าที่รับข้อมูลจากเครื่องคอมพิวเตอร์ และสร้างสัญญาณให้กับตัวไอซีที่จะทำการโปรแกรมโดยใช้ไมโครคอนโทรลเลอร์เบอร์ AT89S8252 ดังที่แสดงในรูปที่ 4.1 ซึ่งส่วนนี้มีความสำคัญเป็นอย่างมากเพราะจะเป็นส่วนที่เชื่อมต่อระหว่างตัวไอซีที่จะโปรแกรมกับเครื่องคอมพิวเตอร์



รูปที่ 4.1

<u>ส่วนควบคุมการจ่ายแรงดันไฟ</u>

ในการนำตัวไอซีลงสู่ Textool และออกจาก Textool จำเป็นต้องมีการตัดแหล่งจ่ายไฟออกจากวงจรก่อนเพื่อป้องกันความเสียหายที่จะเกิดขึ้นกับตัวไอซี โดยจะมีวงจรดังรูปที่ 4.2



รูปที่ 4.2

โดยจะออกแบบค่าตัวต้านทาน(R) ที่ใช้ในการไบอัสได้จาก

I_{BSAT} = 2.5 mA

แรงดันควบคุม = 5 V

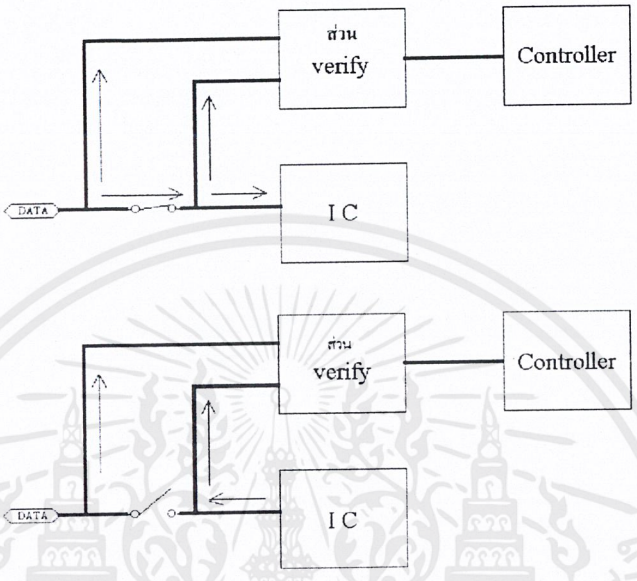
จะได้ 5 V = I_{BSAT} R + 0.6 V

R = (5 V - 0.6 V) / I_{BSAT}

= 1.7 kΩ

การ CONTROL SWITCH

ข้อมูลที่ส่งผ่านไปยังไอซี และสัญญาณที่ตอบกลับออกมาเพื่อตรวจสอบนั้นเป็นสัญญาณบนสายเส้นเดียวกันแต่แตกต่างกันในช่วงเวลาและทิศทางของข้อมูล โดยเมื่อส่งข้อมูลไปแล้วจำเป็นต้องตัดเส้นสัญญาณนี้ จะเหลือไว้เฉพาะเส้นตรวจสอบสัญญาณ ดังรูป 4.3



รูปที่ 4.3

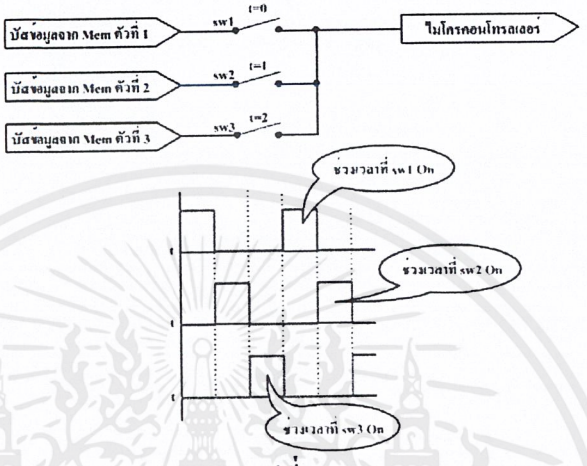
จากรูปจะเห็นได้ว่าต้องใช้สวิทช์ที่สามารถควบคุมโดยโปรแกรม ดังนั้นเราอาจเลือกใช้อนาลอก สวิทช์หรือบัฟเฟอร์ที่เป็น Tri-state แต่เนื่องจากสวิทช์ที่ใช้เป็นสวิทช์ทางเดียว ถ้าเลือกใช้อนาลอกสวิทช์ สายข้อมูล 8 เส้นต้องใช้ไอซีถึง 2 ตัว ดังนั้นเราจึงเลือกใช้บัฟเบอร์ 74LS244 แทน โดยมีหลักการควบคุมดังนี้

โครงสร้างส่วนที่ 2

ในการนำโปรแกรมข้อมูลลงสู่หน่วยเมมโมรี่(memory) จำเป็นต้องมีการตรวจสอบว่าข้อมูลที่จะโปรแกรมลงไปนั้นถูกต้องหรือไม่ เพื่อให้โปรแกรมทำงานได้ถูกต้องโดยไม่ผิดพลาด ถ้าหากเราโปรแกรมเพียงเมมโมรี่เดียว การตรวจสอบความผิดพลาดก็จะสามารถทำได้ง่าย แต่หากเป็นหลายๆเมมโมรี่ก็จำเป็นต้องมีวิธีการตรวจสอบที่ถูกต้องและเน้นความรวดเร็วดังนี้

1. <u>การแบ่งช่วงเวลาการตรวจสอบ</u> (TDM)

จะมีรูปแบบการทำงาน ดังรูปที่ 4.4

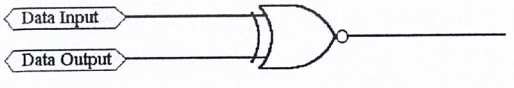


รูปที่ 4.4

จะเห็นได้ว่าวิธีการแบ่งช่วงเวลาการตรวจสอบไม่สามารถตรวจสอบเมมโมรี่พร้อมๆกันได้ ดังนั้นจะมีปัญหาในเรื่องความเร็วที่ใช้ในการตรวจสอบข้อมูล

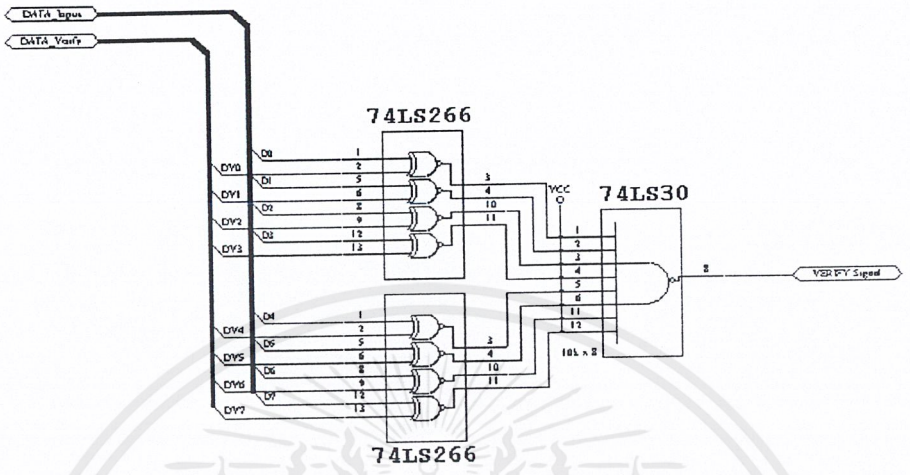
2. <u>การตรวจสอบโปรแกรมแบบขนานโดยใช้วงจรตรวจสอบ</u>

เป็นวิธีที่เน้นความรวดเร็วที่ใช้ในการตรวจสอบข้อมูลโดยใช้วงจรเอกคลูซิฟนอร์ (Exclusive Nor)ในการตรวจสอบข้อมูลที่เข้าและออกมาจากเมมโมรี่ แต่วิธีนี้จะสิ้นเปลืองพอร์ทเป็นจำนวนบิทเท่ากับจำนวนของเมมโมรี่ที่ทำการโปรแกรม โดยมีวงจรดังรูปที่ 4.5



รูปที่ 4.5

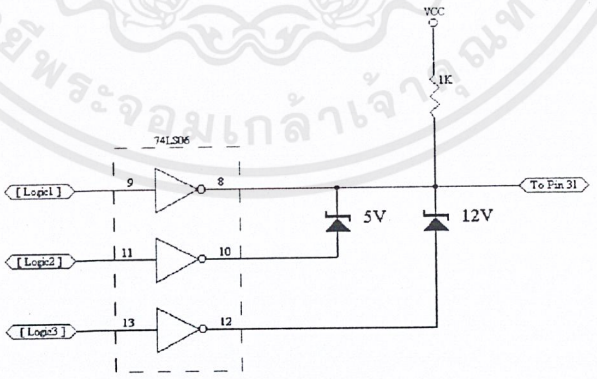
เนื่องจากในโครงงานนี้เน้นที่ความรวดเร็วในการโปรแกรม ดังนั้นวิธีการแบ่งเวลาในการตรวจสอบจึงไม่เหมาะสมที่จะใช้ เราจึงใช้วิธีการตรวจสอบโปรแกรมแบบขนานแทนโดยใช้วงจรดังรูปที่ 4.6



รูปที่ 4.6

โครงสร้างส่วนที่ 3

การที่จะโปรแกรมข้อมูลลงสู่เมมโมรี่จำเป็นต้องมีแรงดันโปรแกรม 12 V ตามคาต้าชีทของแต่ละเบอร์ และขาที่ใช้สำหรับแรงดันโปรแกรม(V_{PP})ของแต่ละเบอร์จะอยู่ไม่ตรงกัน ดังนั้นจึงต้องมีวงจรที่เปลี่ยนระดับแรงดันลงอีกจาก 0 V และ 5 V เป็น 5 V และ 12 V ดังรูปที่ 4.7



รูปที่ 4.7

1.) เมื่อต้องการระดับแรงดัน 0 V ป้อนLogic 100

2.) เมื่อต้องการระดับแรงดัน 5 V ป้อนLogic 010

3.) เมื่อต้องการระดับแรงดัน 12 V ป้อนLogic 001

บทที่ 5
โปรแกรมควบคุมการทำงาน
เครื่องโปรแกรมไมโครคอนโทรลเลอร์แบบกลุ่ม

5.1 หน้าที่ของโปรแกรม

เนื่องจากเครื่องโปรแกรมไมโครคอนโทรลเลอร์แบบกลุ่มนั้นมีการออกแบบให้รับข้อมูล, ตำแหน่งข้อมูล และคำสั่งในการควบคุมการทำงานวงจร และอ่านข้อมูลจากตัวไมโครคอนโทรลเลอร์ ดังนั้นตัวโปรแกรมควบคุมจึงมีหน้าที่สำคัญคือเป็นส่วนติดต่อกับผู้ใช้ ซึ่งจะให้ตัวผู้ใช้เป็นคนเลือกชนิดของไมโครคอนโทรลเลอร์ที่ต้องการโปรแกรม แฟ้มข้อมูลที่ต้องการใช้เป็นต้นฉบับในการโปรแกรม, เป็นตัวจัดลำดับการส่งชุดคำสั่งในการสั่งงานวงจร Flash Programmer, และทำหน้าที่เป็นตัวรับข้อมูลที่ส่งมาจากการอ่านข้อมูลภายในตัวไมโครคอนโทรลเลอร์ และสร้างเป็นไฟล์ HEX เพื่อใช้งานต่อไปอีกด้วย

5.2 ความสามารถของโปรแกรม

- สามารถเลือกชนิดของไมโครคอนโทรลเลอร์ที่ต้องการใช้โปรแกรมได้ 7 เบอร์
    1. AT89C51
    2. AT89C52
    3. AT89S53
    4. AT89S8252
    5. AT89C1051
    6. AT89C2051
    7. AT89C4051
- ใช้งานกับไฟล์ประเภท HEX เป็นต้นฉบับซึ่งสะดวกกับผู้ใช้ในการแก้ไข
- ติดต่อกับคอมพิวเตอร์ผ่านพอร์ตอนุกรมด้วยความเร็ว 19200 bps
- ใช้งานกับระบบปฏิบัติการแบบ Windows 9x/NT
- สามารถอ่านข้อมูลจากไมโครคอนโทรลเลอร์ได้ผ่านทางเครื่องโปรแกรม

5.3 ข้อจำกัดของโปรแกรม

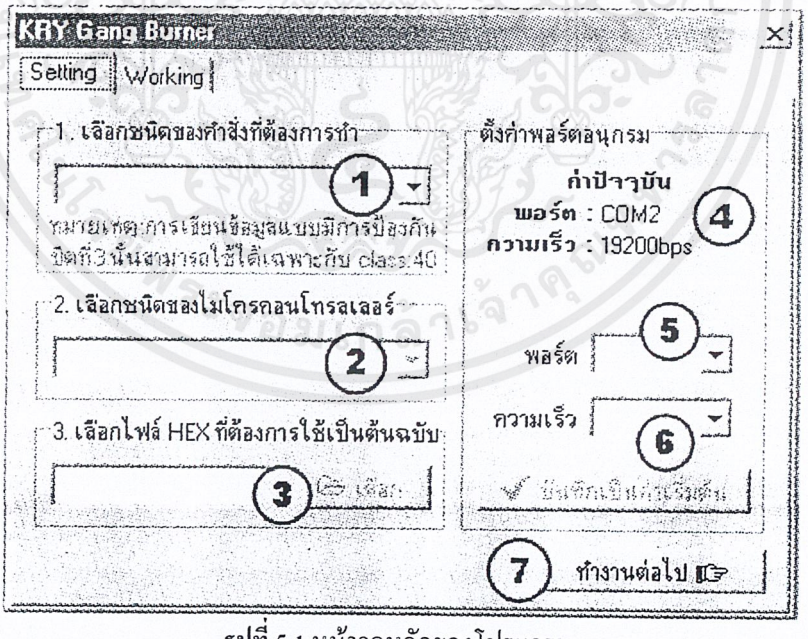
- ต้องทำงานกับพอร์ตอนุกรมที่ความเร็ว 19200 bps เท่านั้น
- ต้องใช้กับระบบปฏิบัติการแบบ Windows เท่านั้น
- ยังต้องใช้งานกับไฟล์ HEX ที่มีจำนวนข้อมูลบรรทัดละ 16 ไบต์เท่านั้น

5.4 การใช้งานโปรแกรม Kry Gang Programmer

โปรแกรม KRY Gang Programmer เป็นโปรแกรมที่ถูกออกแบบมาให้ทำงานกับวงจรควบคุมการทำงานโดยเฉพาะ โดยหน้าที่หลักของโปรแกรมคือเป็นส่วนติดต่อกับผู้ใช้ แล้วส่งผ่านคำสั่งไปสู่แผงวงจรหลักที่เป็นตัวทำงานต่างๆอีกทีหนึ่ง ซึ่งตัววงจรหลักจะทำหน้าที่รับคำสั่งและตีความเพื่อทำงานตามต้องการ

ขั้นตอนการใช้งานโปรแกรม

1. เมื่อเปิดโปรแกรมจะมีหน้าจอดังนี้



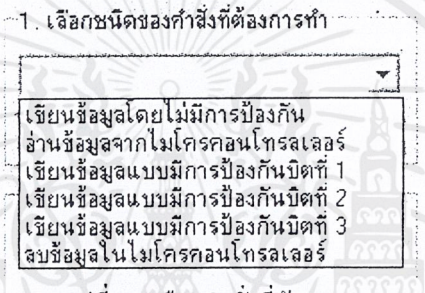
รูปที่ 5.1 หน้าจอหลักของโปรแกรม

<u>คำอธิบาย</u>

1. เลือกคำสั่งที่ต้องการสั่งให้เครื่องโปรแกรมกระทำ
2. เลือกเบอร์ของตัวไมโครคอนโทรลเลอร์ปลายทางในการเขียน หรือเลือกเบอร์ที่ใช้เป็นต้นฉบับในการอ่าน
3. เลือกไฟล์ต้นฉบับ, เซฟ สำหรับการอ่าน/เขียน

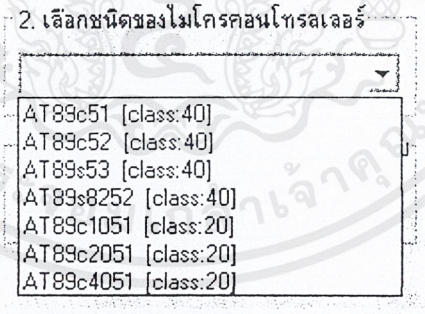
4. แสดงค่าพอร์ตอนุกรมที่โปรแกรมใช้อยู่ในขณะนี้
5. เลือกพอร์ตอนุกรม/ความเร็วการติดต่อที่ต้องการ
6. บันทึกค่าของพอร์ตอนุกรมลง Windows Registry
7. สั่งให้เริ่มทำการติดต่อกับเครื่องโปรแกรม

2. คลิกคอมโบบอกซ์เลือก "เลือกชนิดของคำสั่งที่ต้องการกระทำ" ซึ่งจะขึ้นหน้าจอดังนี้



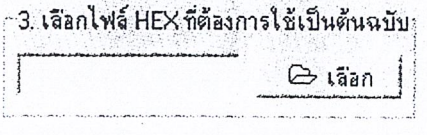
รูปที่ 5.2 เลือกคำสั่งที่ต้องการ

3. คลิกคอมโบบอกซ์เลือก "เลือกชนิดของไมโครคอนโทรเลอร์" จะขึ้นหน้าจอดังนี้

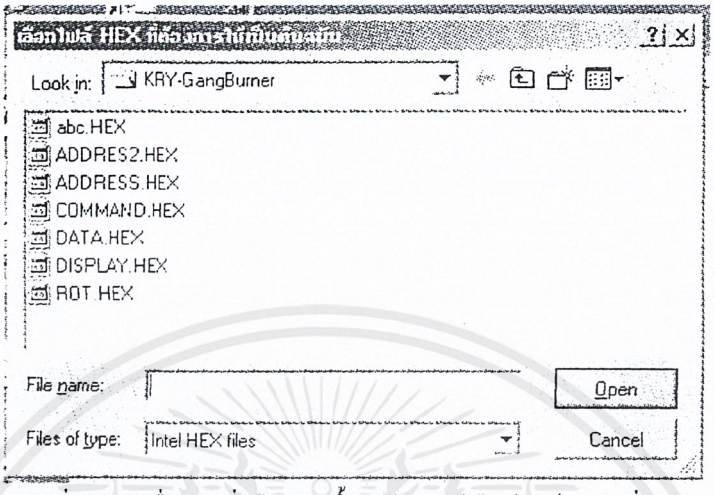


รูปที่ 5.3 เลือกเบอร์ของไมโครคอนโทรลเลอร์ที่ต้องการ

4. หากเป็นการเลือกคำสั่ง เขียนข้อมูล หรือเป็นคำสั่ง อ่านข้อมูล ที่ส่วนของ "เลือกไฟล์ HEX ที่ต้องการใช้เป็นต้นฉบับ" จะสามารถเลือกได้ ให้คลิกที่ "เลือก" เพื่อเลือกไฟล์ HEX ที่ต้องการ เลือกชื่อไฟล์ที่ต้องการแล้วคลิกที่ Open/Save เพื่อดำเนินการขั้นต่อไป



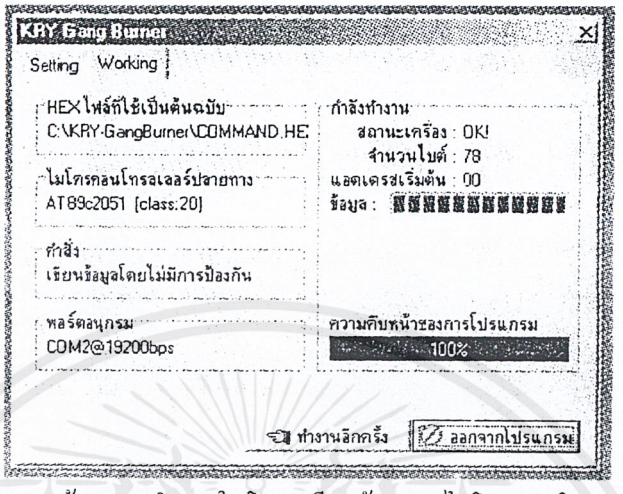
รูปที่ 5.4 หากเลือกคำสั่ง เขียน, อ่าน ส่วนนี้จะสามารถเลือกได้



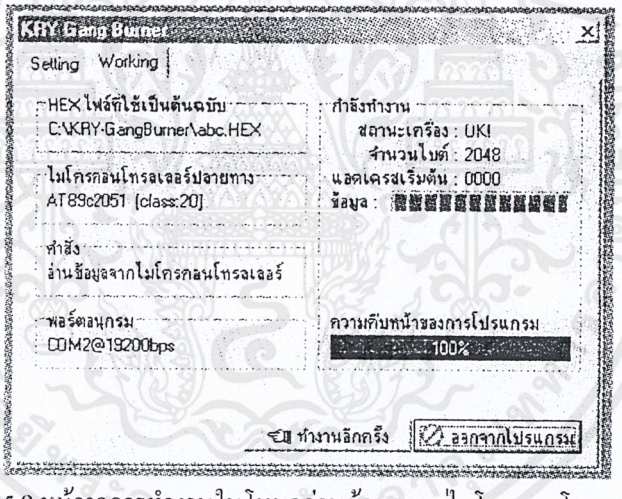
รูปที่ 5.5 หากเลือกคำสั่ง เขียน เมื่อคลิกที่ เลือก จะขึ้นหน้าจอให้เลือกไฟล์ HEX ที่ต้องการใช้เป็นต้นฉบับ

รูปที่ 5.6 หากเลือกคำสั่ง อ่าน เมื่อคลิกที่ เลือก จะขึ้นหน้าจอให้เลือกไฟล์ปลายทางที่ต้องการเซฟ

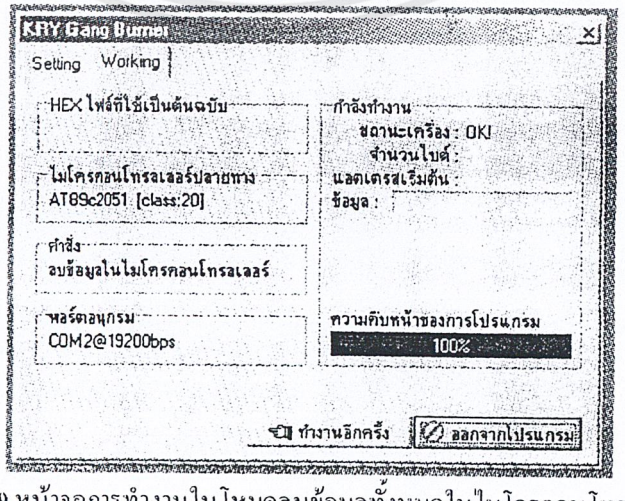
5. คลิกที่ "ทำงานต่อไป" เพื่อให้โปรแกรมเริ่มทำการติดต่อกับตัวเครื่อง ซึ่งเมื่อคลิกแล้วจะขึ้นหน้าจอการทำงานดังรูป



รูปที่ 5.7 หน้าจอการทำงานในโหมดเขียนข้อมูลลงไมโครคอนโทรลเลอร์



รูปที่ 5.8 หน้าจอการทำงานในโหมดอ่านข้อมูลจากไมโครคอนโทรลเลอร์



รูปที่ 5.9 หน้าจอการทำงานในโหมดลบข้อมูลทั้งหมดในไมโครคอนโทรลเลอร์

6. เมื่อโปรแกรมทำงานเสร็จสิ้นแล้ว (สังเกตุได้จาก ความคืบหน้าของโปรแกรมเป็น 100% ก็สามารถคลิกที่ “ทำงานอีกครั้ง” หรือ “ออกจากโปรแกรม”

บทที่ 6
การออกแบบโปรแกรม

โครงสร้างของโปรแกรมควบคุมการทำงานของเครื่อง จะประกอบไปด้วยยูนิตสำคัญๆ อยู่ 5 ยูนิต

1. ยูนิต Main เป็นยูนิตหลักของโปรแกรมนี้ จะประกอบไปด้วยการควบคุมการทำงานของโปรแกรมนี้ ซึ่งทำงานตามอีเวนต์ต่างๆ ที่เกิดขึ้นในโปรแกรมนั่นเอง และทำหน้าที่ติดต่อกับผู้ใช้
2. ยูนิต MSCommLib_TLB เป็นยูนิตที่สร้างขึ้นจากตัวโปรแกรม Delphi เอง ซึ่งจะทำหน้าที่ติดต่อสื่อสารกับพอร์ตอนุกรม
3. ยูนิต FileWorkShop เป็นยูนิตที่เขียนขึ้นเองทำหน้าที่ติดต่อกับไฟล์, สร้างไฟล์ และแปลงไฟล์ ทั้งไฟล์ประเภท Intel HEX format และ ไฟล์ Binary
4. ยูนิต HexWorkShop เป็นยูนิตที่เขียนขึ้นเอง ทำหน้าที่แปลงจำนวนที่เกี่ยวข้องกับเลขฐาน 16 ที่จำเป็นต้องใช้ภายในโปรแกรม
5. ยูนิต BurnIT เป็นยูนิตที่เขียนขึ้นเอง จะทำหน้าที่ควบคุมลำดับการส่ง, รับข้อมูลกับตัวเครื่องโปรแกรม

6.1 การสร้างยูนิต MSCommLib_TLB

เนื่องจากในโครงงานนี้ใช้คอนโทรล MSComm32 ของไมโครซอฟท์ ซึ่งเป็นคอนโทรลที่เป็น ActiveX ทำให้ในการสร้างจำเป็นจะต้องมีการรีจีสเตอร์ไฟล์ MSComm32.OCX กับ Windows ก่อน โดยปกติคอนโทรล MSComm32.OCX นี้จะติดมากับโปรแกรม Visual Basic หากมีการติดตั้งโปรแกรม Visual Basic หรือ Visual Studio ตัวโปรแกรมจะทำการรีจิสเตอร์ตัวคอนโทรลนี้ให้โดยอัตโนมัติ แต่โดยทั่วไปตัว Windows จะไม่มีการรีจิสเตอร์ตัวคอนโทรลนี้ไว้ ซึ่งผู้สร้างสามารถดาวน์โหลดตัวคอนโทรลนี้ได้จากเว็บไซต์หลายแห่ง เช่นที่ http://www.yes-tele.com/MSComm.html เป็นต้น และเมื่อดาวน์โหลดตัวคอนโทรล MSComm32.OCX มาแล้วก็จำเป็นต้องทำการรีจิสเตอร์โดยการเรียกโปรแกรม RegSVR32 โดยมีลำดับการติดตั้งดังนี้

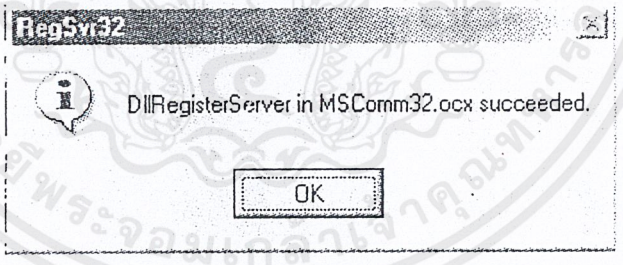
1. ก็อปปี้ไฟล์ MSComm32.OCX ไปที่ C:\Windows\System (สำหรับ Windows 95, 98, ME) หรือที่ C:\WinNT\System32 (สำหรับ Windows NT/2000) ดังรูป

```
C:\WINDOWS\SYSTEM>dir MSComm32.ocx

 Volume in drive C is DISK1_VOL1
 Volume Serial Number is 8AA0-5D38
 Directory of C:\WINDOWS\SYSTEM

MSCOMM32 OCX       103,744  06-24-98 12:00a MSCOMM32.OCX
         1 file(s)        103,744 bytes
         0 dir(s)       1,876.18 MB free

C:\WINDOWS\SYSTEM>
```

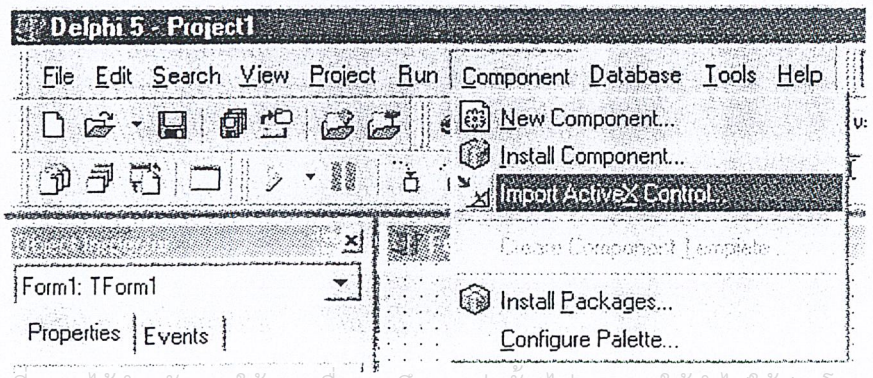
2. ไปยังไดเรคทอรี C:\Windows\System และเรียกโปรแกรม RegSVR32.exe โดยพิมพ์ ดังรูป

```
C:\WINDOWS\SYSTEM>RegSVR32 MSComm32.ocx
```

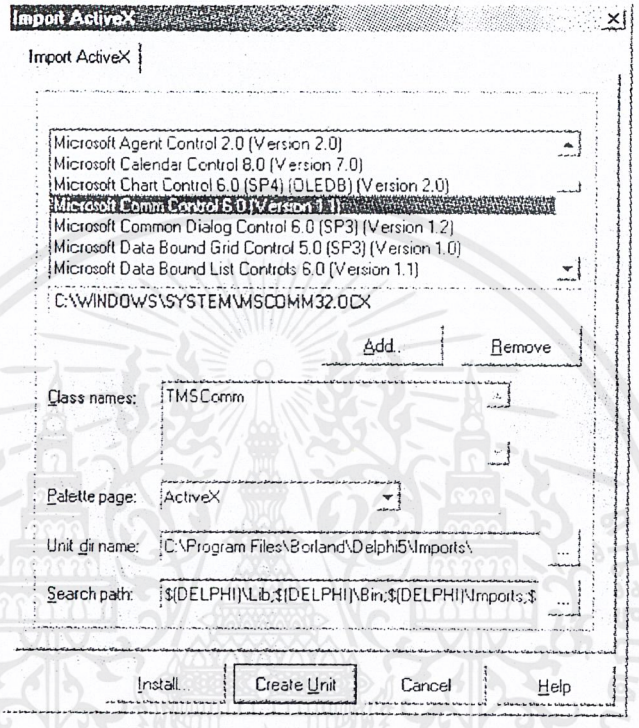
3. วินโดวส์จะแสดงผลว่าทำการรีจิสเตอร์คอนโทรลเสร็จเรียบร้อย ซึ่งตอนนี้คอนโทรล MSComm32 พร้อมจะใช้งาน ดังรูป



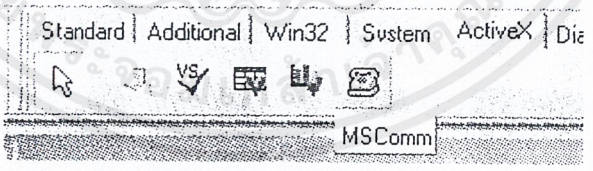
4. เรียกโปรแกรม Delphi
5. เลือกที่ Component/Import ActiveX Control ดังรูป



6. คลิกเลือก Microsoft Comm Control 6.0 จากลิสต์ ซึ่งจะแสดงรายละเอียดต่างๆ ของตัวคอนโทรล จากนั้นคลิกที่ Install... เพื่อให้ Delphi สร้างยูนิต MSCommLib_TLB



7. คอนโทรลพร้อมใช้งาน เมื่อเรียกสร้างโปรแกรมขึ้นใหม่ ยูนิตสำหรับเรียกใช้คอนโทรล MSComm32 นั้นจะอยู่ในแท็บ ActiveX ดังรูป



6.2 คอนโทรล MSComm

MSComm จัดเตรียมทางเลือกเอาไว้ 2 ทางเพื่อความสะดวกในการสื่อสารข้อมูล ทางแรก คือ การสื่อสารข้อมูลที่กระตุ้นด้วยเหตุการณ์ (Event-Driven Communication) เป็นรูปแบบการใช้งานที่มีประสิทธิภาพมากสำหรับการตอบสนองแบบทันทีทันใด เช่น เมื่อตัวอักษรถูกส่งมาที่พอร์ตอนุกรม หรือเกิดการเปลี่ยนแปลงที่ขา Data Carrier Detect (DCD) หรือขา Request To Send (RTS) เหตุการณ์ OnComm จะสามารถตรวจสอบสัญญาณนั้นได้ทันที ส่วนทางเลือกที่สองเป็นการคอยตรวจสอบค่าเหตุการณ์และความผิดพลาดที่เกิดขึ้นด้วยการดูค่าที่เปลี่ยนแปลงภายใน Properties ของ MSComm ซึ่งวิธีนี้ใช้การได้ดีในกรณีที่โปรแกรมมีขนาดเล็ก

คอนโทรล MSComm 1 ตัวสามารถควบคุมการทำงานของพอร์ตอนุกรมได้ 1 พอร์ต ถ้าในโปรแกรมที่ใช้งานต้องการติดต่อกับพอร์ตอนุกรมมากกว่า 1 พอร์ต จะต้องใช้คอนโทรล MSComm ตามจำนวนพอร์ตที่ต้องการติดต่อ แอดเดรสของพอร์ตอนุกรมและแอดเดรสของการเกิดอินเตอร์รัปต์สามารถเปลี่ยนแปลงได้จากการแก้ไขค่าที่ Control Panel

ถึงแม้ว่า คอนโทรล MSComm จะมีคุณสมบัติ (Properties) มากมาย แต่ที่สำคัญๆ ที่ใช้ในโครงงานนี้มีดังนี้

- CommPort ใช้ในการกำหนดและอ่านค่าพอร์ตอนุกรมที่ติดต่ออยู่ การเขียนโปรแกรมจำเป็นจะต้องมีการกำหนดตำแหน่งของพอร์ตอนุกรมทุกครั้งก่อนมีการเปิดพอร์ตนั้นเพื่อใช้งาน
- Setting ใช้กำหนดและอ่านค่าอัตราบอด, พาริตี้, จำนวนบิตข้อมูล, จำนวนบิตปิดท้าย ต้องทำการกำหนดก่อนทำการเปิดพอร์ตนั้นๆ
- PortOpen ใช้กำหนดและอ่านค่าสถานะของพอร์ตอนุกรมว่าเปิดใช้อยู่หรือไม่
- Input อ่านและลบค่าขบวนข้อมูล (ในรูปของ String) จากบัฟเฟอร์ภาครับ
- InBufferCount ใช้ตรวจสอบจำนวนตัวอักษรที่อยู่ในบัฟเฟอร์ภาครับ
- Output ใช้ส่งขบวนข้อมูล (ในรูปของ String) ไปยังบัฟเฟอร์ส่งข้อมูล

ตัวอย่างการเขียนโปรแกรมติดต่อกับ MSComm

```
Var MSComm1 : TMSComm;
   TargetStr : String;
Begin
MSComm1.CommPort := 2;
MSComm1.Settings := "19200,N,8,1";
MSComm1.PortOpen := True;
MSComm1.Output := "Hello";
If MSComm1.InBufferCount <> 0 then TargetStr:=MSComm1.Input;
MSComm1.PortOpen := False;
End.
```

สามารถหารายละเอียดเพิ่มเติมได้ทั้งจากที่ http://www.yes-tele.com/mscomm.html หรือที่เว็บของไมโครซอฟท์ หรืออาจค้นหาได้ในอินเตอร์เน็ต

6.3 ยูนิต FileWorkShop

ทำหน้าที่จัดการเกี่ยวกับไฟล์ HEX และไฟล์ BIN ที่ใช้ในการโปรแกรม หน้าที่สำคัญคือสร้างไฟล์ที่จะใช้ในการโปรแกรม สำหรับไมโครคอนโทรลเลอร์ที่มี 40 ขา (AT89C51, AT89C52, AT89C53, AT89S8252) นั้นสามารถทำการโปรแกรมแบบข้ามแอดเดรสได้ ไฟล์ที่ใช้จึงเป็นแบบเรคคอร์ดของข้อมูลแต่ละชุด ส่วนไมโครคอนโทรลเลอร์ประเภท 20 ขา (AT89C1051,

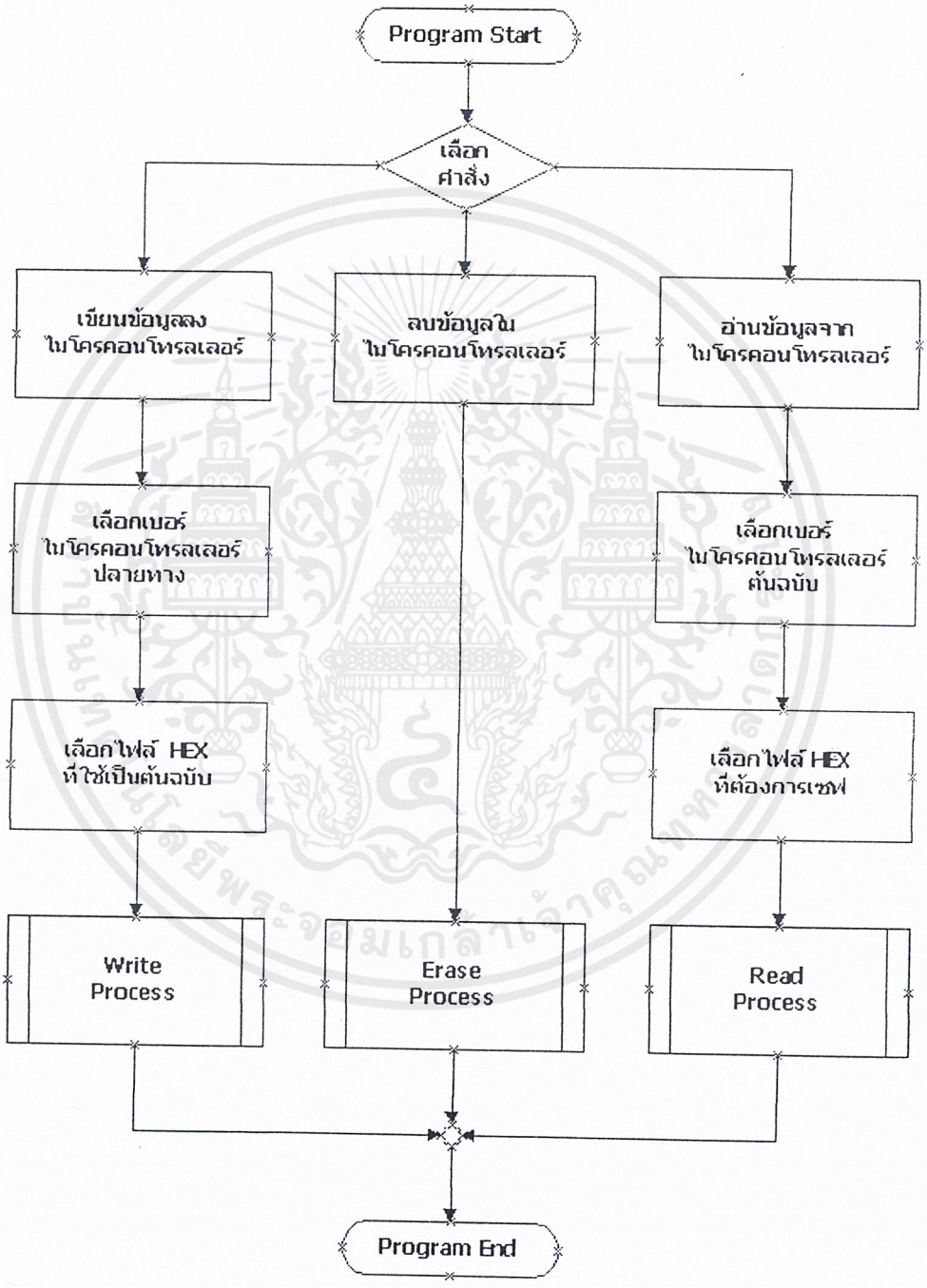
AT89C2051, AT89C4051) นั้นไม่สามารถทำการโปรแกรมแบบข้ามแอดเดรสได้ จึงจำเป็นต้องทำการแปลงไฟล์จาก Intel HEX Format ไปเป็นแบบ Binary ก่อนซึ่งจะอ่านข้อมูลเรียงทีละไบต์เพื่อทำการโปรแกรมไปจนครบทุกข้อมูล และหน้าที่ที่สำคัญอีกอย่างของยูนิตนี้คือ การแปลงข้อมูลที่อ่านมาจากไมโครคอนโทรลเลอร์ไปเป็นไฟล์ประเภท Intel HEX Format เนื่องจากเมื่อทำการอ่านข้อมูลที่อยู่ในตัวไมโครคอนโทรลเลอร์นั้น เราไม่สามารถรู้ได้ว่าขนาดของข้อมูลที่มีอยู่ในไมโครคอนโทรลเลอร์นั้นมีขนาดอยู่เท่าไร จึงจำเป็นต้องทำการอ่านเรียงตั้งแต่เริ่มจนครบทุกไบต์ ดังนั้นไฟล์ที่อ่านเข้ามาได้นั้นจะอยู่ในรูปของไฟล์ประเภท Binary จากนั้นจึงค่อยทำการแปลงไปเป็น Intel HEX Format อีกขั้นหนึ่ง

6.4 ยูนิต BurnIT

ยูนิตนี้จะมีหน้าที่ติดต่อสื่อสารกับตัวเครื่องโปรแกรมไมโครคอนโทรลเลอร์ ซึ่งจะคอยจัดลำดับข้อมูลที่ต้องส่งไปให้กับตัวเครื่องโปรแกรม และรับข้อมูลที่ส่งมาจากตัวเครื่องโปรแกรม ลำดับของการส่งข้อมูลนั้น จะขึ้นอยู่กับชนิดของคำสั่งที่ต้องการสั่งให้เครื่องโปรแกรมทำนั่นเอง สามารถแบ่งคำสั่งที่สามารถสั่งให้เครื่องโปรแกรมทำได้นั้น แบ่งได้เป็น 3 หมวดหมู่คือ

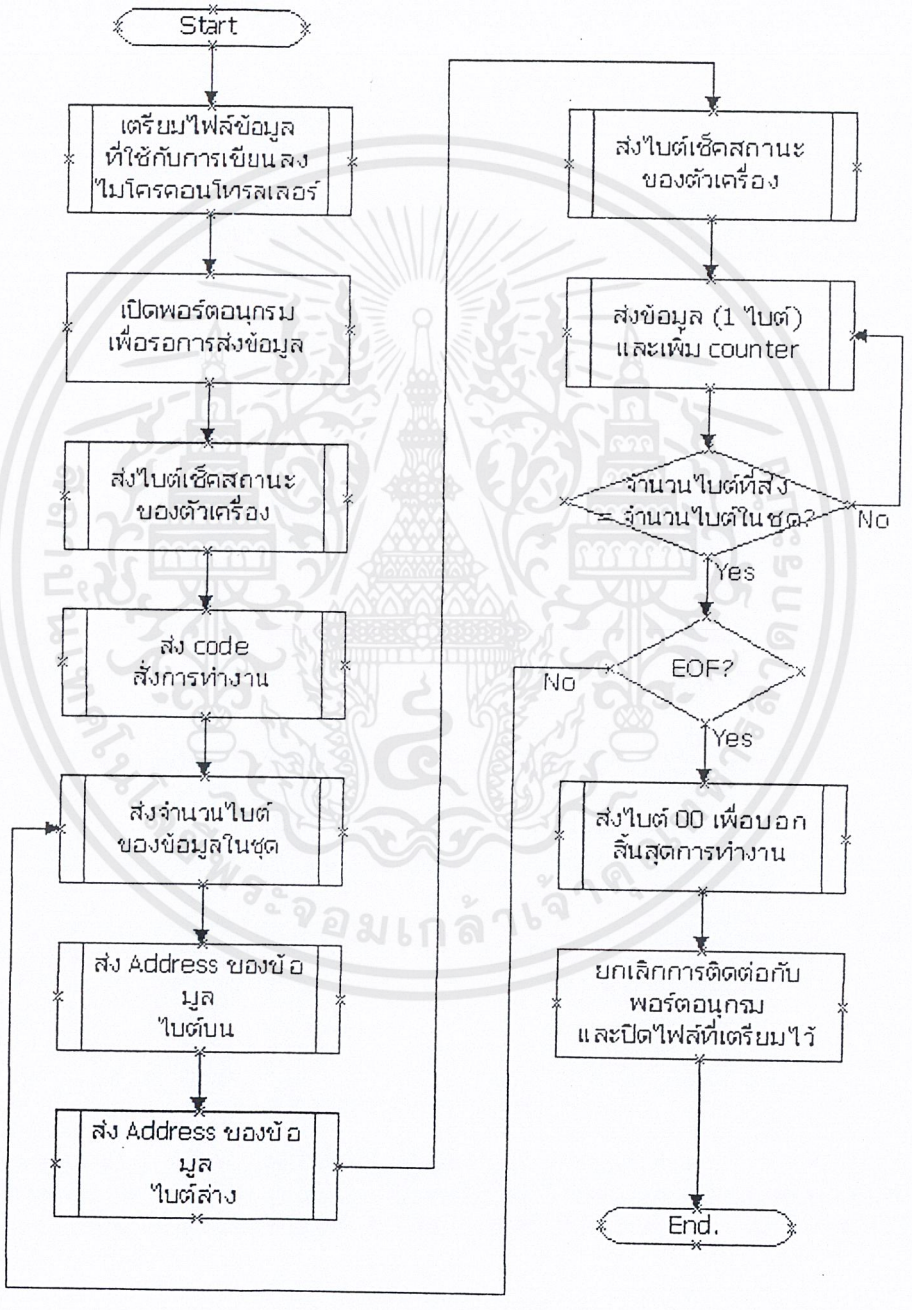
1. คำสั่งเขียนข้อมูลลงไมโครคอนโทรลเลอร์
   - เขียนแบบไม่มีการป้องกัน
   - เขียนแบบมีการล็อกบิต 1
   - เขียนแบบมีการล็อกบิต 2
   - เขียนแบบมีการล็อกบิต 3
2. คำสั่งอ่านข้อมูลจากไมโครคอนโทรลเลอร์
3. คำสั่งลบข้อมูลภายในไมโครคอนโทรลเลอร์

6.4.1 การทำงานของยูนิต BurnIT



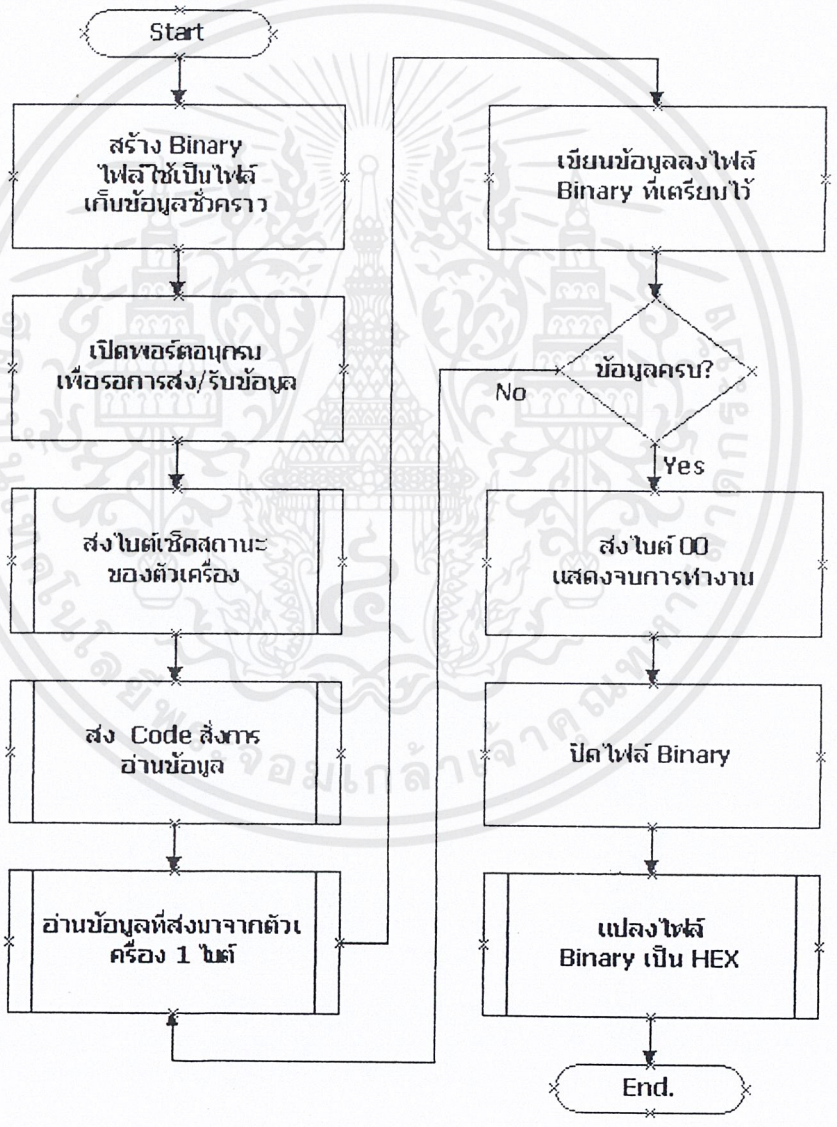
รูปที่ 6.1 โฟลวชาร์ตการทำงานส่วนหลักของโปรแกรมควบคุม

6.4.2 Write Process



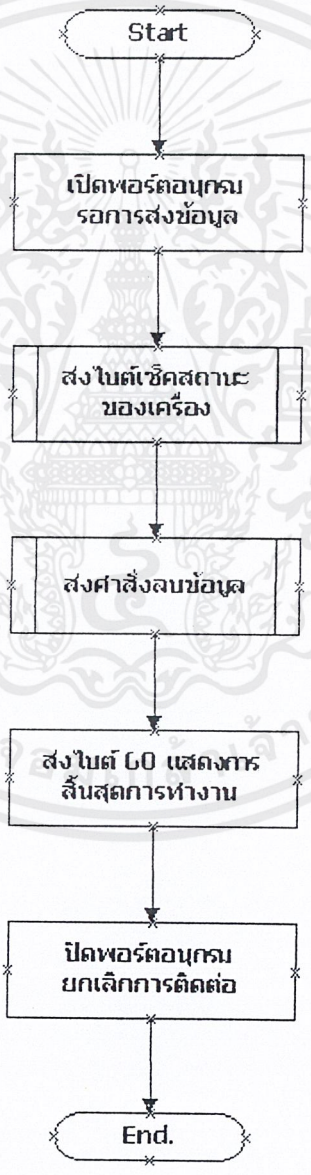
รูปที่ 6.2 โฟลวชาร์ตแสดงการทำงานของโปรแกรมส่วนการสั่งงานเขียนข้อมูลลงไมโครคอนโทรลเลอร์

6.4.3 Read Process



รูปที่ 6.3 โฟลวชาร์ตแสดงการทำงานของโปรแกรมส่วนการสั่งงานอ่านข้อมูลจากไมโครคอนโทรลเลอร์

6.4.3 Erase Process



รูปที่ 6.4 โฟลวชาร์ตแสดงการทำงานของโปรแกรมส่วนการลบข้อมูลภายในไมโครคอนโทรลเลอร์

บทที่ 7

การทดลอง

การทดลองจะใช้ไอซีเบอร์ AT89C2051, AT89C52, AT89S53, AT89S8252 ซึ่งได้ผลการทดลองตามตาราง

ตารางที่ 7.1 แสดงผลการทดลองความเร็วในการโปรแกรม

เบอร์	จำนวน ADDRESS (BYTE)	เวลา (วินาที)
AT89C2051	1k	19
AT89C52	1k	42
AT89C53	1k	42
AT89S8252	1k	42

ตารางที่ 7.2 ตารางแสดงผลการทดลองความเร็วในการอ่านข้อมูล

เบอร์	จำนวน ADDRESS (Byte)	เวลา (วินาที)
AT89C2051	2k	26
AT89C52	8k	45
AT89C53	10k	60
AT89S8252	8k	45

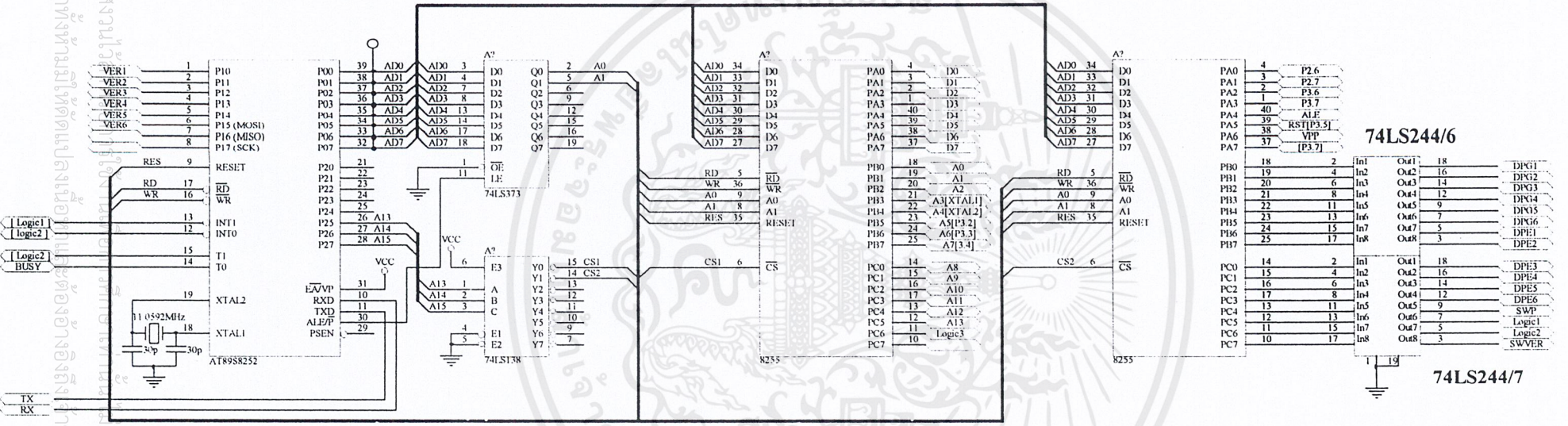
จากผลการทดลองจะเห็นได้ว่า ความเร็วในการเขียน Flash Memory ยังไม่มากนัก เมื่อเทียบกับเครื่องโปรแกรมโดยทั่วไป ตัวอย่างเช่นเครื่องโปรแกรมโดยทั่วไป 1Kbytes ใช้เวลาเขียนประมาณ 20 วินาที แต่เมื่อนับเป็นจำนวน Flash Memory แล้ว จะถือว่าเร็วกว่ามาก

บทที่ 8
สรุปและวิจารณ์ผลการทดลอง

โครงงานเครื่องโปรแกรมชุดไอซีชุดนี้สามารถโปรแกรมชุดไอซีแบบขนานและตรวจสอบการผิดพลาดแบบขนานได้ถูกต้องและสามารถอ่านค่าจากไอซีที่มีโปรแกรมอยู่แล้วได้ หากเพิ่มจำนวนชุดไอซีอีกก็จะเกิดประโยชน์อย่างมากในแง่ของอุตสาหกรรม เนื่องจากต้องการความรวดเร็วในแง่ของการผลิต แต่ยังมีปัญหาที่เกิดขึ้นเนื่องจากการทดลองคือ ความเร็วในการโปรแกรมแต่ละครั้งยังไม่มากพอที่จะทำให้เกิดประโยชน์สูงสุด อีกทั้งขนาดของวงจรยังมีขนาดที่ใหญ่เกินไปเราอาจแก้ปัญหาดังกล่าวโดย การรวมโครงสร้างในส่วนที่ 1 และ 2 เข้าไว้ด้วยกัน และแก้ไขโครงสร้างในส่วนที่ 3 ซึ่งจากเดิมที่ใช้วงจรในการตรวจสอบความผิดพลาดเปลี่ยนมาเป็นการแบ่งช่วงเวลาในการตรวจสอบ ซึ่งถ้าหากทำได้จะเป็นการลดวงจรทำให้ประหยัดต้นทุนในการผลิตได้

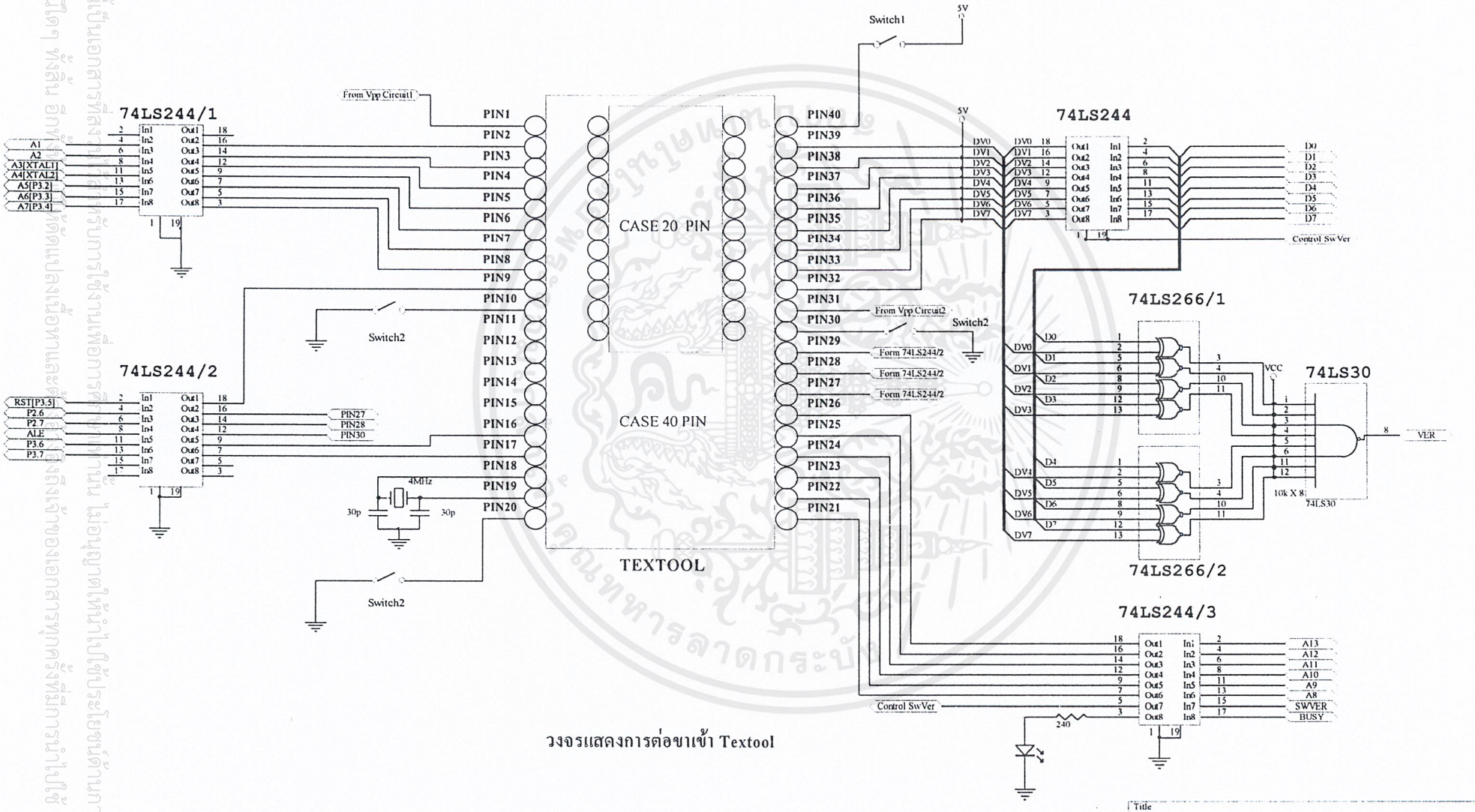
อย่างไรก็ตามในโครงงานชุดนี้ได้ก่อเกิดความรู้ให้แก่ผู้จัดทำเป็นอย่างมาก เพราะต้องใช้ความรู้หลายด้าน เช่นความรู้ด้านไมโครคอนโทรลเลอร์, การเขียนโปรแกรมบน PC เป็นต้น และจะได้นำความรู้เหล่านี้ไปใช้ประโยชน์ต่อไปในอนาคต

ภาคผนวก

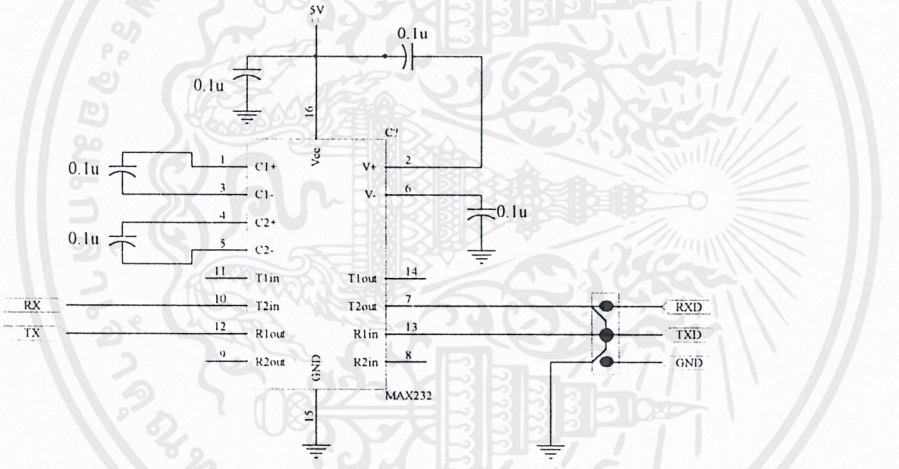


วงจรส่วนประมวลผล

Title			
Size: B	Number		Revision
Date: 19-Mar-2001		Sheet of	
File: C:\PROJECTS\PROTEL\CONTROL.SCH	Drawn By:		

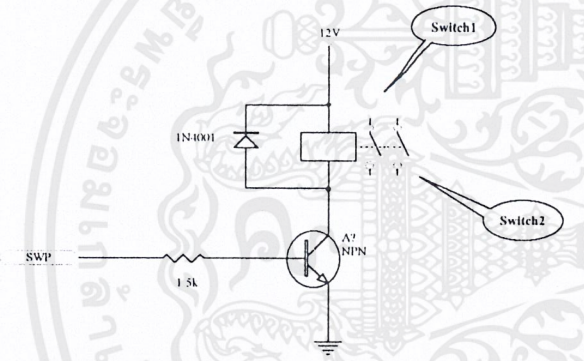


วงจรแสดงการต่อขาเข้า Textool



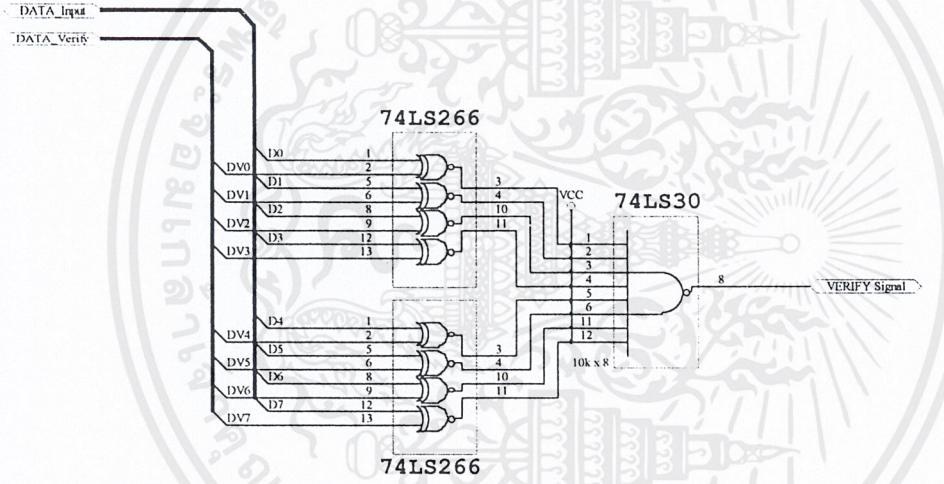
วงจรส่งและรับแบบอนุกรม

Size B	Number	Revision
Date:	19-Mar-2001	Sheet of
File:	C:\PROJECTS\PROTEL\MAX232.SCH	Drawn By:



วงจรควบคุมแหล่งจ่ายแรงดัน

Title		
Size: B	Number	Revision
Date: 19-Mar-2001		Sheet of
File: C:\PROJECTS\PROTELARELAY.SCH		Drawn By:



วงจรส่วนตรวจสอบความผิดพลาด

Title		
Size: B	Number	Revision
Date: 19-Mar-2001	Sheet of	
File: C:\PROJECTS\PROTEL\VERIFY.SCH	Drawn By:	

```
unit burnit;

interface
uses MSCommLib_TLB;

procedure Burn( gngFilename : String;
                gngCMD : byte;
                uCtarget : byte;
                CommObj : TMSComm);
procedure initCommPort;
procedure openCommPort;
procedure closeCommPort;
functionbyteOutChk(outByte:byte):boolean;
function byteIn :byte;
procedureWrite40pin(CmdSendOut:byte);
procedureWrite20pin(CmdSendOut:byte);
procedure uCerase(CmdSendOut:byte);
procedure uCread(CmdSendOut:byte);
//=============================================================
implementation
uses SysUtils, main, fileWorkshop, hexWorkshop, Dialogs, Registry, Windows;
//==========declare variable use in unit here==============
var WorkingFilename : String;
    WorkingComm     : TMSComm;
    WorkingCommPort : Byte;
    WorkingCommBaud : Word;
    WorkingCommand  : byte;
    WorkinguCtype   : byte;
    WorkingHEXfile  : file of hexFileType;
    WorkingBINfile  : file of byte;
    tmpHEX          : hexFileType;
    tmpHEXbyteCount : byte;
    tmpHEXaddrHi    : byte;
    tmpHEXaddrLO    : byte;
    tmpHEXdata      : array[1..16] of byte;
    tmpBIN          : byte;
//=============================================================

procedure Burn( gngFilename : String;
                gngCMD : byte;
                uCtarget : byte;
                CommObj : TMSComm);
begin
WorkingFilename := gngFilename;
WorkingComm := CommObj;
WorkingCommand := gngCMD;
WorkinguCtype := uCtarget;
case WorkingCommand of
    1,2,7,8,9,10,11,12 : Write40pin(WorkingCommand);
    5,13,14            : Write20pin(WorkingCommand);
    16,17,18           : uCerase(WorkingCommand);
    3,4,6              : uCread(WorkingCommand);
end;
//initCommPort;
end;
//=============================================================
procedure OpenHEXfile;
begin
assignFile(WorkingHEXfile,WorkingFilename);
reset(WorkingHEXfile);
end;

procedure CloseHEXfile;
begin
closefile(WorkingHEXfile);
end;

procedure OpenBINfile;
begin
assignFile(WorkingBINfile,WorkingFilename);
reset(WorkingBINfile);
end;

procedure CloseBINfile;
begin
closefile(WorkingBINfile);
end;

procedure foundError;
begin
messageDlg('Hardware Error!', mtError, [mbOK], 0);
end;

function countTotalHexByte(kghFilename : String): word;
var totalbyte:word;
    tmpKGHfile : file of hexfiletype;
    tmpHEX : hexfiletype;
begin
totalbyte:=0;
assignFile(tmpKGHfile, kghFilename);
reset(tmpKGHfile);
while not(eof(tmpKGHfile)) do
      begin
      read(tmpKGHfile,tmpHEX);
      totalbyte:=totalbyte+tmpHEX.byteCount;
      end;
closefile(tmpKGHfile);
countTotalHEXbyte := totalbyte;
end;

function countTotalBinByte(kgbFilename : string):word;
var tmpKGBfile : file of byte;
begin
assignfile(tmpKGBfile, kgbFilename);
reset(tmpKGBfile);
countTotalBinByte := filesize(tmpKGBfile);
closefile(tmpKGBfile);
end;

//=============================================================
procedureWrite40pin(CmdSendOut:byte);
var total_byte : Word;
    i : word;
    tmpAddr : String;
label EndThisProcess;
begin
total_byte := countTotalHEXbyte(WorkingFilename);
Form1.Gauge1.Progress := 0;
Form1.Gauge1.MaxValue := Total_Byte;
str(Total_Byte, tmpAddr);
Form1.lblByteCount.Caption := tmpAddr;
OpenHEXfile;
OpenCommPort;
if byteOutChk(170) then             //perform Hardware status check
   begin
   Form1.lblHWstatus.Caption := 'OK!';
   Form1.Refresh;
   end
   else
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(100);
if not(byteOutChk(CmdSendOut)) then    //send code
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(3000);
while not(eof(WorkingHEXFile)) do
      begin
      read(WorkingHEXFile, tmpHEX);
      tmpHEXbyteCount := tmpHEX.byteCount;
      Form1.ProgressBar1.Position := 0;
      Form1.ProgressBar1.Max := tmpHEXbyteCount;
      tmpHEXaddrHi := tmpHEX.addrHiByte;
      str(tmpHEXaddrHi, tmpAddr);
      Form1.lblBeginAddr.Caption := tmpAddr;
      tmpHEXaddrLo := tmpHEX.addrLoByte;
      str(tmpHEXaddrLo, tmpAddr);
      Form1.lblBeginAddr.Caption := Form1.lblBeginAddr.Caption + tmpAddr;
      Form1.Refresh;
      for i:=1 to 16 do tmpHEXdata[i]:=tmpHEX.data[i];
      if not(byteOutChk(tmpHEXbyteCount)) then        //send bytecount
         begin
         foundError;
         goto EndThisProcess;
         end;
      sleep(100);
      if not(byteOutChk(tmpHEXaddrHi)) then           //send addr hi
         begin
         foundError;
         goto EndThisProcess;
         end;
      sleep(100);
      if not(byteOutChk(tmpHEXaddrLo)) then           //send addr lo
         begin
         foundError;
         goto EndThisProcess;
         end;
      sleep(100);
      if not(byteOutChk(170)) then                    //send free byte
         begin
         foundError;
         goto EndThisProcess;
         end;
      sleep(100);
```

```
      for i:=1 to tmpHEXbyteCount do
          if not(byteOutChk(tmpHEXdata[i])) then      //send data
             begin
             foundError;
             goto EndThisProcess;
             end
             else
             begin
             Form1.ProgressBar1.StepIt;
             Form1.Gauge1.AddProgress(1);
             Form1.Refresh;
             end;
       end;
if not(byteOutChk(00)) then                          //send 00 to end.
   begin
   foundError;
   goto EndThisProcess;
   end;
EndThisProcess:
closeCommPort;
closeHEXfile;
end;
//-------------------------------------------------------
procedure Write20pin(CmdSendOut:byte);
var total_byte : Word;
    tmpTotalByte : Word;
    tmpAddr : String;
    i:byte;
label EndThisProcess;
begin
total_byte := countTotalBINbyte(WorkingFilename);
str(total_Byte, tmpAddr);
Form1.lblByteCount.Caption := tmpAddr;
Form1.Gauge1.Progress := 0;
Form1.ProgressBar1.Position := 0;
Form1.ProgressBar1.Max := Total_Byte;
Form1.Gauge1.MaxValue := Total_Byte;
Form1.Refresh;
OpenBINfile;
OpenCommPort;
if byteOutChk(170) then              //perform Hardware status check
   begin
   Form1.lblHWstatus.Caption := 'OK!';
   Form1.Refresh;
   end
   else
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(10);
if not(byteOutChk(CmdSendOut)) then  //send code
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(3000);
//---------------begin loop
while not(eof(WorkingBinFile)) do
begin
if Total_Byte <= 255 then TmpTotalByte := Total_Byte
  else
      begin
      Total_Byte := Total_Byte - 255;
      TmpTotalByte := 255;
      end;
if not(byteOutChk(TmpTotalByte)) then  //send total bytes
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(10);
if not(byteOutChk(170)) then          //send begin addr hi
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(10);
if not(byteOutChk(170)) then          //send begin addr lo
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(10);
Form1.lblBeginAddr.Caption := '00';
if not(byteOutChk(170)) then         //send free byte
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(10);
for i:=1 to 255 do
   if not(Eof(WorkingBINfile)) then
     begin
     read(WorkingBINFile, tmpBIN);         //read data from file byte by byte.
     if not(byteOutChk(tmpBIN)) then       //send data
        begin
        foundError;
        goto EndThisProcess;
        end
        else
          begin
          Form1.ProgressBar1.StepIt;
          Form1.Gauge1.AddProgress(1);
          Form1.Refresh;
          end;
      end;
end;  //end of while
if not(byteOutChk(00)) then                          //send 00 to end.
   begin
   foundError;
   goto EndThisProcess;
   end;
EndThisProcess:
CloseCommPort;
closeBINfile;
end;
//-------------------------------------------------------
procedure uCerase(CmdSendOut:byte);
label EndThisProcess;
begin
Form1.lblBeginAddr.Caption := '';
Form1.lblHEXfilename.Caption := '';
Form1.lblBeginAddr.Caption := '';
Form1.ProgressBar1.Position := 0;
Form1.Refresh;
OperCommPort;
if byteOutChk(170) then              //perform Hardware status check
   Form1.lblHWstatus.Caption := 'OK!'
   else
   begin
   foundError;
   goto EndThisProcess;
   end;
   sleep(100);
if not(byteOutChk(cmdSendOut)) then
   begin
   foundError;
   goto EndThisProcess;
   end;
WorkingComm.Output := chr(00);
Form1.Gauge1.MaxValue := 1;
Form1.Gauge1.Progress := 1;
Form1.Refresh;
EndThisProcess:
CloseCommPort;
end;
//-------------------------------------------------------
procedure uCread(CmdSehdOut:byte);
var SizeToRead : Word;
    TargetBINfile : file of Byte;
    TargetBinFileName : String;
    BINbuffer : byte;
    i:word;
label EndThisProcess;
begin
case WorkinguCtype of
     1: SizeToRead := 4096;
     2: SizeToRead := 8192;
     3: SizeToRead := 10240;
     4: SizeToRead := 8192;
     5: SizeToRead := 1024;
     6: SizeToRead := 2048;
     7: SizeToRead := 4096;
     else SizeToRead:=0;
     end;
Form1.lblByteCount.Caption := inttoStr(SizeToRead);
Form1.lblBeginAddr.Caption := '0000';
TargetBinFileName := ChangeFileExt(WorkingFileName, '.BIN');
assignFile(TargetBINfile, TargetBinFilename);
rewrite(TargetBINfile);
OpenCommPort;
if byteOutChk(170) then              //perform Hardware status check
   Form1.lblHWstatus.Caption := 'OK!'
   else
   begin
   foundError;
   goto EndThisProcess;
```

```
    end;
    sleep(100);
if not(byteOutChk(CmdSendOut)) then
    begin
    foundError;
    goto EndThisProcess;
    end;
    sleep(3000);
Form1.ProgressBar1.Max := SizeToRead;
Form1.Gauge1.MaxValue := SizeToRead;
Form1.ProgressBar1.Position := 0;
Form1.Gauge1.Progress := 0;
for i:=1 to SizeToRead do
    begin
    BINbuffer := byteIN;
    Write(TargetBINfile, BINbuffer);
    Form1.ProgressBar1.StepIt;
    Form1.Gauge1.AddProgress(1);
    Form1.Refresh;
    end;
WorkingComm.Output := chr(00);
//byteOutChk(00);
EndThisProcess:
CloseCommPort;
closefile(TargetBINfile);
ConvBinToHex(TargetBinFileName);
end;
{=============================================================}
Procedure openCommPort;
begin                                   //open Comm function
WorkingComm.CommPort := CommPort;                //Assign CommPort Number
WorkingComm.Settings := inttostr(CommBaud)+',n,8,1';  //Choose Comm Baudrate
WorkingComm.Handshaking := 0;
if WorkingComm.PortOpen then                    //if port altrady opened then close it.
   WorkingComm.PortOpen := False
else                                    //or open it.
   WorkingComm.PortOpen := True;
end;
{=====================================}
Procedure closeCommPort;
begin                                   //close Comm function
//WorkingComm.CommPort := CommPort;                //Assign CommPort Number
WorkingComm.PortOpen := False;                  //Close Comm whatever it opened or not.
end;
{=====================================}
function byteOutChk(outByte : Byte) : Boolean;
var byteRead : String;
    TickCount : Word;
    commNoData : boolean;                   //mean Not any data waiting in buffer.
label CheckError;
begin
TickCount := 0;
Workingcomm.Output := chr(outByte);             //Put a byte to Output.
commNoData := True;
while commNoData do                     //loop for checking for data input to buffer
    begin                               //if not any data come yet then keep looping.
    if (TickCount>100) then //if HW not response within 100 millisecs
       begin                    //mean there Error occur!!!
       byteOutchk := False;
       goto CheckError;
       end
    else
    if Workingcomm.InBufferCount<>0 then
       begin
       byteRead := WorkingComm.Input;       //read input buffer to var byteRead.
       commNoData := False;                 //still no data return from HW.
       end
       else
       commNoData := True;                  //some data return from HW.
       sleep(1);
       inc(TickCount);
    end;
if ord(byteRead[1]) = outByte then          //read only first byte of input as response of HW.
   byteOutChk := True                       //if response = output then return 'TRUE' mean send
   successful.
   else
   byteOutChk := False;                     //if not return 'FALSE' mean send fail.
CheckError:
end;
{=====================================}
function byteIn : Byte;
var byteRead : String;
    commNoData : boolean;
begin
commNoData := True;
WorkingComm.Output := chr(170);
while commNoData do
      begin
      if Workingcomm.InBufferCount<>0 then
         begin
         commNoData := False;
         byteRead := Workingcomm.Input; //read byte from Hardware
         end
         else
         commNoData := True;
      end;
byteIn := ord(byteRead[1]);    //use only first byte.
//WorkingComm.Output := byteRead[1]; //return read data to Hardware to send next byte
end;
{=====================================}
procedure initCommPort;
var SettingReg : TRegistry;
begin
SettingReg := TRegistry.Create;                 //Create Registry Object
try                                             //begin of 'try'
   SettingReg.RootKey := HKEY_LOCAL_MACHINE;
   if SettingReg.OpenKey('software\KRY',False) then //Key Open Successful
      begin
      WorkingCommPort := SettingReg.ReadInteger('Port'); //read CommPort from Registry
      WorkingCommBaud := SettingReg.ReadInteger('Baudrate'); //read CommBaud from Registry
      end;
finally
   SettingReg.CloseKey;
   SettingReg.Free;
end;                                            //end of 'try'
end;
end.
```

```
unit fileWorkshop;

interface
type
    hexFileType = Record
      byteCount : byte;
      addrHiByte : byte;
      addrLoByte : byte;
      data : array[1..16] of byte;
    end;
function PrepareFile(hexFilename : String; uCtype : byte):String;
function makeKghFile(hexFilename :String) : String;
function createBinFile(kghFileName : String) : String;
function SortHexFile(inFileName:string) : String;
function calcAddr(HiByte, LoByte : Byte):word;
procedureConvBINtoHEXFT(TheFileName:String);
ProcedureConvHEXFTtoHEX(TheFileName:String);
Procedure ConvBINtoHEX(TheFileName : String);
Procedure BINfilter(TheBinFileName : String);
implementation
uses hexWorkShop, SysUtils;

function makeKghFile(hexFilename : String):String;
var hexFile : TextFile;
    hexTmpFile : File of hexFileType;
    hexTmpFileName : String;
    hexTmpRec : hexFileType;
    tmpData : ShortString;
    tmpHexString : String;
    count : byte;
begin
for count:=1 to 16 do hexTmpRec.data[count]:=0;
tmpData := '';
tmpHexString := '';
assignfile(hexFile,hexFileName);
hexTmpFileName := ExtractFileDir(hexFileName)+'\hexTmpFile.kgh';
assignfile(hexTmpFile, hexTmpFileName);
reset(hexFile);        //open hexFile
rewrite(hexTmpFile);   //create hexTmpFile
while not(eof(hexFile)) do
  begin
  readln(hexFile, tmpHexString); //read hexString
  tmpHexString:=StringReplace(tmpHexString,' ','',[rfReplaceAll]); //replace space with nospace
  if ((Length(tmpHexString)<>0) and (tmpHexString<>':00000001FF')) then
  begin
  sepaHex(tmpHexString, hexTmpRec.byteCount, hexTmpRec.addrHiByte, hexTmpRec.addrLoByte, tmpData);
  //sepate hexString to --> bytecount(XX) HiAddress(XX) LoAddress(XX) datastring(XXXX.....XXXX) with no
  checksum
  if (Length(tmpData) div 2)=hexTmpRec.byteCount then //number of byte in datastring must equal to
  ByteCount
    for count:=1 to hexTmpRec.byteCount do
      hexTmpRec.data[count]:=sepaByte(tmpData); //separate 1 byte from datastring
  write(hexTmpFile,hexTmpRec); //save Record to hexTmpFile
  for count:=1 to 16 do hexTmpRec.Data[count]:=0; //clear Record.Data
  end;
  end;
CloseFile(hexFile);
CloseFile(hexTmpFile);
makeKghFile := hexTmpFileName;  //return filename.
end;

function createBinFile(kghFileName : String) : String;
var kgbTmpFile : File of byte;
    kgbTmpFileName : String;
    nextAddr : Word;
    readAddr : Word;
    kghTmpFile : File of hexFileType;
    kghTmpRec : hexFileType;
    count : Byte;
    blankByte : Byte;
begin
nextAddr := 0; //begin of address
assignfile(kghTmpFile,kghFileName);
kgbTmpFileName := ExtractFileDir(kghFileName) + '\binTmpFile.kgb';
assignfile(kgbTmpFile, kgbTmpFileName);
reset(kghTmpFile); //open KGH file
rewrite(kgbTmpFile); //create KGB file
while not(eof(kghTmpFile)) do
  begin
  Read(kghTmpFile, kghTmpRec); //read a record from KGH file
  readAddr:=(kghTmpRec.addrHiByte*256)+kghTmpRec.addrLoByte;
  if nextAddr=readAddr then //if next address equal to read address
    begin                   //write data of record to file
      for count:=1 to kghTmpRec.byteCount do
          write(kgbTmpFile,kghTmpRec.Data[count]);
      nextAddr:=nextAddr + kghTmpRec.byteCount;
      end
  else                      //if next address is less than read data
      begin                 //fill file with 'FF' to read data
      while nextAddr<>readAddr do
        begin
        BlankByte:=255;
        write(kgbTmpFile,BlankByte);
        inc(nextAddr);
        end;                //next address is equal to read data
      for count:=1 to kghTmpRec.byteCount do //write data of record to file
        write(kgbTmpFile,kghTmpRec.Data[Count]);
      nextAddr:=nextAddr + kghTmpRec.byteCount;
      end;
  end;
closefile(kghTmpFile);
closefile(kgbTmpFile);
createBinFile := kgbTmpFileName;  //return KGB filename
end;

function SortHexFile(inFileName:string) : String;
  function FindAddr(item:hexFileType) : word;
  begin
  findaddr:=(item.addrHiByte*256) + item.addrLoByte;
  end;
var kghFile0 : File of hexFileType;
    hexTmpRec0 : hexFileType;
    kghFileArray : array[1..128] of hexFileType;
    count : byte;
    i,j,k : byte;
begin
assignFile(kghFile0, inFilename);
reset(kghFile0);
count:=0;
while not(eof(kghFile0)) do
  begin
  read(kghFile0, hexTmpRec0);          //read data from file
  inc(count);                          //count++
  kghFileArray[count]:=hexTmpRec0;     //store data in array[count]
  end;                                 //all data done.
closefile(kghFile0);
for i:= 1 to count-1 do //begin of sort....
  begin
  k:=i;
  hexTmpRec0:=kghFileArray[i];
  for j:=i+1 to count do
    if findaddr(kghFileArray[j])<findaddr(hexTmpRec0) then
      begin
      k:=j;
      hexTmpRec0:=kghFileArray[j];
      end;
  kghFileArray[k]:=kghFileArray[i];
  kghFileArray[i]:=hexTmpRec0;
  end;                 //sort finished...
assignFile(kghFile0, inFilename);
rewrite(kghFile0);                     //overwrite old KGH file
for i:= 1 to count do write(kghFile0, kghFileArray[i]);
closefile(kghFile0);
sortHexFile := inFilename;     //return input KGH filename
end;

function PrepareFile(hexFilename : String; uCtype : byte):String;
// return filename of ready to use hexfile or binfile;
var tmpFileName:string;
begin
case uCtype of
1,2,3,4 : begin //selected microcontroller is 40 pins type
          tmpFileName := makeKghFile(hexFileName); //make KGH file
          PrepareFile := sortHexFile(tmpFilename); //sort KGH file
          end;
5,6,7   : begin //selected microcontroller is 20 pins type
          tmpFileName := makeKghFile(hexFileName); //make KGH file
          tmpFileName := sortHexFile(tmpFileName); //sort KGH file
          PrepareFile := createBinFile(tmpFileName); //make BIN file
          end;
end; //end of case
end;

function calcAddr(HiByte, LoByte : Byte):word;
  begin
  calcaddr:=(HiByte*256) + LoByte;
  end;
//-------------------------------------------------------------------------
ProcedureConvBINtoHEXFT(TheFileName:String);
//input as '.HEX' file name
Var BinTmpFile : File of Byte;
    BinTmpFileName : String;
    BinTmp : Byte;
    HexTmpFile : File of HEXfileType;
    HexTmpFileName : String;
    HexTmpRec  : hexFileType;
    i,j : byte;
    Addr : Word;
```

```
label ExitLoop;
begin
Addr := 0; j:=0;
HexTmpFileName := ChangeFileExt(TheFileName, '.KGH');
BinTmpFileName := ChangeFileExt(TheFileName, '.BIN');
AssignFile(HexTmpFile, HexTmpFileName);
AssignFile(BinTmpFile, BinTmpFileName);
ReWrite(HexTmpFile);
Reset(BinTmpFile);
While not(Eof(BinTmpFile)) do
   begin
   HexTmpRec.addrHiByte := Addr div 256;
   HexTmpRec.addrLoByte := Addr mod 256;
   for i:=1 to 16 do
      if not(Eof(BinTmpfile)) then
         begin
         Read(BinTmpFile,BinTmp);
         HexTmpRec.data[i] := BinTmp;
         j:=i;
         end
      else
         begin
         Goto ExitLoop;
         j:=i;
         end;
   ExitLoop:  //for loop ended.
   Addr := Addr + j;
   HexTmpRec.byteCount := j;
   Write(HexTmpFile, HexTmpRec);
   for i:=1 to 16 do HexTmpRec.Data[i] := 00;
   end; //end while loop
closeFile(HexTmpFile);
closeFile(BinTmpfile);
end;

ProcedureConvHEXFTtoHEX(TheFileName:String);
//input as '.KGH' File name
   function FindCheckSum(inHexRec : HexFileType):byte;
   var SumOfData : Word;
       i         : byte;
   begin
   SumOfData := 0;
   SumOfData := SumOfData + inHexRec.byteCount;
   SumOfData := SumOfData + inHexRec.addrHiByte;
   SumOfData := SumOfData + inHexRec.addrLoByte;
   for i:=1 to 16 do SumOfData := SumOfData + inHexRec.data[i];
   SumOfData := not(SumOfData);
   i := byte(SumOfData);
   FindCheckSum := i+1;
   end;
   function CHexStr(inByte : byte):ShortString;
      function CHexByte(inlByte : Byte):Char;
      begin
      case inlByte of
           0: CHexByte:='0';
           1: CHexByte:='1';
           2: CHexByte:='2';
           3: CHexByte:='3';
           4: CHexByte:='4';
           5: CHexByte:='5';
           6: CHexByte:='6';
           7: CHexByte:='7';
           8: CHexByte:='8';
           9: CHexByte:='9';
           10: CHexByte:='A';
           11: CHexByte:='B';
           12: CHexByte:='C';
           13: CHexByte:='D';
           14: CHexByte:='E';
           15: CHexByte:='F';
      else CHexByte := '0';
           end;
      end;
   var X,Y : Byte;
   begin
   X := inByte mod 16;
   Y := inByte div 16;
   CHexStr := CHexByte(Y)+CHexByte(X);
   end;
   procedure ChangeByteCount(var inHexRec : HexFileType);
   var i,j:byte;
   begin
   i:= inHexRec.byteCount+1;
   repeat
   dec(i);
   j:= inHexRec.data[i];
   if j=255 then inHexRec.data[i]:=0;
   until j<>255;
   inHexRec.byteCount := i;
   end;

Var HEXfile        : TextFile;
    HEXfilename    : String;
    HEXString      : ShortString;
    CheckSum       : Byte;
    HexTmpFile     : File Of HexFileType;
    HexTmpFileName : String;
    HexTmpRec      : HexFileType;
    i              : byte;
    DataProduct    : byte;
begin
HexFileName := ChangeFileExt(TheFileName, '.HEX');
HexTmpFileName := ChangeFileExt(TheFileName, '.KGH');
AssignFile(HexFile, HexFileName);
AssignFile(HexTmpFile, HexTmpFileName);
ReWrite(HexFile);
ReSet(HexTmpFile);
WriteLn(HexFile);
While not(eof(HexTmpFile)) do
   begin
   Read(HexTmpFile, HexTmpRec);
   DataProduct := 255;
   for i:=1 to HexTmpRec.ByteCount do DataProduct := DataProduct AND HexTmpRec.data[i];
   if DataProduct <> 255 then
      begin
      if HexTmpRec.data[HexTmpRec.byteCount] = 255 then ChangeByteCount(HexTmpRec);
      HEXString := ':';
      HexString := HexString + CHexStr(HexTmpRec.byteCount);
      HexString := HexString + CHexStr(HexTmpRec.addrHiByte);
      HexString := HexString + CHexStr(HexTmpRec.addrLoByte);
      HexString := HexString + '00';
      for i:= 1 to HexTmpRec.ByteCount do HexString := HexString + CHexStr(HexTmpRec.data[i]);
      CheckSum := FindCheckSum(HexTmpRec);
      HexString := HexString + CHexStr(CheckSum);
      WriteLn(HexFile, HexString);
      end;
   end;
CloseFile(HexTmpFile);
WriteLn(HexFile, ':00000001FF');
CloseFile(HexFile);
end;

Procedure BINfilter(TheBinFileName : String);
var BinTmpFileName : String;
    BinTmpFile     : File of Byte;
    BinFileSize    : Word;
    i              : Word;
    StillFF        : Boolean;
    BinTmp         : Byte;
begin
BinTmpFileName := ChangeFileExt(TheBinFileName, '.BIN');
assignFile(BinTmpFile, BinTmpFileName);
Reset(BinTmpFile);
BinFileSize := FileSize(BinTmpFile);
StillFF := True;
i:=1;
While StillFF do
      begin
      Seek(BinTmpfile, BinFileSize-i);
      Read(BinTmpFile, BinTmp);
      if BinTmp = 255 then inc(i)
         else
         begin
         Truncate(BinTmpFile);
         StillFF:=False
         end;
      end;
closefile(BinTmpFile);
end;

Procedure ConvBINtoHEX(TheFileName : String);
begin
BINFilter(TheFileName);
ConvBinToHexFT(TheFileName);
ConvHexFTtoHex(TheFileName);
end;
end.
```

```
unit hexWorkshop;

interface

uses SysUtils;

function strTohex(inString : shortString) : Longword;
//strTohex use for change HEX-string format to its value in integer.
function shopStr(var inString:ShortString; chrToCut: byte): shortString;
//shopStr use for return number of char from begin of string and cut it from string.
procedure sepaHEX( inString : ShortString;
                   var byteCount : byte;
                   var addrHiByte : byte;
                   var addrLoByte : byte;
                   var data : ShortString);
//sepaHEX use for separate HEX format to data need.
//HEX format ==> :BB AAAA TT XXXX...XXXX CC
// : = begin of line <==this no use.
// BB = bytecount <==use this! return as 'byteCount'
// AAAA = begin of address for this data <==use this! return as 'addrHiByte' and 'addrLoByte'
// TT = data type <==this no use.
// XXXX... = data <==use this! return as 'data'
// CC = checksum <==this no use.
function sepaByte(var inString : ShortString):byte;
// sepaByte use for separate byte value from data-string.

implementation
//=============================================================================
function strTohex(inString : shortString) : Longword;
var count:byte;
    outWord : Longword;
    outByte : byte;

begin
outByte := 0;
outWord := 0;
for count := 1 to length(inString) do
   begin
   case upcase(inString[count]) of
   '0','1','2','3','4','5','6','7','8','9' : outByte := byte(strtoint(inString[count]));
   'A': outByte := 10;
   'B': outByte := 11;
   'C': outByte := 12;
   'D': outByte := 13;
   'E': outByte := 14;
   'F': outByte := 15;
   end;
   outWord := outWord*16 + outByte;
   end;
strTohex:=outWord;
end;
//=============================================================================
function shopStr(var inString:ShortString;chrToCut: byte): shortString;
begin
shopStr:=copy(inString,1,chrToCut);
delete(inString,1,chrToCut);
end;
//=============================================================================
procedure sepaHEX(inString : ShortString;
                  var byteCount : byte;
                  var addrHiByte : byte;
                  var addrLoByte : byte;
                  var data : ShortString);
begin
inString:=StringReplace(inString, ':', '', [rfReplaceAll]); //remove :
Delete(inString, Length(inString)-1, 2); //delete checksum
byteCount := strtohex(shopstr(inString,2));
addrHiByte := strtohex(shopstr(inString,2));
addrLoByte := strtohex(shopstr(inString,2));
delete(inString,1,2); //remove datatype
data := shopstr(inString,byteCount*2);
end;
//=============================================================================
function sepaByte(var inString : ShortString):byte;
begin
if not(odd(length(inString))) then
sepaByte:=strtohex(shopstr(inString,2))
else
sepaByte:=0;
end;
//=============================================================================
end.
```

Features

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

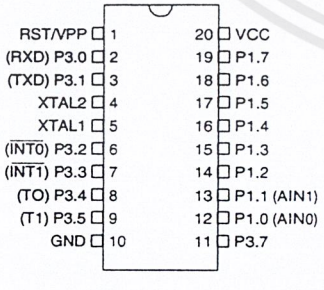
Description

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

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

Pin Configuration

PDIP/SOIC



8-bit Microcontroller with 2K Bytes Flash

AT89C2051

Rev. 0368E–02/00

Programming The Flash

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

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

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

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

To Program and Verify the Array:

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

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

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

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

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

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

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

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

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

Programming Interface

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

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

Flash Programming Modes

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

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

Figure 3. Programming the Flash Memory

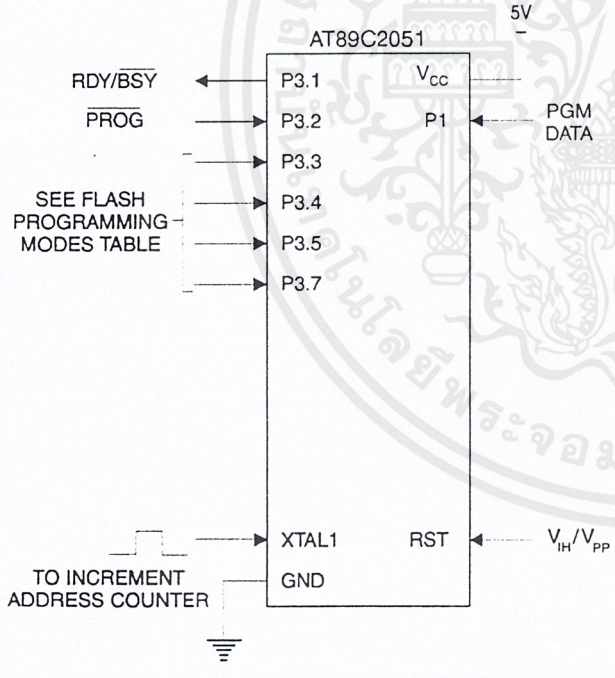
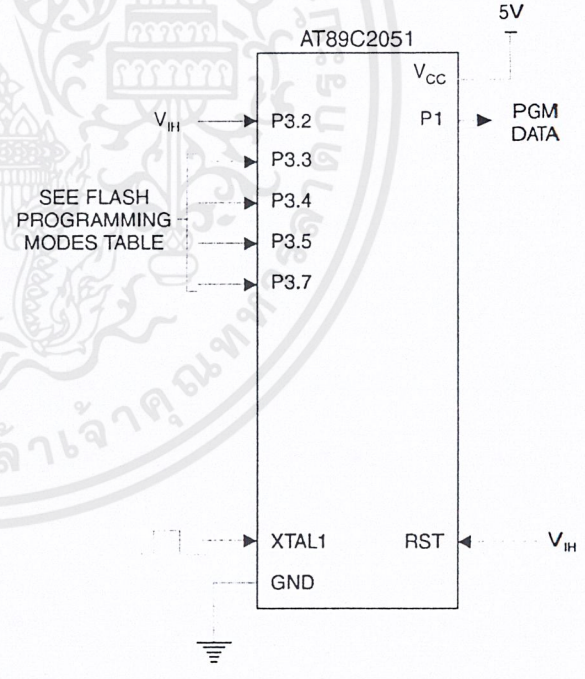


Figure 4. Verifying the Flash Memory



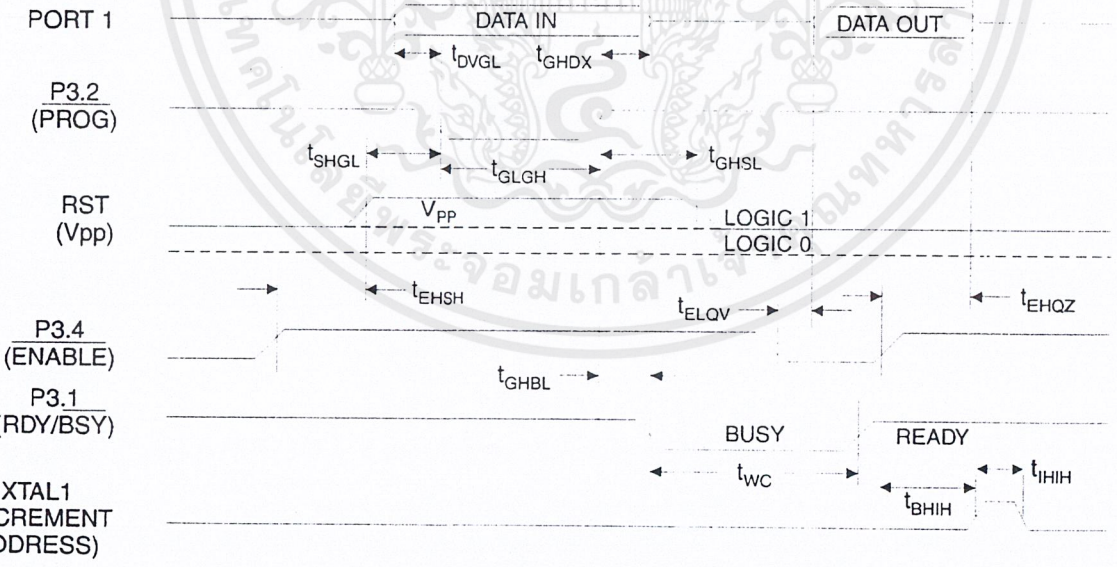
Flash Programming and Verification Characteristics

T_A = 0°C to 70°C, V_{CC} = 5.0 ± 10%

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

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

Flash Programming and Verification Waveforms



Absolute Maximum Ratings*

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

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

DC Characteristics

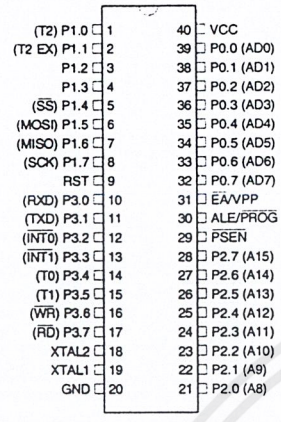
T_A = -40°C to 85°C, V_{CC} = 2.0V to 6.0V (unless otherwise noted)

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

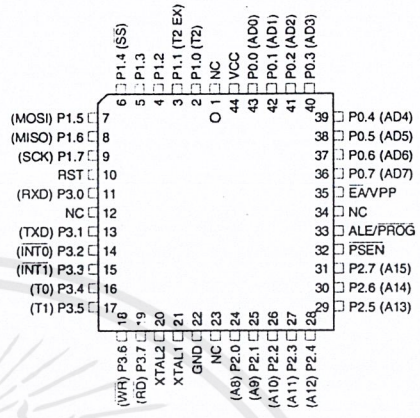
Notes:
1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
   Maximum I_{OL} per port pin: 20 mA
   Maximum total I_{OL} for all output pins: 80 mA
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed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

Pin Configurations

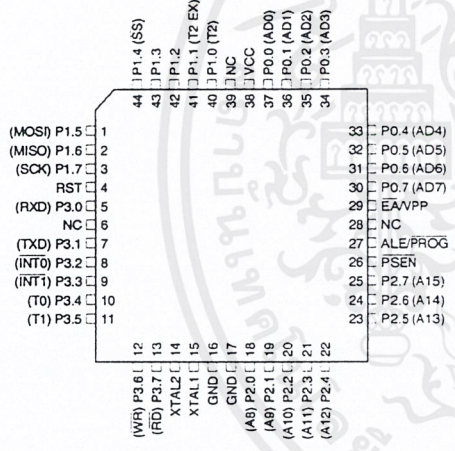
PDIP



PLCC



PQFP/TQFP



Pin Description

VCC

Supply voltage.

GND

Ground.

Port 0

Port 0 is an 8-bit open drain bbi-didirectional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 can also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode, P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Features

- Compatible with MCS-51™ Products
- 8K Bytes of In-System Reprogrammable Downloadable Flash Memory
  - SPI Serial Interface for Program Downloading
  - Endurance: 1,000 Write/Erase Cycles
- 2K Bytes EEPROM
  - Endurance: 100,000 Write/Erase Cycles
- 4V to 6V Operating Range
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 256 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Three 16-bit Timer/Counters
- Nine Interrupt Sources
- Programmable UART Serial Channel
- SPI Serial Interface
- Low-power Idle and Power-down Modes
- Interrupt Recovery From Power-down
- Programmable Watchdog Timer
- Dual Data Pointer
- Power-off Flag

Description

The AT89S8252 is a low-power, high-performance CMOS 8-bit microcomputer with 8K bytes of downloadable Flash programmable and erasable read only memory and 2K bytes of EEPROM. The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard 80C51 instruction set and pinout. The on-chip downloadable Flash allows the program memory to be reprogrammed in-system through an SPI serial interface or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with downloadable Flash on a monolithic chip, the Atmel AT89S8252 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

The AT89S8252 provides the following standard features: 8K bytes of downloadable Flash, 2K bytes of EEPROM, 256 bytes of RAM, 32 I/O lines, programmable watchdog timer, two data pointers, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry. In addition, the AT89S8252 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset.

The downloadable Flash can be changed a single byte at a time and is accessible through the SPI serial interface. Holding RESET active forces the SPI bus into a serial programming interface and allows the program memory to be written to or read from unless Lock Bit 2 has been activated.

ATMEL

8-bit Microcontroller with 8K Bytes Flash

AT89S8252

Rev. 0401E–02/00

Idle Mode

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

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

Status of External Pins During Idle and Power-down Modes

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power-down	Internal	0	0	Data	Data	Data	Data
Power-down	External	0	0	Float	Data	Data	Data

Power-down Mode

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

To exit power-down via an interrupt, the external interrupt must be enabled as level sensitive before entering power-down. The interrupt service routine starts at 16 ms (nominal) after the enabled interrupt pin is activated.

Program Memory Lock Bits

The AT89S8252 has three lock bits that can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the following table.

When lock bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value and holds that value until reset is activated. The latched value of $\overline{EA}$ must agree with the current logic level at that pin in order for the device to function properly.

Once programmed, the lock bits can only be unprogrammed with the Chip Erase operations in either the parallel or serial modes.

Lock Bit Protection Modes[1][2]

Program Lock Bits				
	LB1	LB2	LB3	Protection Type
1	U	U	U	No internal memory lock feature.
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory. $\overline{EA}$ is sampled and latched on reset and further programming of the Flash memory (parallel or serial mode) is disabled.
3	P	P	U	Same as Mode 2, but parallel or serial verify are also disabled.
4	P	P	P	Same as Mode 3, but external execution is also disabled.

Notes: 1. U = Unprogrammed
2. P = Programmed

Flash and EEPROM Parallel Programming Modes

Mode	RST	$\overline{PSEN}$	ALE/$\overline{PROG}$	$\overline{EA}$/V_{PP}	P2.6	P2.7	P3.6	P3.7	Data I/O P0.7:0	Address P2.5:0 P1.7:0
Serial Prog. Modes	H	h(1)	h(1)	x						
Chip Erase	H	L	⌐ (2)	12V	H	L	L	L	X	X
Write (10K bytes) Memory	H	L	⌐	12V	L	H	H	H	DIN	ADDR
Read (10K bytes) Memory	H	L	H	12V	L	L	H	H	DOUT	ADDR
Write Lock Bits:	H	L	⌐	12V	H	L	H	L	DIN	X
Bit - 1									P0.7 = 0	X
Bit - 2									P0.6 = 0	X
Bit - 3									P0.5 = 0	X
Read Lock Bits:	H	L	H	12V	H	H	L	L	DOUT	X
Bit - 1									@P0.2	X
Bit - 2									@P0.1	X
Bit - 3									@P0.0	X
Read Atmel Code	H	L	H	12V	L	L	L	L	DOUT	30H
Read Device Code	H	L	H	12V	L	L	L	L	DOUT	31H
Serial Prog. Enable	H	L	⌐ (2)	12V	L	H	L	H	P0.0 = 0	X
Serial Prog. Disable	H	L	⌐ (2)	12V	L	H	L	H	P0.0 = 1	X
Read Serial Prog. Fuse	H	L	H	12V	H	H	L	H	@P0.0	X

Notes:
1. "h" = weakly pulled "High" internally.
2. Chip Erase and Serial Programming Fuse require a 10 ms $\overline{PROG}$ pulse. Chip Erase needs to be performed first before reprogramming any byte with a content other than FFH.
3. P3.4 is pulled Low during programming to indicate RDY/$\overline{BSY}$.
4. "X" = don't care

Figure 13. Programming the Flash/EEPROM Memory

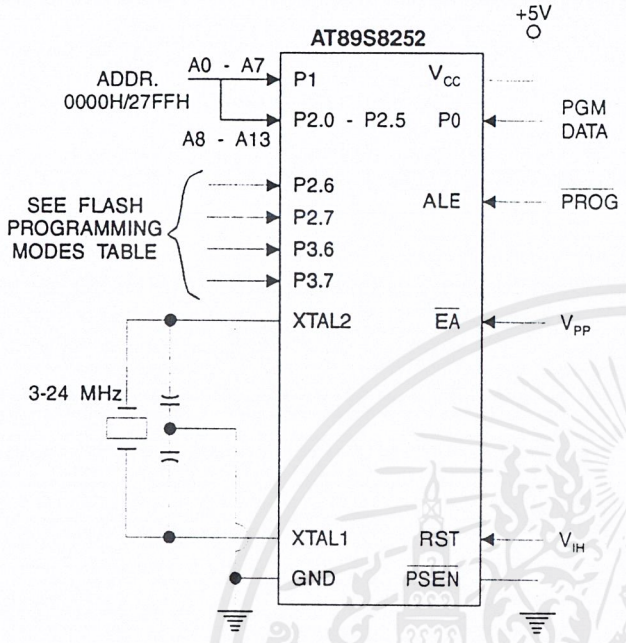


Figure 14. Verifying the Flash/EEPROM Memory

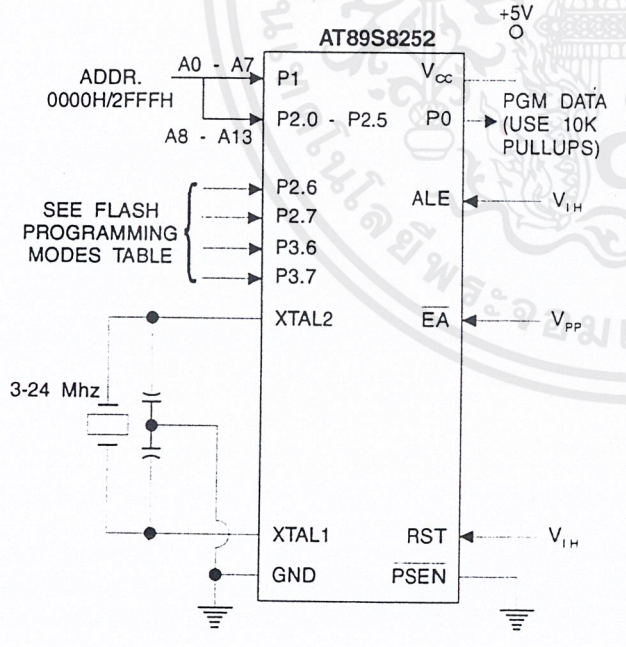
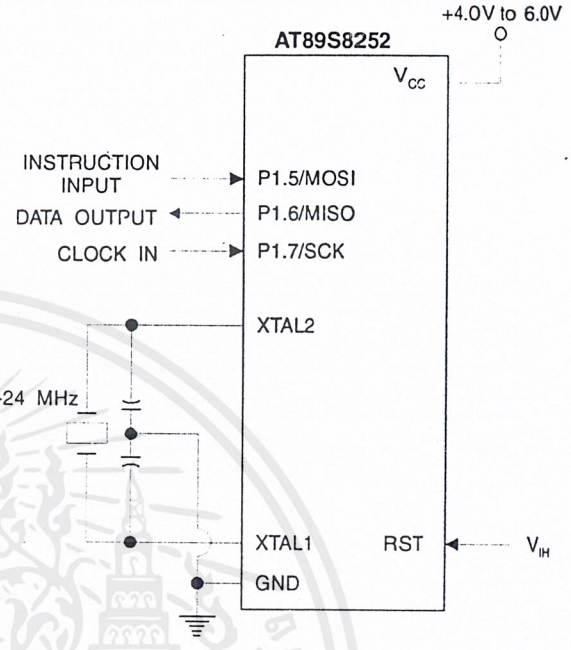


Figure 15. Flash/EEPROM Serial Downloading

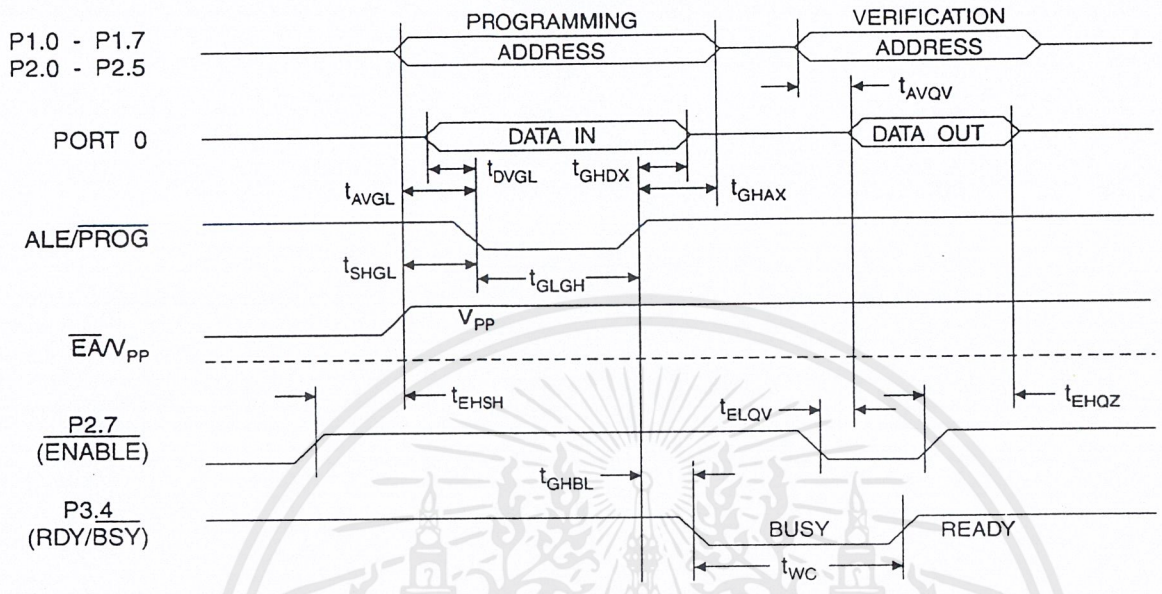


Flash Programming and Verification Characteristics – Parallel Mode

T_A = 0°C to 70°C, V_{CC} = 5.0V ± 10%

Symbol	Parameter	Min	Max	Units
V_{PP}	Programming Enable Voltage	11.5	12.5	V
I_{PP}	Programming Enable Current		1.0	mA
$1/t_{CLCL}$	Oscillator Frequency	3	24	MHz
t_{AVGL}	Address Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHAX}	Address Hold after $\overline{PROG}$	$48t_{CLCL}$		
t_{DVGL}	Data Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHDX}	Data Hold after $\overline{PROG}$	$48t_{CLCL}$		
t_{EHSH}	P2.7 ($\overline{ENABLE}$) High to V_{PP}	$48t_{CLCL}$		
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GLGH}	$\overline{PROG}$ Width	1	110	µs
t_{AVQV}	Address to Data Valid		$48t_{CLCL}$	
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		$48t_{CLCL}$	
t_{EHQZ}	Data Float after $\overline{ENABLE}$	0	$48t_{CLCL}$	
t_{GHBL}	$\overline{PROG}$ High to $\overline{BUSY}$ Low		1.0	µs
t_{WC}	Byte Write Cycle Time		2.0	ms

Flash/EEPROM Programming and Verification Waveforms – Parallel Mode



Serial Downloading Waveforms